Media Engineering and Technology Faculty
German University in Cairo

Audio and Video Indexing

Bachelor Thesis

Author: Alaa Hussein Moharram
Supervisor: Dr. Mohamed Elmahdy

Submission Date: 15 May, 2016

Media Engineering and Technology Faculty
German University in Cairo

Audio and Video Indexing

Bachelor Thesis

Author: Alaa Hussein Moharram
Supervisor: Dr. Mohamed Elmahdy

Submission Date: 15 May, 2016

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Alaa Hussein Moharram
15 May, 2016

Acknowledgments

First and foremost, I thank Allah (SWT) for all the blessings He bestowed upon me, and for giving me the ability and the strength to be where I am today.

I sincerely thank my supervisor, Dr. Mohamed Elmahdy, for the continuous guidance, assistance and help he offered me throughout the course of this work. The knowledge and expertise he generously shared with me were invaluable to the completion of this thesis.

I extend my deepest gratitude and appreciation to my parents for their unconditional love and support. And to my sisters, for their genuine love and their joyful presence.

I wholeheartedly thank all my friends for making my college years as fun and enjoyable as they were. A special thank you, however, goes to Sarah Desouky, Dalia Maarek and Alia ElShennawy, the best friends one could ever have, college would have never been the same without you.

A special thank you goes to my dear sister and friend, Ayah Tarek, for her continuous encouragement and her invaluable presence in my life. And to my mentor, Mrs. Azza Wanis, for her continuous motivation, her timely advice and her tender words.

Finally, this acknowledgment would not be complete without thanking the person whom his words continue to guide my path to this day, Dr. Tarek Elghandour, the forever remembered, the forever missed.

Abstract

The world is currently experiencing a fast-paced breakthrough in multimedia technologies and applications, where videos are becoming a vital part of the every day life. Consequently, the web is transformed from a text-only platform to a multimedia-rich platform that hosts all types of videos in all the different formats. With that arises the need to perform searches based on the videos' content, to be able to make use of this increasing amount of unstructured data. In this thesis, an audio and video indexing and retrieval system is proposed. The aim of the system is to help users search the videos' spoken content for certain words or phrases using regular text search queries, and is composed of two main modules: a video processor and a web application. The video processor is responsible for indexing the videos based on their audio information, where it processes a video's audio content and builds an index structure for the words spoken in the video. The web application then utilizes the generated index structure to answer the users search queries. The application provides an interface through which users submit their queries and then receive the list of related video matches.

Contents

Chapter 1

Introduction

1.1 Motivation

With the ever increasing advances in the development of multimedia technologies, the development of faster microprocessors and the availability of larger inexpensive storage capacities, the number of audio and video documents produced is increasing exponentially.
The web is currently hosting several thousands of audio and video files in all the different formats. From newscasts, webcasts, sports events, recordings of meetings. lectures and conferences to TV shows, movies and entertainment videos, the number of media content hosted on the web is only expected to increase aggressively.
And with that arises the need for preserving and organizing this voluminous media content, so that this large amount of unstructured data can be used efficiently in multimedia mining, multimedia analysis and multimedia searching.

1.2 Aim of Work

The aim of this project is to implement a content-based audio/video indexing and retrieval system that exploits recent speech recognition techniques and information retrieval algorithms.
The system would provide transcriptions for the speech contained in videos then using these generated transcripts, allow users to browse and search the videos' content.
The system is intended to be based on open source tools as mush as possible, and all scripts are written in Python programming language so that the system can be easily maintained and developed.

1.3 Thesis Outline

This thesis is structured as follows:
Chapter 2 gives a general overview of audio mining technology, its different techniques

currently in use and its different applications. *Chapter 3* gives a general overview of the system's design and workflow. *Chapter 4* discusses the system's first module, the transcoder module. It discusses the tools used and explains the module's workflow. *Chapter 5* discusses automatic speech recognition, gives a brief overview for the technology behind speech recognition and explains how this project's recognizer, CMU PocketSphinx, is used. *Chapter 6* explains the process of building an inverted index for the videos' transcriptions. *Chapter 7* explains how the different system components are integrated together in a web application to provide the user with the capability to search the videos' content. *Chapter 8* summarizes all the work presented in this thesis and discusses ways to improve and enhance the system.

Chapter 2

Background

2.1 Audio Mining

Audio mining, or audio indexing, is an approach to automatically analyze and search audio or video files for the presence of words or phrases in the spoken content [16]. It utilizes speech recognition techniques to construct a structured searchable index of spoken words and their locations.

In other words, audio mining answers text-based queries through locating the search term in the audio or the video file, giving users the ability to search audio and video files in a manner similar to that used for searching text files.

This section sheds some light on the two main audio mining techniques used, discusses the difference between them and gives an overview of some current audio mining applications.

2.1.1 Audio Mining Techniques

The main two audio mining techniques are: 1) the Large Vocabulary Continuous Speech Recognition (LVCSR) audio mining technique and 2) the phonetic audio mining technique. Both techniques represent a two step process; the first step, known as the preprocessing or the indexing phase, is the phase in which the audio mining system processes the audio data and generates an index file. The second step, known as the searching phase, is the phase in which the system uses its generated index file to search for search terms identified by the user. However, the main difference between both techniques is the type of index file generated, as discussed below.

2.1.1.1 LVCSR Audio Mining

LVCSR audio mining, also known as text-based indexing, is a technique in which the entire audio data is first converted to text then this text is used for searching the audio

content for occurrences of words or phrases.
Figure 2.1 presents a block diagram for the LVCSR audio mining process. In the first step, the audio data is passed through a large vocabulary speech recognizer that processes the speech content and produces an equivalent text transcription. This is achieved by first recognizing all the phonemes in the speech content being processed, after which the phonemes are recognized in a preconfigured dictionary of several thousand word entries and in the case of a word not being found, the system decides on the closest match to it, consequently creating a full text transcription for the audio data. A searchable index file which contains information about words spoken and their locations is then generated.
In the second step, a text search query, inputted by the user, is defined and matched against one or more index files using text-based searching algorithms and the results are displayed [16].

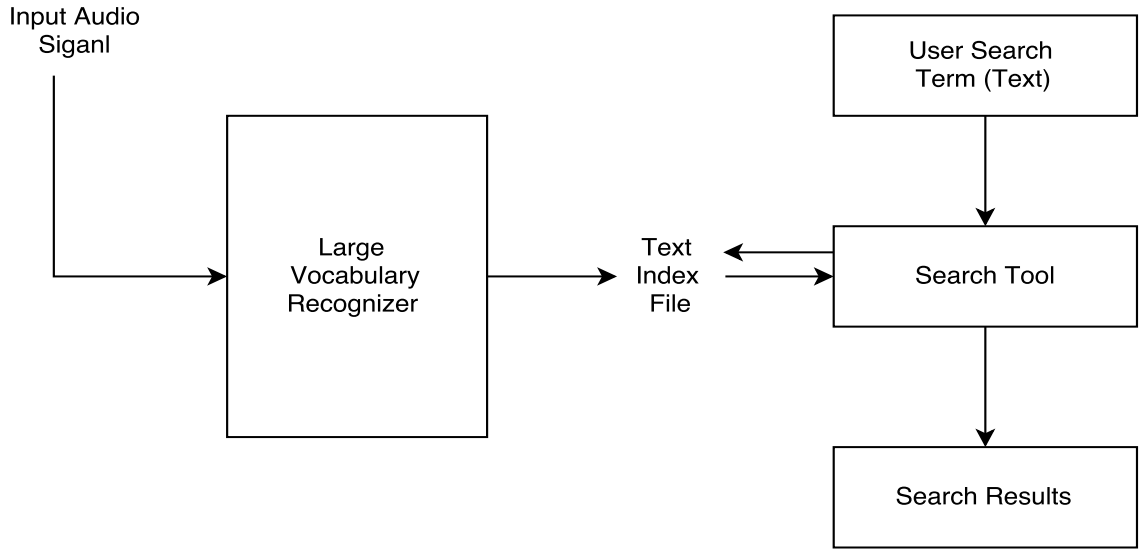

Figure 2.1: A typical LVCSR audio mining system diagram

2.1.1.2 Phonetic Audio Mining

Phonetic audio mining, also known as phonetic-based indexing, is a technique that, unlike the LVCSR technique, utilizes sounds only and does not require speech transcription into text.
In its first step, a phonetic audio mining system passes the audio data through a phonetic recognizer that analyzes and distinguishes sounds in the speech content of this data consequently generating a phonetic index file; an index file that contains information about the phonetic constituents of the audio data.
In the second step, a text search query inputted by the user is matched against a dictionary of several hundred phoneme entries to be converted to its equivalent phonetic string. The phonetic index file generated by the phonetic recognizer is then searched for

the occurrences of this phonetic string and results are displayed.
Figure 2.2 presents a block diagram for the phonetic audio mining process [16].

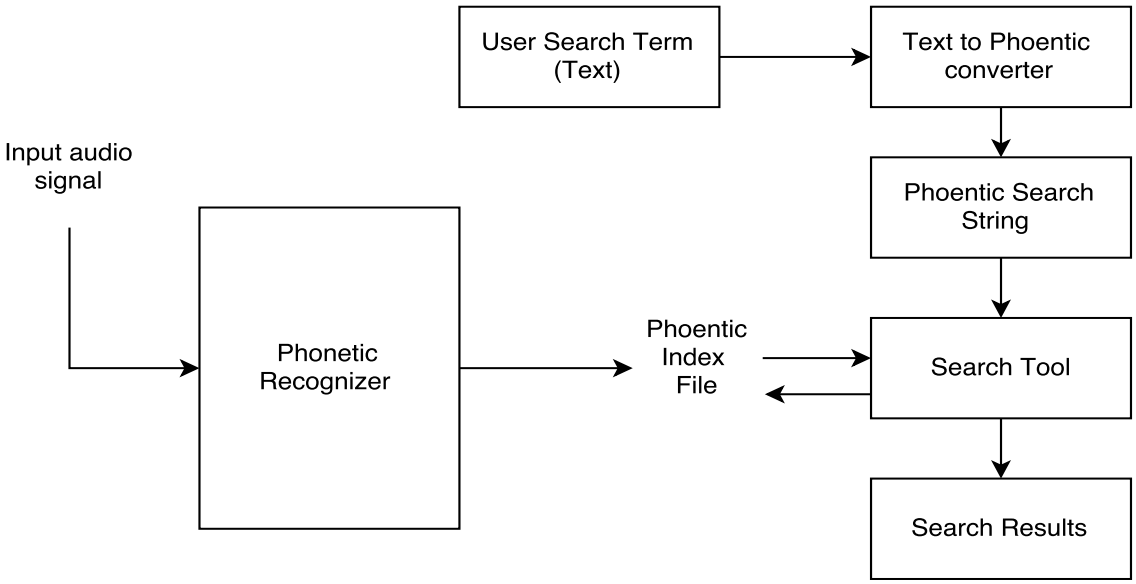

Figure 2.2: A typical phonetic audio mining system diagram

2.1.1.3 LVCSR Technique vs. Phonetic Technique

The variance with which the two previously discussed techniques approach audio mining results in noticeable advantages and disadvantages between them. Discussed below are the major differences between both techniques [19].

Indexing speed: Phonetic audio mining has significantly faster processing and indexing speed; since the dictionary used is that of language phonemes and there are only few tens of unique phonemes in most languages. On the other hand, the LVCSR technique uses large dictionaries of thousands of word entries, thus requiring much more computational power for audio processing and indexing and consequently slowing the process.

Searching speed: The fact that phonemes cannot be as efficiently indexed as text transcripts causes the searching stage in the phonetic technique to require more computational power than that required for searching text indexes in the LVCSR technique, where the usage of text-based searching algorithms on the text indexed data structures in the LVCSR technique makes searching almost instantaneous.

Recognizing new words: In the phonetic technique, totally new words that were not previously predefined can still be recognized if their phonemes can be identified. While in the LVCSR technique, the system cannot recognize words that are not previously defined in the dictionary, thus the dictionary has to be updated in case of presence of new words and the audio content has be re-indexed.

Precision and recall rate: Precision is a measure of how accurate the returned results are.
Recall rate is a measure of how many terms in the files being searched were returned in the results' set.
The phonetic technique usually has high recall rates but low precision. This is due to the fact that the phonetic technique gives many false positives; that is the search term may be found in several places in the file but was not actually said there. This is because some words sound similar and some words are parts of other words.
LVCSR on the other hand usually has high precision but low recall rates because of the presence of new words or because of recognition inaccuracy.

2.1.2 Audio Mining Applications

With the ability to index and search large volumes of audio data with much greater speed than that of previous manual methods, audio mining is becoming a powerful tool in several major applications. An overview of audio mining applications is given below.

Telephony applications: It enables businesses that offer customer services through telephone channels to examine and analyze the behavior and the responses of telephone agents through searching the recorded calls and telephone conversations, thus giving them the ability to control and improve the quality of their offered services [16].

Law enforcement: Audio mining facilitates the search of large volumes of legal archives for required legal or forensic evidences. In addition, it enables the search through prisoners' recorded calls and the analysis of intercepted telephone conversations for any illegal behavior [14].

Business intelligence: With the increasing volumes of media content in different types of businesses, audio mining facilitates the translation of raw audio data into useful cohesive data, thus giving deep and actionable insights into the patterns and trends of the business data allowing business professionals to better understand their business status and consequently consider suitable actions [19].

2.2 Similar Work

2.2.1 The Indexer

The Indexer, previously known as MAVIS, is a media processor that utilizes state of the art LVCSR speech recognition techniques to make audio and video files with speech content easily searchable by users [3]. Developed by Microsoft Research and running on the Microsoft Azure Platform as a cloud service [3], the Indexer processes spoken content

in media files and generates three output files: a binary index file, a keyword file and a closed captions file. These output files are used, together with a SQL server, to create a text search engine with which users are able to search spoken content in audio and video files using text search queries. Figure 2.3 shows the Indexer system architecture [23].

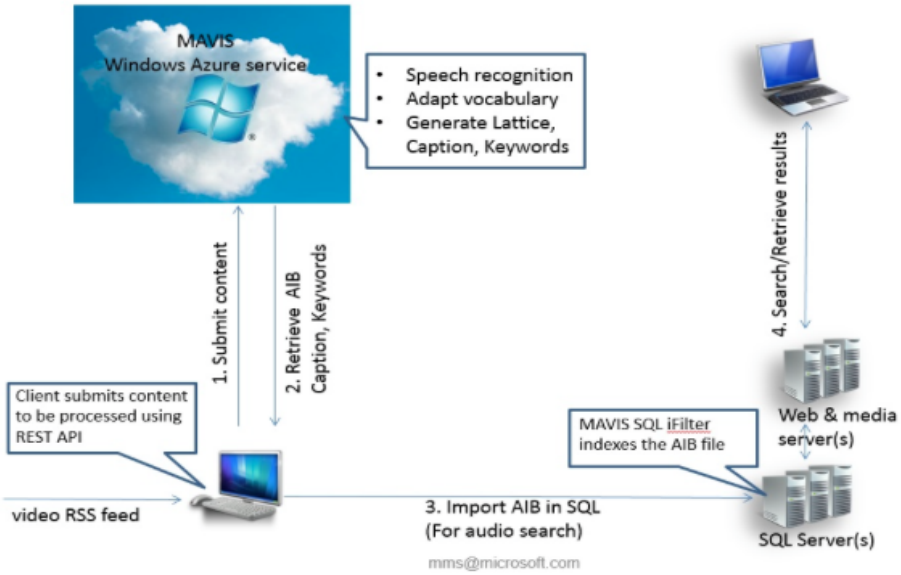

Figure 2.3: The Indexer architecture

2.2.2 Audio Indexing System for Election's Video Material

A team of developers designed and developed a content-based video searching system for the YouTube videos posted during the 2008 presidential elections in the United States. The aim of the system was to index the available video material to allow users to effortlessly search the videos' content and easily find relevant video clips. Figure 2.4 presents the system's architecture. The system consists of a "waveform transcoder" that extracts audio signals from videos stored in a database and resamples them. A "transcription client" then transcripts the videos through passing the extracted audio signals to an ASR service. The ASR service segments then transcripts the audio signals returning time-aligned transcripts in addition to confidence scores for words. The transcription process' information is stored in an "utterance database" which is then used by the indexing system. The indexing system builds an inverted index for the text transcripts and an index that stores position-related information for every word. Finally, the user interface (UI) utilizes the indexes to search the videos' content. It takes a text search term and returns to the user relevant videos in addition to text snippets of relevant sections in each video [6].

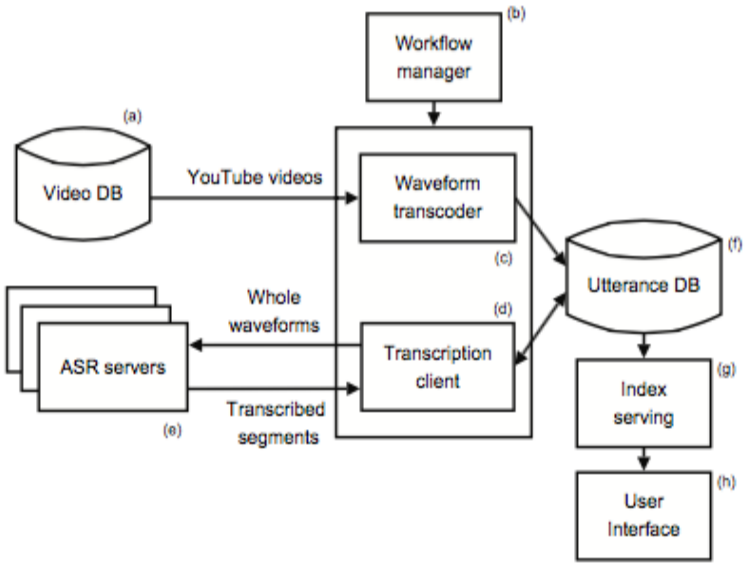

Figure 2.4: Block diagram for the indexing system of election's video material

2.2.3 SCANMail

SCANMail is a system that, with the use of a graphical user interface (GUI), allows users to search and skim through the content of their voicemail messages. In this system, messages are fetched from a voicemail server then transcribed using an ASR server. The transcripts are then indexed by an information retrieval (IR) server that utilizes the SMART IR engine. A GUI then provides users with a search panel that enables them to search the content of their voicemail using text search queries. In addition, the GUI allows users to view a message's header, thumbnail, transcription and any other notes related to the message. Figure 2.5 shows a screenshot for the SCANMail's GUI [8].

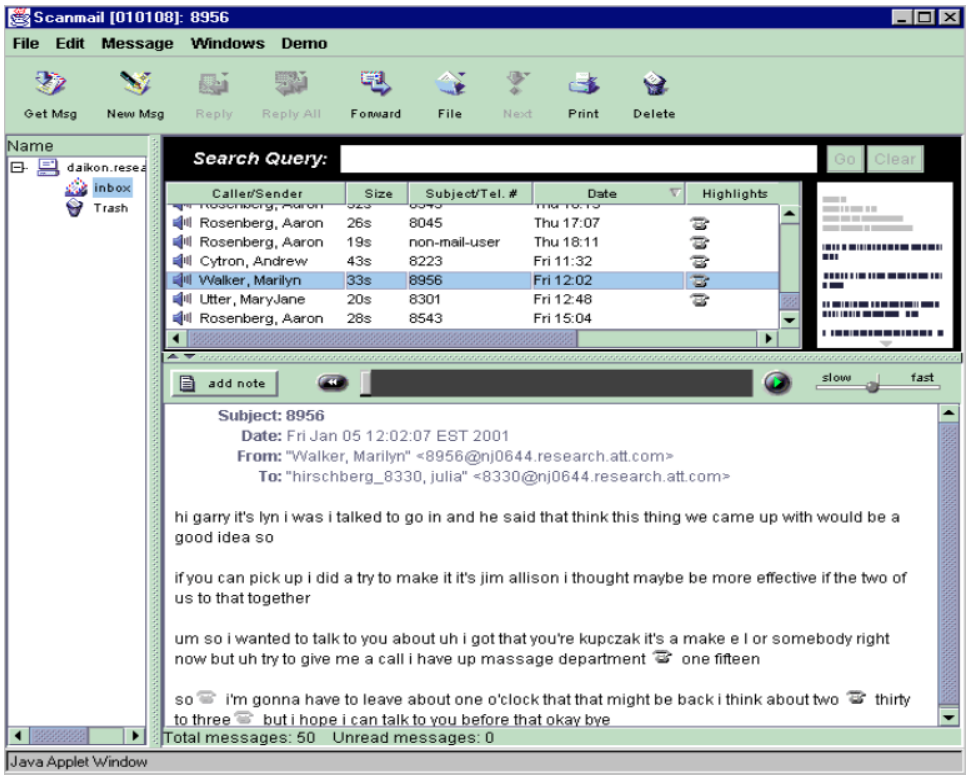

Figure 2.5: The SCANMail graphical user interface

Chapter 3

System Overview

An overall overview of the system's workflow is presented in Figure 3.1. The main goal behind this system is to enable users to search through videos' spoken content for the occurrence of certain words or phrases. It is composed of two main modules:

1. **A video processor:**
   The role of the video processor is to index the audio information contained in the videos. It consists of three modules that operate as follows:

   (a) A **transcoder** that fetches the videos residing in the database, extracts the audio data contained in them, downsamples the data to 16kHz, 16 bit, mono channel signals to be compatible with the speech recognizer and finally segments the data into segments of few seconds each to be more easily handled by the recognizer.

   (b) An **automatic speech recognizer** that takes in the audio segments and produces their equivalent text transcription. The transcription of each individual segment is stored in the database and the full transcription file is passed to the following module, the indexer.

   (c) An **indexer** that takes in the output transcription file of the recognizer and creates index files for the different words uttered. These index files are stored in a local repository to be used by the web application.

2. **A web application:**
   The role of the web application is to answer users' search queries. It has a client-server architecture and consists of two modules:

   (a) A **server** that takes the search queries inputted by users then, using the index files, searches the database for matching videos and returns the list of results.

   (b) A **UI** through which users input their text search query and receive the list of results produced by the server, where for each video in the list, the text extracts of relevant clips are displayed with the search term in each extract highlighted.

These modules and their implementation are discussed in the following chapters.

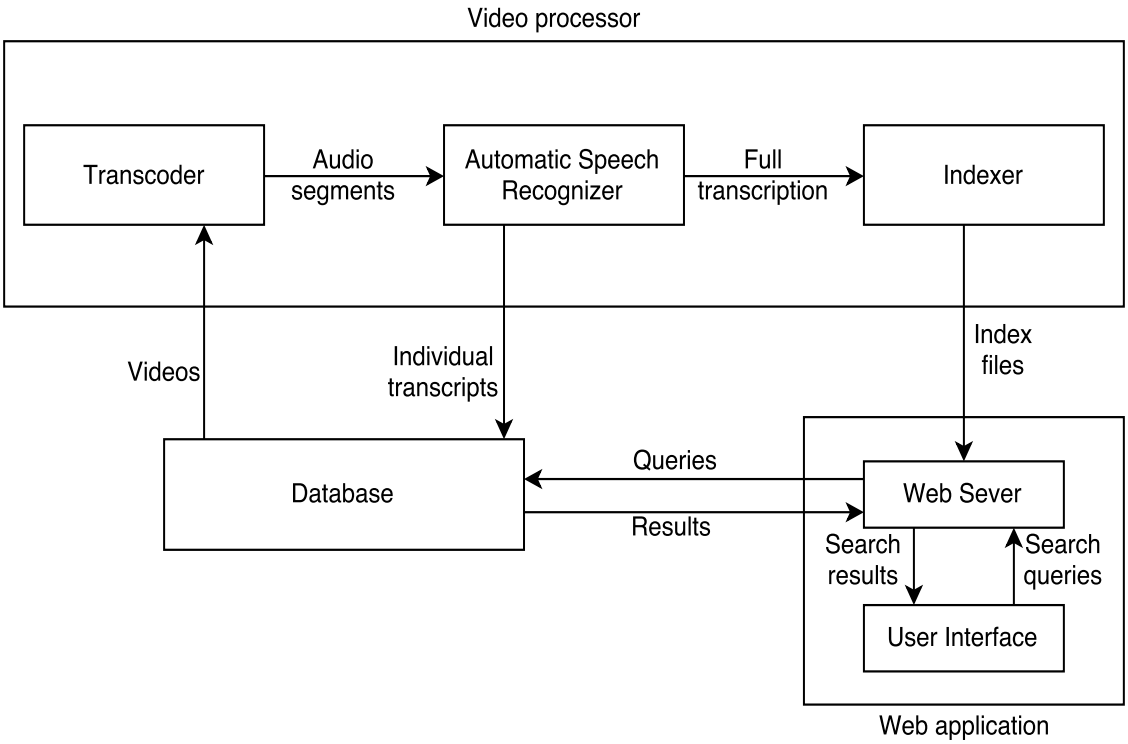

Figure 3.1: Overall workflow of the system

Chapter 4

The Transcoder

The transcoder module has two main roles:

1. It extracts and resamples the audio signals from the video files.
2. It segments the resulting audio files.

It is composed of two main open source tools, the FFmpeg open source software and the LIUM speaker diarization toolkit. This chapter discusses the usage of both tools and gives an overview of the module's workflow.

4.1 FFmpeg

FFmpeg is a free, open source software project that provides a complete, cross-platform solution to all multimedia-related processes, such as conversion, streaming, filtering, recording or playing and that supports nearly all the different audio and video formats [9]. It is developed under the Linux operating system and its operation is based on the command line utility. Because of its high ability to efficiently process any given audio or video, FFmpeg is used by pioneering software companies, such as Google and Facebook, and is widely used in the multimedia field in general [13].

The FFmpeg project is composed of four main tools: ffmpeg, ffplay, ffserver and ffprobe [9]. However, this project's transcoder module utilizes the ffmpeg tool only.

The ffmpeg tool:
The ffmpeg tool is a command line tool that acts as a video and audio converter [9], where it provides reliable manipulation of multimedia files with numerous formats. The general format of an ffmpeg command is as follows:

```
$ ffmpeg [input_options] -i input_file [output_options] output_file
```

where the desired options are applied to the next specified file.
The transcoder module utilizes this tool to extract, downsample and segment audio signals from video files as will be discussed later in this chapter.

4.2 LIUM_SpkDiarization Toolkit

Developed at the University of Maine, France and made available for public use at the Sphinx workshop in 2010 [22], LIUM_SpkDiarization is a toolkit dedicated for the diarization of audio streams, or in other words, the identification of "who speaks when" in an audio stream.
It is the successor of a previous segmentation tool, known as mClust, which was developed for the French ESTER campaign in 2005 and from which LIUM inherits much of its internal system architecture [22].
To reduce the obstacle of the different dependencies needed for the different operating systems, LIUM is developed in Java and is provided as a single precompiled jar file that contains all needed models and packages, and that can be run instantly without requiring any external third-party packages [22].
In this project, LIUM_SpKDiarization's latest version, LIUM_spkDiarization-8.4.1 [1], is used for the segmentation of the audio files so that they are ready for transcription by the automatic speech recognizer.

Running the toolkit:
As previously mentioned, LIUM operation is done through the direct running of the jar file only. The command for running the toolkit is as follows [15]:

```
$ /usr/bin/java -Xmx2024m -jar ./LIUM_SpkDiarization.jar /
 --fInputMask=input.wav --sOutputMask=output.txt
```

where

- "/usr/bin/java" specifies the path to the Java Virtual Machine (JVM)
- "-Xmx2048m" assigns 2048MB to the JVM memory, sufficient for processing a one hour file
- "-jar ./LIUM_SpkDiarization.jar" specifies the jar file to be run
- "–fInputMask=input" specifies the name of the audio file. The audio file can be in either sphere(.sph) or wave(.wav) file formats
- "–sOutputMask=output.txt" is the output segmentation file

The diarization output of this LIUM configuration is in the form of individual segments of duration less than 20 seconds for each segment, where each segment is from only one speaker over one channel [18]. These segments' information are summed up in the LIUM output, the "segmentation file".

[1] The LIUM_SpkDiarization toolkit can be downloaded from http://www-lium.univ-lemans.fr/diarization/doku.php/download

Segmentation file:
A segment is explained to be a distinctive region in the audio signal and is defined by three values: the name of the audio file it belongs to, its starting time and its duration [22].

As shown in the segmentation file sample in Figure 4.1 below, clusters are identified, where each cluster belongs to a specific speaker. Below each cluster head the related segments are listed, where each line represents a single segment. Different speakers are identified by the label "Sn" where n is a unique value assigned to each speaker during the diarization process.
Terms in a line -ordered from left to right- representing a segment are defined as follows:

- The name of the audio file the segment belongs to.
- The number of channels, set by default to 1.
- The start of the segment in frames.
- The duration of the segment in frames.
- The speaker's gender; F for female, M for male, U for unknown.
- The segment's band; S for Studio, T for telephone.
- The segment's environment type.
- The speaker's label.

```
;; cluster S160 [ score:FS = -33.29020872072402 ] [ score:FT = -34.07828994744826 ] [ score:MS = -33.17936644570786 ]
[ score:MT = -34.02366985889806 ]
audio 1 102001 821 M S U S160
audio 1 188966 235 M S U S160
audio 1 189215 514 M S U S160
audio 1 189985 395 M S U S160
;; cluster S171 [ score:FS = -34.39230069640813 ] [ score:FT = -35.13315497117413 ] [ score:MS = -34.52712489038468 ]
[ score:MT = -34.9958929300543 ]
audio 1 110164 171 F S U S171
;; cluster S172 [ score:FS = -33.132933765503694 ] [ score:FT = -34.537586234233096 ] [ score:MS = -33.93480047629456 ]
[ score:MT = -34.789982936334475 ]
audio 1 51921 764 F S U S172
audio 1 91923 430 F S U S172
audio 1 110685 678 F S U S172
audio 1 133925 1404 F S U S172
audio 1 165162 458 F S U S172
```

Figure 4.1: Screenshot for a sample of the segmentation file

4.3 Transcoder's Workflow

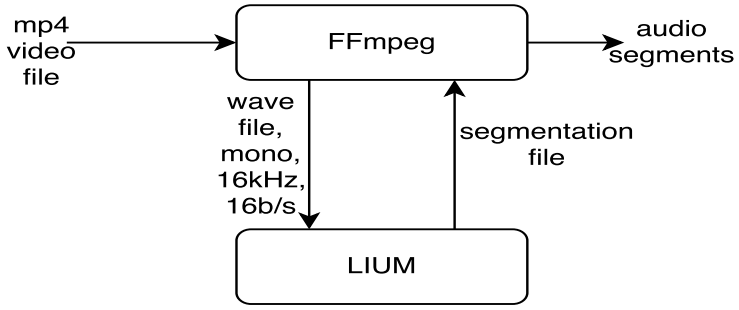

Figure 4.2: Overall workflow of the transcoder module

As illustrated in Figure 4.2, the transcoder goes through three main steps:

1. **Audio extraction and resampling:**
   FFmpeg is used to extract the audio signals from the given MP4 video files, after which the audio signals are downsampled to a sampling rate of 16kHz with 16 bits per sample, and produced as mono-channel wave files that are stored locally for later segmentation. This audio preprocessing is necessary to provide compatibility with the LIUM tool and the automatic speech recognizer input specifications.
   The extraction and resampling is done through the execution of the following command:

   ```
   $ ffmpeg -i input.mp4 -acodec pcm_s16le -ac 1 -ar 16000 output.wav
   ```

   where

   - "-i" specifies the input file, in this case an mp4 video file
   - "-acodec" specidies the desired audio codec, in this case it is 16-bit signed little-endian traditional wave like format
   - "-ac" specifies the number of the audio file's channel, in this case it is specified to be a mono channel audio file
   - "-ar" specifies the audio file's sampling frequency, in this case it is specified to be 16kHz

2. **Running LIUM and producing the segmentation file:**
   LIUM is invoked using the command previously discussed in section 4.2, where it fetches the audio wave files from the local repository, performs the diarization process then produces the corresponding segmentation file.

3. **Audio segmentation:**
   The program then takes in the produced segmentation file, reorders the segments in ascending order with respect to their start time then loops over the ordered file line by line.

In each loop iteration, the program extracts the segment's start time and duration, converts them from frames to seconds then calls the ffmpeg tool. At this step, the ffmpeg tool is used to slice the audio file and produce the specified segment. This is achieved through executing the following command:

```
$ ffmpeg -ss startTime -i input.wav -t duration -c copy output.wav
```

where

- "-ss" specifies the position in the audio file from which to start copying
- "-i" specifies the input audio file
- "-t" the duration of the required segment
- "-c" copies the specified stream portion to the output file
- "output.wav" specifies the name and the format of the output file

The produced segment is then stored in a local repository.
Each segment's start time, end time, calculated as the addition of the start time and the duration, and the video it belongs to are stored in the database.
After all segments are created, the program creates a "control file". The control file is a file that contains the names of all the created segments, where the name of each segment excluding the extension is written on a separate line. This control file is needed by the speech recognizer that will be discussed in the next chapter.

Chapter 5

The Automatic Speech Recognizer

5.1 Automatic Speech Recognition

Automatic Speech Recognition, or ASR in short, is the technology that facilitates automatic speech to text conversion. That is, it allows a machine to recognize and identify spoken speech and be able to translate it into written words. In a nutshell, it is what enables a machine to know "what is said".

The field of ASR has interested researchers for decades and has experienced significant developments. Currently, ASR is being used on a day-to-day basis in different services and applications [11], such as:

- Rapidly creating and editing medical reports using voice commands.
- Data entry and database management in business systems.
- Routing customer calls and providing interactive voice commands in telecommunication and telephone services.
- Generating subtitles in real-time for news broadcasts.

5.1.1 ASR System Classification

ASR systems can be classified based on either speaker types or word recognition types.

- Based on types of speakers, ASR systems can be:
    1. Speaker Dependent, in which the system is designed for a specific group of users.
    2. Speaker Independent, in which the system is designed for any kind of users.

    Speaker dependent ASR systems yield more accurate results, however, speaker independent ASR systems are more flexible and can be easily scaled since they are not limited to particular users.

- Based on types of word recognition, ASR systems can be:

  1. Isolated word recognition systems, in which a single word is recognized at a time and words are separated by silences.
  2. Connected word recognition systems, in which a sequence of words is recognized and sequences are separated by silences.
  3. Continuous word recognition systems, in which a stream of words with no silence in between is recognized.
  4. Spontaneous word recognition systems, in which continuous speech that contains inessential words, such as "uhm" and "ah", is recognized. This is the closest to natural speech.

5.1.2 ASR System Architecture

As previously mentioned, the goal of an ASR system is to efficiently convert a speech signal into its corresponding text transcription of spoken words. This can be mathematically defined as a function that maps the acoustic features of a given signal to a single word or a series of words [5].
So, given a sentence W that belongs to a predefined fixed set of viable words, ω, and an acoustic feature X that is generated from the given speech signal and that belongs to a predefined set of acoustic sequences, χ, the goal of the ASR system would be to find the sentence $\widetilde{W}$ that best estimates W, matches X and has the highest probability in ω. This can be demonstrated with the following equation:

$$\widetilde{W} = argmax_{W \in \omega} P(W|X) \tag{5.1}$$

which can be simplified to become:

$$\widetilde{W} = argmax_{W \in \omega} P(W)P(X|W) \tag{5.2}$$

where $P(W)$ is the sentence's probability and $P(X|W)$ is the probability of detecting X when the speaker utters W.
Equation (5.2) forms the components of an ASR system, where a language model calculates $P(W)$, an acoustic model calculates $P(X|W)$ and a decoder searches through all viable word sequences to maximize the product of both probabilities. Figure 5.1 illustrates the main constituents of an ASR system [5].

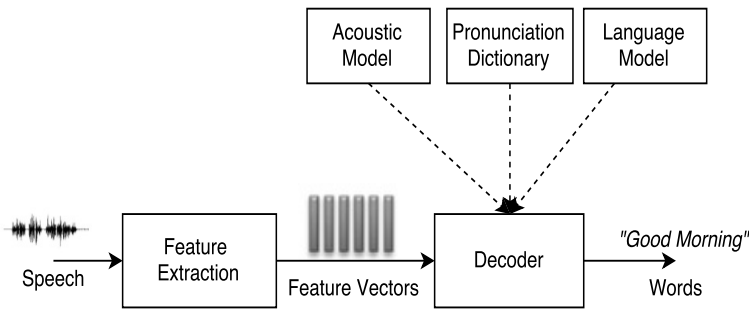

Figure 5.1: Block diagram for the ASR system architecture

5.1.2.1 Feature Extraction

The goal of the feature extraction process is to digitize the speech signal and obtain acoustic feature vectors; which are explained as the relevant features that can characterize and represent the given speech signal. A standard feature set used in ASR is the mel-frequency cepstral coefficients (MFCCs). The MFCC feature set is a representation for speech that tries to model the human ear sensitivity and simulate some behaviors and characteristics of the human auditory system [21].

5.1.2.2 Feature Matching

After the speech signal is represented as a set of acoustic feature vectors, the decoder matches these feature vectors with the acoustic model, the language model and the pronunciation dictionary to find the best estimation, $\widetilde{W}$, of the spoken content.

- Acoustic Models:
  An acoustic model is used to represent the relation between an audio signal and the phonemes contained in the language to be recognized [21]. It is used to calculate $P(X|W)$, or the observation likelihood, which is the probability of detecting X when W is uttered as previously mentioned. An acoustic model is trained using speech corpora and their corresponding text transcripts.

- Language Models:
  A language model seeks to determine the likelihood of the presence of the sequence W in the target language, in other words, it describes whether the given word sequence represents a valid sentence in the language or not. It is used to calculate the probability of the occurrence of a sequence of words W in the language to be recognized [5], or $P(W)$ as previously mentioned. A language model is trained using large sets of text corpora. N-gram language models are one of the most common language models used.
  An N-gram language model approximates the conditional probability that a word w is going to occur given the complete previous sequence of words W by estimating the probability of w to occur given the previous $N-1$ words in the sequence. The N-gram model is defined with the following equation [5]:

$$P(W) \approx \prod_{i=1}^{n} P(w_i|w_{i-N+1}, w_{i-N+2}, ..., w_{i-1})$$

- Pronunciation Dictionaries:
  A pronunciation dictionary is a file that describes how each word in a language is pronounced [21]. It contains a list of words and their corresponding pronunciations, e.g. an entry for the word 'gain' would be the word 'gain' itself and its pronunciation represented as 'G EH N'.

5.1.3 ASR System Evaluation

An ASR system can experience any of the following three types of errors during the recognition process:

- Word insertion errors, in which the recognizer adds extra words that were not actually present in the speech signal.
- Word substitution errors, in which the recognizer incorrectly identifies a word, mistakenly replacing it with an incorrect one.
- Word deletion errors, in which the recognizer omits some words that were actually present in the speech signal.

A standard evaluation metric to assess the ASR system performance is the word error rate (WER), which is evaluated as follows [21]:

$$WER = \frac{NI + NS + ND}{|W|} \tag{5.3}$$

where NI is the number of word insertions, NS is the number of word substitutions, ND is the number of word deletions and $|W|$ is the total number of words in the sentence W being evaluated.

5.2 CMU PocketSphinx

CMU Sphinx is a speaker independent, open source large vocabulary continuous speech recognizer. It is developed at Carnegie Mellon University and contains various toolkits and packages for building speech recognition systems [10].

CMU PocketSphinx is a lightweight speech recognition library built in C. It contains utility programs that can be called from the command line. In this project, the latest version of CMU PocketSphinx[1] is used.
To run PocketSphinx and produce the text transcriptions of the transcoder's output audio segments discussed in Chapter 4, the following command is used:

```
$ pocketsphinx_batch -hmm [dir] -dict [dir] -lm [dir] -cepdir [dir]
-ctl ControlFile -cepext .wav -adcin yes -hyp transcript.txt
```

where:

- "pocketsphinx_batch" runs PocketSphinx in batch mode, in which a collection of audio files are transcribed sequentially.
- "-hmm" specifies the path to the acoustic model files.
- "-dict" specifies the path to the pronunciation dictionary.
- "-lm" specifies the path to the language model files.
- "-cepdir" specifies the path to the repository in which the audio segments reside.

[1] CMU PocketSphinx and its required files -the acoustic models, language models and pronunciation dictionaries- can be downloaded from https://sourceforge.net/projects/cmusphinx/files/

- "-ctl" specifies the control file to be used in locating the input files.
- "-cepext" specifies the input audio files extension. This would be prefixed with the files' names in the control file to locate them.
- "-adcin" indicates that the input is raw audio data.
- "-hyp" specifies the output file name.

After PocketSphinx performs the speech recognition process, the text transcription of each individual segment is saved in the database and the full text transcription file is saved in a local repository to be indexed.
A sample of the output text transcription file is shown below in Figure 5.3. Each line corresponds to a single segment and consists of the segment's text and the the segment's file name.

```
we're taking this to our next level by using the exact technology this distracting us from the present moment to help smokers
they enterprise and that's right (50seg5582 -34148)
we can actually deliver this using videos and animation we can even get people in the evil exercises to help them really right
out there koreans the moment they come up (50seg5583 -39590)
we also wanna see what's going on it (50seg5584 -6748)
when they're getting out of their own way so here we brought an experienced managers folks were really good it just need to see
what their brains with look like when your mother to suggest or a new here's a brain annie is a brand new car down the middle
minus all the one that's (50seg5585 -66939)
and here's the net what a great cobb brain regions called the default mode network how do you thing (50seg5586 -27242)
remember that they presented a time when we're not present in fact mark radio in his colleagues at washington university in
saint louis staring get vigorously discovered this by giving people the simple task lay in the scanner and don't do anything in
particular so what we do when we don't do (50seg5587 -61095)
```

Figure 5.2: Transcription file sample

5.2.1 Recognizer Accuracy

Using the *word_align.pl* tool provided by the CMU Sphinx project, the recognizer accuracy was determined. The tool calculates a speech recognizer's accuracy by comparing the recognizer's output text transcription with a reference text transcription of the video or the audio file and calculating the WER.
A number of already transcribed videos on the web were collected together with there text transcripts. The videos were transcribed using the system's recognizer module, then using the word_align.pl tool, the output transcription was compared with the reference text transcription and the accuracy was calculated.
Using 160 minutes of transcribed video, the accuracy of the recognizer module was 60.5%

Chapter 6

The Indexer

Using the recognizer's output transcription file discussed in Chapter 5, the indexer module builds an inverted index for the different words or sequences of words contained in the videos' transcription and their positions in the videos.

6.1 Inverted Indexes

An inverted index is an index data structure that maps a term back to the set of documents in which it occurs. Each unique term in a document is a key in the index, where its value is the list of documents in which the term appears, or its postings list. In the postings list, each document is referenced by a unique identifier known as the document identifier, or docID [17]. Figure 6.1 illustrates the basic idea of an inverted index [17].

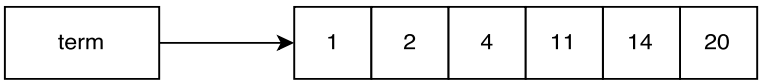

Figure 6.1: The two parts of an inverted index,
the term (key) and its postings list (value)

The main data structure of this system's search engine is the inverted index, where python's dictionary -or a hashtable- is used to store the inverted index in memory, so that it can be later used in answering users' search queries. The fact that this system would require performing lots of lookup and insertion operations combined with the fact that hashtables have an average lookup and insertion times of O(1) makes hashtables a suitable data structure for this system.

6.2 Search Query Types

This system's search engine answers three types of queries:

1. One Word Queries, in which the search query is composed of a single word, e.g. energy
2. Free Text Queries, in which the search query is composed of a sequence of unordered words separated by spaces, e.g. japan nuclear
3. Phrase Queries, in which the search term is composed of a sequence of ordered words written between a pair of curly brackets. That is, retrieved documents should contain the search term in the exact order defined, e.g. {nuclear plants in japan}

Accordingly, different inverted indexes are built to support these variable query types. A unigram index, an index in which the term is composed of a single word, is built to be used in answering one word and free text queries. A bigram, trigram and 4-gram indexes are built to be used in answering phrase queries composed of up to four words.

6.3 The Indexing Process

6.3.1 Preprocessing

Before building the inverted indexes, the text transcription has to be processed to decide on which words are going to be used as terms in the index. Each line of the transcription file is processed and converted into a list of terms through performing the following three operations:

1. **Tokenization**
   Given a sequence of words, tokenization is the process of splitting this sequence into its constituent words while discarding unnecessary characters such as punctuation marks [17].
   Each line is tokenized by splitting it on spaces, lower casing all words, removing any non-alphanumeric characters and discarding words composed of only one character. So, tokenizing the sentence "How are you? asked John's brother" would result in the following list of tokens ['how', 'are', 'you', 'asked', 'John', 'brother'].
   Tokenization is performed using the build_tokenizer() function provided by the CountVectorizer class in the scikit-learn library [20].
2. **Filtering stop words**
   After the tokenization step, stop words are filtered out from the list of tokens. Stop words are words that occur frequently in the text and have no important significance to be used in search queries, such as "in" and "the". So, a list of tokens such as ['how', 'are', 'you', 'asked', 'John', 'brother'] would be ['asked', 'John', 'brother'] after filtering out the stop words. The indexer uses the list of stop words provided by the Natural Language Toolkit's (NLTK) corpus [7].
3. **Lemmatization**
   Lemmatization is the process of removing inflectional endings from words, reducing them to their root or base form known as the lemma [17]. For example, "orange" and "oranges" are lemmatized to their base "orange". This helps decrease entries in the index since the various forms of a word, such as the singular and the plural forms, are reduced to the

dictionary form of the word. The indexer uses the NLTK lemmatizer, which is based on the WordNet's lemmatization function, to lemmatize all tokens and generate a list of terms to be indexed.

However, a modification to these preprocessing steps is applied to generate terms for building the N-gram indexes used for answering exact phrase queries. Since the original wording structure of the text transcription should not be altered, the line of text is only tokenized to generate a list of index terms, where the terms are neither lemmatized nor filtered from the presence of stop words. Figures 6.2 and 6.3 illustrate the workflow of the preprocessing steps.

Figure 6.2: Preprocessing steps for generating unigram index terms

Figure 6.3: Preprocessing steps for generating N-gram index terms

6.3.2 Building the Index

As previously discussed, each term will be a key in the dictionary and its postings list is its value. However, since it is required to keep track of the positions in the video in which each term appears, the postings list of each term will be a list of lists, where each list corresponds to a certain video and contains the video as the first element and the list of segments in which the term appears as the second. Segments and videos are represented with their database ids. An entry in the inverted index looks like this:

'term' : [[v1, [s1,s2,s3]], [v2, [s1,s2,s3]]]

This index structure can be visualized as shown in Figure 6.4.

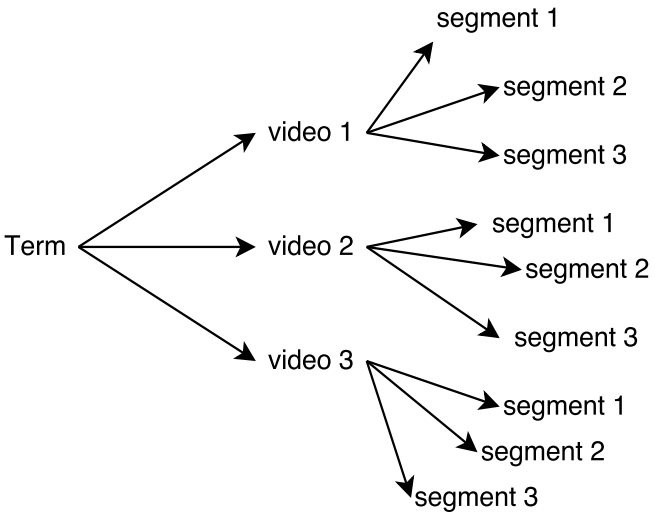

Figure 6.4: Visual representation for an index entry

Building the inverted index is performed as follows:

- Each line of the text transcription file is extracted, preprocessed as previously discussed and returned as a list of (key, value) pairs, where a term is the key and the database id of the segment to which the term belongs is the value.
  For example, the line "in these countries we are trying (1seg3)" is returned as follows: [('country', 3), ('trying', 3)] in the case of building a unigram index, or as follows: [('in these', 3), ('these countries', 3), ('countries we', 3), ('we are', 3), ('are trying', 3)] in the case of building a bigram index. The same follows for building trigram and 4-gram indexes with the exception of having different number of words in the index term.

- After the transcriptions of all segments belonging to the same video are processed, all the resulting lists of index terms are merged together to create an index for all the terms present in a video. In this step, the value of each term is a list which has the video's id as the first element and the list of segments in which the term appears as the second. An entry in a a video's index at this stage looks like this:

  'term' : [video, [s1,s2,s3,s4]]

- Next, the current video's index is merged with the main index of the system. This is achieved by appending the current postings list of a term to the postings list of the corresponding term in the main index to reach the final structure of the term's entry, as was shown previously at the beginning of this section.

This process continues in the same manner until all the transcriptions of all videos are processed and the main inverted index of all the videos is built.

Since the query answering program is separate from the program that creates the indexes, the main inverted indexes have to be saved to a file so that they can be later read by the query answering program to answer users' search queries.
The index is arranged in a text file with the following structure:

term|v1:s1,s2,s3;v2:s1,s2;v3:s1,s2,s3,s4;...

In the index file, each term is written on a separate line. The term is written first, followed by the character '|', which separates the term from its postings list. The postings list is written with the following format: the video's id followed by a colon followed by the list of segments in which the term appears separated by commas. Each video is separated from the other with a semicolon.

Chapter 7

The Web Application

7.1 Client-Server Architecture

The client-server architecture is a system architecture in which computers act as either clients or servers, where servers process and respond to clients' requests.
A client is a normal computer or device with working network connection that provides the user with functionalities and interaction capabilities. A server,on the other hand, is a computer that manages the data processing and handles the different client requests, and is usually connected to a database. Servers require larger memory sizes and higher processing capabilities than clients [1].
In a client-server architecture, the client sends a query requesting some data from the server, the server receives the client's request, processes it, accesses the data files and alters them if required then responds to the client with the required information. Figure 7.1 illustrates the client-server architectural model.
This system architecture allows the server to be accessed from any client, meaning, a user can access the server from any available computer anywhere. In addition, different clients can simultaneously access the server at any given time.

In this project, the web application has a client-server architecture. The client sends a search query through the interface to the server, the server processes this query and searches through the database then responds back to the client with a list of relevant videos.

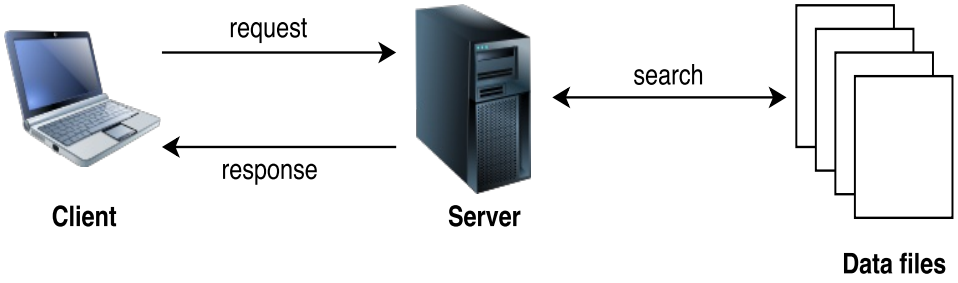

Figure 7.1: Client-Server Architecture

7.2 Django

Django is a free, open source, high-level Python web framework that facilitates the development of web applications. It is based on the model-view-controller (MVC) architectural structure that adheres to the "don't repeat yourself", or the DRY, principle, which enables the easy creation of web applications using minimal, non-repetitive code.
Django has its own server that automatically compiles the scripts and handles the database relations and processes [12].
In this project, Django is used to develop the web application, where the Django server is used to process user queries and retrieve relevant results and the Django templates serve as the application's user interface.

7.2.1 MVC

The MVC is a software architectural pattern in which models handle data definition and access through managing the application's database, controllers handle the application's logic and the view handles the user interface [12]. Figure 7.2 illustrates the workflow of the MVC structure.

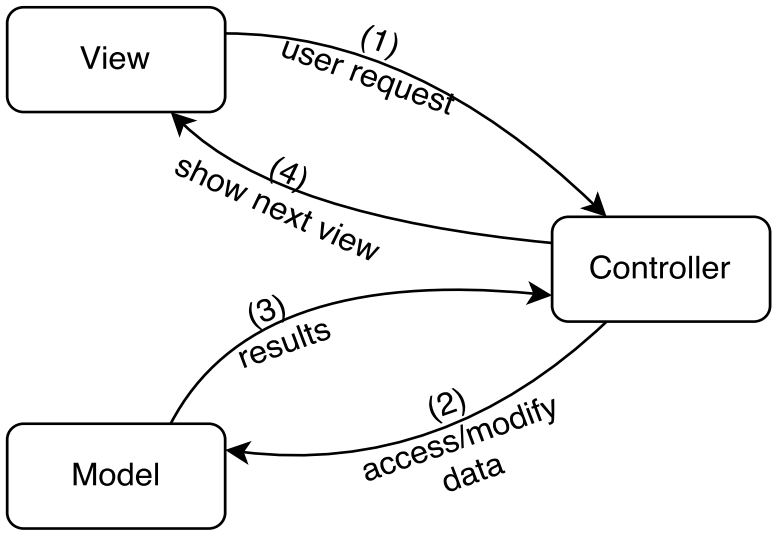

Figure 7.2: The MVC structure workflow (figure adapted from [2])

7.3 Database

With the aid of the index files, the web application uses a MySQL database that stores information about indexed videos and their contents to answer user search queries.
MySQL is an open source relational database management system that runs on all different operating systems. It is scalable, fast and provides almost all features needed for database

development [4].
To integrate the MySQL database in the web application, the MySQLdb module is used.
In this project, videos and their related content information are stored. For each video, the title, duration, YouTube link and the segments are stored. For each segment, the start time, the end time and the text transcription are stored. Figure 7.3 shows a diagram for the database design.

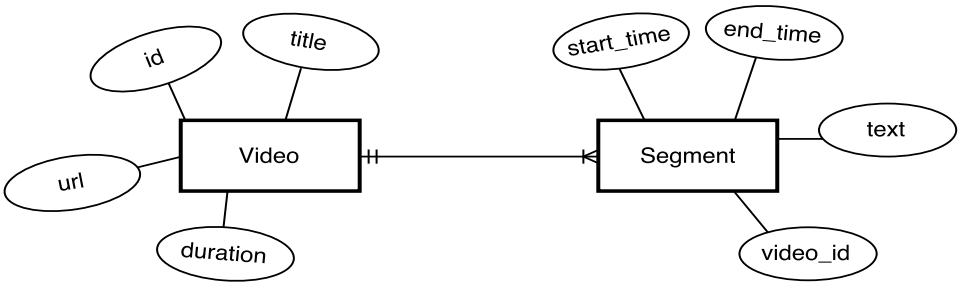

Figure 7.3: The database design

7.4 How it Works

7.4.1 Loading the Indexes

Once the server starts, it reads the index files from disk and loads the index into memory in the same format with which it was created. Recall from Chapter 6 that the format of a line in the index file was:

term|v1:s1,s2,s3;v2:s1,s2;v3:s1,s2,s3,s4;...

and that an entry in the index structure was:

'term' : [video, [s1,s2,s3,s4]]

where the index structure is a dictionary in which the term is the key and the term's postings list is the value. The server constructs the index back in memory through performing the following steps for each line in the file:

1. Read a line from the file.
2. Split the line on '|' to get separate the term and its postings list.
3. Split the postings list on ';' to get the lists of videos in which the term appears.
4. Split each video's list on ':' to separate the video's id from the list of segments in which the term occurs.
5. Split the segments list on ',' to separate the ids of the segments.
6. Create an entry for the term in the inverted index.

These same steps are performed on the four different index files to construct the unigram, bigram, trigram and 4-gram indexes.

7.4.2 Retrieving Information

Recall the three query types the server answers: one word queries, free text queries and phrase queries.
Once the server receives the search query from the user, it decides the query type and accordingly starts searching for relevant information. However, before starting the search process, the server processes the query the same way it preprocessed the videos' transcripts. It lower cases all words, removes any non-alphanumeric characters, filters out stop words and lemmatizes the words. If the query is a phrase query, it only lower cases the words and removes any non-alphanumeric characters.

One Word Queries:
If the search query is only one word with no curly brackets, it is a one word query. In this query type, the server searches the unigram index for the term, and if exists, retrieves its posting list. Using the posting list, it retrieves the videos and the segments in each video in which the term appears. The results is a list of videos in which the term appears.

Free Text Queries:
If the search query is a sequence of words that are not surrounded with curly brackets, it is a free text query. In this query type, the server searches the unigram index for each term in the word sequence and retrieves the postings list of each one, if exists. For each postings list retrieved, it retrieves the corresponding videos and their segments. Finally, it takes the union of all the resulting video lists to produce a list of videos in which any of the words in the sequence appears. In other words, it performs a one word query for each word and takes the union of the results.

Phrase Queries:
If the search query is a sequence of words surrounded by curly brackets, it is a phrase query. Answering phrase queries follows the same steps for that of one word queries with the addition of one extra step. Before following the same searching technique, the server determines the number of words contained in the phrase and accordingly, decides the suitable index file to be used.
The result is the set of videos in which the phrase appears.

7.4.3 Ranking the Results

Before results are returned and displayed to the user, they are ordered with respect to relevancy, where the videos that are most relevant to the user's search query are displayed first. To achieve that, documents are ranked using a common weighting scheme known as the term frequency-inverse document frequency, or TF-IDF, weighting scheme.
In the TF-IDF weighting scheme, each term in a document is given a weight depending on its term frequency and its inverse document frequency. The higher the weight of the term, the more important it is in a document [17]. But before discussing the TF-IDF weighting scheme, a model called the vector space model should be discussed.

Vector Space Model:
The vector space model (VSM) is an algebraic representation of the information contained in textual data where each document is represented as a vector. [17]. The vector components can

be any kind of information, such as the absence or the presence of a term or the importance of a term in a document. The VSM is commonly used in different information retrieval algorithms since the representation of documents as vectors eases the process of extracting different features and information.

Term frequency:
Term Frequency (TF) is the number of times the term appears in a document. It can be represented follows:

$$tf_{t,d} = N_{t,d}$$

where t represents the term and d represents the document [17]. To find the TF of all terms in a document, the document is represented in the VSM, where the vector contains an entry for each term in the document with the value being the number of occurrences of the term in the document. However, using this calculation for the TF favors longer documents, since the number of occurrences of the term would be consequently higher. Taking this into account, term frequencies are normalized, so that a TF representation becomes:

$$tf_{t,d} = \frac{N_{t,d}}{||D||}$$

where D is the document vector's magnitude, calculated by summing up the square values of all values in the vector then taking the square root of the result.

Inverse document frequency:
The inverse document frequency of a term (IDF), however, is the log of the result of dividing the total number N of documents by the document frequency of that term. The document frequency is the number of documents in which the term appears and can be represented as follows:

$$df_t = N_t$$

The IDF of a term can be represented with:

$$idf_t = log\frac{N}{df_t}$$

TF-IDF ranking:
Using the TF alone in weighting terms will not yield accurate results since it considers all terms to be equally relevant, however in real practice, some terms are more rare than others and thus have higher importance in a document [17]. That is why TF is combined with IDF to create a more accurate scheme for weighting a term in a document. The TF-IDF weight of a term t in a document d is simply the multiplication of both weights [17], represented with the following equation:

$$tf - idf_{t,d} = tf_{t,d}.idf_t$$

Cosine similarity:
The cosine similarity is a method to compute the similarity between two vectors by calculating the angle between them. The more the two vectors are similar to each other, the smaller the angle between them in the vector space [17]. The cosine similarity is calculated with the following equation:

$$cos\theta = \frac{\vec{a}.\vec{b}}{||\vec{a}||.||\vec{b}||}$$

The server utilizes the TF-IDF weighting scheme and the cosine similarity measure to rank the list of results. It uses scikit-learn's TF-IDF Vectorizer class to transform each video's transcript into a TF-IDF vector and keeps them in memory. For each user query retrieved, the server treats the query as a document, where it transforms it into a TF-IDF vector. It then uses the cosine similarity function provided by the scikit-learn library to compute the similarity between the query vector and each video's vector. Finally, and according to the similarity scores obtained, the list of videos is reordered to have the videos with the highest similarity score appear first.

However, this ranking technique is not applicable to phrase queries since it views queries as a "bag of words". In the bag of words model, only information on the number of occurrences of a term is retained without any information on term positions being considered. This contradicts the goal of having a phrase query, in which the aim is to retrieve videos that contain the phrase in its exact given order.

7.5 User Interface

The application provides users with an interactive UI through which they can submit their queries and receive the list of results containing the matching videos.
The homepage through which the users submit their queries is shown in Figure 7.4.

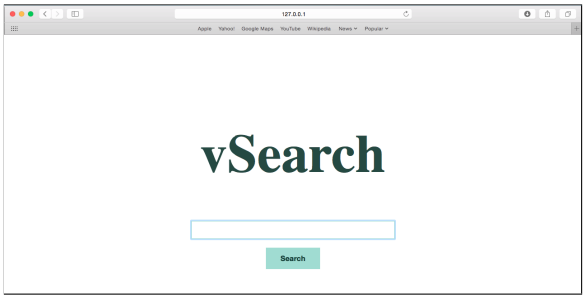

Figure 7.4: The application's homepage

Figure 7.5 shows the result page of a user searching for the word "world".

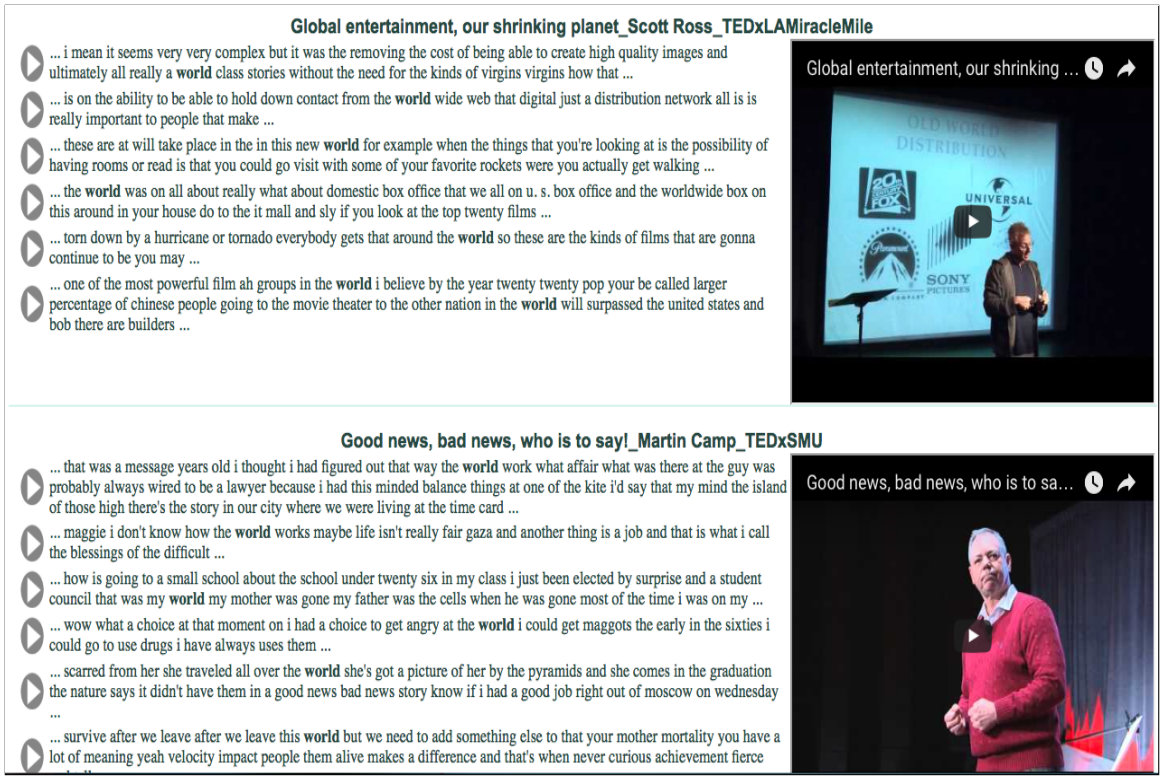

Figure 7.5: View of the result's page

Videos containing the search term are displayed. For each video, text transcripts of clips in which the specified search term was uttered, are listed in order of occurrence in the video, with the search term in each text snippet highlighted. In addition, the interface enables the user to play the video of interest through embedding a YouTube player.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, a system was developed to index a database of videos using the videos' audio information and consequently, provide users with the ability to search for certain words or phrases in the videos' spoken content. The system is composed of two main modules: a video processor module and a web application module.

The video processor is responsible for indexing the videos' audio content. Utilizing the FFmpeg tool, it first extracts and resamples the audio content of the videos. After that, it uses the LIUM_SpkDiarization toolkit to segment this audio content into small segments to be easier for transcription and to help determine the approximate timings of words' utterances in a video. Using the CMU PocketSphinx speech recognizer, the segments are transcribed and their text transcription is saved in the database. An indexer then processes the text transcription and produces and index file that stores information about the terms uttered, in which videos did they occur and the approximate times of their occurrence in each video.

The web application is then used to answer users' search queries. A user enters his search query through the application's user interface. The server retrieves the search query, processes and cleans up the query to produce useful words, then utilizes the index structure to search the database for the videos in which these words were uttered. The resulting list of matching videos is sorted in order of relevancy to the search query using the term frequency-inverse document frequency weighting scheme and the cosine similarity measure. The list of sorted results is returned back to the user through the user interface, where the interface displays the list of matching videos, and for each video, displays the transcription of all the segments in which the search term occurred, with the search term in each text extract highlighted.

8.2 Future Work

A number of enhancements and functionalities can be added to improve the system's performance.

- A module can be added that automatically scans the web, fetches videos and adds them to the index since up until this stage, video processing and indexes' generation is done offline and needs to be manually run whenever new videos are to be added.
- The speech recognition engine can be ported to other languages such as Arabic and French for example.
- Train language models for specific domains such as finance, law and medicine. This would help extend the system to include filed-specific subsystems.
- The text transcription output of the speech recognizer can be formatted for better readability
- A spelling correction functionality can be added to the system, where user queries that contain spelling errors are corrected, thus improving retrieval accuracy.
- A synonym or thesaurus matching functionality can also be added to the system to enhance the user experience. So, for example, a search for "car" would return the videos in which "car" or "automobile" was present.

Appendix

Appendix A

Lists

ASR	Automatic Speech Recognition
GUI	Graphical User Interface
IDF	Inverse Document Frequency
IR	Information Retrieval
JVM	Java Virtual Machine
LVCSR	Large Vocabulary Continuous Speech Recognition
MVC	Model-View-Controller
SQL	Structured Query Language
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
UI	User Interface
VSM	Vector Space Model
WER	Word Error Rate

List of Figures

Bibliography

[1] Client Server Network Architecture. http://www.ianswer4u.com/2011/05/client-server-architectures.html#axzz48I7ktSbY.

[2] Final Project. http://rbedig.com/rbedig-final-project/.

[3] Microsoft Audio Video Indexing Service (MAVIS). http://research.microsoft.com/en-us/projects/mavis/.

[4] Overview: MySQL. http://www.novell.com/documentation/nw65/web_mysql_nw/data/ah7vjv4.html.

[5] A.G. Adami. Automatic speech recognition: From the beginning to the Portuguese language. In *The International Conference on Computational Processing of Portuguese (PROPOR). Rio Grande do Sul: Porto Alegre*, 2010.

[6] C. Alberti, M. Bacchiani, A. Bezman, C. Chelba, A. Drofa, H. Liao, P. Moreno, T. Power, A. Sahuguet, M. Shugrina, et al. An audio indexing system for election video material. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4873–4876. IEEE, April 2009.

[7] Steven B., Ewan K., and Edward L. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[8] M. Bacchiani, J. Hirschberg, A. Rosenberg, S. Whittaker, D. Hindle, P. Isenhour, M. Jones, L. Stark, and G. Zamchick. SCANMail: Audio navigation in the voicemail domain. In *Proceedings of the first international conference on Human language technology research*, pages 1–3. Association for Computational Linguistics, 2001.

[9] F. Bellard, M. Niedermayer, et al. Ffmpeg. *http://www.ffmpeg.org*, 2012.

[10] CMU Sphinx. CMU Sphinx: Open Source Speech Recognition Toolkit. http://cmusphinx.sourceforge.net.

[11] C. Gaudard, G. Aradilla, and H. Bourlard. Speech Recognition based on Template Matching and Phone Posterior Probabilities. Technical report, IDIAP, 2007.

[12] A. Holovaty and J. Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2009.

[13] F. Korbel. FFmpeg Basics: Multimedia handling with a fast audio and video encoder. 2012.

[14] N. Leavitt. Lets Hear It for Audio Mining. 35:23–25, Oct. 2002.

[15] LIUM. LIUM Speaker Diarization Wiki: Quick Start, August 2013. http://www-lium.univ-lemans.fr/diarization/doku.php/quick_start/.

[16] M.K. Mand, D. Nagpal, and Gunjan. An Analytical Approach for Mining Audio Signals. *International Journal of Advanced Research in Computer and Communication Engineering*, 2, Sept. 2013.

[17] C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2008.

[18] S. Meignier and T. Merlin. LIUM SpkDiarization: an open source toolkit for diarization. In *CMU SPUD Workshop*, volume 2010, 2010.

[19] M. Meteer. Choosing the Right Technology for your Speech Analytics Project. White paper, Dec. 2003.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] L.R. Rabiner and B-H. Juang. Speech Recognition: Statistical Methods. *Encyclopedia of linguistics*, pages 1–18, 2006.

[22] M. Rouvier, G. Dupuy, P. Gay, E. Khoury, T. Merlin, and S. Meignier. An open-source state-of-the-art toolbox for broadcast news diarization. Technical report, Idiap, 2013.

[23] A. Solanki. Introducing: Azure Media Indexer, Sept. 2014. https://azure.microsoft.com/en-us/blog/introducing-azure-media-indexer/.