Indexing and Keyword Search to Ease Navigation in Lecture Videos

Tayfun Tuna, Jaspal Subhlok, Shishir Shah
University of Houston
Computer Science Department
Houston, TX 77204-3010
ttuna@uh.edu, jaspal@uh.edu, shah@cs.uh.edu

***Abstract*——Lecture videos have been commonly used to supplement in-class teaching and for distance learning. Videos recorded during in-class teaching and made accessible online are a versatile resource on par with a textbook and the classroom itself. Nonetheless, the adoption of lecture videos has been limited, in large part due to the difficulty of quickly accessing the content of interest in a long video lecture. In this work, we present "video indexing" and "keyword search" that facilitate access to video content and enhances user experience. Video indexing divides a video lecture into segments indicating different topics by identifying scene changes based on the analysis of the difference image from a pair of video frames. We propose an efficient indexing algorithm that leverages the unique features of lecture videos. Binary search with frame sampling is employed to efficiently analyze long videos. Keyword search identifies video segments that match a particular keyword. Since text in a video frame often contains a diversity of colors, font sizes and backgrounds, our text detection approach requires specialized preprocessing followed by the use of off-the-shelf OCR engines, which are designed primarily for scanned documents. We present image enhancements: text segmentation and inversion, to increase detection accuracy of OCR tools. Experimental results on a suite of diverse video lectures were used to validate the methods developed in this work. Average processing time for a one-hour lecture is around 14 minutes on a typical desktop. Search accuracy of three distinct OCR engines – Tesseract, GOCR and MODI increased significantly with our preprocessing transformations, yielding an overall combined accuracy of 97%. The work presented here is part of a video streaming framework deployed at multiple campuses serving hundreds of lecture videos.**

Keywords: video search; optical character recognition; video indexing; lecture videos; distance learning.

I. INTRODUCTION

Video of classroom lectures is a versatile learning resource. It is often made available as additional material to a conventional course, as the core of distance learning coursework, or posted publicly for community learning or as reference material. An evidence of popularity of lecture videos is thousands of complete courses posted on portals such as MIT OpenCourseware and Apple's iTunes University. At the University of Houston, video lectures have been widely used for over a decade to enhance STEM coursework. Typically, Tablet PCs that allow free mixing of prepared (PowerPoint) viewgraphs with hand annotations and illustrations are employed for teaching and the lectures simultaneously recorded on the PC itself.

Prior research has established that the recorded Tablet PC lectures are a powerful resource on par with a textbook and classroom experience, and that a major weakness of the video format is the inability to quickly access the content of interest in a video lecture [5,13]. The importance of videos in relation to other resources is also highlighted in Fig. 1.

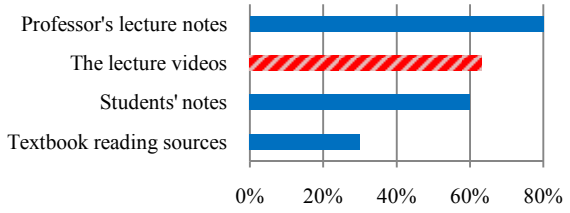

Figure 1. Student reported relative importance of lecture video and other learning resources

This paper reports on how indexing and search is implemented for ***ICS videos***: videos enhanced with *Indexing, Captioning,* and *Search* that are designed for quick access to video content. **Indexing** adds logical index points, each in the form of a snapshot representing a video segment that can be accessed directly; **Captioning** adds the text of the video lecture in a separate panel; **Search** enables identification of video segments that match a keyword.

A key goal of this project is fully automatic conversion of standard videos to ICS Videos. The framework of ICS video project is depicted in Fig. 2. First, index points are identified with analysis of video frames by applying automatic indexing algorithms.. Second, index points are saved to a database and all the images extracted by the indexer are sent to the search framework. Search framework is developed with the adoption of OCR technologies along with a suite of image transformations. After the text from the images is gathered, the information is stored in the database for the use of the ICS video player. Currently, captioning is done manually. All these aspects are integrated in a unified ICS Video player shown in Fig. 3.

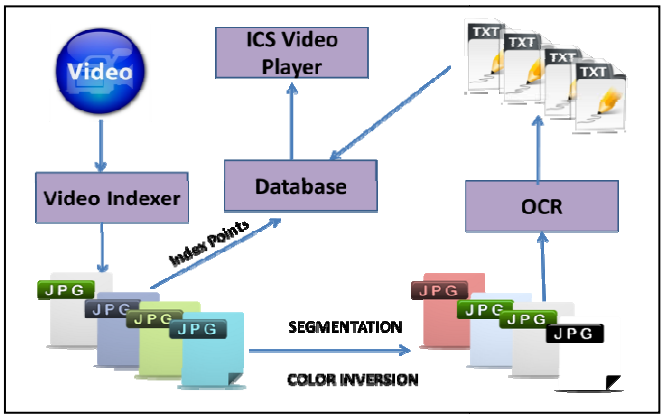

Figure 2. ICS video project framework

Figure 3. ICS video player

II. Related Work

The idea of automatically capturing videos of classroom lectures, conference presentations, and talks followed by presenting / distributing them as videos has been widely used for several years. These videos are generally recorded by camera(s) operated by professionals or installed in the lecture/presentation rooms [8]. Lectem II employs the touch-sensitive screen technology to build a "digital desk," which is shown to be able to effectively support and transparently capture the standard classroom lecturing activity. Recorded lectures can be edited and automatically uploaded to a Web server and then viewed by students with a standard streaming video player [3].

Project Tuva from Microsoft Research features searchable videos along with closed captions and annotations [7]. It also features division of the video time-line into segments where a segment represents a topic taught in the lecture. However, the division of video into segments is done manually. A related technology known as hypervideo can synchronize content inside a video with annotations and hyperlinks [14]. Hypervideo allows a user to navigate between video chunks using these annotations and hyperlinks. However, the addition of annotations and hyperlinks is manual.

A fully automatic method for summarizing and indexing unstructured presentation videos based on the text extracted from the projected slides is proposed in [8]. They use changes of text content in the slides as a means to segment the video into semantic shots. Once text regions are detected within key frames, a novel binarization algorithm, Local Adaptive Otsu (LOA), is employed to deal with the low quality of video scene text. We are inspired by this work and employ the concepts of thresholding to images and the use of OCR tools. Authors of [10] from Augsburg University present a project named MOCA where they employ Automatic Text Segmentation, a combination of text localization, and text recognition, for video indexing. They used OCR engines to detect the text from TV programs. The idea of using multiple snapshots of the same scene is generally not applicable to classroom lecture videos. An index structure-based method to efficiently search short video clips in large video collections is presented in [6]. ClassX [2] allows interactive pan/tilt/zoom while watching the video. It also offers automatic tracking of the lecturer and employs slide recognition technology, which allows automatic synchronization of digital presentation slides with those appearing in the lecture video.

The key features of the work presented in this paper that stands it apart from most previous work are i) automation of segmentation of videos, ii) deployment of image transformations with OCR for enhanced video search, iii) an approach based on video frame analysis that is independent of any hardware or presentation technology. Focus is on PC based screen capture that leads to excellent resolution and low production cost. Furthermore, we are only interested lecture videos which allow us to deploy several context specific optimizations.

III. Indexing

Indexing is the task of dividing a lecture video into segments that contain different topics. This task is accomplished by first identifying all *transition points*, i.e., places where the scene on the video changes significantly. A subset of these transition points are selected as *index points* and are the starting points of video segments presented to the user.

A. Detection of Transition Points

Detection of transition points is based on a comparison of successive frames in the video. Corresponding pixels in successive frames are considered different if they differ by a minimum *RGB threshold* when the RGB values of the pixels are compared. Successive frames constitute a transition point if the fraction of pixels that are different based on the RGB criteria exceeds a minimum threshold that we refer to as the *transition point threshold*. The reason for using thresholds to identify transition points is that frames corresponding to the same scene in practice (e.g. exactly the same viewgraph) also have minor differences in the RGB spectrum that must be ignored to avoid false transition points. The threshold values are chosen empirically after evaluation of a large number of diverse lectures. A value of 10% was selected for both *RGB threshold* and *transition point threshold* for the system used

in this paper. Fig. 4 illustrates a transition point in a sequence of video frames.

Figure 4. Transition Point in video: third frame is a new transition point

A *Canonical Search* through the video to identify all transition points by comparing pairs of all successive frames in a video is illustrated in Fig. 5a. We also propose two approaches to speed-up this process; *Sequential Search* and *Binary Search*.

For both these approaches we use sampling, which means only one frame per sample interval is examined rather than all frames. For instance, if the sample interval is 2 we only use the frames 1,3,5,7… If the sample interval is 3 we processes the frames 1,4,7,10... Fig. 5b and Fig. 5c show how sampling is used. Every other frame (light-color frames) is not processed.

1) Sequential Search: In a video lecture scene transitions are relatively infrequent, so the search is optimized by skipping over blocks of identical frames. With this optimization, instead of comparing successive frames, the current frame is compared to a frame that is *Jumping Interval* ahead. If the two frames are identical, then in all likelihood, all frames in between are also identical. Hence the search marker is moved up by Jumping Interval and the frames in between are ignored, as shown in Fig. 5b. If the current frame and the frame Jumping Interval ahead are different, then the frames in between are compared sequentially starting from the current frame to identify all transition points. Essentially, when a large sequence of frames is identical, most of the comparisons are skipped.

2) Binary Search: Binary search indexing algorithm is shown in Fig. 5c. After the sampling, this procedure starts by splitting the video into two segments – from the first frame to the midpoint frame and the midpoint frame to the end frame. For each of the segments, the starting and the ending frames are compared; if they are identical, then it is assumed that the segment contains no transition points and if they are different, the segment is again subdivided into 2 segments and the procedure is applied recursively. When a segment size of 1 is reached, a transition point is identified and the search terminates.

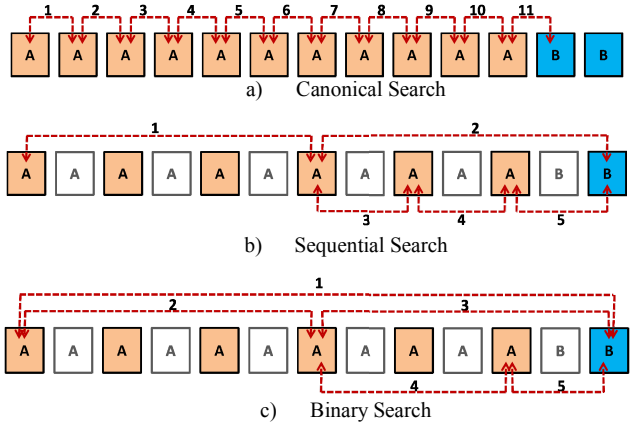

Figure 5. Indexing Algorithm steps for finding transition points in a video. a) Canonical search takes 11 steps to find the transition point, b) Sequential search requires 5 steps with the help of sampling and jumping , and c) Binary search also requires 5 steps.

B. *Detection of Index Points*

The algorithms discussed above typically find all slide transitions in a lecture video with very few false positives. However, a video may have a large number of such transition points, e.g., over 100 transition points is not unusual when an instructor is writing on a Tablet PC screen frequently. The goal is to identify a smaller number of index points that are more meaningful and easier to use. To identify the index points from the detected transition points, a filtering algorithm is applied that tries to eliminate unwanted points. The algorithm takes two parameters as input: the number of index points desired and the minimum time gap between successive index points, and then applies iterative elimination. The basic idea is to repeatedly find the closest pair of remaining transition points on the timeline and eliminate one from the pair. The algorithm ensures that, when there is a cluster of transition points on the timeline, a representative one is included as an index point. The transition points are eliminated till the number is less than the desired number of index points, and the time difference between the remaining transition points has exceeded the predefined minimum time gap. The remaining transition points are index points.

C. *Indexing Experiments and Performance Results*

For evaluation we selected 20 different videos prepared by 15 different instructors. The average video length is 86 minutes and the average number of transition points was found to be 78. These videos are selected for their diversity in templates and color styles from Computer Science, Biology, and Geoscience departments. Fig. 6 and Fig. 7 are based on experiments with this video suite.

Fig. 6 shows the average number of transition points detected on these videos. Canonical indexing found all transition points but sequential and binary missed a few transition points due to optimizations.

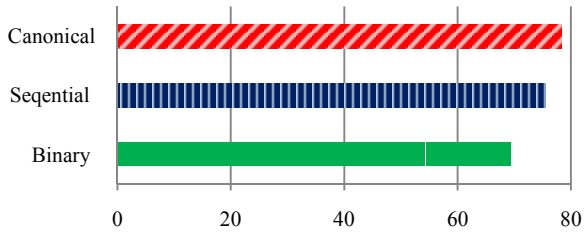

Figure 6. Average number of transition points detected with different methods in a lecture video

Fig. 7 shows the average execution times of indexing algorithms for 20 videos. We need to note that the canonical indexing algorithm execution time is about 85 minutes, which is almost same as the average video length. Sequential algorithm is 4 times faster than canonical and the binary search indexing is 8 times faster than canonical.

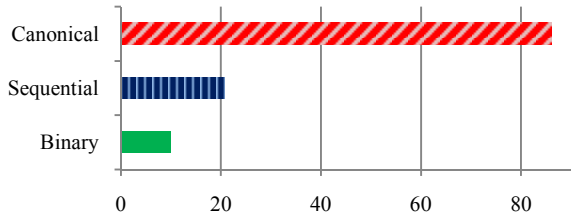

Figure 7. Average Execution Times (in minutes) for indexing algorithms

Employment of sampling, jumping interval and binary search substantially improves performance, but there is a chance that a legitimate transition point may be missed. This can happen when frames at the endpoint of a segment are identical masking transition points inside the segment, e.g., when an instructor goes back and forth between viewgraphs in a lecture. This also depends on the size of *Jumping Interval* and *Sample Interval.* A larger interval implies faster but more error-prone processing. Based on experiments with our videos suite that is mostly recorded at 4 frames/second, sampling interval is set to 4 and jumping interval is set to 8 for our deployment

IV. KEYWORD SEARCH AND OCR TOOLS

Keyword search is the process of identifying all video segments that match a keyword. The steps to support this feature are as follows. The indexer creates the video segments as well as transition point frames. Text on these frames is detected and stored in a database. ICS Video Player loads the keyword database, and when a user searches for a keyword, the player navigates to the related video segment. The framework is illustrated in Fig. 2.

This process requires that the text contained in the images corresponding to the transition points be detected. This can be accomplished by the use of Optical Character Recognition (OCR) tools. Several OCR tools are available for use, both commercial and open source. For our experiments we selected the following OCR tools: GOCR, Tessaract and MODI. Selection is based on usability, accuracy, and ease of integration.

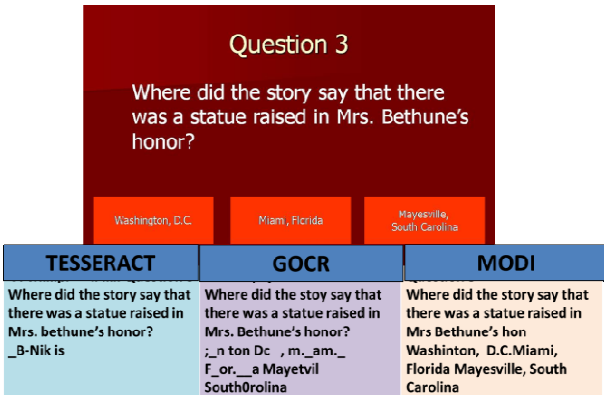

Figure 8. Detection of the same text block in a video frame with different OCR Tools; Tesseract, GOCR and MODI

In general, we found that OCR tools tend to have limited effectiveness at recognizing text in the presence of 1) certain text and background color and shade combinations, 2) text mingled with colorful shapes, and 3) small and exotic fonts. An example of text detection by different OCR engines for the same image is shown in Fig. 8. We can conclude that OCR tools are not perfect and different OCR tools have different accuracies. For example, Tessaract tool missed a lot of words in contrast to MODI, but it detected "honor" which is not detected by MODI, as shown in Fig. 8.

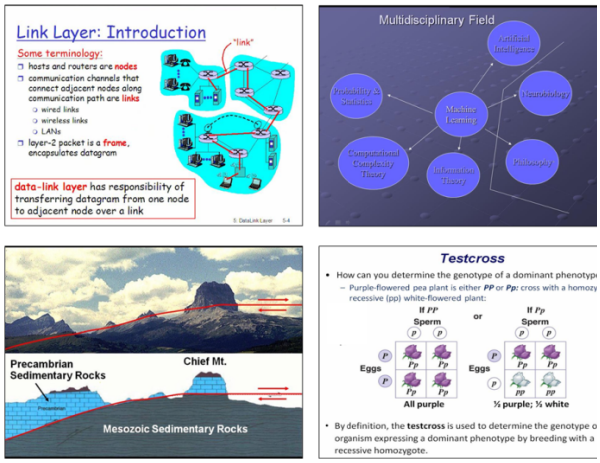

Figure 9. Example of ICS video Frames

A. Image Enhancements

Lecture video frame often include complex images, graphs, and shapes in different colors with non-uniform backgrounds, with examples shown in Fig. 9. Text detection requires a more specialized approach than is provided by off-the-shelf OCR engines, which are designed primarily for recognizing text within scanned documents in black and white format.

To improve the detection efficiency of text on ICS video images, we have used simple image processing techniques for image enhancement (IE) prior to the application of OCR

tools. IE operations employed are segmentation of text, enlargement with interpolation, and color inversion.

1) Segmentation of text: This involves defining and extracting the text regions in an image. For text segmentation, first we convert the color image to a binary black and white image by using Simple Image Statistics based thresholding. Then by using dilation operation, which allows for growth of separate objects, or joining objects, we join the characters and words. After that we employ Sobel edge detection. Lastly, we use blob extraction to extract standalone objects in the image using connected components labeling algorithm. Segmentation of text is illustrated in Fig. 10.

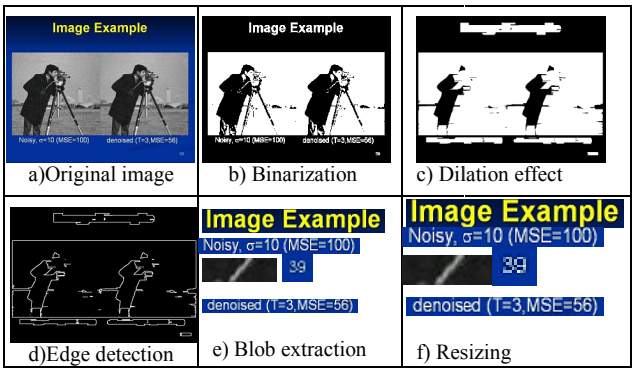

Figure 10. Image Segmentation steps

Enlargement with interpolation is implemented for the segmented blocks. By this operation, small sized text is enlarged so that some of the small size text becomes visible to OCR engines. Below are the details of segmentation:

a) Binarization: Images in ICS videos are in color; therefore we need to convert colored images to black and white images, for segmentation and morphological operations. We do so by performing image binarization. We used the SIS filter in AForge Image Library, a free opensource image processing library for C#, which performs image thresholding by calculating the threshold automatically using the simple image statistics method. Threshold is calculated as the sum of weighted pixel values divided by the sum of weight [11]. Thresholded image is shown in Fig. 10b.

b) Dilation: After binarizing the image, we use dilation, which is removing object holes of too-small size on black foreground and white background image. This effect is performed by using morphological operations to connect the characters. For erosion and dilation operations we used the structured element : [0,0,0;1,1,1;0,0,0]. Thus, dilation effect allows for growth of separate objects, or joining of objects. We use it for joining the characters and creating groups for segmentation. We choose a horizontal window so that the characters tend to merge in the right and left direction in the image, as shown in Fig. 11.

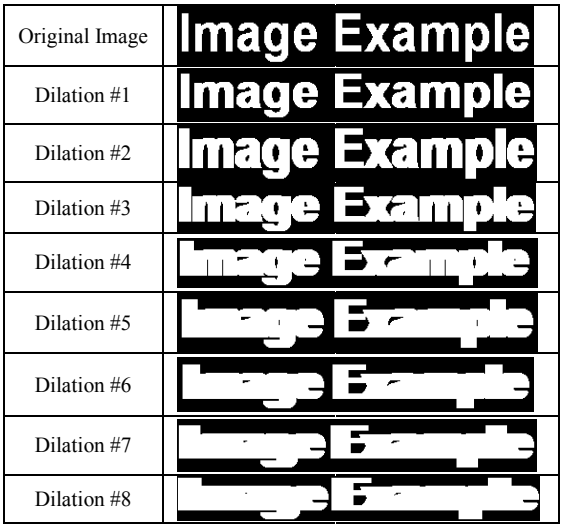

Figure 11. Dilation effect on an image: Dilation joins the small objects (characters) and fills the small holes; text is converted to square objects

Dilation removes object holes of too-small size on black foreground and white background image. Erosion is the reverse of dilation, but when we use it for a white foreground and black background image, it gives the same effect and removes object holes of too-small size. So we need to decide which one to use according to our image. If the image has a white foreground and a black background, dilation will tend to remove the foreground. This is not desirable, so we make a decision by Average Optical Density calculation:

$$AOD(I) = \left(\frac{1}{N^2}\right) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j) \qquad (1)$$

We calculate AOD of a binary image which has 0(white) and 1(black) value. This puts the AOD value between 0 and 1. We found that when AOD >0.15 for ICS video frames, it refers to a black foreground and white background image using erosion. In the other case, when AOD <=0.15, it refers to an image with white foreground and black background and we choose to use dilation.

After deciding which morphological operation to use for segmentation, we need to determine the number of iterations for the process. Through trial and error, we found that 8 iterations of dilation/erosion process are reasonable for joining characters in most ICS video images.

c) Edge Detection: We grouped every small object, such as characters and small items, to a single object in the dilation process. But there can still be incomplete borders after dilation, which may lead to wrong segmentation. Edge detection will help connect the borders of objects. We choose Sobel operator to detect the edges in AForge Image Library. Detection of the edges will unify the objects and provide more accurate detection of groups. Searching for objects' edges by applying Sobel operator, each pixel in the resulting image is calculated as an approximate absolute gradient magnitude for the corresponding pixel of the source image:

$$|G| = |Gx| + |Gy] \quad (2)$$

where Gx and Gy are calculated utilizing Sobel convolution kernels:

Gx			Gy		
-1	0	+1	+1	+2	+1
-2	0	+2	0	0	0
-1	0	+1	-1	-2	-1

Using the above kernel, the approximated magnitude for pixel **x** is calculated using the next equation: [12]

P1	P2	P3
P8	x	P4
P7	P6	P5

$$|G| = |P_1+2P_2+P_3-P_7-2P_6-P_5| + |P_3+2P_4+P_5-P_1-2P_8-P_7| \quad (3)$$

Edge detection effect is shown in Fig. 10d.

d) Blob Extraction: In this next step we count and extract standalone objects in the image using connected components labeling algorithm. It is an algorithmic application of the graph theory, where subsets of connected components are uniquely labeled based on a given heuristic of the AForge Image Library. The filter performs labeling of objects in the source image. It colors each separate object with a different color. The image processing filter treats all non-black pixels as the objects' pixels and all black pixels as the background and the AForge Library blob extractor extracts blobs in the input images (which in our case are thresholded and dilated images). However, we need the original image as an input for OCR, thus we use blob extraction to detect the place of the objects in dilated image and use this information to extract the blobs from the original image. A blob extraction example is shown in Fig. 10d. In the extracted blob, one would expect more blobs; however, they were filtered using the following two criteria. If a blob contains other blobs, or if the blob-width / blob-height <1.5, we do not extract it. (The text we want to detect is at least two characters long; since we dilated text to the right and left, in all cases the width will be more than the height.). In Fig. 10d, the man's body is not extracted because of the threshold on height to width ratio. Also, very small size blobs are not included in the blobs.

e) Resizing: We separated the text from other complex background by segmentation in the previous section. What if the segmented text font size is too small to be detected correctly? We know that the best font size for OCR is 10 to 12 points and use of smaller font sizes leads to a poor quality OCR. Font sizes greater than 72 are not considered as text and thus should be avoided. Usually dark, bold letters on a light background, and vice versa, yield good results. The textual content should be ideally placed with good spacing between words and sentences [1]. We need large enough fonts to be able to detect the text. Increasing the size of a font can be achieved easily by increasing the size of the image. For instance, if we resize the image to 1.5x it will make everything inside the image including text 1.5x bigger. We have plenty of small characters (mostly occurring in the explanation for the images or graphs), so before the image processing we increase the size of the images by a factor of 1.5 as a default. Resizing is done by bilinear image interpolation method; it works in two directions, and tries achieving the best approximation of a pixel's color and intensity based on the values at surrounding pixels. Considering the closest 2x2 neighborhood of a known pixel values surrounding the unknown pixel, the weighted average of these 4 pixels is taken to arrive at the final interpolated value.

2) Inversion: Segmentation procedure often leads to new text being recognized but can also prevent the recognition of other text. Hence, color inversion is done separately by altering the RGB values of images aimed at increasing the contrast between the text and the background.

Figure 12. Inversion example: Original image and color inverted images

In image file formats such as BMP, JPEG, TGA, or TIFF, that are common in 24-bit RGB representations, color value for each pixel is encoded in 24 bits per pixel fashion where three 8-bit unsigned integers (0 through 255) represent the intensities of red, green, and blue. Inverting colors is basically altering the RGB values. When we invert an image in a classical way, we take the inverse RGB values. For example, the inverse of the color (1,0,100) is (255-1,255-0,255-100) = (254,255,155). In our approach, we expand this technique from 1 to 7 inversions shown in Fig. 12. R' is referring to 255-R value. OCR engines give different results for inverted images. In this example the 3rd inversion seems better than the first one. But this will differ in different images that have different color combinations.

Instead of using only the original image, additionally using inverted images improved the OCR results. However, since we do not know which inversion will be the best, OCR engines are applied to the original images as well as enhanced images and the union of the results is taken.

B. OCR Performance Results

To test the impact of OCR Tools and image transformations, we evaluated 1387 different images that are created by the indexer from 20 diverse videos discussed in the previous section. Images from these videos contain 20007 unique words, 27201 total words (of more than 1 character length) for a total of 144613 characters.

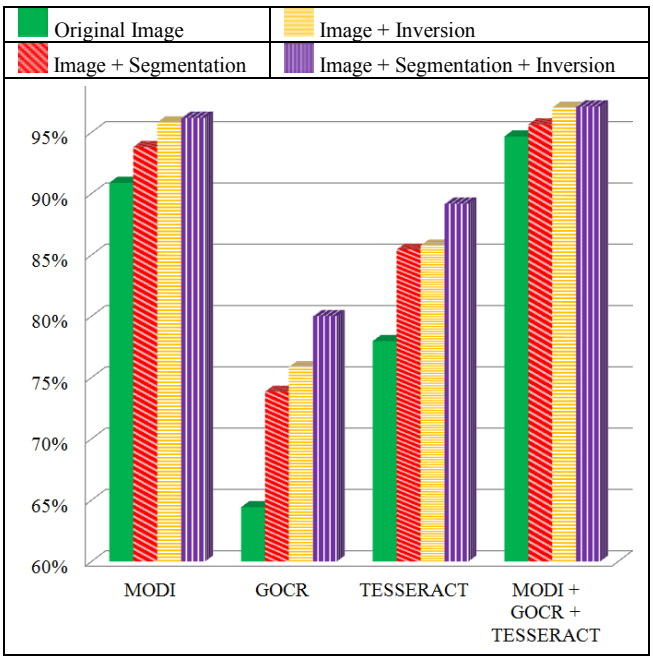

Figure 13. Search accuracy rate of OCR tools: Individual and combination of three OCR engines's results

Experimental results, presented in Fig. 13, show that the search accuracy of three distinct OCR engines -Tesseract, GOCR and MODI, improved as a result of image transformations. *Search accuracy* is defined as the detected number of unique words divided by the total number of unique words. The words have to be at least 2 character in length and only alphabetic characters and numbers are counted as part of the words. Segmentation and inversion both increased the accuracy but inversion is slightly more effective than segmentation. The maximum accuracy obtained by applying all OCR engines with segmentation and inversion is 97.1%. Alternately stated, the miss rate was 8.9% for the best single OCR engine, 5.2% for all OCR tools combined, and 2.9% for all OCR engines combined with image enhancement.

Segmentation and inversion provided this accuracy improvement but it increased the processing time, partly because OCR engines have to be applied on original and enhanced images. Inversion is a simple process but segmentation includes several steps and several iterations. Therefore transformation time for segmentation is more than the transformation time of inversion. On the other hand, inversion creates 7 more full size copies of the image and segmentation has only some part of the image as an input for OCR tool. Hence OCR time for inversion is more than for segmentation. The execution time overhead of these transformations is shown in Fig. 14. The overall processing time remains modest for a typical video. The processing time is dependent on the number of transition points, but it is typically in the range of 2-3 minutes on an average desktop. Image transformations also increased the number of false positives of OCR engines, i.e., some words were detected that were not present in the video. This typically happens when an OCR engine misses a character in a word leading to false identification of a different word. Since the main aim of text recognition is to let the user find words of interest, the "extra words" resulting from false positives do not diminish the functionality in a meaningful way.

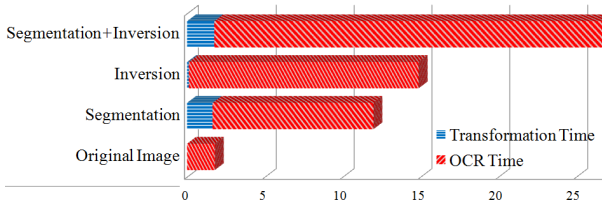

Figure 14. Execution times (in minutes) of OCR tools and image transformations

V. Discussion and Future Work

The underlying significance of this direction of research is that video is a dynamic and versatile resource, rather than a poor substitute for a classroom. The challenge of quick access to the content of a video is addressed to a large extent with indexing and search. Extending the concept of "Keyword Search" to "Semantic Search" seems the logical next step to increase the quality and speed of access to the content of a lecture video. An interface can guide the user with text suggestions with topics and words predefined in dictionaries. These dictionaries may be automatically created from collections of videos, constructed from text content of powerpoint or pdf files that a video lecture is based on, or other sources such as course textbooks.

Enabling instructors to customize videos with easy interfaces for selection of indexes and other basic editing and manipulation is another way to improve the accuracy and value of indexing and search. One touch operation from the start of recording a lecture to it being posted online for students is a desirable practical goal. With mobile devices becoming increasingly common, understanding the modifications needed to make use on mobile devices attractive is another important direction of future work.

VI. Conclusion

This paper reports on technologies developed for Indexed and Searchable videos for STEM coursework. We demonstrated that automated indexing and search frameworks are effective and efficient. The processing of a 1 hour video is typically less than 15 minutes on an ordinary desktop and expected to be significantly lower on server class hardware.

The main focus of this paper is on image transformations to enhance the accuracy of OCR tools on video frames. Inversion and segmentation had a substantial

impact on improving text recognition capacity of each of the three OCR engines employed – MODI, Tesseract and GOCR. Overall, the fraction of words that are not recognized by any engine is reduced from 8% to 3% with image enhancements.

The ICS video framework has been used by dozens of courses in the past few semesters at the University of Houston campuses. The student response has been enthusiastic both in terms of the value of lecture videos and the importance of indexing and search features.

ACKNOWLEDGMENT

We would like to acknowledge the contributions of several members of the ICS Videos group over the years. Joanna Li and Gautam Bhatt developed the indexer modules used in this work. Anshul Verma and Varun Varghese developed the ICS video player. Salvador Baez, Pradeep Krishnan, and Andrea Arias helped provide support for the streaming framework used by the students to access the lecture videos. Dr. Lecia Barker, Dr. Zhigang Deng, and Dr. Olin Johnson provided leadership in several aspects of this project.

Partial support for this work was provided by the National Science Foundation's Division of Undergraduate Education under Course, Curriculum, and Laboratory Improvement (CCLI) program with Award No. DUE-0817558. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Best OCR Font Size, Computer vision, http://www.cvisiontech.com/pdf/pdf-OCR/best-font-and-size-for-OCR.html?lang=eng.

[2] D. Pang, S. Halawa, N.-M. Cheung, and B. Girod, "Mobile Interactive Region-of-Interest Video Streaming with Crowd-Driven Prefetching," International ACM Workshop on Interactive Multimedia on Mobile and Portable Devices, ACM Multimedia (MM'11), Scottsdale, Arizona, USA, Nov. 2011.

[3] G. Ahanger and T.D.C. Little, "A Survey of Technologies for Parsing and Indexing Digital Video", Journal of Visual Communication and Image Representation

[4] Image Interpolation, http://www.cambridgeincolour.com/tutorials/image-interpolation.htm.

[5] J.Subhlok, O.Johnson, V. Subramaniam, R.Vilalta, & C.Yun.. "Tablet PC video based hybrid coursework in computer science." Proceedings of the 38th SIGCSE technical symposium on Computer science education - SIGCSE '07 (p. 74). Presented at the Proceedings of the 38th SIGCSE technical symposium, Covington, Kentucky, USA, 2007

[6] J.Yuan, L. Duan, Q. Tian, and C Xu."Fast and robust short video clip search using an index structure". In Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval (MIR '04). ACM, New York, NY, USA, 61-68.

[7] Microsoft Project Tuva, http://research.microsoft.com/apps/tools/tuva/

[8] M. Bianchi, "Automatic Video Production of Lectures Using an Intelligent and Aware Environment" ACM Press Pages: 117 - 123, 2004

[9] M. Merler and, J.R. Kender, "Semantic keyword extraction via adaptive text binarization of unstructured unsourced video", IEEE International Conference on Image Processing, Cairo, Egypt, Nov. 2009 .

[10] R. Lienhart and W.Effelsberg. "Automatic text segmentation and Text recognition for video indexing". ACM/Springer multimedia Systems, Vol. 8. pp.69-81, Jan. 2000.

[11] Sis threasholding, http://www.aforgenet.com/framework/docs/html/39e861e0-e4bb-7e09-c067-6cbda5d646f3.htm.

[12] Sobel Edge Detector, http://www.aforgenet.com/framework/docs/

[13] T.Tuna, J.Subhlok, S.Shah, "Development and Evaluation of Indexed Captioned Searchable Videos for STEM coursework", Proceedings of the SIGCSE technical symposium on Computer science education - SIGCSE '12, Raleigh, North Carolina, Mar. 2012, un published

[14] W. Ma, Y. Lee, David H. C. Du, and M. P.McCahill, Video-based hypermedia for education-on-demand, MULTIMEDIA '96: Proceedings of the fourth ACM international conference on multimedia (New York, NY, USA), ACM, 1996, pp. 449-450.