AUTOMATED TAGGING TO ENABLE FINE-GRAINED BROWSING OF LECTURE VIDEOS

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Master of Technology

by

VIJAYA KUMAR KAMABATHULA

(Roll No. 09305081)

under the guidance of

Prof. Sridhar Iyer

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY - BOMBAY
2011

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled **"Automated Tagging to Enable Fine-Grained Browsing of Lecture Videos"**, submitted by **Vijaya Kumar Kamabathula** (Roll No: **09305081**) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology, Bombay**.

Prof. Sridhar Iyer
CSE, IIT Bombay
Supervisor

Prof. Sahana Murthy
CDEEP, IIT Bombay
Internal Examiner

Prof. Vijay Raisinghani
HoD, NMIMS
External Examiner

Prof. Girish Saraph
EE, IIT Bombay
Chairperson

Place: IIT Bombay, Mumbai
Date: 28^{th} **June, 2011**

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Vijaya Kumar Kamabathula

(09305081)

Date: 28^{th} **June, 2011**

Acknowledgments

I would like to express my sincere thanks and gratitude to my guide *Prof. Sridhar Iyer* for his invaluable guidance and encouragement. He provided me constant support through out the duration of the work. It has been a great learning experience working with him. I am very much thankful to *CSE Department, IIT Bombay* for providing computing facilities in labs.

I would like to thank all the people involved in development of open source tools which are being used in our system. I also thank *Prof. Deepak B Phatak* for providing us with CS 101 lecture contents, which is offered at IIT Bombay during Autumn 2009.

Date: _____ VIJAYA KUMAR KAMABATHULA

Abstract

Many universities offer distance learning by recording classroom lectures and making them accessible to remote students over the Internet. A university's repository usually contains hundreds of such lecture videos. Each lecture video is typically an hour's duration and is often monolithic. It is cumbersome for students to search through an entire video, or across many videos, in order to find portions of their immediate interest. It is desirable to have a system that takes user-given keywords as a query and provides links to not only the corresponding lecture videos but also to the section within the video. In order to do this, lecture videos are sometimes tagged with meta-data to enable easy identification of the different sections. However, such tagging is often done manually and is a time-consuming process.

We proposed an approach to automatically generate tags for lecture videos and make lecture videos easily accessible. The approach is based on extracting textual information from lecture videos and indexing the text data to provide search features in lecture video repository. We used Automatic Speech Recognition (ASR) and optical character recognition (OCR) technologies for the content extraction. In user interface, our system provides links to slides of a lecture. When a slide link is clicked, video also seeks to location where exactly the slide is discussed in that video and starts playing from that part. For automatically generating the slide index, we proposed an algorithm based on slide title matching which uses OCR. Following the approach and using open source tools mentioned here, a lecture video repository can provide features for its users to access content required by them easily.

We used open source Java libraries available for speech recognition and text search purposes. We designed a generalized framework for creating language models which are required during automatic speech recognition process. We performed experiments to test the performance of our system and achieved a recall of 0.72 and an average precision of 0.87 as video retrieval results. We also tested the performance of our slide detection algorithm and on an average it is able to detect 95% of the slides of a lecture.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recording of lectures during class room presentations and making them accessible for students is a very attractive and useful activity in distance learning. Many universities around the world maintain recordings of lecture videos to provide facility to access them over web. Usually a university maintains hundreds of lecture videos covering various fields. To make best use of these available lecture video recordings, search facility in should be provided so that users can access the videos of their interest easily.

A System called as *Browsing System* is required which takes user query (keywords) as input and provides videos matching the query as results. The system should identify appropriate videos in the video repository and also highlight relevant portions in each video that matches user given keywords.

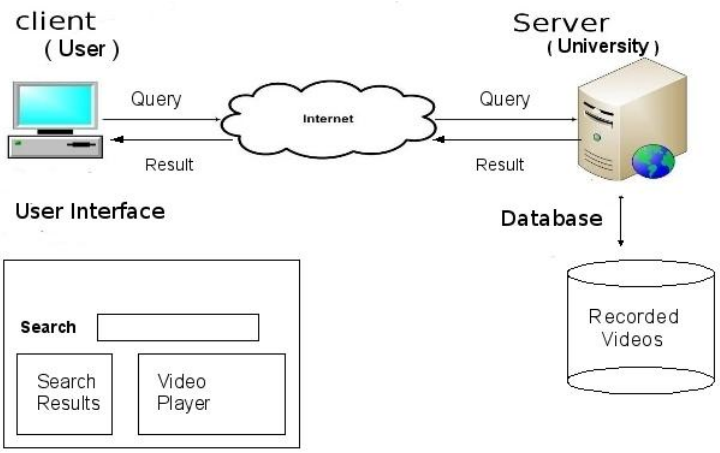

Figure 1.1: Lecture Video Browsing System

The figure 1.1 shows various components of a browsing system. Our Browsing System

runs at the server and handles user given queries. Other components of the system are content repository and user interface. Content repository contains all the recordings of available lecture videos and any other related material like presentation slides. Typically user interface contains a video player in which the lecture video is played.

1.1 Motivation

Duration of a lecture video is usually around 90 minutes. Within a video, user queried content may last for only few minutes. Usually a lecture video cover more than one topic. If a student want to refer one particular topic in a lecture video, then he/she want to see some particular sequences in that video where the required topic is discussed. For example in a Data Structures lecture video of one hour duration, discussion on *quick sort* may last for only 10 minutes. If we provide the user with entire video then it will be difficult to search for the required content manually. The problem is exacerbated if the user is not familiar with the area, or if the topic is very specific, since the user may not be able to decide the videos to be scrutinized. So, The browsing system should search this content among thousands of other lecture videos in the repository and present video sequences which matches user needs. This type of features are very useful for students since

- Students can get required information very easily.
- It is not necessary to watch full video of the lecture for finding required content.

If a video repository contains hundreds of lecture videos of various courses, following scenarios describe uses of providing search facility for them.

Example Scenarios

1. While preparing for next day exam (Data Structures) students may want to listen some part of lectures in which they are having doubts. If a student want to find portion of lectures where the topic **double hashing** is discussed. In this case, the student enters the keyword **double hashing** in search field, server should give response with list of videos in which Double Hashing is discussed and also for each video, portions which contain the word should be highlighted so that the student can directly watch the video from that point instead of watching whole video for the topic.

2. Finding video sequences where **matrix multiplication** is discussed in a programming course lecture videos.

3. Suppose we have Data Structures lecture videos and we need to find the video in which **quick sort** is discussed. In this case also we need search feature so that we can directly find the required video instead of looking at all the videos.

Hence, it is desirable to have a system that takes user-given keywords as a query and provides a link to not only the corresponding lecture videos but also to the section within the video. A few such systems exist (discussed in Chapter 2). In some of them, meta-data is associated (manually) with the lecture videos, while others use proprietary software to generate an index from the video contents. In this paper, we describe the design and implementation of a system for automatic tagging and fine-grain browsing of lecture videos. We use available open source tools for speech recognition engine and text search engine to develop the overall system. In the experiments to test the performance of our system, We achieved a recall of 0.72 and an average precision of 0.87 as video retrieval results. We indexed automatically generated speech transcript and a slide index file to provide search features in the repository. We provided navigation features using slide navigation mechanism through which users can move from one slide portion of the lecture to other portion just by clicking on links provided in the interface. For the slide navigation process, we proposed an algorithm which detects presentation slides in a lecture video and determines when their transitions occur within it. It gives slide titles present in the lecture video and their time of occurrences in the video. The algorithm is based on comparing slide titles and which uses Optical Character Recognition (OCR) for extracting slide text.

Organization of the Report :

In Chapter 2, we presented some existing lecture video repositories and observations we made while searching for some content in them. In Chapter 3, project details such as problem definition and solution scope are mentioned. In Chapter 4, overview of our solution approach and different phases present in the approach is presented. It discusses various decisions we made related to the approach and selection of available tools to use for building the overall system. In Chapter 5, System architecture is presented. It explains various components present in the proposed method and how to integrate them to develop the final system. In Chapter 6, results of

various experiments we conducted during development of the system are included. In Chapter 7, conclusion and future work is presented. Chapter 8 contains is a publication of our work.

Chapter 2

Background

In this section we provide a brief background on techniques for searching in lecture videos and a survey of search facilities currently supported in some well known courseware repositories. It discusses various features they provide to users and also presents issues present in them. In each repository, we looked for videos related to a particular topic and our experiences in finding the required content are described.

2.1 Techniques for searching in lecture videos

Search facilities can be provided in lecture video repositories in two ways [30]. They are:

- ***Meta-data based***: Meta data is textual data that is applied to a piece of multimedia content in order to describe it. These methods use meta data, such as video title, video description and user comments to identify video results matching given set of keywords. Such a meta data based approach may be able to identify videos that contain the keywords but they cannot locate where those keywords appear in the video time line.

- ***Content based***: Lecture videos typically contain the following contents [31]: (i) Lecturer Speech: Portion of video that shows the instructor talking, (ii) Presentation Slides: Portion of video that shows the current slide of the presentation, and (iii) Notes: Portion of video that shows the board/paper on which instructor is writing. Content based approaches extract meta-data from appropriate portions of the video and create an index that can be used for searching within the video. Such techniques are difficult to automate and time-consuming to do manually.

2.2 Existing Lecture Video Repositories

2.2.1 CDEEP

Center for Distance Education and Engineering Programme(CDEEP)[4] is a web interface for providing access to lecture videos of courses offered in various departments at IIT Bombay.
Topic looked for : *Matrix Multiplication*

Here we wanted to find lecture video sequences which explains Matrix Multiplication. It does not provide any search facility for users. It just displays list of courses for which recorded video lectures are available.

We found a course with the name CS101 Computer Programming & Utilization by Prof. D.B.Phatak and clicked on the course link. We got a list of 35 lectures of the course as shown in figure 2.1. Each has a topic name and link to the video file. We found two lectures on matrices lecture 14 and lecture 15 each are of 55 minutes duration. As we did not have any clue of where exactly matrix multiplication is discussed, we started watching the video of lecture 14 but we could not find the matrix multiplication in the video.

Figure 2.1: Screenshot-CDEEP Lectures list of CS101 course [4]

It took us around 55 minutes to find that the lecture 14 is irrelevant to our need. Then we started watching lecture 15 and at 33^{rd} minute of the video we found discussion on matrix multiplication. So it took us around **88 minutes** to locate video portions where matrix multiplication is discussed.

2.2.2 NPTEL

NPTEL[17] maintain lecture videos from different IIT's and makes them accessible over the web.

Topic looked for : *Double Hashing*

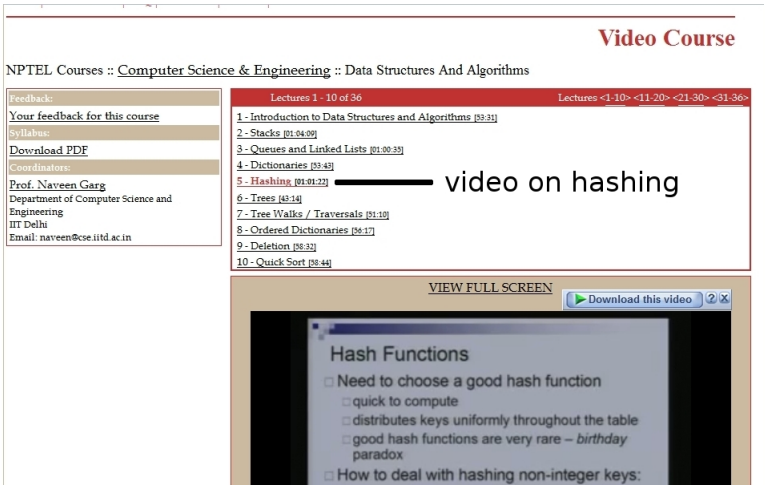

Figure 2.2: Screenshot-NPTEL lectures list of Data Structures course [17]

Here we wanted to find video sequences that explains double hashing. It also does not provide any search facility on lecture contents. It displays available lecture videos department wise or we can find lecture videos of a particular course by giving its name.

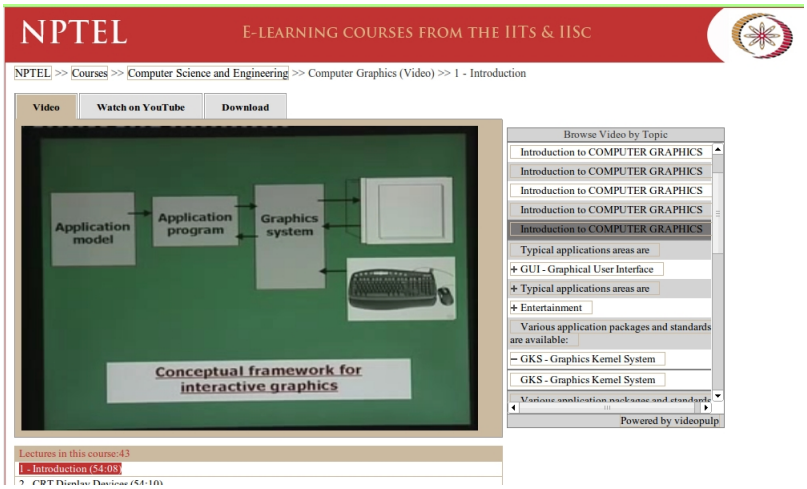

Figure 2.3: Screenshot-Slide Navigation Feature provided by NPTEL [17]

We selected Computer Science & Engg link for video lectures related to that department.

We found a list of 41 courses under this department. Each course has information such as course name, professor and type which is web or video.

We chose course named Data Structures and Algorithms by Prof. Naveen Garg and it displayed list of 36 lectures in the course as shown in the figure 2.2. We selected lecture 5 on Hashing which is of 1 hour duration. As there was no indication of where double hashing is discussed, we had to watch the video from starting and we were able to find the required information at 45^{th} minute. So it took us around **46 minutes** to locate content related to double hashing.

NPTEL recently started providing slide synchronization feature for the lecture videos. In the user interface links to presentation slides are provided as shown in the figure 2.3. This feature is being implemented by an official partner of them called videopulp [26], which is a video processing company.

2.2.3 freevideolectures.com

freevideolectures.com[7] is a website that provides free access to lecture videos of various universities.

Topic looked for : *Double Hashing*

Here we wanted to find video sequence explaining Double Hashing. It provides search facility based on meta data, we can give keywords in search box and it gives list of videos matching the keywords.

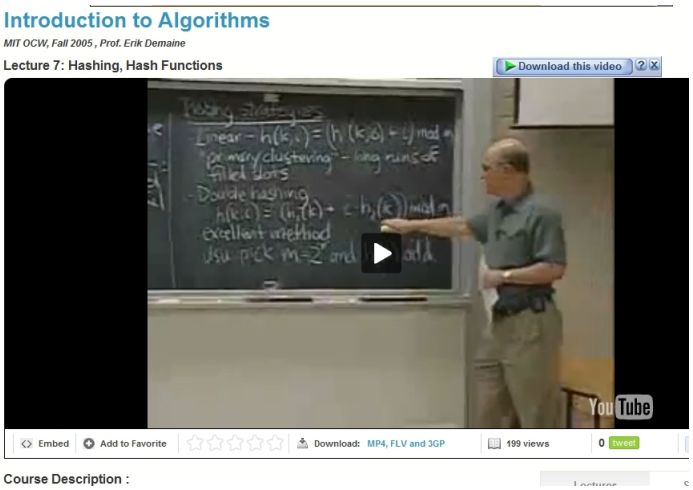

Figure 2.4: Screenshot-User Interface of freevideolectures.com [7]

We entered double hashing in the search box and we did not get any lectures matching the keywords. Then we entered hashing in the search box and we got 5 results matching the term.

We selected the first link and it was a course on Introduction to Algorithms by Prof. Charles E. Leiserson. Duration of the lecture is 78 minutes and we had to watch the video from its starting. Finally we found out the required information at 63^{rd} minute of the lecture as shown in the figure 2.4. It took us around **64 minutes** to find the required video contents.

2.2.4 videolectures.net

videolectures.net[25] is a website that maintains lecture videos of various universities and provides free access to them.

Topic looked for : *Double Hashing*

Here we wanted to find video sequences on Double Hashing. It also provides search facility based on meta data, we can give keywords and it gives list of videos matching the keywords.

We selected the link of Lecture 7: Hashing, Hash Functions by Prof. Charles E. Leiserson. Here user interface contains three portions as shown in the figure 2.5. Portion on the left side contains video player, right side top portion displays current slide and right side bottom portion shows hyper links to each slide and it title. So we can directly go to a particular slide of interest by clicking on the link.

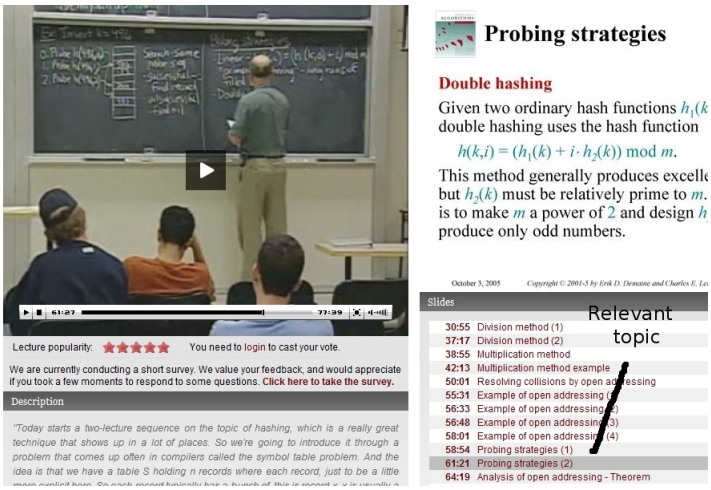

Figure 2.5: Screenshot-User Interface of videolectures.net [25]

For the selected video lecture on Hashing we found a link to probing strategies. As double hashing is one of the probing strategy we selected the link and it took me to 58^{th} minute of the

lecture. At 63^{rd} minute of the lecture we found the required information on double hashing. So it took **6 minutes** to find the required content.

Technical Details :

- The code for this site was written in Python and It is about 30,000 lines of code which is not publicly available
- It is using Postgres database and is running on Ubuntu linux.
- Synchronization of video with slides is done through manual process completely.

The synchronization of the slides can be automated by generating a tag file containing slide transition times.

2.2.5 Lecture Browser

MIT Lecture Browser[14] is a web interface for accessing lecture videos in MIT Open Course Ware.

Topic looked for : *Matrix Multiplication*

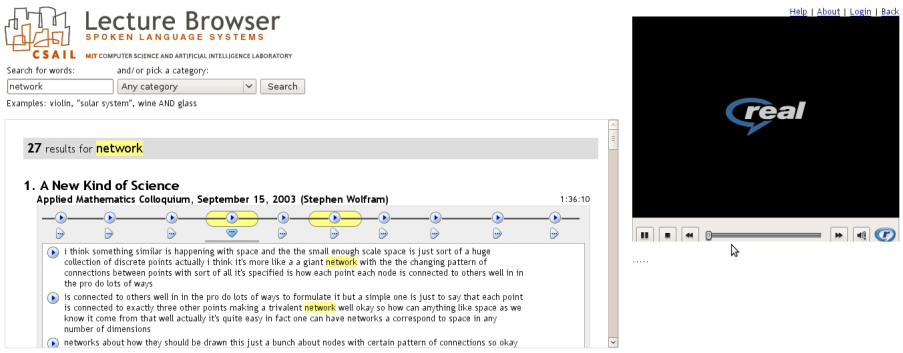

Figure 2.6: Screenshot-MIT Lecture Browser [14]

The Figure 2.6 shows the user interface provided by the MIT Lecture Browser. At the top left corner, user selects a category and enters some keywords in search box. Videos list matching search criteria are displayed at the left side portion. Each of the video contain arrows at some points which indicate highlighted portions. If user clicks on any arrow video starts playing from that point in video player provided at the right side of the interface.

Here we entered matrix multiplication and got video results on that which some highlighted portions. So we directly clicked on those portions and viewed the video. It took us around **2 minutes** to find the required content.

Technical Details[32]:

- It uses speech recognition engine to generate speech transcripts from lecture videos
- Textual search is performed on the *speech transcript*. It gives video results matching search criteria and highlights segments where the search keywords appear
- It is not an open source tool. They are using only to provide search facility in MIT lecture videos
- It is developed using speech recognition engine developed at MIT Speech group which is also not publicly available

The following table shows comparison of features available in the previously discussed video repositories.

Repository	**Search**	**Navigation Features**
NPTEL	No	No
freelecturevideos.com	Meta data	No
videolectures.net	Meta data	Slide Synchronization (manual)
Lecture Browser, MIT	Content	Speech Transcript &
Our System	Content	Speech Transcript Slide Synch(Automatic)

Table 2.1: Lecture Video Repositories Comparison

Last row in the table indicates features provided in our proposed system. As mentioned in the table, our system combines the features found in videolectures.net and lecture browser. videolectures.net involves manual process in slide synchronization but our proposed system does it automatically.

Chapter 3

Problem Definition

This Chapter covers details about the project such as inputs and outputs of the proposed system, its scope and solution assumptions.

At present, there is no open source tool available which automatically extracts information from lecture videos and makes them searchable. Lecture Browser of MIT provides the search facility on speech transcript but the tool is not publicly available. They provided a web interface for the lecture browser to provide the search feature on their lecture videos only. They are using speech recognition engine developed by MIT speech group which is also not an open source tool.

There will be major influence of accent of the speech during speech to text conversion using a automatic speech recognition. So if the system is used with a non-native English speaker whose accent and vocabulary it hasn't been trained to recognize, the accuracy can drop to 50 percent. So an existing speech recognition can not be used directly for our purposes. We need to tune the speech recognition engine so as to recognize the our English accent which varies largely.

So, our goal is to build a complete system for providing search facility for available lecture video recordings. The system makes use of available open source speech recognition engine and text search engine to build overall system.

Input

Input to the system is keywords entered by the user. In user interface we provide a search box so that user can enter a set of keywords in it. These keywords represent the video content for

which user is interested to watch. For example, if a student is looking for video lectures in which a networking topic called TCP (Transmission Control Protocol) is discussed, he/she can enter tcp in the search box.

The user queries can be simple keywords, keywords with operators such as +,- and also users can retrieve videos of particular course by specifying title or course code in the input query.

Output

The browsing system gives a list of lecture videos matching the user entered keywords as output. In each video portions where the keywords occur in the speech are highlighted. When user clicks on a particular portion video starts playing in the media player present in user interface. Along with the media player, user interface also shows hyper links to slide transitions, current slide and speech transcript. User can click on any word present in *speech transcript* and directly go to the particular time in the currently playing video where the word occurred. Hence users can get better browsing features within a video through the speech transcript and slide links present in the interface.

3.1 Solution Scope

We are dealing with lecture videos which are in English. And it also restricted to lecture videos related to computer science domain.

Reasons for restricted scope :

One of the major component of our system is Speech Recognition Engine (SRE). Performance of the SRE is mainly influenced by *language models* and *acoustic models*. Language models are used during speech recognition to capture the grammar or sequence of words of the language. Language models are domain dependent and should be generated using a *corpus* that is suitable to a particular domain for getting better accuracy from Automatic Speech Recognizer. So to apply our system in any other domains such as video lectures of other departments(eg. Electical, Mechanical etc), News videos or Sports videos corresponding language models have to generated or use them if they are already available. In this report we discussed a generalized

framework for creating text corpus. Using the methodology, we can generate large amounts of text corpus related to a particular domain.

Acoustic models capture speech characteristics and maps the voice to phoneme representation during automatic speech recognition.

Solution Assumptions

We made following assumptions in developing our system.

- It is assumed that the sound quality of the recorded lecture videos is good and negligible noise from surroundings. As speech recognition performance is also influenced by background noise, it is assumed that lectures are available with sufficient sound quality for recognition.

3.2 Related Work

An on line video browsing system is presented in [33], which stores individual clips of video separately at the server side and provide keyword search based on meta data and it does not deal with contents of the video. A method for synchronization of lecture videos with slides is presented in [35]. This method is based on using a plug-in in power point during class room presentation. The plug in is to store the slide transition timings. For currently recoding lecture videos the method can be applied, but for already available lecture videos we can not use that approach. A video browser based on closed captions available in the video is presented in [34]. A lecture browser using speech recognition is developed by MIT [14]. Our proposed system which will be discussed later provides more navigation features to users through speech transcript and automatic slide synchronization.

Chapter 4

Solution Overview

This Chapter presents the outline of our proposed system and gives the details of various design decisions. We follow a content based approach for providing search features in lecture video repositories. There are two technologies useful for extracting textual information from lecture videos which represents what exactly present inside those video contents. One is Optical Character Recognition (OCR) and the other is Speech Recognition.

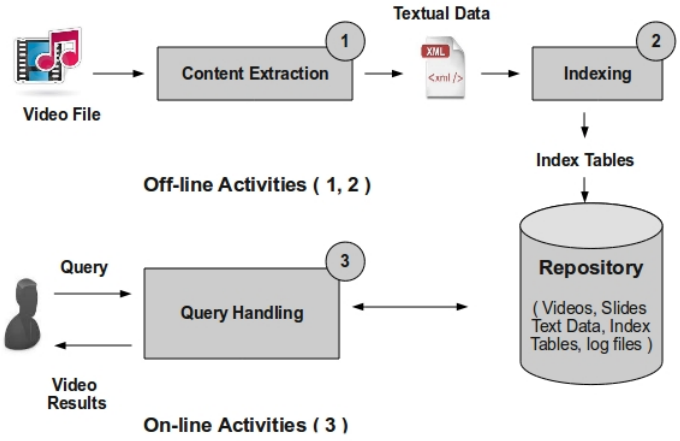

Figure 4.1: Solution Approach

The figure 4.1 shows solution approach of our system. The various activities involved in development of browsing system can be divided into two types.

- **Off-line Activities :** These are one time activities that are performed at the server side. It involves processing of each lecture video in the repository and prepare them for providing

search facility based on contents of the lectures.

- **On-line Activities :** These are activities performed by the server through the Internet. These include requests made by users for particular content and the responses given by the system.

4.1 Off-line Activities

These activities involve

1. Content Extraction

2. Index Generation

4.1.1 Content Extraction

Major aim of the content based video searching is to extract some textual information from videos which represents their contents. Once the extracted textual information is available, normal text search can be done on the available text data and retrieve respective video files from the repository. Lecture videos mainly contain following components.

- Lecturer Speech : Portion of video that shows instructor talking.

- Presentation Slides : Portion of video that shows slides of a presentation.

- Handwritten Notes : Portion of video that shows white paper on which instructor is writing while explaining the things.

We can make use of the above components and extract textual information from them. Optical Character Recognition (OCR) and Speech Recognition technologies are useful for the purpose of automatic extraction of textual information that represent contents of the video.

Using OCR, text present in the slides and hand written notes can be detected and extracted. Using Automatic Speech Recognition(ASR) a speech transcript can be generated which contains each word present in lecture speech.

Optical Character Recognition (OCR):

Optical Character Recognition (OCR) takes an image as a input and produces textual information present in the image as output.

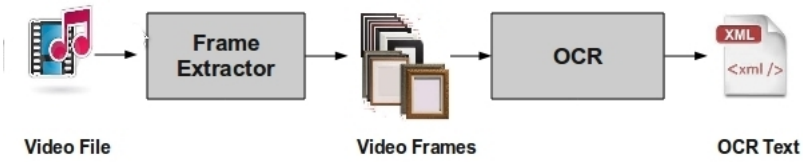

Figure 4.2: OCR Approach

To apply this technique to videos, a video is to be divided into individual frames and these frames should be given as input to OCR. The figure 4.2 shows overall process of OCR.

Open source OCR Tools

The following open source OCR tools are tested with the various image types as input and measured accuracy of their recognition. Tesseract ocr [24] is found to give better recognition accuracy when compared to others.

- **ocrad** [18]

  ocrad is able to decode only three image formats which are in pbm, pgm or ppm format.

  Commands:

  1. Convert image to pnm format

  $*convert ex.png ex.pnm*

  2. Run ocrad to store text of the image in a file

  $*ocrad -o output.txt ex.pnm*

- **gocr/jocr** [11]

  Command :

  $*gocr -i ex.pnm -o output.txt*

- **tesseract-ocr [24]**

  It takes .tif images as input and produces output text file.

Command :

$*tesseract ex.tif output.txt*

Speech Recognition :

Content Based Lecture video retrieval based on speech recognition uses automatic speech recognizer which takes an audio file of lecture and converts into textual transcription. The figure 4.3 shows the process of generating speech transcript from a lecture video.

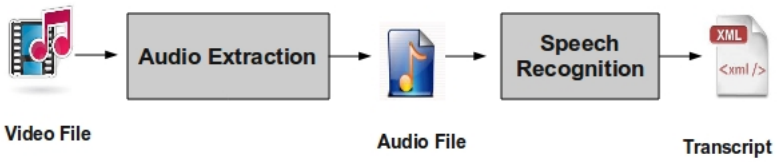

Figure 4.3: Speech Recognition Approach

First audio track has to be extracted from video lecture and it should be given to Speech Recognizer to produce its speech transcript. The speech transcript contains individual spoken words in textual form along with their time of occurrence in the lecture video.

```
<transcript>
  <tt>
     <text> deals with </text>
     <time> 7 </time>
  </tt>
  <tt>
    <text> searching </text>
    <time> 11 </time>
  </tt>
<tt>
     <text> of lectures </text>
     <time> 14 </time>
```

```
  </tt>
</transcript>
```

Example Speech Transcript

A sample speech transcript which is an XML file, looks as shown in the following paragraph. The **tt** (timed text) tags represent each individual utterance in speech. The text between **text** tags are the words present in the lecture speech. Value between **time** tag denotes starting time (in seconds) of corresponding words in the speech.

Open Source Speech Recognition Tools :

We want to use an open source speech recognition tool for our speech recognition task. We have identified the following open source tools available.

- **Sphinx 4** [22]

  Sphinx is a open source speech recognition engine from Carnegie Mellon University(CMU). It is Platform independent as it is Purely Written in Java. It can be operated in batch mode or live mode. In batch mode we can give an audio file as input and we can generate text transcript. In live mode text can be generated from spoken words. There are tools available for *acoustic models* and *language models* creation. Sphinx Train is useful for generation of *acoustic models*. CMU Language Modeling Tool kit is useful for language model creation.

- **HMM ToolKit (HTK)** [8]

  The Hidden Markov Model Toolkit(HTK) is a set of tools for building and manipulating systems based on Hidden Markov Models. HMM toolkit is developed at the Machine intelligence laboratory of the Cambridge University Engineering Department (CUED). HTK is primarily used for speech recognition. It is written in C language.

Decision on Speech Recognition Engine :

Out of the above two, we found **sphinx** suites best for our needs.The major reasons for choosing Sphinx-4 are

1. It provides easy configuration management where we need to set various parameters related to speech recognition such as *acoustic model*,language model etc.

2. It provides suitable Pre trained acoustic and language models for speech recognition purpose.

3. We can create *acoustic models* of any speech from SphinxTrain tool.

4. Sphinx 4 is developed in Java and provides *API*(Application Programming Interface)s, so it can be integrated easily into any application.

5. Good Documentation is available for sphinx.

6. As it is completely written in Java, it is highly modular and platform independent.

4.1.2 Slide Navigation

Using the slide index file generated during content extraction phase, slide navigation feature can be provided. The slide index file contains slide title and the time at which the slide occurred in the lecture video. We have proposed an algorithm for this based on slide title matching. We used OCR for extracting text from slides and implemented the algorithm to identify slide transitions in a lecture video. The figure 4.4 shows the process of generating slide index file of slide detection from a lecture video

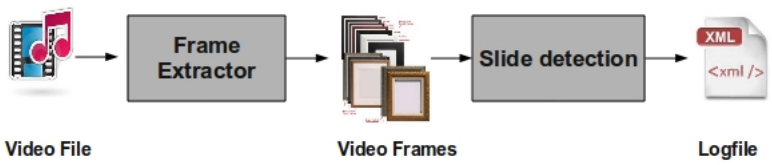

Figure 4.4: Slide Transition Detection

This component generates a slide index file for slide navigation which looks as shown in the following paragraph and is an XML file. The **title** is title of the corresponding slide. The time in seconds at which slide appears in the lecture video is indicated in **time** tags.

```
<slides>
   <slide>
    <title> Overview </title>
    <time> 13 </time>
  </slide>
  <slide>
    <title> Introduction </title>
    <time> 79 </time>
  </slide>
</slides>
```

Example slide index file for navigation

4.1.3 Index Generation

Input to this phase is the text extracted in the previous phase (Content Extraction). In this phase index tables are generated using the extracted text documents and are stored in the Content Repository. The figure 4.5 shows the index table generated using inverted indexing.

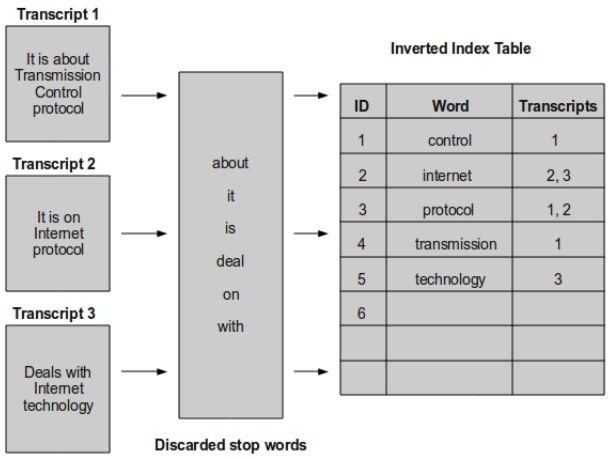

Figure 4.5: Inverted Index Table

For each video we get two files containing extracted textual information related to it which are speech transcript and slide index file for slide based navigation. We index these to files and store the indices in repository.

4.2 On-line Activities

These activities involve interaction of users with the system. Query handling is performed in this phase.

4.2.1 Query Handling

This phase takes keywords from the user input and prepares a query. Using the query, search is performed on the index tables generated as shown in the previous section. After searching the index tables, matched results are returned to user. For Index Generation and Query Handling a text search engine can be used.

Open Source Text Search Engines :

Following are some of the open source text search engines available.

- **Lucene** [2]

  Lucene is a popular text search engine library written entirely in Java. It is developed from the Apache Software Foundation. Lucene offers powerful features through a simple *API*. It can be used to index pdf files, word documents, html and xml files.

- **Indri** [9]

  Indri is a search engine from the Lemur[15] project. It is developed jointly by the University of Massachusetts and Carnegie Mellon University to build language modeling information retrieval tools. It can parse PDF, HTML and XML documents.

- **Xapian** [28]

  Xapian is also useful for providing indexing and search features for websites. It is written in C++ and can be ported to other languages also.

- **Zettair** [29]

  It is developed by the search engine group at Royal Melbourne Institute of Technology(RMIT) university. It can index large amounts of text data. It provides Modular C *API* for applications to integrate into their websites.

Decision on Text Search Engine :

Out of the above text search engines we have chosen **Lucene** because of the following reasons.

1. Integration with existing application
   Lucene is completly written in java. As our chosen speech recognition engine (sphinx4) is also written in java, It will be easy to integrate our system with Lucene.

2. It is the mostly used text search engine. List of more than 120 applications and websites that are using Lucene to provide search facility are listed at this link. [16]

3. It creates index of smaller size and search time is also very less [19].

4. It provides simple *API*s to build applications on top of it.

5. It supports ranked searching, best results returned first.

6. It can handle many powerful query types: phrase queries, wild card queries, proximity queries, range queries and more.

Following steps describe the overall process of our solution approach.

1. Take a video lecture and generate audio file containing corresponding speech of the lecture.

2. Input the audio file to automatic speech recognition engine to get a transcript which is time aligned textual representation of actual speech of the lecture.

3. Extract one frame per second from the lecture video and store them as image files in a directory.

4. Give the extracted images to Optical Character Recognizer to produce textual information present in the slides.

5. Using the slide text generated in the previous step, apply slide detection algorithm (described in section 5) to create a tag file containing slide title and time at which its transition occurs in the lecture.

6. Index the speech transcript generated in step 2 and tag file of step 5, store the index tables in content repository

7. Develop user interface of the system that takes queries from users, Searches the indexes developed in previous steps and presents them with lecture video results from repository.

Chapter 5

System Architecture and Implementation

This chapter describes various functional units in our approach and shows how they are integrated to build complete system. It contains implementation details of each component in the system architecture and also include details of using various tools related to audio extraction and speech recognition.

The figure 5.1 shows building blocks of our proposed method.

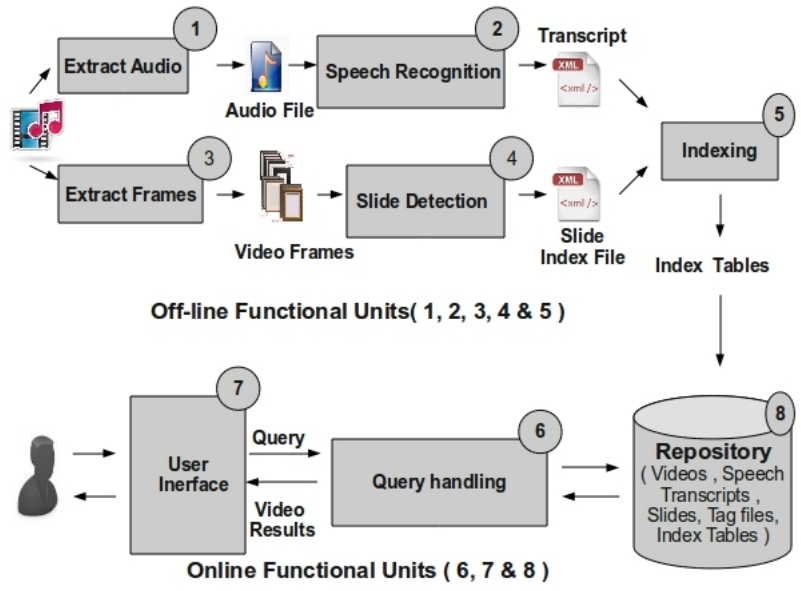

Figure 5.1: Proposed Method

5.1 Audio Extraction

This component of the system takes a lecture video as input and produces an audio file which contains lecturer speech. As speech recognition engine takes audio file as input, we need to extract audio track from the video.

5.1.1 ffmpeg

FFmpeg [6] is an open source tool which provide libraries and programs for handling multimedia data. One of the command line program provided by that software is *ffmpeg* and is useful for transcoding multimedia files. *ffmpeg* can extract audio track from a given video file and is completely independent of type of speech in it. It just takes a given video file of one format as input and produces an audio file in specified format. There will not be any effect on the original video file.

Features of *ffmpeg*:

- convert video file in one format into another format and save that as a new file
- convert audio file in one format into another format and save that as a new file
- extract audio file from a video file and save the audio file in a separate file

Running *ffmpeg*:

ffmpeg can be called from terminal using the following command.

$ *ffmpeg* -i *CS101_L10_Strings.mp4* -ar 16000 -ac 1 *CS101_L10_Strings.wav*

The above command takes a video file (*CS101_L10_Strings.mp4*) as input and produces an audio file named *CS101_L10_Strings.wav*. Properties of the output audio file are set by options -**ar** (audio rate) as 16 KHz and -**ac** (number of channels) as Mono.

5.2 Automatic Speech Recognition

This component takes audio file generated in the audio extraction unit as input and creates a speech transcript.

5.2.1 Sphinx-4 Speech Recognizer

We are using an open source speech recognition engine from CMU called Sphinx-4 [23] in our system. As mentioned earlier, any speech recognition engine uses *acoustic model* and *language model* for automatically converting spoken words into text. It also requires a dictionary that maps each word to its phoneme representation. The figure 5.2 shows sphinx-4 architecture and major components in it [23].

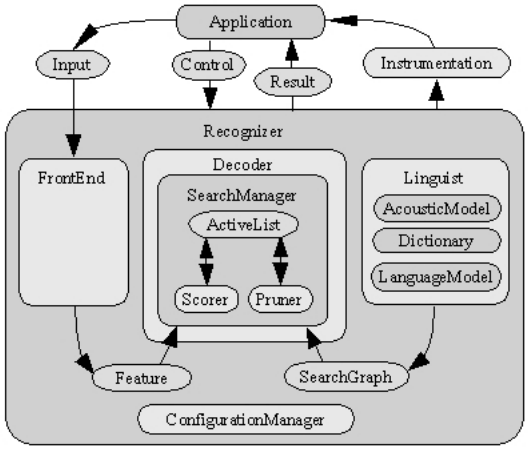

Figure 5.2: Sphinx-4 Architecture [23]

Brief description of those components is given in the following paragraph.

- **Front End**: It generates features from speech signals. The features are a sequence of vectors that represent certain characteristics of the signal.

- **Linguist**: This component creates a search graph, which contains nodes and transitions among the nodes. It represents all possible sequences of phonemes in the entire language of the task under consideration. It uses acoustic model, dictionary and language model during construction of the search graph.

- **Decoder**: It uses search graph created by linguist and features extracted from front end to map given speech signals into words. During this mapping, it creates a *search manager* which constructs scorer, pruner and active lists. The task of decoder is to search through the graph created by linguist and generate words representing the input sounds. A*ctive list* keeps track of all the current active paths in the search graph. *Scorer* is used to score each of those paths against next feature vector obtained from the front end. *Pruner* then prune the active paths through certain heuristics.

- **Configuration Manager**: It acts likes an interface and stores various parameters related to speech recognition. Some of the example parameters are like which acoustic models to use and their location in the system.

Language Model

Language model tries to capture the properties of a language, and to predict the next word in a speech sequence. A language model can be represented in JSGF Grammar or as a statistical model. JSGF grammar is useful when a small set of sentences need to be recognized. For large vocabulary continuous speech recognition statistical language model is required.

A *statistical language model* represents grammar of a language and is a file containing the probabilities of sequences of words. It gives the probability of word (X) being followed by word (Y) so that the word sequence which is having higher probability will be chosen during decoding process. Language modeling requires supplemental text files, such as journal articles, book chapters, etc., to adapt the vocabulary of the system.

CMU Sphinx provides statistical language modeling toolkit to create language models. Using the tools provided by toolkit we can generate language model from available text corpus. We have collected text corpus related to computer science courses to create language model for Sphinx-4 decoder. The text corpus includes electronic versions of text books, presentation slides and available manual transcriptions.

Creating statistical language model using CMU SLM Toolkit

1. **Text Preparation**:

   A text corpus is required for generating a statistical language model. So a set of documents have to be collected such as HTML pages, XML files, PDF files or normal text files related to the domain in which speech recognition is to be done. Input to CMU SLM Toolkit is a single text file called as reference text containing each sentence in $< s >$ and $< /s >$ tags. Process each of the collected documents and add each sentence of it to the reference text file. If the document is an XML or a HTML file corresponding parsers can be used to extract text in it. For processing PDF files, Java PDF library provided by PDFBox [10] is useful. After that cleaning of the extracted text has to be performed which includes expanding abbreviations, converting numbers to words and removing non-word items. More the text corpus better the language model created.

Generalized Framework for Generating Text Corpus :

We have developed a framework for freely collecting a large amount of text corpus related to any domain. Wikipedia provides a dump file of all the articles available in it and latest copy of the file can be found at [13]. The dump file is an XML file whose size is around 6.7gb. We parsed the XML file and indexed contents of all the articles using Lucene. So text corpus can be generated easily by firing queries from a particular domain and retrieve results from the indexes. The figure 5.3 displays different steps involved in creating large amounts of text corpus for language modeling

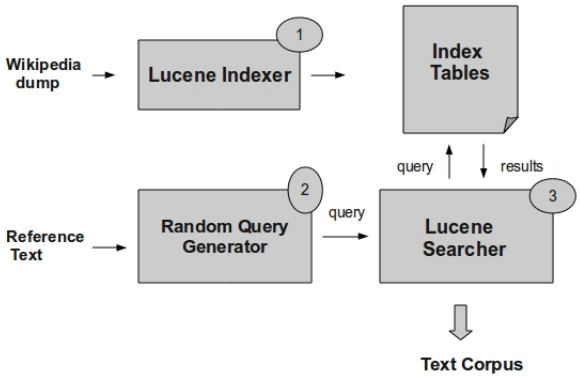

Figure 5.3: Framework for Creating large amount of Text Corpus

We collected documents related to data structures and algorithms which are in PDF format. Those documents include e-books, presentation slides, available manual transcriptions of lectures. Text is extracted using PDFBox java library and cleaning operation is performed to generate the reference text in a text file as *corpus.txt*. We also generated corpus from Wiki indexes through random queries consisting of Computer Science terms and added these sentences to *corpus.txt*. The following represents some sentences of the created text corpus.

$< s >$ there are occasions when it is useful to refer to memory locations directly and that facility is provided by pointers $< /s >$

$< s >$ in this lecture we will discuss pointer variables pointer assignment pointer arithmetic and some example programs $< /s >$

$< s >$ these values are assigned to local variables a and b $< /s >$

2. **Vocabulary Creation** :

   Generate vocabulary file which contains all the words present in text corpus generated in previous step. *text2wfreq* and *wfreq2vocab* are command line tools provided by CMU SLM Toolkit. *text2wfreq* takes a text file and gives list of words occurred in the text, along with number of occurrences of each word. *wfreq2vocab* takes output produced by *text2wfreq* and gives a list of top 20000 frequent words in the given text. The following command stores list of most frequent words present in *corpus.txt* into a file called *corpus.vocab*.

   $ *text2wfreq* < corpus.txt | *wfreq2vocab* > corpus.vocab

3. **Create language model in ARPA format**:

   text2idngram creates a file containing list of n-grams (3-gram means sequence of 3 words), and number of occurrences of each of those n-gram. In the following command, n-grams will be stored in *corpus.idngram* file.

   $ *text2idngram* -vocab corpus.vocab -idngram corpus.idngram < corpus.txt

   idngram2lm generates a list of n-grams and their probability of occurrence. After executing the following command, final ARPA format language model will be stored in *corpus.arpa*.

   $ *idngram2lm* -vocab_type 0 -idngram corpus.idngram -vocab corpus.vocab -arpa corpus.arpa

   The following are some line in the language model generated, number in the beginning of each line represents probability of occurrence corresponding sequence of words.

```
-1.8772 text file format
-1.8772 text file may
-1.1161 text file name
-1.8772 text file till
-1.8772 text file to
```

4. **Convert language model from ARPA to binary (DMP) format** :

   Sphinx-4 accepts a language model in binary (DMP) format, which can be created by using *sphinx_lm_convert* tool provided by CMU language modeling toolkit.

   $ *sphinx_lm_convert* -i weather.arpa -o weather.lm.DMP

Pronunciation Dictionary

It contains phoneme representation of each word present in the language. CMU Sphinx provides pronunciation dictionary *cmudict* [5] for English which contains over 125,000 words and their transcriptions (phoneme sequences). Pronunciations need to be generated for words which are not available in the dictionary. Using *pronounce* tool [20] of CMU Sphinx the pronunciations for the new words in our vocabulary can be created as shown in the following command. *newWords.dic* file contains pronunciations of newly found words in our language.

$ *pronounce* < newWords.vocab > newWords.dic

The following represents pronunciation for some words which are not available in the CMU dictionary.

```
DEEPAK D IY EH P AH K
PHATAK F AE T AH K
DUMBO D AH M B OW
KRESIT K R EH S AH T
```

Add the contents of the dictionary of new words to *cmudict* for building complete pronunciation dictionary of all words in language model.

Acoustic Model

The *acoustic model* represents how individual characters are pronounced, also known as phonemes. It gives statistical representation of distinct sounds that make up each word in a language.

Acoustic model creation needs large amount of speech corpus and their corresponding accurate speech transcriptions. As it is a time consuming process to generate new acoustic models, we configured sphinx-4 system with open source acoustic models provided by CMU Sphinx. It provides Wall Street Journal (WSJ) and HUB4 acoustic models which are trained for the recognition of American English accents. So recognition accuracy is very low (less than 10%) for our lectures which are in Indian English accent.

In order to improve the recognition accuracy of Sphinx-4 decoder, we have adapted the HUB4 models for recognizing our lecturer speech. *Sphinx base* and *Sphinx Train* are the tools from CMU Sphinx which are useful for the acoustic model creation and adaptation. After the adaptation the recognition accuracy is increased to 62%.

The adaptation requires manually transcribing 3 to 4 minutes of speech of lecturer. This process takes the segmented audio files and manually transcribed sentences as input and generates acoustic models adapted for the lecturer speech.

Adapting available acoustic models

Adaptation of an existing acoustic model improves accuracy of speech recognition. That means we can adapt the models to match the accents of our users or to match our recording environment. HUB4 acoustic models are created by using speech corpus of American English. The HUB4 models can be adapted to recognize other accents such as UK English, Indian English etc. The following steps explain the adaptation process.

1. Preparation of Adaptation corpus

   The following files are needed for adapting an existing acoustic model to our speakers.

   i. adapt.txt : List of sentences used for adaptation.

   Example :

   ```
   so each operator works on one particular item
   two names defined a and b as integer numbers
   but this program is easily understood by all
   ```

   ii. adapt.dic : Contains phoneme representations for the words present in *adapt.txt* whose creating is discussed in the previous section.

   Example :

   ```
   CALCULATES K AE L K Y AH L AH T S
   DEFINED D IH F AY N D
   DISCUSS D IH S K AH S
   DISCUSSED D IH S K AH S T
   EACH IY CH
   EASILY IY Z IY L IY
   ```

   iii. Recorded Audio Files : An audio file in *wav* format is required for each of the sentences present in *adapt.txt*.

   iv. adapt.fileids : contains list of audio file names(with out extension) which are created in previous step Example :

```
ad_01
ad_02
ad_03
```

v. adapt.transcription: Each line of the file contains two parts and part is separated by blank spaces. First part is one of the sentence of *adapt.txt* file which is included in $< s >$ and $< /s >$ tags. Second part is file id of audio file of the corresponding sentence.
Example :

```
<s>SO EACH OPERATOR WORKS ON ONE PARTICULAR ITEM</s>(ad_01)
<s>TWO NAMES DEFINED A AND B AS INTEGER NUMBERS</s>(ad_02)
<s>BUT THIS PROGRAM IS EASILY UNDERSTOOD BY ALL</s>(ad_03)
```

2. Generating acoustic feature files
   sphinx_fe from Sphinx base is useful in creating acoustic model feature files from the audio WAV files specified in *adapt.fileids*. We have to use the same parameters used by the existing acoustic model to extract these features. Here we used HUB-4 acoustic model whose parameters can be found in *feat.params*. Here *hub4* is a directory containing existing acoustic model files.

```
$ sphinx_fe -argfile hub4/feat.params \
    -samprate 16000 \
    -c adapt.fileids \
    -di . -do . -ei wav -eo mfc \
    -mswav yes
```

3. Accumulating observation counts
   bw implements a part of Baum-Welch algorithm [3], which is useful in collecting necessary statistics for parameter estimation of the adapted models.

```
$ bw -hmmdir hub4 \
    -moddeffn hub4/mdef \
    -ts2cbfn .cont. \
    -feat 1s_c_d_dd \
    -dictfn adapt.dic \
```

```
    -ctlfn adapt.fileids \
    -lsnfn adapt.transcription \
    -accumdir .
```

4. Updating the acoustic model files with Maximum a posteriori (MAP) estimation.
   map_adapt creates adapted acoustic models by updating given models. It uses the statistics generated in previous step. Adapted acoustic models will be stored in a directory called *adapted_hub4*.

```
$ cp -a hub4 adapted_hub4
$ ./map_adapt \
    -meanfn hub4/means \
    -varfn hub4/variances \
    -mixwfn hub4/mixture_weights \
    -tmatfn hub4/transition_matrices \
    -accumdir . \
    -mapmeanfn adapted_hub4/means \
    -mapvarfn adapted_hub4/variances \
    -mapmixwfn adapted_hub4/mixture_weights \
    -maptmatfn adapted_hub4/transition_matrices
```

Configuration Management

It is an XML file containing various parameters set for sphinx-4 system. The following shows a part of the Configuration file which is the list of parameters set for acoustic models.
Acoustic Model Configuration :
Each parameter contains a list of properties and each property is specified by name, value pair. For example location property of Sphinx3Loader gives actual path of the acoustic models.

```
<component name="hub4"
  type="edu.cmu.sphinx.linguist.acoustic.tiedstate.
TiedStateAcousticModel">
  <property name="loader" value="sphinx3Loader"/>
  <property name="unitManager" value="unitManager"/>
</component>
```

```
<component name="sphinx3Loader" type="edu.cmu.sphinx.
linguist.acoustic.tiedstate.Sphinx3Loader">
  <property name="logMath" value="logMath"/>
  <property name="unitManager" value="unitManager"/>
  <property name="location"
  value="file:/home/vijay/AM/adapted_hub4"/>
</component>
```

Speech to Text Conversion

The following code shows sequence of steps in recognizing an audio file using Sphinx-4 recognizer. First it initializes configuration manager whose path is specified in *configURL* variable. Then it calls the recognizer using parameters set in the configuration file.

```
ConfigurationManager cm = new ConfigurationManager(configURL);
Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
Result result;
AudioFileDataSource dataSource = (AudioFileDataSource)
cm.lookup("audioFileDataSource");
dataSource.setAudioFile(audioURL, null);
FileWriter fw = new FileWriter("transcription.txt");
while ((result = recognizer.recognize())!= null)
{
  String resultText = result.getBestResultNoFiller();
  fw.write(resultText+" ");
}
```

Transcript Generation

Using AlignerGrammar class of sphinx-4 time stamps of each word in the transcript can be generated. The time stamps generated by sphinx is as shown below.

```
quest(1728.89,1729.26) and(1729.26,1729.75)
acknowledgement(1729.75,1730.37)
associates(1730.37,1731.49)
```

The numbers in brackets indicates start and end times (in seconds) of occurrence of the word in the lecture video. We converted the above format into an XML format and stored that in a separate file.

5.3 Video Frames Extraction

This takes lecture video as input and generates individual frames.

5.3.1 ffmpeg

We can use the same command line tool *ffmpeg* provided by FFmpeg for this purpose also. It takes a video file as input, extracts frames at specified timings and store each of the extracted frame as an image file. Following command shows running *ffmpeg* to extract one frame per second from a lecture video.

```
$ ffmpeg -i CS101_L10_Strings.mp4 -r 1 -s 4cif \
      -f image2 image_%4d.jpeg
```

After extracting one frame per second from the given lecture video, image files will be saved as image_0001, image_0002, etc.

5.4 Slide Detection

Identifies slide timings in a lecture video and creates a file containing slide title and time at which slide occurred in the video. This takes frames extracted from video and examines each frame to identify slide transition. We have designed an algorithm to identify slide transitions to generate the slide index file for providing slide navigation feature.

5.4.1 Slide Text Extraction

Our algorithm is based on text present in each of the generated image. So we need to extract textual data present on those images. For this purpose Optical Character Recognition(OCR) is useful and Tesseract OCR [24] is one of the open source tool which gives better accuracy.

Tesseract OCR

It takes a binary, gray scale or color image and save text present in the image into a separate file. It accepts images which are in TIFF format, so we need to convert images in other formats to TIFF. The following commands are used for text extraction from an image in JPEG format.

```
$ convert image_0001.jpeg image_0001.tif
$ tesseract image_0001.tif image_0001.txt
```

The text file image_0001.txt contains text present in the given image file.

5.4.2 Slide Detection Algorithm

The algorithm is based on text generated from the images of lecture video frames. After extracting text from some images following observations can be made. Different portions that present in a lecture video are instructor element, presentation portion and instructor writing element.

- If the image is of instructor, extracted text contains nothing or some times it also give few random characters due to errors while recognizing.
- If the image is of a presentation slide or Written text portion, extracted text contains the data present in it. As the title will be in larger font than other data, the title can be extracted with more than 95% accuracy.
- After removing, non alpha numeric characters from extracted data of a slide/written text, we get text data whose first line indicates title of the slide/written text.

Extract images from a lecture video one per a second. Name the images like image_0001, image_0002, etc., such that the number part 0001, 0002 represents time in seconds at which the image is present in the lecture. The naming can be done automatically through option provided while running the *ffmpeg* command for extracting images. Process each of the image and extract text, store title and time of slide/written portion in separate arrays.

Title Matching Algorithm for Slide Detection

Input : Two arrays title[] and time[] which contains title and time of each slide image of the lecture video.

Output : An XML file which contains title of each detected slide and time at which it occurs in

the lecture video.

Overview : The algorithm checks each title in title array and if three successive titles matches then that will be taken as starting time of the particular slide. Then it finds next title in the title *array* such that it does not match with the previously detected title and matches with next two of it.

Algorithm SlideTransitionDetection(title[],time[])

```
Integer i = 0
String prev = null;
List indices = null;
while i < titles.length-1
begin
    if !titles[i].equals(prev) && matchesNextTwo(titles,i)
      indices.add(i);
      i = findNextSlide(titles,title[i],i+3)
        if i == -1
            return;
        endif
      prev = titles[i];
      indices.add(i);
      i = i + 2;
    endif
    i = i + 1;
end

function findNext(titles[],title,index)
{
  Integer i = index, next = -1;
  while i < titles.length -2
    begin
      if !title.equals(titles[i]) && matchesNextTwo(titles,i)
        next = i;
        return next;
```

```
      endif
      i = i + 1;
    end
  return next;
}

function matchesNextTwo(titles[],index)
{
  Integer i = index;
    if (i < titles.length-2) &&
      (titles[i].equals(titles[i+1])&&
        titles[i].equals(titles[i+2]))
      return true;
  return false;
}
```

At the end of the algorithm *indices* list contains each of the index of detected slide. From the index we can get title, time of that particular slide and write those two entries in an XML file to generate slide index file.

5.5 Index Generation

Index Generation constructs index tables from speech transcripts and stores them in database. The XML files which are generated in the transcript generation phase and slide index file created in slide detection algorithm are input to this component.

5.5.1 Lucene

Lucene Java library contains classes for creating indices of a set of documents. IndexWriter class is one of them and useful to create indices and store them in a given directory. Lucene provides libraries which works for indexing textual data. A normal text file can be read line by line or in any order and use IndexWriter to generate indices. In our case speech transcript is an XML file and so first we need to perform parsing on the XML to extract text, then use that data

for indexing. Parsing of XML transcripts can be done using a Java library provided by Apache called commons digester [1].

The following code shows adding contents of a speech transcript to index directory. The index generation component processes each speech transcript file and store indices in the directory which is given as argument to IndexWriter.

```
IndexWriter writer = new IndexWriter(indexDir, analyzer,
     createFlag, IndexWriter.MaxFieldLength.UNLIMITED);
Document transDoc;
transDoc.add(new Field("transcript", transcript,Field.Store.YES,
  Field.Index.ANALYZED));
writer.addDocument(transDoc);
```

We created a Graphical Interface for performing all the above discussed off-line activities. The figure 5.4 shows starting window of the tool, where input video lecture is to be selected. The interface contains a tab on the top for each of the processing step involved in off-line.

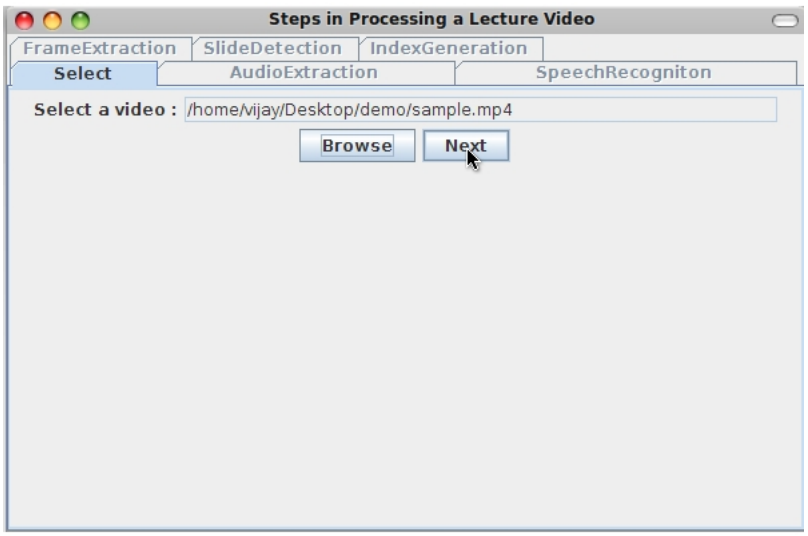

Figure 5.4: Starting window of our off-line processing tool

The figure 5.5 shows Speech Recognition step in the process.

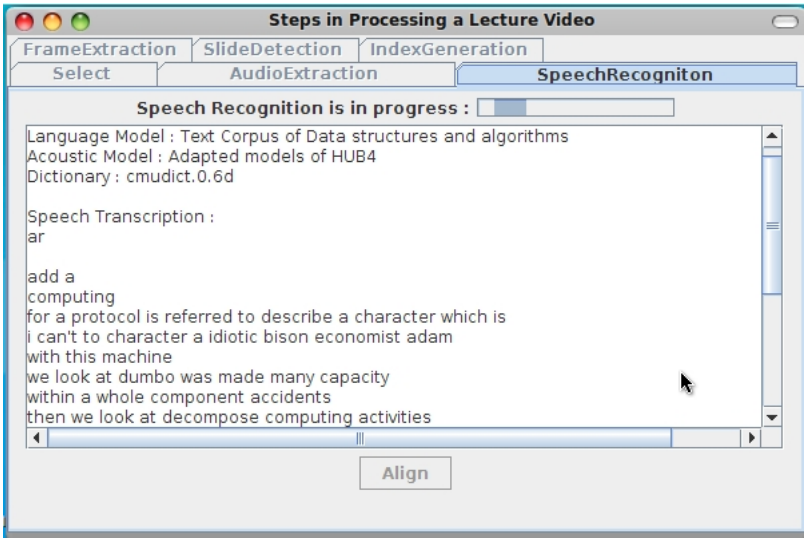

Figure 5.5: Speech Recognition step of our off-line processing tool

The figure 5.6 shows a window which indicates progress of Slide Detection step of off-line processing of a lecture video.

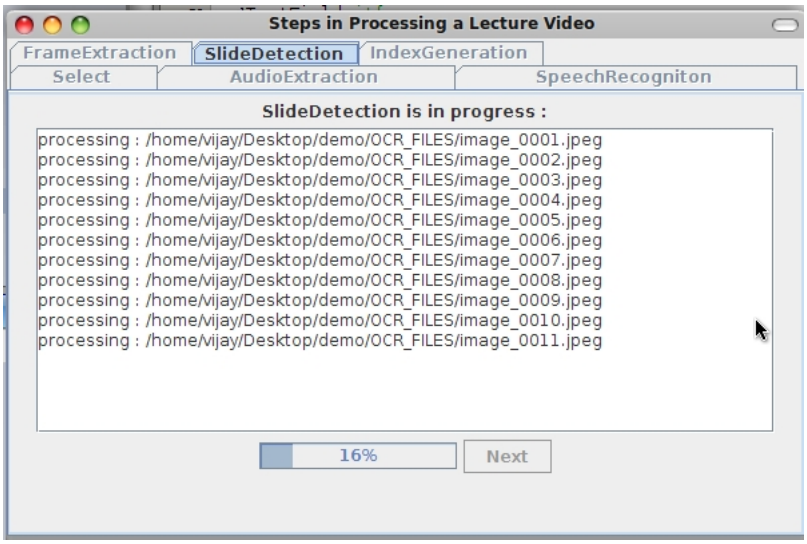

Figure 5.6: Slide Detection step of our off-line processing tool

5.6 Query Handling

It takes user input keywords and searches in database for matching video lectures and gives the lecture video results as output. This component development involved server side programming which has been done using Java Servlets.

5.6.1 Lucene

The Lucene Java library provides classes for performing search functions on index tables created using IndexWriter. IndexSearcher is the useful for searching in the available indexes. The following lists major block of code that performs search operation. IndexSearcher takes index directory and query string as inputs, provide results in the form of TopDocs.

```
FSDirectory        dir = FSDirectory.open(new File(indexDir));
IndexSearcher searcher = new IndexSearcher(dir,true);
QueryParser parser = new QueryParser(Version.LUCENE_CURRENT,
                  "transcript", analyzer);
Query query = parser.parse(queryStr);
TopDocs hits = searcher.search(query, 20);
```

5.7 User Interface

User interface contains a search box in which user can enter keywords to search for videos in which the particular topic is discussed. After retrieving results from repository, list of lecture videos are displayed in the interface. In each video of the list, portions where the searched keyword is present in lecture speech as shown the figure 5.7 which show search results for the query *binary search*.

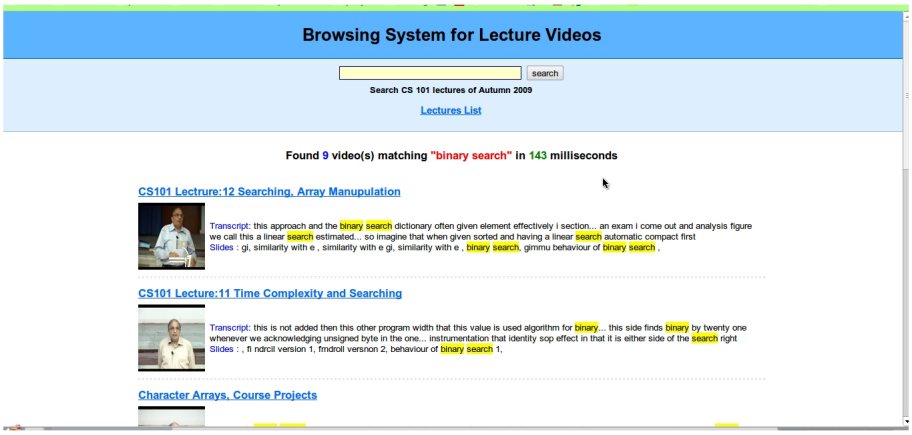

Figure 5.7: Search Results Page of Proposed System

On clicking on any highlighted portion, a new window appears which contains media player, speech transcript and slide navigation links. So in the user interface we provide more

navigation features for the users. Users can navigate to any particular time within the video by clicking on links of slide index or on any word in speech transcript. User also can go to any time in video duration by clicking at a point in video time line of media player.

In the *user interface* we also display list courses for which lecture videos are available. By clicking on any course, users can get a list of all the available lecture videos in that particular course. When a user clicks on any video then a new window, which contains above mentioned navigation features.

We provided web interface for the users, where they enter keywords and get results. These web pages have been developed using HTML, CSS, Java Script and JSPs.

The figure 5.8 shows user interface of our system. As shown in the figure, the interface contains the following three sections

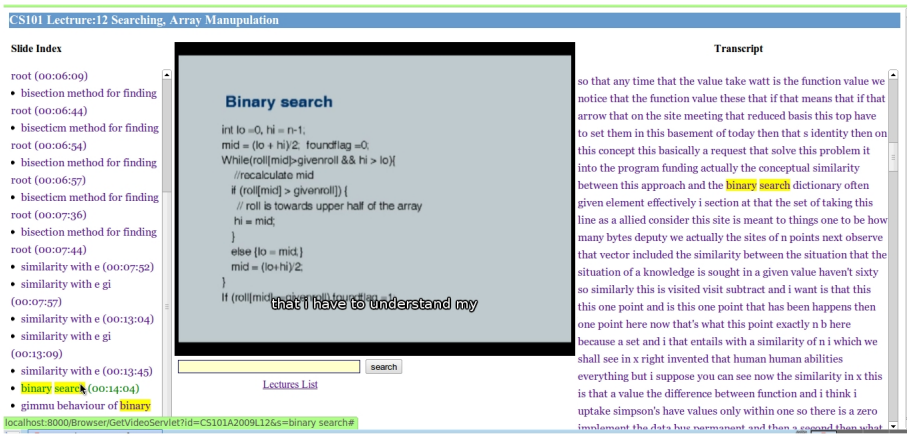

Figure 5.8: User Interface of Proposed System

1. *Video player*: As shown in the middle part of the figure 5.8, it plays video of the selected lecture. Users can navigate to any particular location by clicking on video time line of the player.

2. *Speech transcript*: It is the right part of the figure 5.8. It displays textual representation of the lecturer speech. User can click on any particular word of the speech transcript and can jump directly to that location.

3. *Slide Navigation*: As shown in the figure 5.8, It gives hyper links to slides of the video being played on the left part. Each link represent title of the slide and time at which it starts. So users can click on any required slide and go directly to that location of the video.

5.7.1 JW Player

For playing video in user interface we are using freely available version of JW player [12]. It supports various plug ins for controlling the player and handling events. Navigation based on speech transcript has been done using java script and JW Player APIs.

5.8 Content Repository

It stores all the information related to lecture videos in the repository. It contains actual video recordings of the lectures, presentation slides, speech transcriptions generated during content extraction and slide index files created for slide navigation and any other related courseware.

Chapter 6

Evaluation and Experiments

This section presents evaluation metrics and experimental results we have got during the implementation. To build basic functionality of the system, we took the lectures of CS 101: Computer Programming and Utilization by Prof. Deepak B Phatak. We collected text corpus related to programming and data structures to create language model using CMU language modeling toolkit. Then we adapted HUB4 acoustic models to voice of the professor using tools provided by sphinx train. We configured sphinx4 recognizer with the created language and acoustic models to provide speech transcriptions. We applied slide detection algorithm on those lectures to generate slide index files. We have provided the search facility in those 20 lectures by indexing the speech transcripts and slide index files using lucene text search libraries. User can enter keywords and our system searches in the indexed transcripts, and then retrieves videos that match user input and they can browse lecture video by using slide index provided in the interface.

6.1 Evaluating Speech Recognition

In order to evaluate the Sphinx-4 system, we have to compare the output text called hypothesis with the actual transcription which is called as reference. Three error types are distinguished in this process

- *Substitution*: deals with words that are wrongly recognized (including singular and plural differentiation).
- *Insertion*: additional word in the hypothesis that is absent in the reference.
- *Deletion*: word present in the reference but not in the hypothesis.

Word Error Rate (WER) and Word Accuracy (WA) are useful for measuring the performance of speech recognition [27]. WER is defined as

$$WER = \frac{S + D + I}{N}$$

where

- N is the total number of words in reference text
- S is the number of substitutions
- D is the number of deletions
- I is the number of insertions

WA is defined as

$$WA = 1 - WER = \frac{N - S - D - I}{N}$$

We have tested the performance of Speech Recognition using audio files collected during adaptation phase. We created 180 wav files for adaptation of HUB4 acoustic models. In those 180, we randomly selected 30 files for testing. Out of the remaining 150 files, we again randomly selected some set of files, adapted HUB4 models using those files and tested the recognition accuracy of Sphinx 4 recognizer after configuring that with the adapted acoustic models. The following table 6.1 shows the results of speech recognition The figure 6.1 shows results of

Adaptation files	Words in test files	Matches	Accuracy(%)
0	127	22	13
30	119	43	31
60	124	70	52
90	120	76	59
120	110	69	61
150	123	82	62

Table 6.1: Speech Recognition Results

speech recognition graphically.

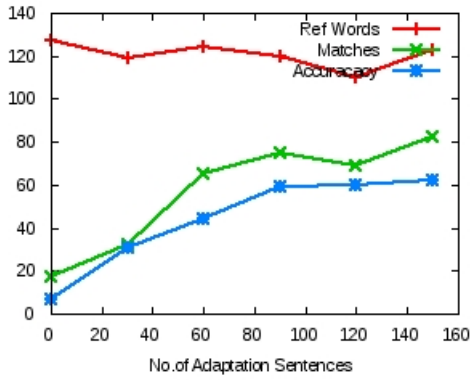

Figure 6.1: Results of Speech Recognition

6.2 Evaluating Slide Transition Detection

Accuracy of our algorithm for detecting slide transitions can be evaluated using following metrics.

Recall :

This measures the ability of our algorithm in detecting all the slides present in a lecture video

$$Recall = \frac{Number\ of\ correctly\ detected\ slides}{Actual\ number\ of\ slides\ present\ in\ video}$$

Precision :

The metric measures the false detection rate out of the detected slides

$$Precision = \frac{Number\ of\ correctly\ detected\ slides}{Total\ number\ of\ detected\ slides}$$

Average Timing Error (ATE) :

It indicates performance of the algorithm in identifying slides accurately according to their timing in the corresponding lecture video. Let N be total number of slides detected by the algorithm.

$$AverageTimingError(ATE) = \frac{\sum_{i=1}^{N} ATE_i}{N}$$

Where

ATE_i = Actual time of the slide(secs) $-$ Time detected by our algorithm(secs)

Here the time means at which second in lecture video the slide occurs.

We have evaluated the performance of the slide detection algorithm using first five lectures of CS 101. We manually identified, slides in each video and compared them with results of our algorithm. The table 6.2 shows the results of slide detection of five lectures.

Video	Actual slides	Detected slides	Correctly detected	Duplicates	Recall (%)	Precision (%)	ATE (sec)
Lecture_01.mp4	14	14	12	0	100	85	0.6
Lecture_02.mp4	20	20	16	6	100	80	6
Lecture_03.mp4	12	11	11	2	91.6	100	0
Lecture_04.mp4	32	30	26	9	93.7	86.6	1.25
Lecture_05.mp4	32	30	28	5	93.6	93.3	2.1
Total	110	105	93	18	95.4	88.5	2.1

Table 6.2: Slide Detection Results

6.3 Evaluating Video Retrieval

An information retrieval system can be evaluated based on two parameters called recall and precision [21]. Here document means video lecture in our case. As our system also deals with information retrieval we are going to use those metrics for evaluation of the system. Recall and Precision values describe the search quality of a information retrieval system.

Recall :

performance of the search system in finding relevant documents can be measured using Recall. It is a measure of the ability of a retrieval system to present all relevant documents.

$$Recall = \frac{Number\ of\ relevant\ videos\ retrieved}{Total\ number\ of\ existing\ relevant\ videos}$$

Precision :

Precision measures how well the system filters out the irrelevant. It is a measure of the ability of a system to present only relevant documents.

$$Precision = \frac{Number\ of\ relevant\ videos\ retrieved}{Total\ number\ of\ videos\ retrieved}$$

The goal of an information retrieval system is to maximize both recall and precision.

We have built the system to provide search facility in the videos lectures of CS 101 course. The table 6.3 shows search quality of Lucene based on the metrics mentioned in the above section. Quality package under benchmark of Lucene allow us to measure Recall and Precision of the Lucene indexing mechanism.

The process of measuring recall and precision requires following three files as input.

1. *Index file*: Lucene generated file during indexing process
2. *Topics file*: This contains list of search queries for which we want to measure the recall and precision.
3. *Qrel file*: This is filled with file names of expected video lectures for each of the query mentioned in topics file.

No.of queries tested	30
Avg Search seconds	0.004
Recall	0.72
Avg Precision	0.91

Table 6.3: Search Quality Results

Chapter 7

Conclusion and Future Work

We have proposed design of a web based browsing system for lecture videos and described the details of its implementation. It facilitates users to easily find required video contents at a fine-level of granularity. By following the approach and set of tools mentioned here, lecture video repositories can be made easily accessible for their users. We also proposed a generalized framework for creating language models, which is very useful for collecting large amounts of text corpus related to a particular domain. For slide detection component of our system, we designed an algorithm which uses Optical Character Recognition (OCR) and is based on slide title matching.

We have built a prototype of the system by providing search facility in lecture videos on CS 101: Computer Programming and Utilization course. We have performed experiments to test the performance of our system, we achieved a recall of 0.72 and an average precision of 0.87. We have achieved an average word recognition accuracy of 62% in the speech transcription process. The low value of recognition accuracy affected the decrease in recall and precision results. We analyzed the performance of our slide detection algorithm and on an average it is successful in detecting 95% of slides of a lecture video. The performance of speech recognition can be improved in future to get better video retrieval results and more courses can be added to the repository. The detection algorithm also can be improved to reduce number of duplicate slides while detection.

Bibliography

[1] *Apache Commons Digester Component for parsing XML documents*. http://commons.apache.org/digester/.

[2] *Apache Lucene*. http://lucene.apache.org/java/docs/index.html.

[3] *BaumWelch algorithm for parameter estimation during acoustic model adaptation*. http://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm.

[4] *CDEEP , IIT Bombay*. http://www.cdeep.iitb.ac.in/.

[5] *CMU Pronunciation Dictionary*. http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

[6] *FFmpeg*. http://www.ffmpeg.org/.

[7] *freevideolectures.com*. http://www.freevideolectures.com/.

[8] *HTK*. http://htk.eng.cam.ac.uk/.

[9] *Indri*. http://www.lemurproject.org/indri/.

[10] *Java PDF Library*. http://pdfbox.apache.org/.

[11] *jocr*. http://jocr.sourceforge.net/.

[12] *JW Player*. http://www.longtailvideo.com/players/jw-flv-player/.

[13] *Latest dump of wikipedia english articles*. http://download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2.

[14] *Lecture Browser , MIT*. http://web.sls.csail.mit.edu/lectures/.

[15] *Lemure*. http://www.lemurproject.org/.

[16] *List of Applications that are using Lucene*. http://wiki.apache.org/lucene-java/PoweredBy.

[17] *nptel*. http://www.nptel.iitm.ac.in/.

[18] *ocrad*. http://www.gnu.org/software/ocrad/.

[19] *Open Source Text Search Engines Evalution Results*. http://wrg.upf.edu/WRG/dctos/Middleton-Baeza.pdf.

[20] *Pronounce tool to generate pronunciation for new words*. https://cmusphinx.svn.sourceforge.net/svnroot/cmusphinx/trunk/logios/Tools/MakeDict/.

[21] *Recall and Precision*. http://en.wikipedia.org/wiki/Precision_and_recall.

[22] *sphinx*. http://www.speech.cs.cmu.edu/.

[23] *sphinx 4 speech recognizer*. http://cmusphinx.sourceforge.net/sphinx4/.

[24] *tesseract-ocr*. http://code.google.com/p/tesseract-ocr/.

[25] *videolectures.net*. http://www.videolectures.net/.

[26] *VideoPulp: Official Partners for Slide Index feature in NPTEL*. http://www.videopulp.in/.

[27] *Word Error Rate*. http://en.wikipedia.org/wiki/Word_error_rate.

[28] *xapian*. http://xapian.org/.

[29] *zettair*. http://www.seg.rmit.edu.au/zettair/.

[30] Alan F.Smeaton. Techniques used and open challenges to the analysis,indexing and retrieval of digital video. *Adaptive Information Cluster and Center for Digital Video Processing , Dublin City University*, 2006.

[31] Sridhar Iyer Ganesh Narayana Murthy. Study element based adaptation of lecture videos to mobile devices. *National Conferece on Communications,Chennai*, 2010.

[32] Scott Cyphers Igor Malioutov David Huynh James Glass, Timothy J. Hazen and Regina Barzilay. Recent progress in the mit spoken lecture processing project. *INTERSPEECH*, 2007.

[33] Jesse Jin and Ruiyi Wang. The development of an online video browsing system. *ACM International Conference Proceeding series; Vol. 147*, 2001.

[34] Janghwan Lee. A video browser based on closed captions. *IEEE Transactions on Consumer Electronics*, August 2006.

[35] H. Sack and J. Waitelonis. Automated annotations of synchronized multimedia presentations,. *Proc. Workshop Mastering the Gap: From Information Extraction to Semantic Representation (ESWC 06)*, 2006.

Chapter 8

Publication

Vijaya Kumar Kamabathula and Sridhar Iyer, *Automated Tagging to Enable Fine-Grained Browsing of Lecture Videos*, IEEE International Conference on Technology for Education (T4E) - 2011, IIT Madras, July 14-16, 2011.