Automatic Text Detection and Tracking in Digital Video

Huiping Li, David Doermann and Omid Kia

Abstract

Text which appears in a scene or is graphically added to video can provide an important supplemental source of index information as well as clues for decoding the video's structure and for classification. In this paper we present algorithms for detecting and tracking text in digital video. Our system implements a scale-space feature extractor that feeds an artificial neural processor to detect text blocks. Our text tracking scheme consists of two modules: an SSD (Sum of Squared Difference)-based module to find the initial position and a contour-based module to refine the position. Experiments conducted with a variety of video sources show that our scheme can detect and track text robustly.

Keywords

Text Detection, Text Tracking, Video Indexing, Digital Libraries, Neural Network

I. Introduction

The continued proliferation of large amounts of digital video has increased demand for true content based indexing and retrieval systems. Traditionally, content has been indexed primarily by manual annotation [1], closed caption [2] or transcribed audio [3], but some work has also been done on the content analysis of the video itself. One area where significant progress is being made is in the detection and recognition of text. Text which either appears in a scene or is graphically added to video provides an important supplemental source of index information. For example, sports scores, product names, scene locations, speaker names, movie credits, program introductions and special announcements often appear in the image text and supplement

H. Li and D.S. Doermann are with the Language and Media Processing Laboratory, Center for Automation Research, University of Maryland, College Park. Email: {huiping,doermann}@cfar.umd.edu; O. Kia is with the National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899

or summarize the visual content, but may not be present in the transcript. Searches can easily be refined if access to this textual content is available.

At a high level, text in digital video can be divided into two classes, *scene* text and *graphic* text. Scene text appears within the scene and is captured by the camera. Examples of scene text include street signs, billboards, text on trucks and writing on shirts. Graphic text, on the other hand, is text that is mechanically added to video frames to supplement the visual and audio content. Since it is purposefully added it is often more structured and closely related to the subject than scene text. In some domains such as sports, however, scene text can be used to uniquely identify objects (participants in the clip). Most related previous work has focused on the extraction of graphic text [4], [5], [6]. Although scene text is often difficult to detect and extract due to its virtually unlimited range of poses, sizes, shapes and colors, it is important in applications such as navigation, surveillance, video classification, or analysis of sporting events.

Text often spans tens or even hundreds of frames in digital video. Exploiting the temporal coherence of text by tracking it is useful not only for reducing processing time by not applying all steps (detection, extraction, enhancement and OCR) to every frame, but also for maintaining integrity by detecting when new information appears and as a means of enhancement. In the literature Lienhart [6] and Shim [7] make use of multiple frame information to reduce incorrectly identified text regions, but neither addresses the problem of tracking text in the video. In Lienhart's system, the text in each frame is extracted and saved after recognition, resulting in duplicates of the same text string in the database.

Our goal is to detect both graphic and scene text and track it robustly. A simplistic solution would be to perform the text detection frame by frame and then match the corresponding text blocks between consecutive frames. Considering that video is often digitized 30 frames per second, the text detection on every frame in the absence of context would be prohibitive. We can, however, make use of the fact that text remains in the scene for many consecutive frames to reduce the complexity by performing the text detection process periodically and focusing on the tracking process (Figure 1). We use a hybrid wavelet/neural network segmenter to detect text regions and an SSD-based module is used to track the detected text. Text contour information is used to refine the tracking results.

II. Text Detection in Video Frames

In the literature text detection methods are typically either connected component (CC) based [4], [8] or texture based [9], [10]. The CC based methods can extract text efficiently, but have difficulties when text touches itself or other graphical objects, which may happen in digital video since text is often embedded in complex backgrounds. Jain presents a text extraction system that treats text as a distinctive texture and uses unsupervised clustering to classify each pixel as text or non-text [10]. However, in video frames natural scenes like the leaves of a tree or grass in a field have textures similar to text, and in the feature space, text and nontext often overlap.

Our methodology uses a small window (typically 16×16) to scan the image and classify each window as text or non-text using a neural network. We feel that using supervised learning to classify text and nontext will be more effective than the unsupervised clustering techniques used in [9], [10]. An artificial neural network is a natural choice as a classifier because of its ability to learn. Theoretically, a three-layer neural network can approximate any nonlinear function after training. The success of neural networks in related problems [11], [12], [13] provides us with further motivation to rely on a neural network as a classifier to identify text regions. To facilitate the detection of various text sizes, we use a pyramid of images generated from the original image by halving the resolution at each level. The extracted text regions are hypothesized at each level and then extrapolated to the original scale. A diagram of our text detection scheme is shown in Figure 2.

A. Feature Extraction and Selection

Analysis of scale space provides a method of identifying the spatial frequency content in local regions within the image. We use wavelets to decompose the image because they provide successive approximations to the image by downsampling and have the ability to detect edges during the high-pass filtering. The low-pass filter creates successive approximations to the image while the detailed signal provides a feature-rich representation of textual content. This is easily seen in the image decomposition shown in Figure 3 where 3a is the original image and 3b is its first-level wavelet decomposition. Note that the text region shows high activity in the three high-frequency subbands (*HL, LH, HH*). As a result of their local nature, only wavelets which are located

on or near the edge yield large wavelet coefficients, making text regions detectable in the high frequency subbands.

All the features are computed on the decomposed subband images. We use the mean and the second- and third-order central moments as features. For an $N \times N$ subblock I we calculate the mean (m) and the second-order (μ_2) and third-order (μ_3) central moments as

$$\begin{aligned} M(I) &= \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j) \\ \mu_2(I) &= \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (I(i,j) - M(I))^2 \\ \mu_3(I) &= \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (I(i,j) - M(I))^3 \end{aligned} \tag{1}$$

The features described in (1) are calculated only in the subblocks of the first three levels since only one pixel is left in the fourth level. There are 36 features corresponding to each 16×16 window. We conduct feature saliency analysis to reduce the feature set since a larger feature set requires more training samples and time. Specifically, 1000 text and nontext blocks were collected and split into training and testing sets. Each feature was trained on the training set and then applied to classify the testing data. We use the Bayes error rate P_e to analyze the saliency of features since it determines whether or not the feature will yield adequate separation between the classes. In most practical cases, the Bayes error rate is estimated using a finite set of labeled samples from the various classes. Finally, eight features which have the lowest Bayes error rates are selected from the original 36 features and are fed to the neural network. Details can be found in [14].

B. Artificial Neural Network Classifier

After selecting the features, we train the neural network. The neural network consists of 8 input, 12 hidden and 1 output nodes. Although it is easy to get representative samples of text, it is more difficult to get representative samples of non-text since non-text spans a vast space. To handle this problem we use a *bootstrap* method recommended by Sung and Poggio [15] to train the neural network. The idea is that the training samples are collected in part during training rather than before training. The process is as follows:

1. Create an initial set of training samples which includes a complete set of text samples and a partial set of non-text samples.

2. Train the network on these samples.
3. Run the system on a video frame which contains no text.
4. Add image blocks which the network incorrectly classifies as text to the non-text sample set.
5. Repeat Steps 2 - 4 until the process converges.

C. Text Detection

After training the neural network, we use a 16×16 window to scan the video frame to classify each window as text or nontext. We can view the output of the neural network as a mapping function which maps each feature set into a real value between 0 and 1. A threshold of 0.5 is used to indicate whether the window contains text or not. The larger the window step, the smaller the number of windows to be processed but the less refined the result. Considering the trade-off between the precision and speed, we shift the window 4 pixels at each step.

If a single window is classified as text, all the pixels in this window are labeled as text. Those pixels which are not covered by any text window are labeled as nontext. The result of classification is a label map of the original image. Figure 4(a) is a video frame and Figure 4(b) is its corresponding label map. Figure 4(c) shows the extracted text regions. We can see that all of the text is labeled correctly, but there are some small isolated areas which are incorrectly labeled as text. We use size constraints between blocks to filter out these small isolated areas. The bounding box of the text area is generated by a connected component analysis of the text windows (Figure 4(d)).

As depicted in Figure 2 we use a three-level pyramid approach to detect text with a wide range of font sizes. After the detection step is applied at each level, the bounding boxes are mapped back to the original input image. If the bounding boxes at one level do not overlap with the boxes at other levels, we simply merge them into the original image. In some cases, however, text strings or parts of text strings appear at multiple levels so the boxes detected at different levels overlap. In this case we merge them by forming a bounding box which contains all of them.

III. Text Tracking

Once text is detected, the tracking process is started. Text motion in digital video is one of three types: static; simple linear motion (for example, scrolling movie credits); or complex nonlinear motion (for example, zooming in/out, rotation, or free movement of scene text). In the first two cases, a simple image matching technique can track the text well. In the third case, a more robust technique is required. Our goal is to design a scheme to efficiently track text with both simple and complex motions.

An arbitrary motion can encompass a wide range of possible motion transformations. Robust, reliable visual tracking of an object in a complex environment often requires the integration of several different visual modules, each using a different criterion and each employing different assumptions. The modules will be selected so that they complement each other. We can treat a text line as a closed set S in the plane, where the bounding box of the text line corresponds to the boundary of S. The boundary B and the interior I of S provide complementary information: B determines the shape (and size) of S and I determines its content (intensity or texture). We believe that the integration of tracking modules based on image content and shape will make the tracking processing more powerful and stable. We use a general SSD module to measure the similarity of image content (corresponding to the interior I) and then make use of information about the text contour to refine the position of the text block (corresponding to the boundary B). In the rest of this section we give the details of the implementation.

A. SSD Based Module for Text Tracking

As either the camera or the text moves, the pattern of image intensities changes in complex ways. A text block in the current frame can be obtained by moving every point of the text block in the reference frame by an amount $\delta = (\xi, \eta)$, represented by an affine transform:

$$\delta = D\vec{x} + \vec{d} \tag{2}$$

where D is a 2×2 deformation matrix and $\vec{d}$ is a 2×1 displacement vector.

If we choose minimization of the SSD as a matching metric, then the basic tracking algorithm can be described as follows: Given a reference text block I, and a search region W in image J, determine the six

parameters of the deformation matrix D and displacement vector $\vec{d}$, which minimize

$$\epsilon = \int\int_W [J(D\vec{x} + \vec{d}) - I(\vec{x})]^2 d\vec{x} \tag{3}$$

The solution to Equation (3) can be very complex. The quality of the estimate depends on factors such as the texture of the image and the amount of camera motion between frames. Under the assumption of small inter-frame motion, we can simplify the equation by setting the deformation matrix D to the *identity matrix*. Equation (3) then simplifies to

$$\epsilon = \int\int_W [J(\vec{x} + \vec{d}) - I(\vec{x})]^2 d\vec{x} \tag{4}$$

The model described by Equation (4) is a *pure translational* model. In this model only two translational parameters need to be determined. The search space W is within some range of the predicted location. We use a simple prediction technique to locate W more accurately. Suppose $\vec{x}_n, \vec{x}_{n-1}$ are the text positions in the current and previous frames respectively, then the text position in next frame $\vec{x}_{n+1}$ can be expressed as

$$\vec{x}_{n+1} = \vec{x}_n + (\vec{x}_n - \vec{x}_{n-1}) \tag{5}$$

The pure translational model tracks well in the cases of simple motion or static text. Figure 5a shows an example of tracking movie credits. Although the text line being tracked moves from a clean background to a complex background (in the middle of the frame), the text line is correctly tracked. Figure 5b shows the graph of the SSD.

Evidently, the pure translational model is not adequate to handle scale, rotation and perspective distortions [16], [17]. Over a long sequence the error will accumulate to the point that the tracker loses the target. Figure 6 shows the tracking result when we track a text line in a conference video sequence. Although the SSD is small enough in consecutive frames (Figure 6b), the tracker gradually loses the target after several frames.

B. Contour-Based Text Stabilization

When text undergoes complex motion, a pure translational model does not track it well, so we use the text contour to stabilize the tracking process. The stabilization process can be implemented efficiently in the

following way:

1. For matched text position $s = (x1, y1, x2, y2)$, generate a slightly larger text block $s_0 = (x1 - \delta, y1 - \delta, x2 + \delta, y2 + \delta)$. Assume the real text position is included within s_0.
2. Generate the edge map of s_0 by calculating the Canny edges. We use the edge map instead of thresholding the image to avoid the difficulties of identifying normal text and inverse text.
3. Apply a horizontal smearing process so the edge map can be grouped to form a text block.
4. Extract connected components and their positions $s' = (x1', y1', x2', y2')$ to represent the refined text position.

Figure 7a shows the initial matching position and Figure 7e shows the refined position.

The scheme described above works well when text is moving on a relatively clean background. However, when text moves over a complex background, it often touches other graphical objects. Since our contour extraction process operates on a slightly enlarged box (as is necessary since we must leave extra space for contour extraction), the text contour will become larger than its actual size. In order to deal with this case, we check the SSD between two consecutive frames. When text moves in a complex background, the SSD between two consecutive frames is larger than the SSD in a clean background since the pixel values in the background change considerably (Figure 5b). In this case we stop the stabilization process and depend only on the pure translational model to track the text. Once the text moves out of the complex background (SSD becomes smaller again), the stabilization process is started again.

C. Using Multi-Resolution Matching to Reduce Complexity

Our SSD-based module is region-based and its computational cost can be considerable when we track a large text line. To reduce the processing requirements, we perform matching from coarse to fine in a hierarchical fashion on a Gaussian image pyramid. For a frame I_t of size $w \times h$, a Gaussian pyramid G_t^l is formed by combining several reduced-resolution Gaussian images of frame I_t, where t is the frame number and $l = \{0, 1, 2, ..., N\}$ represents the level in the pyramid.

The matching is conducted starting at the coarsest resolution (level N). Each level contributes to determining

the position of the match on the next level. Figure 9 illustrates the process. The search for the minimum SSD starts on level N over a window size of $S = (2s + 1) \times (2s + 1)$. Suppose the matching point found is $P^N(x, y)$. Then at level $N - 1$, the search for the minimum SSD will be conducted around pixel $P^{N-1}(2x, 2y)$ over the same window size $S = (2s+1) \times (2s+1)$. This process continues until the finest resolution (level 0) is reached. At each level, the maximum displacement supported by the search is s, which is much smaller than what is required in a one-step search, but the displacement is doubled after each level. The total displacement reached at the finest level is $s \times 2^N$, with the level of the pyramid depending on the size of the text block. If the text line is small enough, no pyramid will be formed and matching will be conducted directly at the original image scale.

D. Motion Status Analysis

Although the goal of our system is to track text in the general case, understanding text motion can help facilitate the tracking process. If we find, for example, that the text lines are scrolling up (or down), we can deduce that new text lines will appear at the bottom (or top) of the video frame. We can then restrict the costly text detection process to a relatively small region.

When text crosses the frame horizontally, analysis of motion status is necessary to track text correctly. This often happens, for example, in newscasts when announcements cross the bottom of the TV screen. We have no way to track the whole text line in this case since the virtual text line is much bigger than the frame. In this case, we separate the text line into words and track the words. If we find that all the words in a text line are moving horizontally in the same direction and new words continue to appear on the same line, we can deduce that the text is crossing the screen, and we only need to monitor the areas where new text words may appear.

IV. Implementation and Experiments

We have implemented three tools based on the methods described above: A text detection tool (*TextDetect*), a text tracking tool (*TextTracker*), and a text detection and tracking tool (*TextDT*). All the programs are written in C and run on a Sun workstation under *Solaris 2.5*. We evaluated the performance of these tools in

the following experiments.

A. Text Detection

We collected two sets of data for experiments. The first set of data consists of 500 key frames selected automatically from a collection of 22 *MPEG* video clips using a transition key frame detection and selection algorithm [18]. The second data set includes 75 frames containing text selected from cable TV and captured using a *miro VIDEO DC30 plus* digitizing board. The samples include both scene and graphic text with multiple font sizes and styles.

The detection procedure requires about 1 second on a Sun Ultra 1 Workstation to process a 352×240 frame with unoptimized code. Classification (including feature extraction) takes 0.5 seconds; postprocessing and image input and output take another 0.5 seconds.

A.1 Evaluation of Text Frame Detection

Text frame detection checks if a video frame contains text or not and is useful in video browsing applications. We output 1 if a frame contains text and 0 if the frame does not contain text. In 500 video frames, 151 of them contain text and the remaining 349 frames do not contain text. The confusion matrix for text frame detection is shown in Table I.

We use two metrics (*precision* and *recall*) commonly used in information retrieval (IR) to summerize the text frame detection results:

$$\begin{aligned} precision &= \frac{\#\text{ of correctly detected text frames}}{\#\text{ of detected text frames}} \\ recall &= \frac{\#\text{ of correctly detected text frames}}{\#\text{ of text frames}} \end{aligned} \tag{6}$$

Therefore, the precision is $\frac{133}{133+81} = 62\%$ and the recall is $\frac{133}{151} = 88\%$. The precision is relatively low in part because the number of non-text frames is more than that of text frames in our dataset.

A.2 Evaluation of Text Block Detection

The second data set is used to evaluate the detection of text blocks [1]. A text block may contain one or more text lines which are close to each other. There are a total of 153 text blocks in the 75 frames. 142 of them were correctly detected by our algorithm (Figure 8) and 11 of them were missed. On the other hand, 14 non-text blocks are misclassified as text blocks. The precision of text block detection is $\frac{142}{142+14} = 91\%$ and the recall is $\frac{142}{153} = 92.8\%$. Errors occur primarily because of low resolution or small text block size. Further training, domain-specific training, or attempting OCR will overcome these problems.

B. Text Tracking

We collected ten video sequences for experiments on text tracking from a wide variety of video sources, including movie credits, TV programs, football games and news and conference videos. A description of the video sequences motion is given in in Table II.

We conducted our experiments in two steps. First, we studied the performance of text tracking by manually specifying the initial position of the text block, to filter out the possible effects of text detection.

Unfortunately, quantitative evaluation of tracking accuracy is not easy because of the lack of ground truth data. We output the tracking results in the form of MPEG video which can be viewed at website *http://documents.cfar.umd.edu/LAMP/Media/Projects/TextTrack/*. Some of the tracking results are shown in Figure 10.

Our experiments show that the tracker works well when the text is in simple motion or when the motion is complex but the background is clean (Figure 10c). When the text moves arbitrarily on a complex background, the tracker may be confused in some frames but will adjust its position once the text moves to a relatively clean background. In Figure 10a, the initial position specified touches another object (the edge of the map). When the text size increases, the tracking position deviates for some frames but adjusts when the text grows large enough to be separated from the map. Another example tracks the number on an athlete's jersey in a football game. The task is complicated by the athlete's running, jumping, and rotating, as well as by camera

[1] The block level evaluation is not conducted in the first data set since the ground truth is not available in the block level for the first data set.

motion (Figure 10b). The tracker loses part of the target in some frames but then adjusts to the correct position.

The average tracking time for one text block is about 0.2 seconds per frame. If the text line is large enough, we can use multi-resolution matching to reduce the time. By tracking text we can get text blocks as well as temporal correspondences of the blocks. If we perform detection frame by frame, extra time is required to find the correspondences of blocks between consecutive frames.

In the second part of our experiments we combined the text detection and the tracking modules. As shown in Figure 1, the text detection process is performed every five frames to detect new text lines entering the frame. Once text is detected, the tracking process is applied to find the temporal correspondence in consecutive frames. Figure 11 shows a tracking result for the movie *Star Wars*. There are 2600 frames in the sequence, which includes static, zooming, and scrolling text. Figure 12 shows tracking results for a transverse text line. We detect it as horizontal scrolling by analyzing the motion status and we thus divide the line into words and track them.

The processing time changes considerably with the number of text lines per frame. Tracking movie credits takes more time than other video types since there are more text lines per frame in movie credits. For example, for the movie *Star Wars*, it takes about 1 second to track one frame (the average number of text lines in a frame is 5), while it takes only 0.17 seconds to track text in a football game (only one text line is moving in all of the frames). But as we indicated above, we can detect the text as well as the temporal correspondences of the text blocks.

There are several limitations to our system. First, text tracking is started only when text is detected. If the text detection module fails, the system will miss the text. Second, our tracker uses SSD-based image matching to approximate the position and then uses the text contour to refine the position. Therefore, the system can only be used to track text. In addition, since we use speed prediction to predict the position of the text, the text's acceleration is limited. The tracker has difficulties when text moves too abruptly or keeps moving on a complex background. This happens especially in sports video. For example, when tracking the name of an athlete on a jersey, the text may occlude quickly because of the athlete's jumping and rotating.

V. Conclusion

We have presented a system for detecting and tracking text in digital video automatically. A hybrid wavelet/neural network based method is used to detect text regions. The tracking module uses SSD-based image matching to find an initial position, followed by contour-based stabilization to refine the matched position. The system can detect graphical text and scene text with different font sizes and can track text that undergoes complex motions. Our next focus will be on making use of detected and tracked text to build a text-based video indexing and retrieval system.

References

[1] M. Davis, "Media streams: Representing video for retrieval and repurposing," in *Proc. ACM Multimedia 94*, 1994, pp. 478–479.

[2] W. Li, S. Gauch, J. Gauch, and K. M. Pua, "VISION: A digital video library," in *DL'96: Proceedings of the 1st ACM International Conference on Digital Libraries*, 1996, Multimedia Digital Libraries, pp. 19–27.

[3] J. Hernando, "Voice signal processing and representation techniques for speech recognition in noisy environments," *Signal Processing*, vol. 36, pp. 393, 1994.

[4] A.K. Jain and B. Yu, "Automatic text location in images and video frames," in *Proceedings of ICPR*, 1998, pp. 1497–1499.

[5] Hae-Kwang Kim, "Efficient automatic text location method and content-based indexing and structuring of video database," *Journal of Visual Communication and Image Representation*, vol. 7, pp. 336–344, 1996.

[6] R. Lienhart and F. Stuber, "Automatic text recognition in digital videos," in *Proceedings of ACM Multimedia*, 1996, pp. 11–20.

[7] J. Shim, C. Dorai, and R. Bolle, "Automatic text extraction from video for content-based annotation and retrieval," in *Proceedings of ICPR*, 1998, pp. 618–620.

[8] J. Zhou and D. Lopresti, "Extracting text from WWW images," in *Proceedings of ICDAR*, 1997, pp. 248–252.

[9] V. Wu, R. Manmatha, and E.M. Riseman, "Finding text in images," in *DL'97: Proceedings of the 2nd ACM International Conference on Digital Libraries*, 1997, Images and Multimedia, pp. 3–12.

[10] A.K. Jain and S. Bhattacharjee, "Text segmentation using Gabor filters for automatic document processing," *Machine Vision and Applications*, vol. 5, pp. 169 – 184, 1992.

[11] K. Etemad, D. S. Doermann, and R. Chellappa, "Multiscale document page segmentation using soft decision integration," *IEEE Trans. PAMI*, vol. 19, pp. 92–96, 1997.

[12] R. Chellappa, B.S. Manjunath, and T. Simchony, "Texture segmentation with neural networks," in *Neural Networks in Signal Processing*, pp. 37 – 61. Prentice Hall, 1992.

[13] S.K. Rogers, J.M. Colombi, C.E. Martin, and J.C. Gainey, "Neural networks for automatic target recognition," *Neural Networks*, vol. 8, pp. 1153–1184, 1995.

[14] H. Li, D.S. Doermann, and O. Kia, "Automatic text extraction and tracking in digital video," Tech. Rep., University of Maryland, College Park, 1998, LAMP-TR-028.

[15] K. Sung and T. Poggio, "Example-based learning for view-based human face detection," Tech. Rep., MIT, A.I. Memo 1521, CBCL Paper 112, 1994.

[16] G.D. Hager and P.N. Belhumeur, "Efficient region tracking with parametric models of geometry and illumination," *IEEE Trans. PAMI*, vol. 20, pp. 1025–1039, 1998.

[17] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of CVPR*, 1994, pp. 593–600.

[18] V. Kobla, D.S. Doermann, and K.I. Lin, "Archiving, indexing, and retrieval of video in the compressed domain," in *Proc. of the SPIE Conference on Multimedia Storage and Archiving Systems*, 1996, vol. 2916, pp. 78–89.

List of Tables

List of Figures

	text	nontext
text (151)	133 (88%)	18(12%)
nontext (349)	81 (23%)	268 (77%)

TABLE I

The confusion matrix for text frame detection

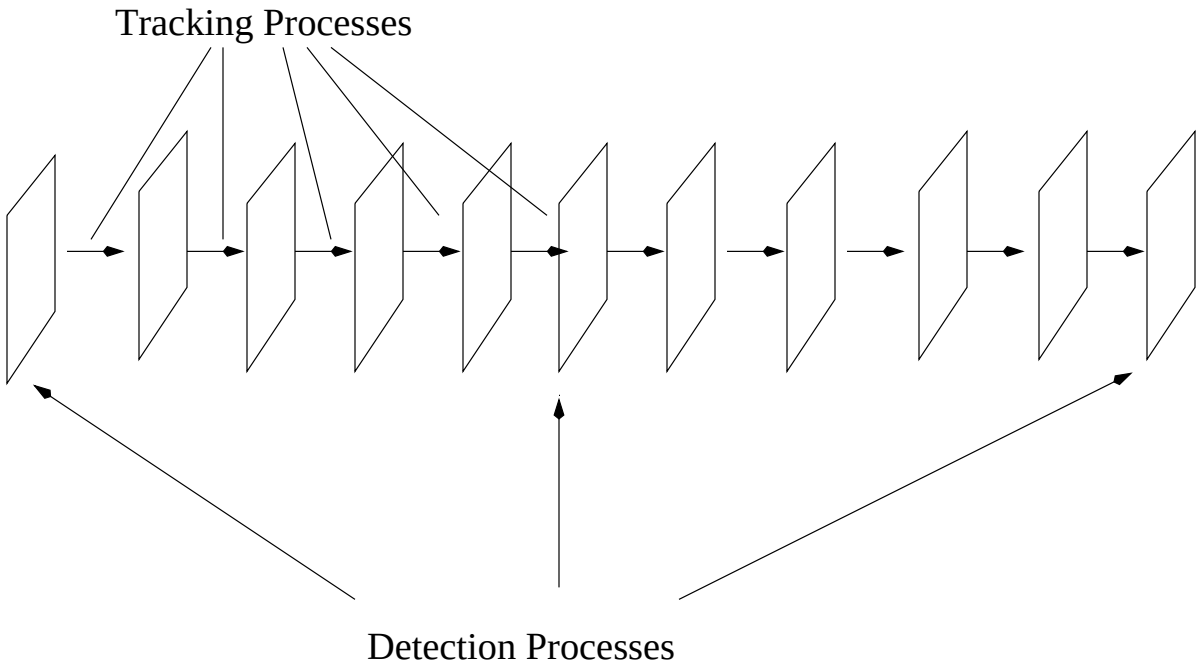


Fig. 1. The scheme for text detection and tracking in digital video.

Video ID	Video Type	Frame Number	Text Line	Motion Description
1	Movie Credit	2600	29	Complex
2	Movie Credit	2600	87	Scrolling
3	News	800	8	Static
4	Sports	200	1	Complex
5	Sports	200	2	Complex
6	Conference	800	4	Complex
7	Conference	800	1	Complex
8	Scene	228	7	Crossing
9	TV program	1200	1	Zooming in
10	Commercial	300	1	Crossing

TABLE II

VIDEO SOURCES USED IN EXPERIMENTS

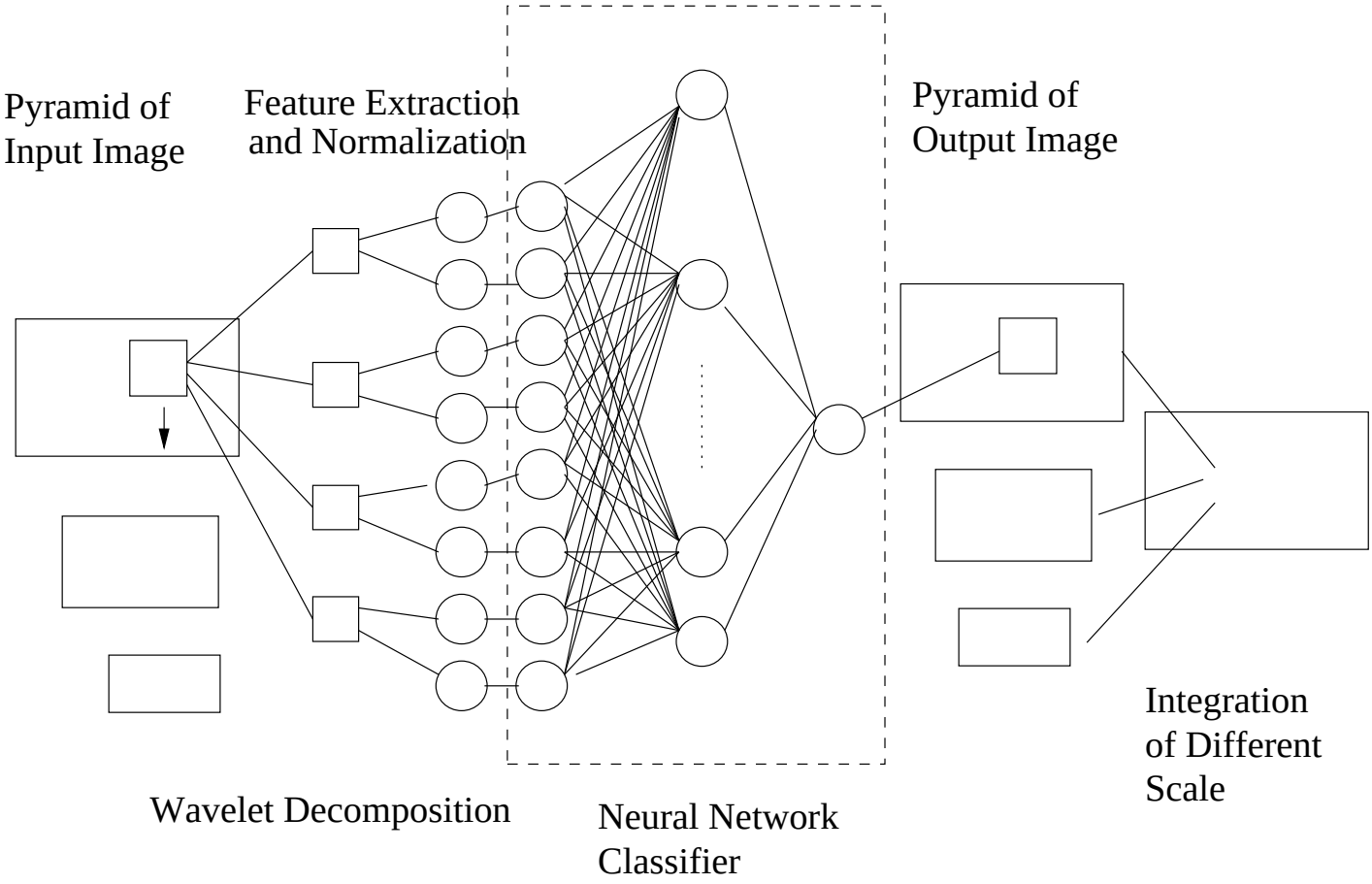
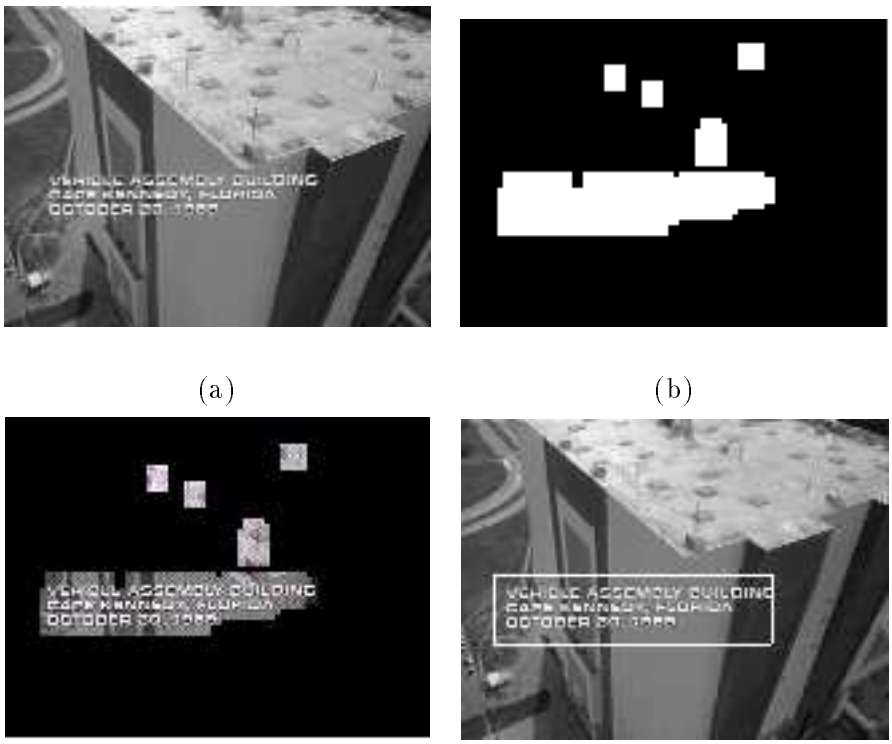


Fig. 2. The architecture for detecting text in video frames.

(a) (b)

Fig. 3. Single-level wavelet decomposition of a video frame: (a) Original image, (b) The first level decomposed images.



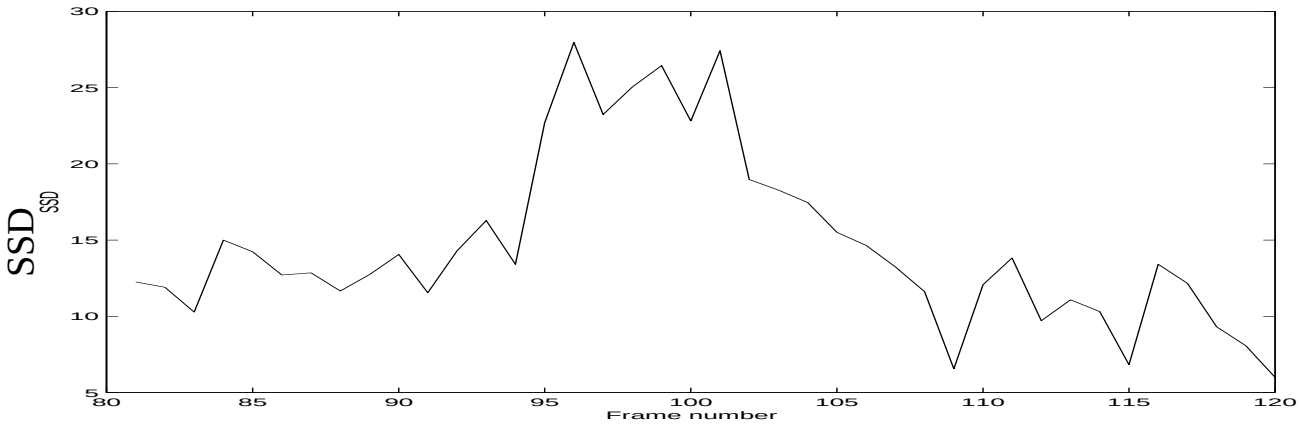
(a) (b)

(c) (d)

Fig. 4. (a) Original frame, (b) Classified label map, (c) Segmented text area, (d) Segmented text area after postprocessing and bounding box generation.

Frame 81 Frame 100 Frame 116

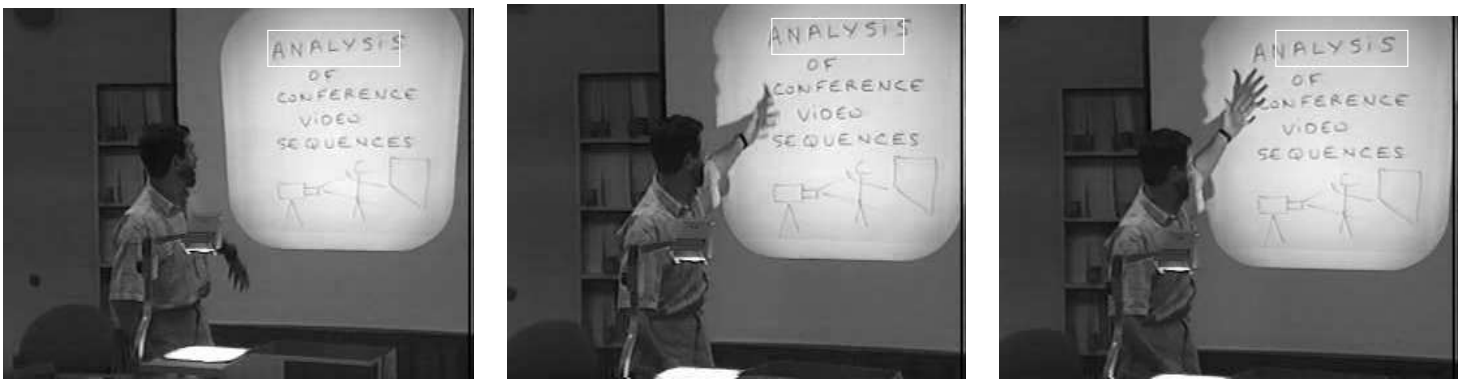
(a)



Frame Number

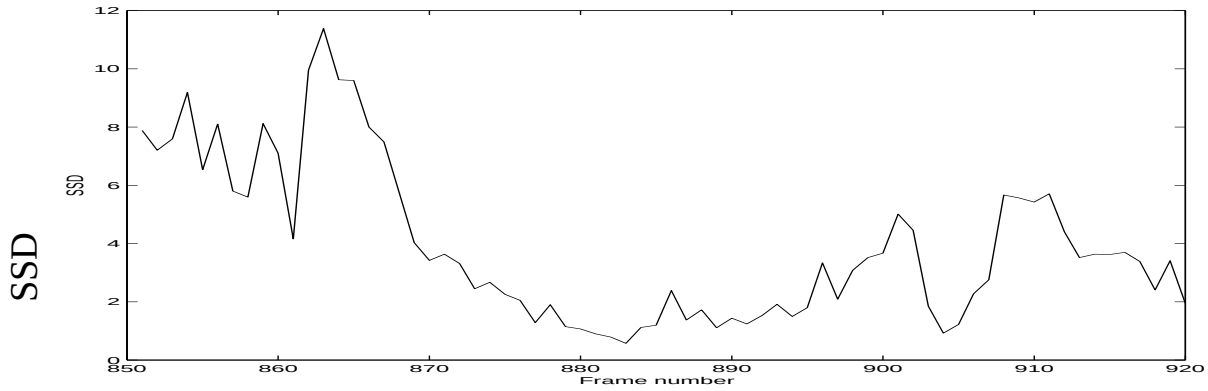
(b)

Fig. 5. SSD-based model for tracking movie credits. The initial position is specified manually. (a) Tracking result, (b) Graph of SSD in consecutive frames. The SSD is higher in frames 95–105 since the text line moves onto a complex background.



Frame 850 Frame 860 Frame 870

(a)



Frame Number

(b)

Fig. 6. Examples of conference video with scale change and rotation.

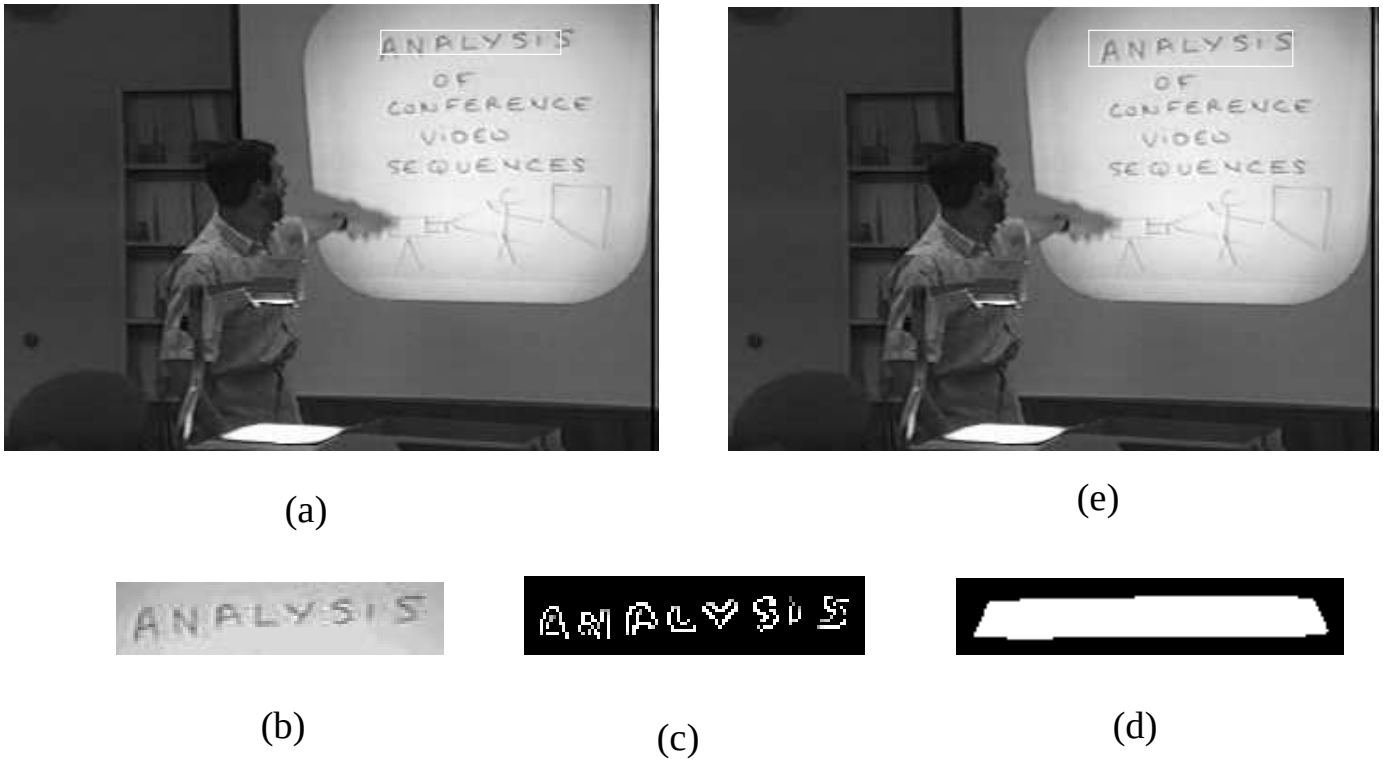
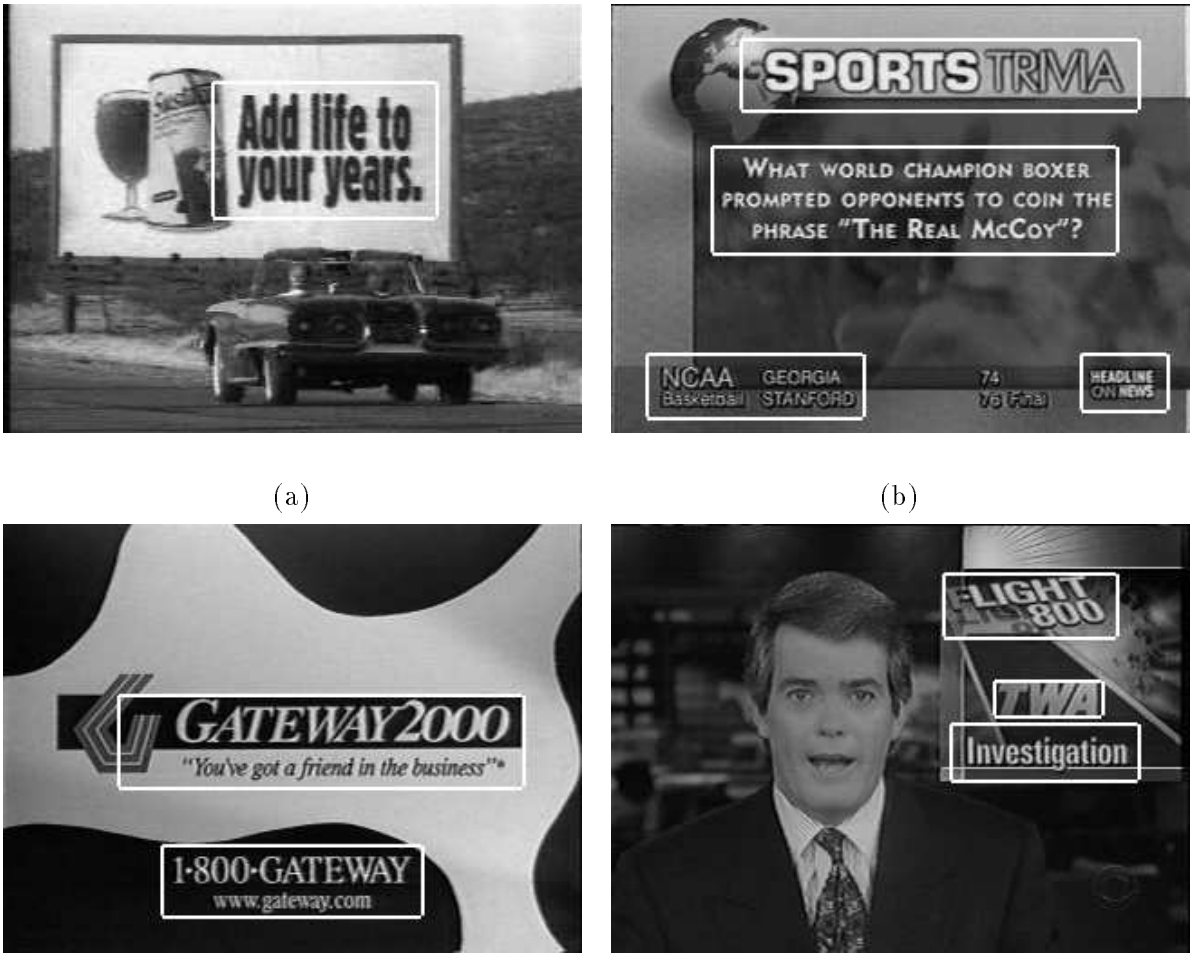


Fig. 7. Text position refinement based on contour stabilization. (a) Initial matched text position, (b) A slightly enlarged text block, (c) Edge map, (d) Smeared image, (e) Refined position.



(a) (b)

(c) (d)

Fig. 8. Detection results. (a) Reverse text with large font size, (b) Text with different font style and size, (c) Normal text and reverse text with different font sizes, (d) Text with different font styles.

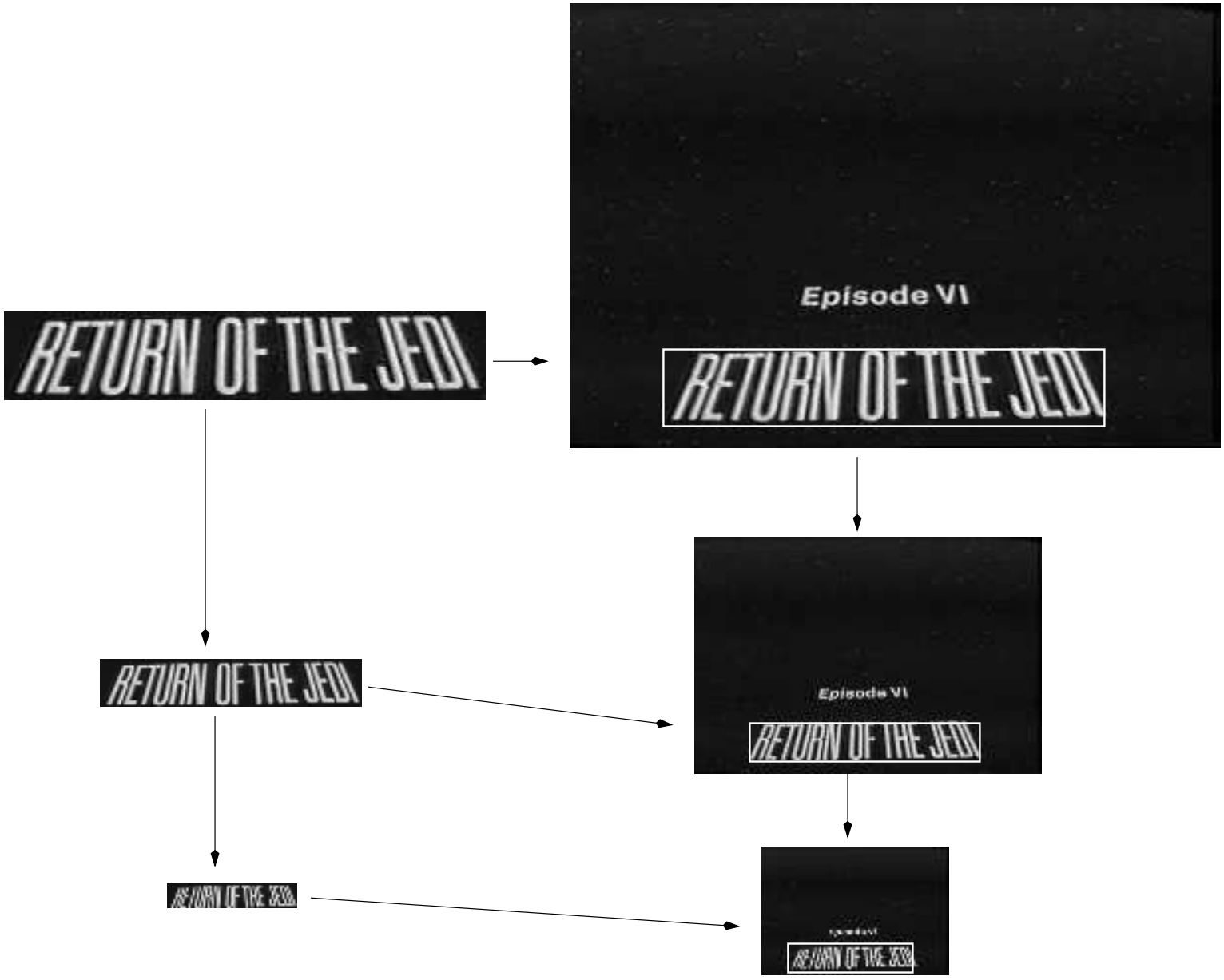
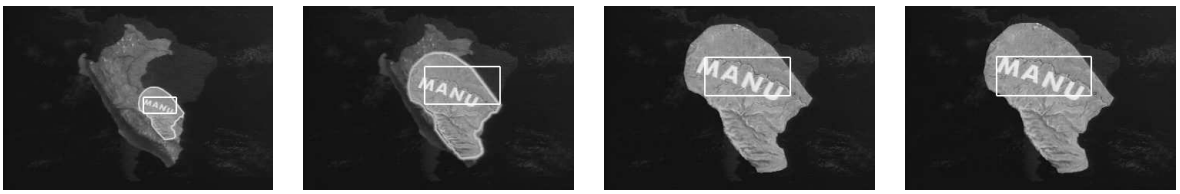
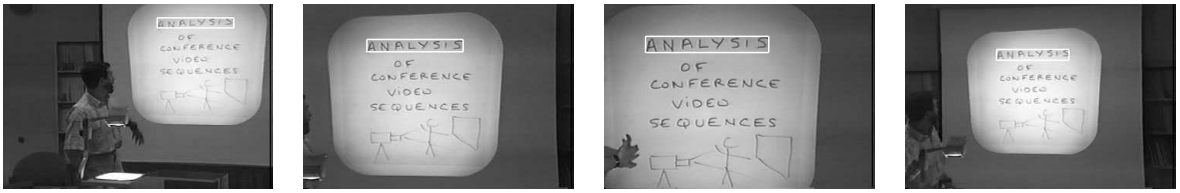


Fig. 9. Multiple resolution based image matching. (a) Text block pyramid in Frame 650, (b) Image pyramid formed with Frame 651. The matching process is conducted between images of the corresponding scale.



(a)

(b)



(c)

Fig. 10. Demonstration of *TextTracker*'s performance on various video sources. The initial position of the text block is manually specified. (a) Zooming scene text. (b) Tracking text on a football athlete's jersey. (c) Tracking text in a conference room scene.

(a) (b) (c) (d)

Fig. 11. Text detection and tracking in the movie *Star Wars*. (a) Static Text, (b) Text zooming out, (c) and (d) Text scrolling up and zooming out.

(a) (b) (c) (d)

Fig. 12. "Welcome to the Language and Media Processing Laboratory". (a) Frame 30, (b) frame 60, (c) frame 114, (d) frame 170.