Localizing and Segmenting Text in Images and Videos

Rainer Lienhart, *Member, IEEE,* and Axel Wernicke

***Abstract*—Many images—especially those used for page design on web pages—as well as videos contain visible text. If these text occurrences could be detected, segmented, and recognized automatically, they would be a valuable source of high-level semantics for indexing and retrieval. In this paper, we propose a novel method for localizing and segmenting text in complex images and videos. Text lines are identified by using a complex-valued multilayer feed-forward network trained to detect text at a fixed scale and position. The network's output at all scales and positions is integrated into a single text-saliency map, serving as a starting point for candidate text lines. In the case of video, these candidate text lines are refined by exploiting the temporal redundancy of text in video. Localized text lines are then scaled to a fixed height of 100 pixels and segmented into a binary image with black characters on white background. For videos, temporal redundancy is exploited to improve segmentation performance. Input images and videos can be of any size due to a true multiresolution approach. Moreover, the system is not only able to locate and segment text occurrences into large binary images, but is also able to track each text line with sub-pixel accuracy over the entire occurrence in a video, so that one text bitmap is created for all instances of that text line. Therefore, our text segmentation results can also be used for object-based video encoding such as that enabled by MPEG-4.**

***Index Terms*—MPEG-4 object encoding, object detection, object segmentation, text detection, text extraction, text segmentation, video indexing, video OCR, video processing.**

I. Introduction

INFORMATION is becoming increasingly enriched by multimedia components. Libraries that were originally pure text are continuously adding images, videos, and audio clips to their repositories, and large digital image and video libraries are emerging as well. They all need an automatic means to efficiently index and retrieve multimedia components.

Text in images—especially in images which are part of web pages and in videos—is one powerful source of high-level semantics. If these text occurrences could be detected, segmented, and recognized automatically, they would be a valuable source of high-level semantics for indexing and retrieval. For instance, in the Informedia Project at Carnegie Mellon University, text occurrences in videos are one important source of information to provide full-content search and discovery of their terabyte digital video library of newscasts and documentaries [26]. Detection and recognition of characteristic text occurrences may also be used to record the broadcast time and date of commercials, helping the people to check whether their client's commercials have been broadcast at the arranged time on the arranged television channel [8].

Detecting, segmenting, and recognizing text in images which are part of web pages is also a very important issue, since more and more web pages present text in images. Existing text-segmentation and text-recognition algorithms cannot extract the text. Thus, all existing search engines cannot index the content of image-rich web pages properly [13]. Automatic text-segmentation and text-recognition also helps in automatic conversion of web pages designed for large monitors to small liquid crystal displays (LCDs) of appliances, since the textual content in images can be retrieved.

Our novel method for robust text detection and segmentation in complex images and videos together with current optical character recognition (OCR) algorithms and software packages enables OCR in multimedia components, the fastest growing media type on the Internet. For video, our novel text-segmentation method is able to not only locate text occurrences and segment them into large binary images, but also to track each text line with sub-pixel accuracy over the entire occurrence in a video, so that one text bitmap is created for all instances of that text line. Thus, our text-detection and text-segmentation method can be used for object-based video encoding, which is known to achieve a much better video quality at a fixed bit rate compared to existing compression technologies. In most cases, however, the problem of extracting objects automatically is not yet solved. Our text-localization and text-segmentation algorithms solve this problem for text occurrences in videos. The rigid text objects have to be encoded only once, while their motion vector have to be updated regularly, and the pixels in the background behind the text have to be colored in such a way that it maximizes compression of the background video.

The paper is structured as follows. Section II reviews earlier related work, while Section III sketches our new approach. This is followed by a detailed description of the text-detection scheme in images (Section IV) and its extension to videos (Section V). Section VI elaborates on the text segmentation. Performance numbers are reported in Section VII. Section VIII concludes the paper.

II. Related Work

Smith *et al.* briefly propose a method to detect text in video frames [21]. They characterize text as a "horizontal rectangular structure of clustered sharp edges" [21] and exploit this feature to identify text in individual frames. They do not utilize the multiple instances of the same text over successive frames to en-

Manuscript received April 4, 2000; revised August 5, 2001. This paper was recommended by Associate Editor R. Koenen.

R. Lienhart is with the Microprocessor Research Lab, Intel Corporation, SC12-303, Santa Clara, CA 95052-8119 USA (e-mail: Rainer.Lienhart@intel.com).

A. Wernicke was with the Microprocessor Research Lab, Intel Corporation, Santa Clara, CA 95052-8119 USA. He is now with Forcont Business Technology Gmbh, Nonnenstrasse 39, 04229 Leipzig, Germany.

hance detection performance. The approach is also scale dependent, i.e., only text within a certain font size range is detected, and does not address its preparation (i.e., text segmentation) for OCR.

Zhong *et al.* propose two simple methods to locate text in complex images [30]. The first approach is mainly based on finding connected monochrome color regions of a certain size, while the second locates text based on its specific spatial variance. Both approaches are combined into a single hybrid approach. Since their methods were designed primarily to locate text in scanned color CD cover images, they are not directly applicable to video frames. Usually, the signal-to-noise ratio (SNR) is much higher in scanned images, while its low value in videos is one of the biggest challenges for text segmentation.

In [6], Jain and Yu introduce a method for text localization that is suitable in a number of applications, including newspaper advertisements, web pages, and images and video in general. As in [7], the text-localization algorithm is based on connected component analysis, and thus requires either text to be monochrome or its background. Visually good results are presented in [6] for advertisement images, web banner images, scanned magazine pages, and video frames. The authors point out problems with small text fonts and cases where the image's 3-D color histogram is sparse, and thus no dominant color prototypes exist. These cases, however, may often occur in videos. No quantitative performance numbers are reported.

Wu *et al.* propose a four-step system that automatically detects and extracts text from images [27]. First, text is treated as a distinctive texture. Potential text locations are found by using three second-order derivatives of Gaussians on three different scales. Second, vertical strokes emerging from horizontally aligned text regions are extracted. Based on several heuristics, strokes are grouped into tight rectangular bounding boxes. These two steps are applied to the input image at all scales in order to detect text over a wide range of font sizes. The boxes are then fused at the original resolution. Third, the background is cleaned up and binarized. Finally, text boxes are refined by repeating steps 2 and 3 with the text boxes detected thus far. By failing to provide any mechanism to infer the text color, two binary images are produced for each text box. These binary images are passed on to standard OCR software for evaluation. Wu *et al.* report a recognition rate of 84% for 35 images. Unfortunately, the algorithms were designed to deal with scanned images, which usually have a better SNR than video frames and, therefore, may not work best on video frames. In addition, the algorithms do not exploit the temporal information available in video streams.

Recently, Sato *et al.* developed a system for segmenting and recognizing static low-resolution caption text in newscasts [19]. They use the method proposed in [21] to detect caption text. Detected text blocks are then magnified by a factor of 4 and integrated over time by means of a time-based minimum pixel value search. This multiframe integration approach assumes that on average captions are brighter than their background pixels. While these restrictions may be acceptable for newscasts, they are too strict for general artificial text occurrences in video. Sato *et al.* also present a new character extraction filter and an integrated approach of text recognition and segmentation. However, it is very tailored to their application domain (newscasts) and video format (MPEG-1) [19]. They report a character recognition rate of 83.5% and a word recognition rate of 70.1% with respect to the 89.6% correctly detected text locations, i.e., a character recognition rate of 74.8% and a word recognition rate of 62.8% with respect to the ground truth. Word recognition was improved by using the Oxford dictionary and a closed caption dictionary.

Two systems for detecting, segmenting and recognizing text in video have been developed by Lienhart *et al.* [7], [9]. Both systems are based on image segmentation exploiting the monochromaticity, the high contrast with its background and the simple texture of text in video. The first system uses a color-based split-and-merge algorithm [7], while the latter system counts on anisotropic diffusion [9]. The system in [7] tracks text only short-term in order to rule out nontext regions. Text, however, is segmented and recognized on a frame basis and not associated over time. An iterative text-recognition algorithm is applied to transcribe the text into ASCII. In [9], text is tracked over its life time, and multiple text occurrences are integrated over time. The commercial Recognita OCR engine is used for text recognition. Recognition rates between 41%–76% are reported.

Li *et al.* use a neural feed-forward network to locate text in videos [10], [11]. The high-frequency wavelet coefficients of a frame are the input to the network. The trained neural network monitors a video for text occurrences, proceeding each time by a certain number of frames if no text was found. If text was found, it is tracked coarsely against a simple background by using plain block matching [10], [11]. Although Li's approach is partially similar to ours, their system has many shortcomings.

1) Their method is restricted to video only.
2) It detects text only at the block level. A text block is defined as text lines which are close to each other. This can cause severe problems if the text lines do not move homogeneously.
3) I does not achieve sub-pixel accurate tracking since their system is not able to determine the text color. Consequently, it cannot remove the background during the search for the actual text location.

A. Contributions

The main contributions of this paper are the following.

- *A truly multiresolution approach.* Our novel text localization and text segmentation algorithms work successfully for small and large images as well as from MPEG-1 video sequences up to HDTV MPEG-2 video sequences (1980×1280) without any parameter adjustment. Character sizes can vary between 8 pixels and half the image height. Only [10] and [27], [28] address the problem of multiresolution. However, our approach uses more scales than their approaches.
- *A truly multimedia text detector and text segmenter.* The proposed system can localize and segment text in images (especially complex images in web pages) and videos. None of the systems in the related work section except [10], [27], and [28] are capable of doing this. Wu *et al.*

show results of text detected in web pages, maps, ads, images, and static video frames, but do not take the temporal redundancy of video into account. Reference [10] claims that their approach works on images and videos, however they only report data for videos.

- *A novel way to estimate the text color reliably by using vector quantization.* None of the existing work is capable of doing this. For instance, [19] and [20] assume that, on average, over time the text color is brighter than its background. For applications where this in not true, Sato *et al.* propose to try out the results for normal and inverse text. Although Hori proposes a scheme to estimate the character intensity distribution by means of intensity mean estimation of high-intensity pixels, his method also depends on the assumption that characters are brighter than their background [5]. In detail, he assumes that the intensity values of all character pixels are equal or larger than $m - 3\sigma$, while all background pixels have lower intensity with m denoting the estimated mean of the high-intensity pixels in the text box and σ their standard deviation.
- *A new scheme to register moving text lines perfectly over time.* This allows use of the temporal segmentation scheme first proposed by [19] and [20]. Overall, previous work that reports text tracking such as [7], [9]–[11] do it more on a qualitative basis in order to identify false alarms. Tracking text is not pixel-accurate.
- *Unlike all existing work, detected text lines are scaled to an aspect-ratio preserving fixed height of 100 pixels during text segmentation.* This scaling improves segmentation for text of smaller font sizes as well as saves time for text of font sizes larger than 100 pixels.

III. Strategy and Overview

A top-down approach is taken in our text localization and text segmentation system. In a first step, potential text lines in images, video frames or web pages are localized (Section IV). These potential text lines are refined in the case of video by exploiting its temporal redundancy (Section V). In a second step, localized text lines/objects are segmented into binary images with black characters on white background and a fixed height of 100 pixels (Section VI). Again, in the case of video, temporal redundancy is used to improve segmentation. The output of the text segmentation is passed on to a standard OCR software package.

Three basic decisions preceded the development of our new text localization scheme.

1) Only horizontal text is considered, since it accounts for more than 99% of all artificial text occurrences. The experiences with our old systems [7], [9], which considered any writing direction, suggest that taking the missing 1% of text occurrences into account must be paid off by a much higher false alarm rate. As long as a performance of 90% and higher of correctly segmented text in videos and complex images is still a challenge, nonhorizontal text can be neglected.
2) Non-text regions are much more likely than text regions. Therefore, we decided to train the raw text detector as narrowly as possible, i.e., it is trained to detect text at a fixed position and scale. Scale and position independence is achieved by applying the raw text detector to all positions at all scales.
3) Text occurrences only matter if they consist of at least two letters and/or digits.

IV. Text Localization in Images

The input to the text localization step may be complex images, images embedded in web pages or videos. It is the task of the text localization to locate and circumscribe text occurrences in all these kinds of multimedia data by tight rectangular boxes. Each so-called text bounding box is supposed to circumscribe only a single text line.

Fig. 1 gives an overview of our text localization scheme. It visualizes the various processing steps and will become clear while we step through in the remainder of this chapter.

A. Image Features

Artificial text occurrences have been commonly characterized in the research community as regions of high contrast and high frequencies [9], [19]. There are many different ways to amplify this feature. In this work, we choose to use the gradient image of the RGB input image $I(x, y) = (I_r(x, y), I_g(x, y), I_b(x, y))$ in order to calculate the complex-values edge orientation image $E(x, y)$. $E(x, y)$ is defined as follows:

$$E(x, y) = \left(\sum_{c \in \{r, g, b\}} \left| \frac{I_c(x, y)}{dx} \right|, \sum_{c \in \{r, g, b\}} \left| \frac{I_c(x, y)}{dy} \right| \right).$$

E maps all edge orientations between 0° and 90°, and thus distinguishes only between horizontal, diagonal and vertical orientations. E serves as our feature for text localization.

B. Fixed Scale Text Detector

Given a 20 × 10 pixel region in an edge orientation image E, the fixed scale text detector is supposed to classify whether the region contains text of a certain size. Many different techniques exist for developing a classifier [14]. For our work, we compared the performance of a real-valued and complex-valued neural feed-forward network [15]. Due to its superior performance, we decided on the complex-valued neural network with hyperbolic tangent activation function. At a comparable hit rate (90%), its false hits (0.07%) on the validation set were more than twice as low as with a comparable real-valued network.

Network Architecture: 200 complex-valued neurons were fed by a 20 × 10 edge orientation region in E (one link from each complex-valued edge pixel to each complex-valued input neuron). This size of the receptive field exhibits a good tradeoff between performance and computational complexity. An input layer of 30 × 15 neurons did not achieve better classification results and was computationally more expensive. On the other hand, using an input layer with fewer than ten rows resulted in substantially worse results. Note that the number of rows of the receptive field determines the size of the font being detected since all training text patterns are scaled such that the font

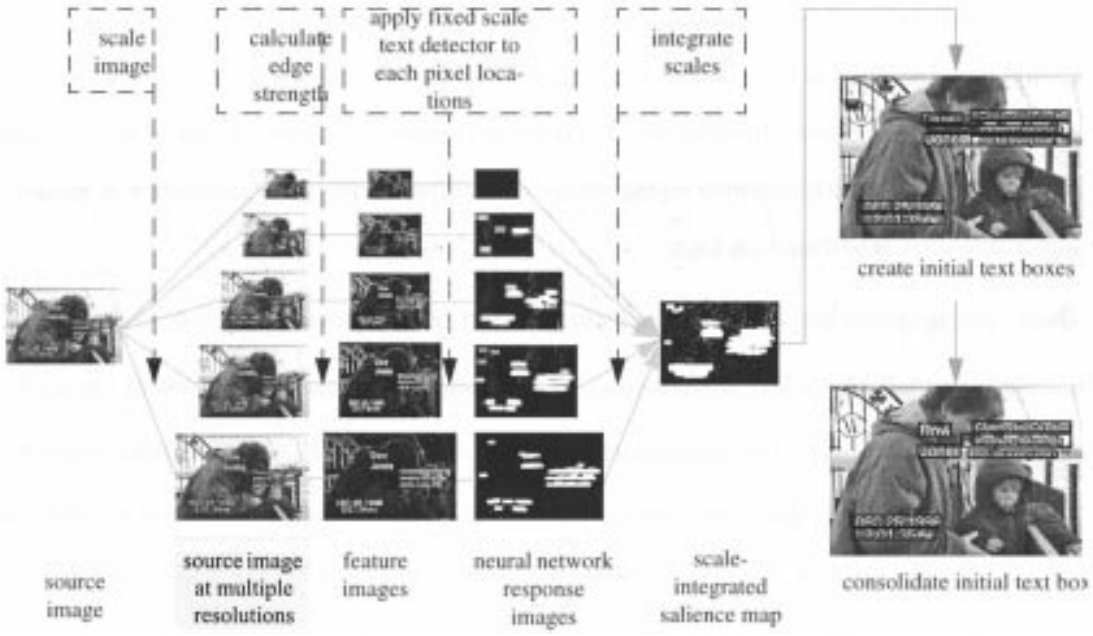


Fig. 1. Scale- and position-invariant text localization.

size is equal to the number of rows. The input layer in turn is fully connected to a hidden layer of two complex-valued neurons using hyperbolic tangent activation functions. Again, using more hidden neurons did not result in any performance improvements, while using only one increased the false alarm rate by a factor of three. The hidden layer is aggregated into one real-valued output neuron of range $[-1, 1]$.

Network Training: The composition of the training set seriously affects a network's performance. We collected a representative set of 30 180 text patterns and 140 436 nontext patterns. Initially 6000 text patterns and 5000 nontext patterns were selected randomly for training. Only the nontext pattern set was allowed to grow by another 3000 patterns collected by means of the so-called "bootstrap" method. This method, proposed by Sung [22], starts with an initial set of nontext patterns to train the neural network. Then, the trained network is evaluated using a validation set distinct from the training set (all patterns minus the training set). Some of the falsely classified patterns of the validation set are randomly added to the training set and a new—hopefully enhanced—neural network is trained with this extended and improved training set. The resulting network is evaluated with the validation set again and additional falsely classified nontext patterns are added to the training set. This cycle of training and adding new patterns is repeated until the number of falsely classified patterns in the validation set does not decrease anymore or, as in our case, 3000 nontext patterns (and only nontext patterns) have been added. This iterative training process guarantees a diverse training pattern set.

Given a properly trained neural network, a 20×10 pixel window slides over the edge orientation image E and is evaluated at each position. The network's response is stored in a so-called response image by filling the associated 20×10 region in the response image with the networks output value if and only if it exceeds $th_{network} = 0$. Since a step size of one is computationally prohibitive for large images or HDTV video sequences, we use a step factor of 3 and 2 in the x and y direction, respectively. We proved experimentally that the subsampling does not cause any decrease in accuracy but a speed-up of $6\times$.

C. Scale Integration

The raw fixed-scale text detector is applied to all scales using a scale down factor of 1.5. In order to recover initial text bounding boxes, the response images at the various scales must be integrated into one saliency map of text. As you can observe from Fig. 1 column 4, text locations stick out as correct hits at multiple scales, while false alarms appear less consistently over multiple scales. Similar results have been observed by Rowley *et al.* for their neural network-based face detector [18] and by Itti in his work on models of saliency-based visual attention [3]. Therefore, a text salience map is created by projecting the confidence of being text (here: the activation level of the neural network output) back to the original scale of the image. Hereto, the salience map is initialized by zero. Then, for each 20×10 pixel window at each scale, its confidence value for text is added to the saliency map S over the size of the bounding box at the original image scale if and only if the window was classified as text. Fig. 1 column 5 shows an example. Confidence in text locations is encoded by brightness.

D. Extraction of Text Bounding Boxes

1) Initial Text Bounding Boxes: To create an initial set of text bounding boxes, a special kind of region-growing algorithm is employed, where each rectangular region is only allowed to grow by one complete row or column of the rectangle and the seed position is required to meet a minimal amount of text saliency. The merge decision is based on the average saliency of the candidate row/column.

In detail, the algorithm starts to search for the next pixel which has not yet been processed in the saliency map with a value larger then a pre-specified threshold th_{core}. The choice of the threshold's value is determined by the goal of avoiding the creation of text boxes for nontext regions. Nontext regions should be less salient than text regions. For our classifier $th_{\text{core}} = 5.0$ worked fine, however, it may have to be adjusted if a new neural network is trained. Semantically, this threshold means that the sum of all text probabilities of the tested windows overlapping with the respective pixel add up to at least 5.0. Once a pixel in the saliency map with $S(x, y) > th_{\text{core}}$ is found (henceforth called a core pixel), it is taken as a seed for a new text box of height and width 1. This new text box is then expanded iteratively based on the average pixel value of the row above the box in the saliency map S: if the average value exceeds $th_{\text{region}} = 4.5$, the row is added to the text box. th_{region} is chosen to be slightly smaller than th_{core} in order to create a text box encompassing all parts of the characters and not only their middle part. The same criterion is used to expand the box to the left, bottom, and right. This iterative box expansion repeats until the text box stops growing. One example of the initial text bounding boxes created by this algorithm is given in Fig. 4(a).

2) Refined Text Bounding Boxes: As shown in Fig. 4(a), the initial bounding boxes circumscribe the text in the image, however, in a sub-optimal fashion: some boxes span more than one line and/or column of text, others contain no text, while in many the background makes up a large portion of the pixels. Fortunately, these shortcomings can be overcome by an iterative post-processing procedure utilizing the information contained in so-called projection profiles [20].

A projection profile of an image region is a compact representation of the spatial pixel content distribution and has been successfully employed in document text segmentation [20]. While histograms only capture the frequency distribution of some image feature such as the pixel intensity (all spatial information is lost), intensity projection profiles preserve the rough spatial distribution at the cost of an even higher aggregation of the pixel content. A horizontal/vertical projection profile is defined as the vector of the sums of the pixel intensities over each column/row. Fig. 2 shows an example. Vertical and horizontal projection profiles are depicted as bar charts along the x and y axes of the feature images, and the upper boundaries of text lines are marked by steep rises in the vertical projection profile while the lower boundaries are marked by steep falls. Similarly, the right and left boundaries of text objects are indicated by steep rises and falls in the horizontal projection profile. These steep rises and falls can be identified as locations where the profile graph crosses an adaptively set threshold line.

In detail, the vertical segmentation algorithm applied to each text box works as follows. First, the box is enlarged at the top and bottom by 25% or half the maximal possible text height, whichever is smaller [steps (1) and (2) in Fig. 3(a)]. This enlargement is necessary, because the correct boundaries may lie outside the current text box and thus accidentally cut off portions of the text. Some rows outside the current text box must be taken into consideration, to recover these boundaries correctly, and half the height of the original text box is a good worst case

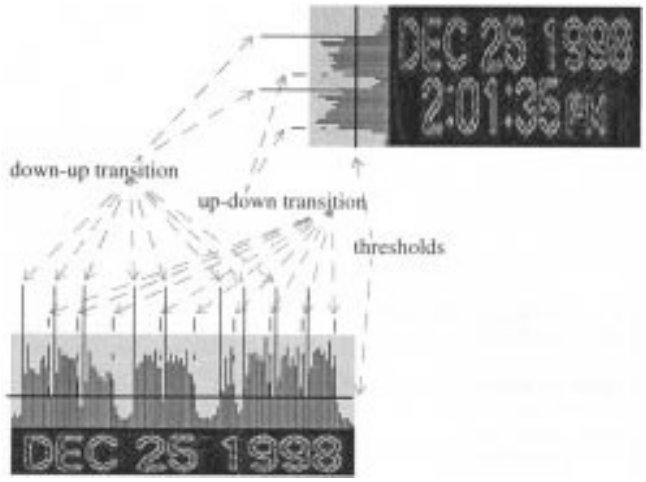


Fig. 2. Examples of projection profiles and their usage for determining individual text lines and words.

estimate. Next, the vertical projection profile over the enlarged text box in $|E|$ is calculated as well as the maximum and minimum values $\max_{\text{profile}}$ and $\min_{\text{profile}}$ in the profile. In order to determine whether a single value in the projection profile belongs to a text line (step (4) in Fig. 3), the adaptive threshold $\text{thresh}_{\text{text}} = \min_{\text{profile}} + (\max_{\text{profile}} - \min_{\text{profile}}) \cdot 0.175$ is calculated. Every line with a vertical profile value exceeding $\text{thresh}_{\text{text}}$ is classified as containing text. The factor of 0.175 was chosen experimentally.

In steps (6)–(8), the vertical segmentation algorithm [see Fig. 3(a)] begins to search for the first down–up transition starting from the top. This row is marked as a potential upper boundary of a text box (9). Then, the next up–down transition is searched in the projection profile (13). If found, a new box with the last upper and lower boundaries is created. The search for a new pair of down–up and up–down transitions continues until all elements in the projection profile have been processed. Finally the original text box is deleted. The original text box is now split into its text lines [see Fig. 4(b)].

A similar horizontal segmentation algorithm is applied to ensure that each box consists of only one line of text [see Fig. 3(b)]. However, there are two minor but important differences.

1) A factor of 0.25, instead of 0.175, is used in the computation of $\text{thresh}_{\text{text}}$. Experimentally, this value has proven to be superior for the horizontal segmentation.
2) The *gap* parameter has been added to avoid splitting up characters and words in the "same" column due to the small gaps between them. Therefore, all newly bounding boxes created by the horizontal segmentation algorithm equivalent to the vertical segmentation algorithm are merged into one wider text box if and only if they are less than *gap* pixels apart, otherwise the text boxes are treated as belonging to separate columns. In the current system, the *gap* parameter is set adaptively to the height of the current text box. The whole algorithm is given in see Fig. 3(b).

Fig. 4(c) shows the result of the horizontal segmentation algorithm applied to our sample image.

```
(1) expand box at the top and bottom by the minimum of
    half the height of the original text box and half
    the possible maximal text height
(2) calculate vertical projection profile of |E|
(3) get minimum and maximum profile values
(4) calculate the segmentation threshold
(5) set change = false

(6) for all rows of the profile
(7)   if (profile[current row] > threshold)
(8)     if (no upper boundary yet)
(9)       set upper boundary = current row
(10)  else
(11)    if (no lower boundary yet)
(12)      set lower boundary = current row
(13)      if (upper boundary)
(14)        create new box using the values of the
            upper and lower boundaries
(15)        unset upper and lower boundaries
(16)        set change = true

(17)  delete processed box
```

(a)

```
(1) expand box at the left and right by the minimum of
    half the height of the original text box and half
    the possible maximal text height
(2) calculate horizontal projection profile of |E|
(3) get minimum and maximum profile values
(4) calculate the segmentation threshold

(5) for all columns of the profile
(6)   if (profile[current column] > threshold)
(7)     if (no left boundary yet)
(8)       set left boundary = current column
(9)     else if (right boundary)
(10)      if (gap between current column and right
              boundary is large enough)
(11)        create new box from left and right
            boundaries
(12)        unset left and right boundaries
(13)      else
(14)        unset right boundary
(15)  else if (no right boundary)
(16)      set right boundary = current column

(17) if (left && no right boundary)
(18)   right boundary = last column
(17) if (left and right boundaries)
(18)   update processed box to current right/left
       boundaries
(19) else
(20)   delete processed box
```

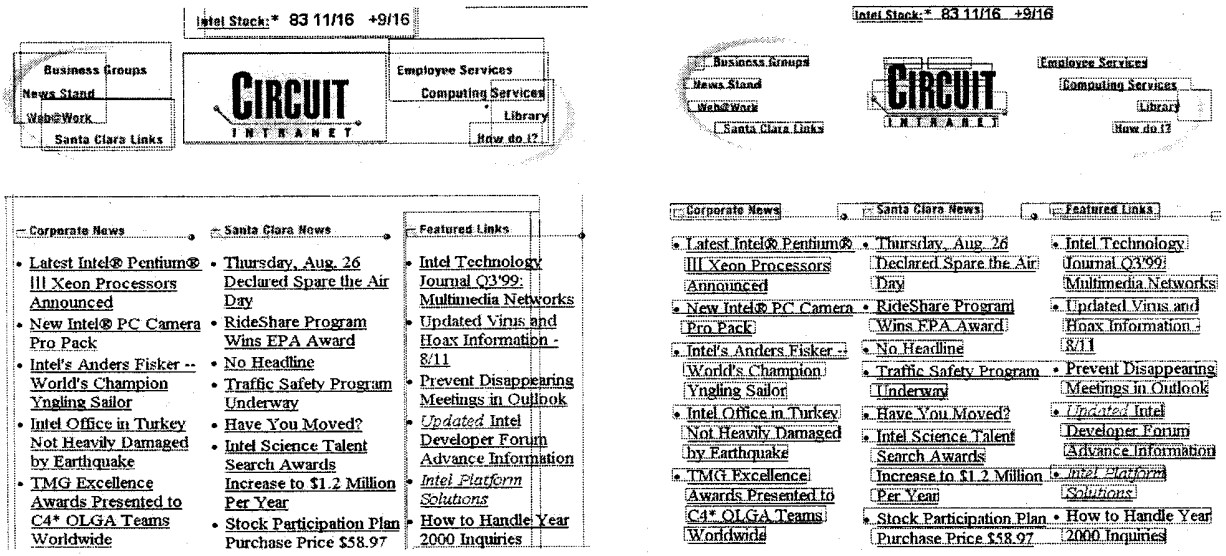
(b)

Fig. 3. (a) Vertical and (b) horizontal segmentation algorithms.

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

Fig. 4. (a) Initial bounding boxes. (b) One vertical segmentation step. (c) One vertical and horizontal segmentation step. (d) Multiple cycles and clean-up.

Although the vertical and horizontal segmentation achieved good results on the sample frame in Fig. 4, one pass does not resolve complex layouts. One such example is shown on the left side of Fig. 4. Obviously, the vertical segmentation could not separate the different text lines of “Commodities trading involves risk and is not for everyone” [see Fig. 4(b)]. The reason for this becomes clear if one imagines what the vertical projection profile for the respective text box in Fig. 4(a) looks like. The text box in the left column obviously masks the vertical profiles of the smaller text to the right which therefore could not be split



(a) (b)

Fig. 5. (a) Initial text bounding boxes. (b) Completely localized web page.

into two text lines. On the other hand, the gap between these two text columns is large enough to be split up after the horizontal segmentation algorithm was applied [see Fig. 4(c)]. Experimentally, it turned out that almost every layout can be divided into its text rows and columns if a few cycles of vertical and horizontal segmentations are applied to each text box [see Fig. 4(d)].

However, as a side effect, a few cycles of vertical and horizontal segmentations may produce some very small boxes and/or boxes of large width-to-height ratios. Since the text height in images as well as in video frames is limited, boxes with

$$\text{height} < \min_{\text{textheight}} = 8\text{pt}$$

or

$$\text{height} > \max_{\text{textheight}} = \frac{\text{image}_{\text{height}}}{2}$$

are classified as nontext regions and, therefore, discarded. Moreover, since horizontal segmentation assures that text boxes contain text objects like words or text lines, the height of correctly segmented text boxes must be smaller than their width. Consequently, boxes with $\text{height} > \text{width}$ are discarded, too. Finally, text boxes sharing the same upper and lower boundary and being close enough to touch or overlap each other are joined into one text box. This reduces complexity and will later enable a more stable text tracking throughout time. Fig. 5 shows an example for a web page.

3) Estimating Text Color and Background Color: Estimates of the text color and background color for each text box are needed later to determine whether a text bounding box contains normal (i.e., dark text on bright background) or inverse text (i.e., bright text on dark background). Unfortunately, images and videos are colorful, and even a visually single-colored region like a character in a video frame consists of pixels of many different but similar colors. Therefore, the complexity of the color distribution in each text bounding box is reduced by quantization to the four most dominating colors using the fast color vector quantizer proposed by Wu [29]. Two color histograms are derived from the color-quantized text box.

1) A color histogram covering the four center rows of the text box.
2) A color histogram covering the two rows directly above and underneath the text box (four rows together).

The latter histogram should describe an image region of little or no text, while the first histogram should be dominated by the text color. The maximum value of the difference histogram between the first and second histogram is very likely to correspond to the text color and the minimum to the dominating background color. This methodology has experimentally proven to be very reliable for homogeneously colored text. Of course, it may fail for multicolored text which, however, is rare.

Based on the estimated text and most dominant background color, we estimate whether a text bounding box contains normal text or inverse text. If the grayscale value of the text color is lower than the dominant background, we assume normal text; otherwise, inverse text.

V. Text Localization in Videos

Videos differ from images and web pages by temporal redundancy. Each text line appears over several contiguous frames. This temporal redundancy can be exploited to:

1) increase the chance of localizing text since the same text may appear under varying conditions from frame to frame;
2) remove false text alarms in individual frames since they are usually not stable throughout time;
3) interpolate the locations of "accidentally" missed text lines in individual frames;
4) enhance text segmentation by bitmap integration over time.

To exploit the redundancy inherent in video, text boxes of the same content in contiguous frames must be summarized in one

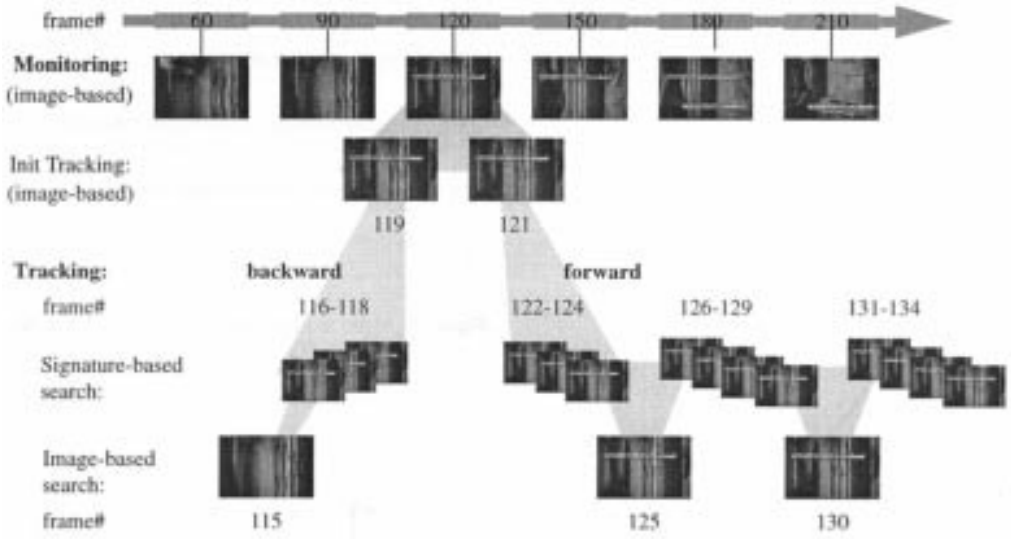


Fig. 6. Relationship between video monitoring (stage 1) and text tracking (stage 2).

text object based on the visual contents of the text boxes. A text object describes a text line over time by its text bitmaps, sizes, and positions in the various frames, as well as its temporal range of occurrence.

Text objects are extracted in a two-stage process in order to reduce computational complexity. In a first stage, a video is monitored at a coarse temporal resolution (see Fig. 6 and [19], [21]). For instance, the image-based text localizer of Section IV is only applied to every 30th frame. If text is detected, the second stage of text tracking will be entered. In this stage, text lines found in the monitor stage are tracked backward and forward in time up to their first and last frame of occurrence. This stage uses a combination of signature-based search of text lines and image-based text localization in order to reduce computational complexity even further.

A. Video Monitoring

Video is monitored for text occurrences at a coarse temporal resolution. For this purpose, the image-based text localizer is only applied to a temporally evenly spaced frame subset of the video. The step size is determined by the objective not to overlook any text line. However, it is completely unimportant whether text lines are localized at the beginning, at the middle or at the end of their temporal occurrence. In any case, the text tracking stage will recover the actual temporal range of each text line.

The maximum possible step size is given by the assumed minimum temporal duration of text line occurrences. It is known from vision research that humans need between 2 and 3 seconds to process a complex scene [12], [17]. Thus, we should be on the safe side to assume that text appears clearly for at least one second. For a 30-fps video, this translates to a step size of 30 frames (see Fig. 6).

If the image-based text localizer does not find any text line in frame_t, the monitor process continues with frame_{t+30}. If, however, at least one text line is found, the image-based text localizer will be applied to frame_{t-1} and frame_{t+1}. Next, for each text line in frame_t, the algorithm searches for a corresponding text

```
(1) video = (frame 0, ..., frame T)
(2) for t = 0 to T step 1 second
(3)   localize text in frame t
(4)   if no text line found
(5)     continue with next t
(6)   localize text in frame t-1 and t+1
(7)   for all text lines in frame t which do
(8)         not belong to any text object yet) {
(9)     search for corresponding text line in t-1, t+1
(10)    if search successful
(11)       create new text object
(12)       track text object backward
(13)       track text object forward
```

Fig. 7. Video-monitoring algorithm for text occurrences.

line in frame_{t-1} and frame_{t+1}. Correspondence between two text lines is assumed if their respective bounding boxes at their original frame locations overlap each other by at least 80%. The percentage of overlap is defined as

$$\mathrm{overlap} = |A \cap B| / |A|$$

given that A and B represent the point sets describing the reference and the second bounding box, respectively. Consequently, two corresponding boxes cannot differ by more than 20% in size if they occur at the same position and/or are only allowed to be slightly shifted with respect to each other if they have the same size. Small shifts are common for nonstatic text. If corresponding boxes in frame_{t-1} and frame_{t+1} are found for a text box in frame_t, a new text object (comprising these text boxes) is created and marked for tracking in time. A summary of the video monitoring process is given in Fig. 7.

B. Text Tracking

Each text object must now be extended backward and forward in time to all frames containing the respective text line. We will restrict our description to forward tracking only, since backward tracking is identical to forward tracking, except in the direction you go through the video.

Basically, our fast text tracker takes the text line in the current video frame, calculates a characteristic signature which allows discrimination of this text line from text lines with other contents, and searches in the next video frame for the image region of the same dimension which best matches the reference signature. This signature-based exhaustive search algorithm resembles the block-matching algorithm for motion prediction [1], except that the similarity measure is based on a signature derived from a feature image of the actual image. The vertical and horizontal projection profiles defined in Section IV-D-2 serve as a compact and characteristic signature, and the center of a signature is defined as the center of the associated bounding text box. Similarity between two signatures is measured by signature intersection, i.e., by the sum of the minimum between respective elements in the signatures [23]. Cases where signatures capture an object of interest as well as changing background signature intersection are known to outperform L-norms [23]. All signatures whose centers fall into a search window around the center of the reference signature are calculated and compared to the reference signature in order to find the precise position of a text line in the next frame. If the best match exceeds a minimal required similarity, the text line is declared to be found and added to the text object. If the best match does not exceed a minimal required similarity, a signature-based drop-out is declared. The size of the search radius depends on the maximal assumed velocity of text. In our experiments, we assumed that text needs at least 2 s to move from left to right in the video. Given the frame size and the playback rate of the video, this translates directly to the search radius in pixels. In principle, we could predict the location by the information contained in the text object so far to narrow down the search space; however, there was no computational need for it.

The signature-based text line search cannot detect a text line fading out slowly, since the search is based on the signature of the text line in the previous frame and not on a fixed master/prototype signature. The frame to frame changes are likely to be too small to be detectable. Further, the signature-based text line search can track zooming in or zooming out text only over a very short period of time. To overcome these limitations, the signature-based search is replaced every xth frame by the image-based text localizer in order to re-calibrate locations and sizes of the text lines. Heuristically, every fifth frame turned out to be a good compromise between speed and reliability. Again, the bounding boxes of corresponding text lines must overlap by at least 80%.

In addition, continuous detection and tracking (i.e., in every frame) of text objects is often not possible due to imperfection in the video signal such as high noise, limited bandwidth, text occlusion, and compression artifacts. Therefore, tracking should be terminated only if for a certain number of contiguous frames no corresponding text line could be found. For this, two thresholds $\max_{\mathrm{DropOut}}^{\mathrm{signature-based}}$ and $\max_{\mathrm{DropOut}}^{\mathrm{image-based}}$ are used. Whenever a text object cannot be extended to the next frame, the drop-out counter of the respective localization technique is incremented. The respective counter is reset to zero whenever the search succeeds. The tracking process is finished as soon as one of both counters exceeds its threshold $\max_{\mathrm{DropOut}}^{\mathrm{signature-based}}$ or $\max_{\mathrm{DropOut}}^{\mathrm{image-based}}$. In our experiments, the thresholds were set to $\max_{\mathrm{DropOut}}^{\mathrm{image-based}} = 3$ and $\max_{\mathrm{DropOut}}^{\mathrm{signature-based}} = 4$. A value of 4 allows for tracking of text lines where signature-based search is very difficult such as for zooming in or zooming out text. A summary of the video-monitoring process is given in Fig. 8.

```
(1) sigBased_DropOuts = 0
(2) imageBased_DropOuts = 0
(3) while (not (beginning or end of video ||
        sigBased_DropOuts > maxSigBased_DropOut ||
        imageBased_DropOut > maxImageBased_DropOut))
(4)   get next frame t
(5)   if (frame has to be localized)
(6)     localize text in frame t
(7)     search localized text box that matches to the
        box in the last frame of the text object
(8)     if (search successful)
(9)       add text box to the text object
(10)      reset image-based_DropOut
(11)    else
(12)      increment image-based_DropOut
(14)  else
(16)    calculate feature image for frame t
(17)    estimate search area A for the text line
(18)    create a window W with the dimension of the
        text box in frame t-1
(19)    get signature s1 of the text box in t-1
(20)    for (each possible position of W in A)
(21)      calculate signature s2 for W
(22)      calculate error between s2 and s1
(23)      memorize minimal error
(25)    if (minimal error < threshold)
(26)      add text box to the text object
(27)      reset sig-based_DropOut
(28)    else
(29)      increment sig-based_DropOut
```

Fig. 8. Forward text-tracking algorithm of a given text object.

C. Postprocessing

In order to prepare a text object for text segmentation, it is trimmed down to the part which has been detected with high confidence: the first and last frame in which the image-based text localizer detected the text line. Next, all text objects which occur for less than a second or show a drop-out rate of more than 25% are discarded.

The first condition rests on our observation that text lines are usually visible for at least one second; shorter text lines are usually false alarms. The second condition removes text objects resulting from unstable tracking which cannot be handled by subsequent processing. Unstable tracking is usually caused by strong compression artifacts or nontext. Finally, a few attributes are determined for each text object.

1) *Text color*: Assuming that the text color of the same text line does not change over the course of time, a text object's color is determined as the median of the text colors per frame.
2) *Text size*: The size of the text bounding boxes may be fixed or change over time. If they are fixed, we determine the fixed width and height by means of the median over the set of widths and heights.
3) *Text position*: The position of a text line might be static in one or both coordinates. It is static in the x and/or

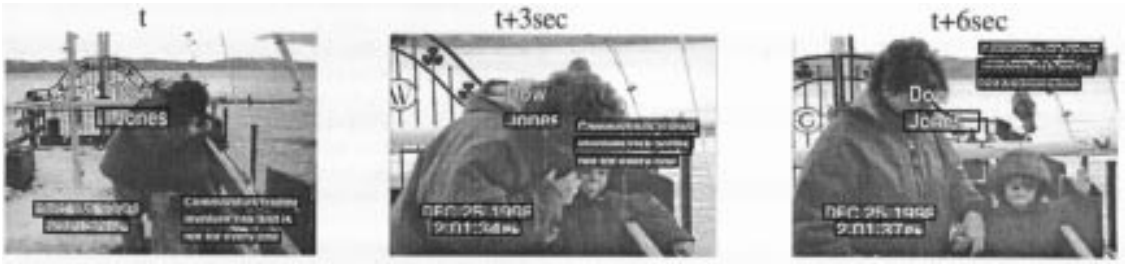


Fig. 9. Example of text tracking of located text lines. All text lines except "Dow" could be successfully tracked. The line "Dow" is missed due to its difficult background (iron gate and face border).

y direction if the average movement per frame is less than 0.75 pixels. The average movement is calculated based on the difference in location between the first and last text occurrence of that text line normalized by the number of frames. If the text line is static, we replace all text bounding boxes by the median text bounding box. The median text bounding box is the box whose left/right/top/bottom border is the median over all left/right/top/bottom borders. If the position is only fixed in one direction such as the x or y axes, the left and right or the top and bottom are replaced by the median value, respectively.

Fig. 9 shows the results of text tracking of located text lines for the sample sequence. All text lines except "Dow" could be successfully tracked. The line "Dow" is missed due to its partially difficult background such as the iron gate and face border. The iron gate's edge pattern is very similar to text in general. It also contains individual characters, thus confusing the image-based text localization system, which in turn renders tracking impossible.

VI. Text Segmentation

A. Resolution Adjustments

All subsequent text segmentation steps are performed on text box bitmaps rescaled by cubic interpolation to a fixed height of 100 pixels, while preserving the aspect ratio for two reasons.

1) Enhancing the resolution of small font sizes leads to better segmentation results. The very low resolution of video is a major source of problems in text segmentation and text recognition. In MPEG-I encoded videos individual characters often have a height of less than 12 pixels. Although text is still recognizable for humans at this resolution, it gives today's standard OCR systems a hard time. These OCR systems have been designed to recognize text in documents, which were scanned at a resolution of at least 200–300 dpi, resulting in a minimal text height of at least 40 pixels. In order to obtain good results with standard OCR systems, it is necessary to enhance the resolution of segmented text lines.

   Enhancing the visible quality of text bitmaps is another and even more important reason for up-scaling small text bitmaps. The higher resolution enables sub-pixel precise text alignment (with respect to the original resolution) in Section VI-B-2 and better text segmentation in general. Similar observations have been made by [19], [10], [11].

2) A text height larger than 100 pixels does not improve segmentation nor OCR performance. Reducing its size lowers the computational complexity significantly. Since our approach is truly multiresolution and also operates on HDTV video sequences up to 1920 by 1280 pixels, larger font sizes are very likely.

B. Removing Complex Backgrounds

1) Images and Web Pages: Text occurrences are supposed to have enough contrast with their background in order to be easily readable. This feature is exploited here to remove large parts of the complex background. The basic idea is to increase a text bounding box such that no text pixels fall onto the border and then to take each pixel on the boundary of the text bounding box as a seed to fill all pixels with the background color which do not differ more than $\mathrm{threshold}_{\mathrm{seedfill}}$. The background color is black for inverse text and white for normal text. Since the pixels on the boundary do not belong to the text and since the text contrasts with its background, the seedfill algorithm will never remove any character pixels. We call this newly constructed bitmap $B^r(x, y)$.

In our experiments, it was necessary to extend each rescaled text box horizontally by 20% and vertically by 40% to ensure that all letters were completely contained in the text bounding box. The Euclidean distance between RGB colors was used as distance function, and the seed fill-algorithm used a 4-neighborhood [2].

Not all background pixels are deleted by this procedure, since the sizes of the regions filled by the seedfill algorithm are limited by the maximum allowed color difference between a pixel and its border pixel. Therefore, a hypothetical 8-neighborhood seedfill algorithm with $\mathrm{threshold}_{\mathrm{seedfill}}$ is applied to each nonbackground pixel in $B^r(x, y)$ in order to determine the dimension of the region that can hypothetically be filled. Background regions should be smaller than text character regions. Therefore, all hypothetical regions with a height less than $\mathrm{min}_{\mathrm{height}}$ pixels and a width less than $\mathrm{min}_{\mathrm{width}}$ or larger than $\mathrm{max}_{\mathrm{width}}$ are deleted, i.e., set to the background color.

2) Videos: Unlike text objects in images, text objects in videos consist of many bitmaps of the same text line in contiguous frames. This redundancy is again exploited here, to remove the complex background surrounding the characters. The basic idea works as follows. Suppose the various bitmaps of a text object are piled up such that the characters are aligned perfectly with each other. Looking through a specific pixel in time, you may notice that pixels belonging to text vary only slightly, while background pixels often change tremendously

through time. Since a text's location is static due to its alignment, its pixels are not supposed to change. Background pixels are very likely to change due to motion in the background or motion of the text line.

Using about 40 temporally evenly spaced bitmaps out of the pile of perfectly aligned bitmaps, a maximum/minimum operator applied through time on the grayscale images for normal/inverse text is generally able separate text pixels from background pixels. This temporal maximum/minimum operation was first proposed by [19] for static text. In our work, however, it is also applied to moving text since we solved the problem of sub-pixel accurate text line alignment as follows.

Likewise for images and web pages, all bounding boxes of a text object are extended horizontally by 20% and vertically by 40%. Next, all bitmaps are converted to grayscale since grayscale is less vulnerable to color compression artifacts. Almost all video compression algorithms represent intensity at a higher resolution than colors. Let $B_0(x, y), \ldots, B_{39}(x, y)$ denote the 40 bitmaps and $B^r(x, y)$ the representative bitmap which is to be derived and which is initialized to $B_0^r(x, y) = B_0(x, y)$. Then, for each bitmap $B_i(x, y)$, $i \in \{1, \ldots, 39\}$, we search for the best displacement $(dx_t^{\text{opt}}, dy_t^{\text{opt}})$, which minimizes the difference between $B_{i-1}^r(x, y)$ and $B_i(x, y)$ with respect to the text color, i.e., see the equation at the bottom of the page.

This partial block-matching search works because only pixels with text color are considered. A pixel is defined to have text color if and only if it does not differ more than a certain amount from the grayscale text color determined for the text object. At each iteration, $B^r(x, y)$ is updated to

$$B_i^r(x, y) = \max\left(B_{i-1}^r(x, y), B_i\left(x + dx_t^{\text{opt}}, y + dy_t^{\text{opt}}\right)\right), \quad \text{for normal text}$$

and to

$$B_i^r(x, y) = \min\left(B_{i-1}^r(x, y), B_i\left(x + dx_t^{\text{opt}}, y + dy_t^{\text{opt}}\right)\right), \quad \text{for inverse text.}$$

Note that if in Section IV-D-3, a text object has been identified to be static, we do not have to search for the best translations. Instead, the translations between the various bitmaps are all set to null. Finally, the same segmentation procedure as described in Section VI-B-1 is applied to $B^r(x, y)$.

C. Binarization

The text bitmap $B^r(x, y)$ is now prepared for recognition by standard OCR engines. Here, the grayscale text bitmaps must be converted to binary bitmaps. From Section IV-D-3, we know the text color, the dominant background color, and whether we have to deal with normal text or inverse text. Since most of the background has been removed, we decided to set the background color to black for inverse text and to white for normal text. Then, an intensity value halfway between the intensity of the text color and the background color is a good binarization threshold. Each pixel in the text bitmap exceeding the binarization threshold is set to white for normal text and black for inverse text. Each pixel in the text bitmap which is lower than or equal to the binarization threshold is set to black for normal text and white for inverse text. Finally, it is recommended that the binary bitmap be cleaned-up by discarding small regions in the same way as described Section VI-B-1.

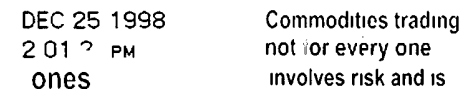


Fig. 10. Segmentation sample results of tracked text lines. The "J" in "Jones" and the "f" in "for" were lost since the upper bounds of the text boxes cut through the "J" and "f," respectively. The seconds of the time of recording were lost since they constantly increment and thus change over the course of the video sequence.

Fig. 10 shows the segmentation results of the sample video sequence. The "J" in "Jones" and the "f" in "for" were lost since the upper bounds of the text boxes cut through the "J" and "f," respectively. The seconds of the time of recording were lost since they constantly increment and thus change over the course of the video sequence.

VII. Experimental Results

Our text-detection and text-segmentation system has been tested extensively on a large, diverse, and difficult set of video sequences. From a video database of tens of hours of home video, newscasts, title sequences of feature films, and commercials, we selected about 23 short video sequences with the objective to be as diverse as possible and to cover difficult—as well as easy—cases for text detection and text segmentation. These 23 video sequences totaled about 10 min, contained 2187 characters, and had a frame size that varied between 352 $\times$ 240 and 1920 $\times$ 1280. In addition, seven web pages were added to the test set.

Text Localization: For each frame and web page, the ground truth for text bounding boxes was created by hand. Two different types of performance numbers were calculated: pixel-based and text box-based performance numbers.

Pixel-based performance numbers calculate the hit rate, false hit rate, and miss rate based on the number of pixels the ground truth and the detected text bounding boxes have in common, as follows:

$$\text{hitrate}_{\text{pixel-based}} = \frac{100}{\sum_{\forall g \subset G} |g|} \cdot \left| \bigcup_{g \in G} \bigcup_{a \in A} a \cap g \right|$$

$$\text{missrate}_{\text{pixel-based}} = 100 - \text{hitrate}_{\text{pixel-based}}$$

$$\text{falschits}_{\text{pixel-based}} = \frac{100}{\sum_{\forall g \subset G} |g|} \cdot \left(\sum_{\forall a \subset A} |a| - \left| \bigcup_{g \in G} \bigcup_{a \in A} a \cap g \right| \right)$$

$$\left(dx_t^{\text{opt}}, dy_t^{\text{opt}}\right) = \arg\min \sqrt{\sum_{(x,y) \in B_{i-1}^r \wedge B_{i-1}^r(x,y) \subseteq \text{textColor}} \left(B_{i-1}^r(x, y) - B_i(x + dx, y + dy)\right)}$$

TABLE I
RESULTS OF TEXT LOCALIZATION

text localization	box-based			pixel-based		
	hit rate	false hits	miss rate	hit rate	false hits	miss rate
Initial bounding boxes (Section 4.4.1)	48.8%	74.0%	51.2%	95.0%	72.7%	5.0%
Refined text bounding boxes (Section 4.4.2)	69.5%	76.3%	30.5%	86.7%	53.5%	13.3%
text tracking (Section 5)	94.7%	18.0%	5.3%	-	-	-

TABLE II
RESULTS OF TEXT SEGMENTATION

character-based		
hit-rate	damaged characters	miss rate
79.6%	7.6%	13.5% (inc. 5.3% from text localization)

TABLE III
RESULTS OF TEXT RECOGNITION

character-based	
hit-rate	miss rate
69.9%	30.1% (inc. 13.5% from text segmentation)

where $A = \{a_1, \ldots, a_n\}$ and $G = \{g_1, \ldots, g_M\}$ are the sets of pixel sets representing the automatically created text boxes and the ground-truth text boxes of size $N = |A|$ and $M = |G|$, respectively, $|a|$ and $|g|$ are the number of pixels in each text box, and $a \cap g$ is the set of joint pixels in a and g.

In contrast, the text box-based hit rate, false hit rate, and miss rate refer to the number of detected boxes that match with the ground truth. An automatically created text bounding box A was regarded as matching a ground truth text bounding box G if and only if the two boxes overlapped by at least 80%

$$\text{hitrate}_{\text{box-based}} = \frac{100}{M} \cdot \sum_{g \in G} \max_{a \in A} \{\delta(a, g)\}$$

$$\text{missrate}_{\text{box-based}} = 100 - \text{hitrate}_{\text{box-based}}$$

$$\text{falsehits}_{\text{box-based}} = \frac{100}{M} \cdot \left(N - \sum_{a \in A} \max_{g \in G} \{\delta(a, g)\} \right)$$

where

$$\delta(a, g) = \begin{cases} 1, & \text{if } \min(|a \cap g|/|a|, |a \cap g|/|g|) \geq 0.8 \\ 0, & \text{else.} \end{cases}$$

On still images such as individual frames or web pages, the text-detection system correctly found 69.5% of all text boxes. The 30.5% that were missed, however, constituted only 13.3% of all text pixels. In other words, small text was more likely to be framed incorrectly. Wu *et al.* similarly observed that the text detection rate falls off drastically for text heights below 10 points—from around 90% down to ~55% [28].

The localization performance could be boosted up to 94.7% by exploiting the temporal redundancy in video sequences as mentioned in Section V. Only 5.3% of the text was missed or falsely framed (see Table I). It took about 0.21 s for color images of size 352 × 240 on a Pentium® 4 at 1.7 GHz to localize the text occurrences.

Text Segmentation: 79.6% of all characters in the test video set (including the ones lost during text localization) were binarized correctly. Correctness was determined by manual visual inspection of all created binary bitmaps. As shown in Table II, 7.6% of the characters were damaged (e.g., some parts were missing) but still recognizable by humans. Considering that 5.3% of the text was not even detected, only 7.2% of the characters were lost in the segmentation stage.

For MPEG-1 video (352 × 240), our text localization and text segmentation operated at 10.6 fps on a Pentium® 4 at 1.7 GHz.

Text Recognition: We used OmniPagePro 10 from Caere to evaluate the overall performance of the system. 87.8% (=(69.9/79.6) · 100) of the correctly segmented characters were also recognized correctly. Over all stages, i.e., text localization, text segmentation, and text recognition, 69.9% of all characters were recognized correctly (see Table III).

Many times, individual characters within words were missing in the input binary bitmaps. Therefore, a higher recognition rate could be achieved if a content specific dictionary were used during the recognition.

Finally, we want to point out that although several threshold values have to be chosen properly, the whole system was not very sensitive to their precise choice. Most threshold values could be chosen from a large range of values without really changing the overall performance of the system. For instance, over-detection as well as under-detection of the fixed-scale text detector was compensated by the text bounding box refinement procedure.

A very detailed experimental analysis of our system on how the different processing step affect the hit and false hit rate can be found in [25].

VIII. CONCLUSION

Text in video is a unique and useful source of information about its content. Text localization, segmentation, and recognition in videos allow extraction of this information for semantic indexing. We have presented a very generic, scale-invariant solution for text localization and text segmentation in images and videos. On a difficult, real-world test set of video frames and web pages, 69.5% of all text boxes were located correctly. The performance rose to 94.7% by exploiting the temporal redundancy in videos. 79.6% of all characters in the test video set could be segmented properly and 88% of them were recognized correctly. These performance numbers are above the ones reported for existing systems, albeit the test set has been very challenging.

In addition, our novel text segmentation method is not only able to locate and segment text occurrences into binary bitmaps,

but also to track each text line sub-pixel accurate over the entire occurrence in a video, so that one text bitmap is be created for all instance of that text line. Thus, our text-detection and text-segmentation methods can be used for object-based video encoding. We have started initial experiments to encode segmented text as MPEG-4 objects and the reminder as background. Initial results are very promising, but need further investigations.

For the future, we also plan to replace the globally adaptive threshold used in the binarization of the text bounding boxes to a locally adaptive threshold. Locally adaptive thresholds are well known to improve segmentation in OCR applications.

Acknowledgment

The authors would like to thank NSTL for the wonderful MPEG Library that made this work possible and which decodes MPEG videos incredibly fast.

References

[1] E. Clan, A. Rodriguez, R. Gandhi, and S. Panchanathan, "Experiments on block-matching techniques for video coding," *Multimedia Systems*, vol. 2, no. 5, pp. 228–241, Dec. 1994.

[2] J. D. Foley, A. Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*. Reading, MA: Addison-Wesley, 1990.

[3] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, pp. 1254–1259, 1998.

[4] B. Jaehne, *Practical Handbook on Image Processing for Scientific Applications*. Boca Raton, FL: CRC Press, 1997.

[5] O. Hori, "A video text extraction method for character recognition," in *Proc. 5th Int. Conf. Document Analysis and Recognition (ICDAR)*, 1999, pp. 25–28.

[6] A. K. Jain and B. Yu, "Automatic text location in images and video frames," *Pattern Recognit.*, vol. 31, no. 12, pp. 2055–2076, 1998.

[7] R. Lienhart, "Automatic text recognition for video indexing," in *Proc. ACM Multimedia 96*, Boston, MA, Nov. 1996, pp. 11–20.

[8] R. Lienhart, C. Kuhmünch, and W. Effelsberg, "On the detection and recognition of television commercials," in *Proc. IEEE Conf. Multimedia and Systems*, Ottawa, Canada, June 1997, pp. 509–516.

[9] R. Lienhart and W. Effelsberg, "Automatic text segmentation and text recognition for video indexing," *Multimedia Syst.*, vol. 8, pp. 69–81, Jan. 2000.

[10] H. Li, D. Doermann, and O. Kia, "Automatic text detection and tracking in digital video," *IEEE Trans. Image Processing*, vol. 9, pp. 147–156, Jan. 2000.

[11] H. Li, O. Kia, and D. Doermann, "Text enhancement in digital videos," in *Proc. SPIE99—Document Recognition and Retrieval*.

[12] P. H. Lindsay and D. A. Norman, *Introduction Into Psychology—Human Information Reception and Processing* (in German). Berlin, Germany: Springer-Verlag, 1991.

[13] D. Lopresti and J. Zhou, "Locating and recognizing text in WWW images," *Info. Retrieval*, vol. 2, pp. 177–206, May 2000.

[14] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[15] T. Masters, *Signal and Image Processing With Neural Networks*. New York: Wiley, 1994.

[16] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical review of OCR research and development," *Proc. IEEE*, vol. 80, pp. 1029–1058, July 1992.

[17] S. Pfeiffer, R. Lienhart, S. Fischer, and W. Effelsberg, "Abstracting digital movies automatically," *J. Vis. Comm. Image Represent.*, vol. 7, no. 4, pp. 345–353, Dec. 1996.

[18] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, pp. 23–38, Jan. 1998.

[19] T. Sato, T. Kanade, E. Hughes, M. Smith, and S.-i Satoh, "Video OCR: Indexing digital news libraries by recognition of superimposed caption," *Multimedia Syst.*, vol. 7, no. 5, pp. 385–395, 1999.

[20] T. Sato, T. Kanade, E. K. Hughes, and M. A. Smith, "Video OCR for digital news archives," in *Proc. IEEE Int. Workshop on Content-Based Access of Image and Video Database (CAVID'98)*, 1998, pp. 52–60.

[21] M. A. Smith and T. Kanade, "Video skimming for quick browsing based on audio and image characterization," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-95-186, July 1995.

[22] K.-K. Sung, "Learning and example selection for object and pattern detection," Ph.D. dissertation, MIT AI Lab, AI Tech. Rep. 1572, Cambridge, MA, Jan. 1996.

[23] M. J. Swain and D. H. Ballard, "Color indexing," *Int. J. Comput. Vis.*, vol. 7, no. 1, pp. 11–32, 1991.

[24] M. Takatoo *et al.*, "Gray scale image processing technology applied to vehicle license number recognition system," in *Proc. Int. Workshop Industrial Applications of Machine Vision and Machine Intelligence*, 1987, pp. 76–79.

[25] A. Wernicke, "Text localization and text segmentation in images, videos and web pages," M.S. thesis, Univ. of Mannheim, Mannheim, Germany, Mar. 2000.

[26] H. D. Wactlar, M. G. Christel, Y. Gong, and A. G. Hauptmann, "Lessons learned from building a terabyte digital video library," *IEEE Computer*, vol. 32, pp. 66–73, 1999.

[27] V. Wu, R. Manmatha, and E. M. Riseman, "Finding text in images," in *Proc. 2nd ACM Int. Conf. Digital Libraries*, Philadelphia, PA, July 1997, pp. 23–26.

[28] V. Wu, R. Manmatha, and E. Riseman, "TextFinder: An automatic system to detect and recognize text in images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 21, no. 11, Nov. 1999.

[29] X. Wu, "YIQ vector quantization in a new color palette architecture," *IEEE Trans. Image Processing*, vol. 5, pp. 321–329, Feb. 1996.

[30] Y. Zhong, K. Karu, and A. K. Jain, "Locating text in complex color images," *Pattern Recognit.*, vol. 28, pp. 1523–1535, Oct. 1995.

Rainer Lienhart (S'97–A'98–M'95) received the M.S. degree in computer science and applied economics and the Ph.D. degree in computer science from the University of Mannheim, Mannheim, Germany. His dissertation was on methods for content analysis, indexing, and comparison of digital video sequences.

He was a core member of the Movie Content Analysis Project (MoCA). Since 1998, he has been a Staff Researcher at Intel Labs, Santa Clara, CA. His research interests include image/video/audio content analysis, machine learning, scalable signal processing, scalable learning, ubiquitous and distributed media computing in heterogeneous networks, media streaming, and peer-to-peer networking and mass media sharing.

Dr. Lienhart is a Member of the IEEE Computer Society.

Axel Wernicke received the M.S. degree in computer science and applied economics from the University of Mannheim, Mannheim, Germany.

During the summer of 1999, he was an Intern at Intel Labs, Santa Clara, CA. Currently, he is an Internet Business Consulting at Forcont Business Technology Gmbh, Leipzig, Germany.