The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

EXTRACTION OF UNCONSTRAINED CAPTION TEXT FROM GENERAL-PURPOSE VIDEO

A Thesis in

Computer Science and Engineering

by

David J. Crandall

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2001

We approve the thesis of David J. Crandall.

Date of Signature

Rangachar Kasturi
Professor of Computer Science and Engineering
Thesis Adviser

Lee D. Coraor
Associate Professor of Computer Science and Engineering

Dale A. Miller
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

Abstract

The popularity of digital video is increasing rapidly. To help users navigate libraries of video, algorithms that automatically index video based on content are needed. One approach is to extract text appearing in video. Such text often gives an indication of a scene's semantic content. This is a more difficult problem than recognition of document images due to the unconstrained nature of general-purpose video. Text can have arbitrary color, size, and orientation. Backgrounds may be complex and changing.

Most work so far has made restrictive assumptions about the nature of text occurring in video. Such work is therefore not applicable to unconstrained, general-purpose video. Also, most work so far has focused only on detecting the spatial extent of text in individual video frames. But text occurring in video usually persists for several seconds. This constitutes a text event that should be entered only once in the video index. Therefore it is also necessary to determine the temporal extent of text events. This is a non-trivial problem because text may move, rotate, grow, shrink, or otherwise change over time. Such text effects are common in television programs and commercials to attract viewer attention, but have so far been ignored in the literature.

This thesis discusses the problems involved in extracting caption text from unconstrained, general-purpose MPEG-1 video. These problems include localizing text in individual video frames, binarizing text, and tracking text as it moves and changes over time. Solutions are proposed for each of these problems and compared with existing work found in the literature.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am indebted to Dr. Rangachar Kasturi for introducing me to the field of Computer Vision early in my academic career. His wholehearted support, gentle encouragement, and constant (though often not deserved) patience have let me achieve far more than I ever thought possible. I would also like to thank my undergraduate advisor, Dr. Lee Coraor, for his support throughout both my undergraduate and graduate studies. I thank the members of the Computer Vision Lab, especially Sameer Antani, Ullas Gargi, Vlad Mariano, Anand Narasimhamurthy, and JinHyeong Park. Their insight has enriched my work; their friendship has enriched my life.

I would like to thank my parents, Ron and Sally, for constantly and selflessly supporting me in everything that I do. Just as their comments have improved the drafts of this thesis, their role models will always help me be the best person that I can be. I could not ask for more perfect parents. Finally, I wish to thank Shawna Daigle for her love and support, and for always helping me to remember that there really is life outside the Vision Lab.

Chapter 1

Introduction

1.1 Motivation for intelligent video indexing

The popularity of digital video is growing at an explosive rate. Hundreds of television stations are now broadcast over digital cable every day. Digital Versatile Discs (DVDs) are quickly replacing analog video tape as the preferred medium for viewing movies at home. Inexpensive video capture cards and plummeting data storage costs are allowing users with even modest workstations to convert home movies to digital form. Surveillance cameras are everywhere, capturing video for detection of suspicious activity. Streaming video clips are becoming increasingly popular on the Internet.

The rapid rise in quantities of digital video carries enormous promise. Given such huge amounts of video data available, it is quite probable that a video clip that a user wants to see exists somewhere in digital form. One can imagine large video databases available on the Internet that would give users access to vast quantities of video data from their home personal computers.

But as quantities of available video data grow, it will become increasingly difficult for users to locate specific video clips of interest. It is analogous to the proverbial problem of finding a needle in an ever-growing haystack of video data. Search engines are required that can automatically identify relevant video clips based on a user's query.

However, the current state-of-the-art in video search technology is quite limited. The video search engines of Lycos and Altavista exemplify the current technology available on the Internet. Lycos [51] requires humans to manually index each video by identifying keywords that describe its content. User queries are matched against this keyword index to find relevant video clips. This approach is clearly intractable for large, growing video libraries because of the large amount of effort required to create video indices by hand. Also, the quality of the search engine is directly limited by the quality and scope of the manually-created index. It is impossible for the human indexer to identify all possible keywords that describe a given video sequence.

Altavista's video search engine [2] attempts to index videos contained in World Wide Web pages automatically. Words near a video in a web page are assumed to describe the content of the video and are used as its keywords. This approach eliminates the dependence on human indexers, but the assumption that words appearing near a video are appropriate keywords is not true in general. This can cause irrelevant words to be placed into the keyword index, and decrease search result quality. Altavista's approach cannot be used unless a textual description is available. It is therefore not applicable to general-purpose video.

Clearly, better video search technologies are required. Algorithms must be developed that can automatically extract semantic information from video using *content* alone. Given an arbitrary video sequence, such algorithms would determine as much information as possible, such as genre (sitcom, movie, sports program, etc.), filming location characteristics (indoor or outdoor, time of day, weather conditions, etc.), identity of important objects, identity of people (politicians, movie stars, sitcom characters, etc.), and human activity and interaction (running, laughing, talking, arguing, etc.). This wealth of information could be used to better identify video sequences of interest to a user.

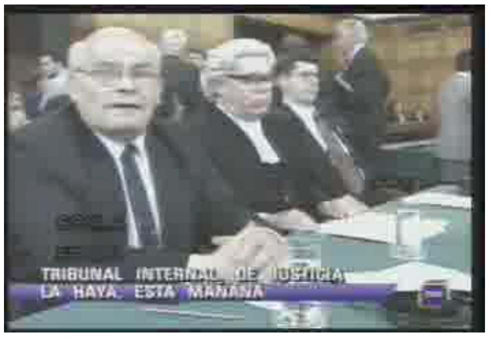
Automatically extracting this information from unconstrained video is very challenging. Solving the underlying computer vision and artificial intelligence problems will undoubtedly occupy these research communities for many years.

1.2 Motivation for extracting text from video

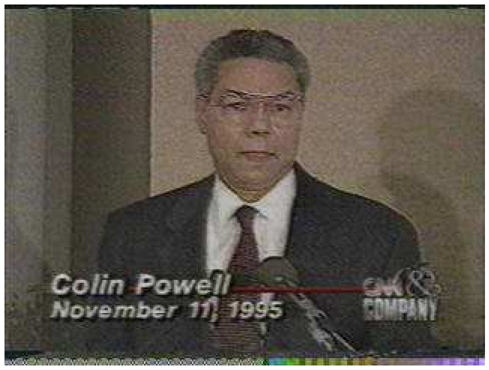
In addition to the features mentioned above, text appearing in a video sequence can provide useful semantic information. Text occurring in video naturally gives clues to the video's content. Words have well-defined, unambiguous meanings. If the text in a video sequence can be extracted, it can provide natural, meaningful keywords indicating the video's content.

Text occurring in video can be classified as caption text or scene text. *Caption text* is artificially superimposed on the video at the time of editing. Caption text usually underscores or summarizes the video's content. This makes caption text particularly useful for building a keyword index. Figure 1.1 presents some examples of caption text.

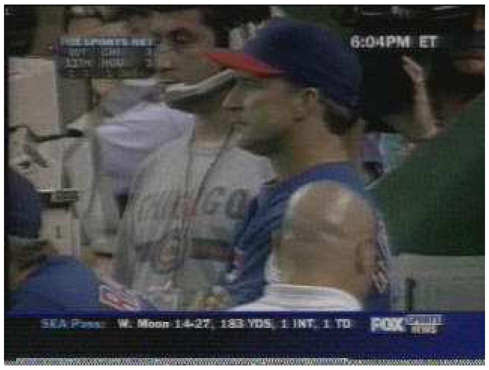
Scene text naturally occurs in the field of view of the camera during video capture. Figure 1.2 presents examples of scene text occurring in video frames. Scene text occurring on signs, banners, etc. gives natural indications as to the content of a video sequence.



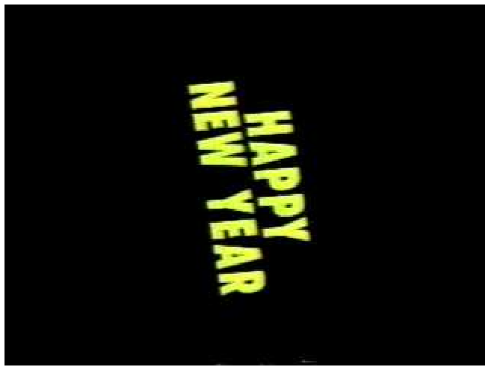
(a)



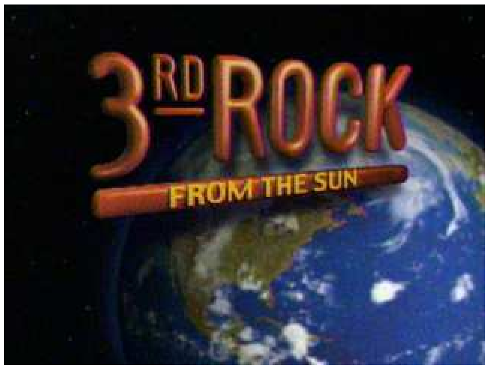
(b)



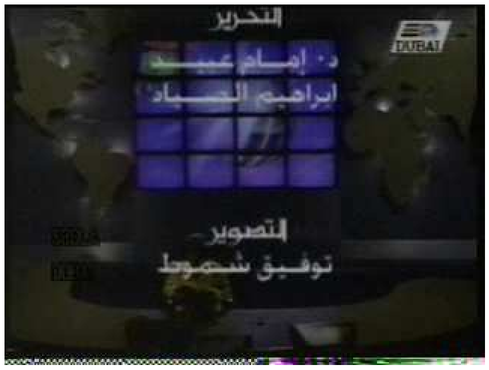
(c)



(d)

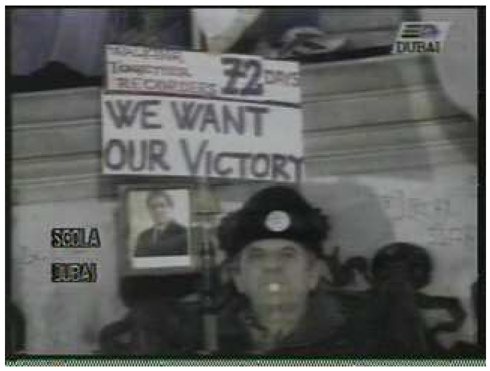


(e)

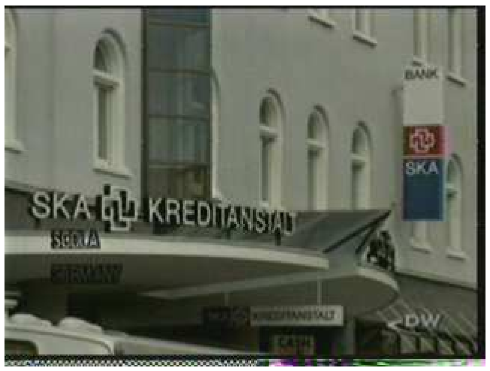


(f)

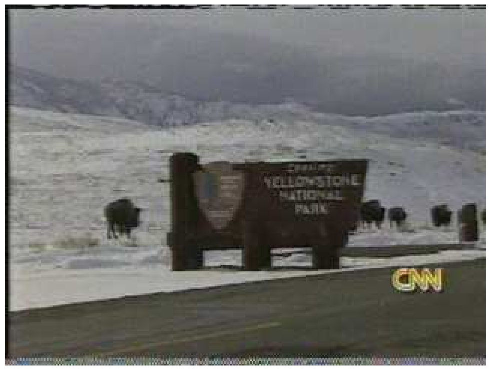
Fig. 1.1. Examples of caption text indicating the semantic content of video.



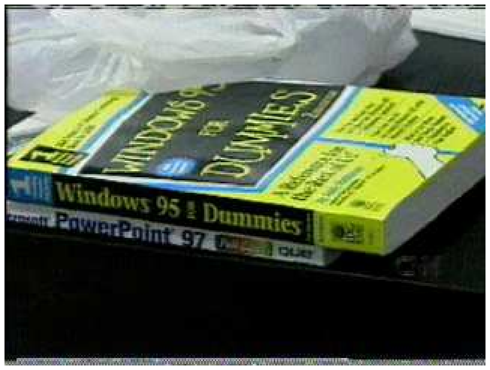
(a)



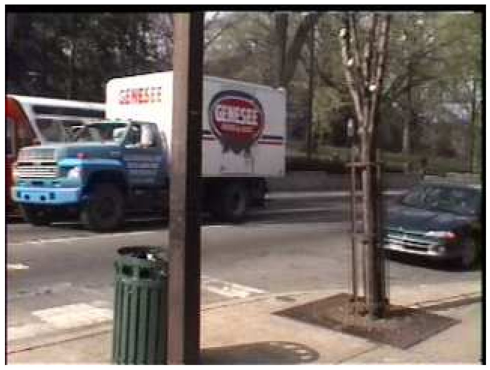
(b)



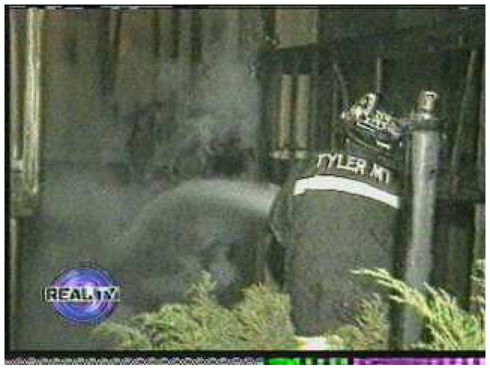
(c)



(d)



(e)



(f)

Fig. 1.2. Examples of scene text indicating the semantic content of video.

1.3 Differences between video text extraction and document OCR

Optical character recognition (OCR) of document images has been studied extensively for decades [35]. Technology has evolved to nearly solve the document OCR problem. Recognition accuracy rates higher than 99% are now achievable.

However, extraction of text from video presents unique challenges over OCR of document images. Document images are usually scanned at high resolutions of 300 dots per inch or higher. In contrast, video frames are usually digitized at much lower resolutions, typically 640×480 or 320×240 pixels for an entire frame. In addition, lossy compression schemes are usually applied to digital video to keep storage requirements reasonable. Video frames therefore suffer from color bleeding, loss of contrast, blocking artifacts, and other noise that significantly increases the difficulty of accurately extracting text.

Many characteristics of the text in a document image are known *a priori*. For example, the text color in a document is nearly always black, and the background is known to be uniform white. There is high contrast between the background color and the text color. The orientation of the text can be assumed to be horizontal, or can easily be inferred by analyzing the structure of the document. In contrast, text in video can have arbitrary and non-uniform stroke color. The background may be non-uniform, complex, and changing from frame to frame. The contrast between the background and foreground may be low. Text size, location, and orientation are unconstrained.

The temporal nature of video introduces a new dimension into the text extraction problem. Text in video usually persists for at least several seconds, to give human viewers the necessary time to read it. Some text events remain unchanged during their lifetimes. Others, like movie credits, move in a simple, rigid, linear fashion. Still others, like scene text and stylized caption text, move and change in complex ways. Text can grow or shrink, or character spacing can increase or decrease. Text color can change over time. Text can rotate and change orientation. Text can morph from one font to another. Text strings can break apart or join together. Special effects or a moving camera can cause changing text perspective.

The problem of text extraction from video is therefore significantly more difficult than the document image OCR problem. It is possible to simplify the problem by making *a priori* assumptions about the type of video, or to extract only certain types of text. However, in a general-purpose video indexing application, it is important to be able to extract as much text as possible. Therefore text extraction systems must be applicable to general-purpose video data and must be able to handle as many types of text as possible.

1.4 Problem statement and scope of this thesis

This thesis discusses the extraction of unconstrained caption text from general-purpose video. In particular, it addresses the extraction of types of text that have largely been ignored by the work in the literature to date. These types of caption text include moving text, rotating text, growing text, shrinking text, text of arbitrary orientation, and text of arbitrary size. The focus of this work is on extraction of caption text, although much of the work could be applied to extracting scene text as well.

Text extraction from video can be divided into the following subproblems:

- **Detection:** The *text detection* problem involves locating regions in a video frame that contain text.
- **Localization:** *Text localization* groups the text regions identified by the detection stage into text instances. The output of a good localization algorithm is a set of tight bounding boxes around each text instance.
- **Tracking:** The *text tracking* problem involves following a text event as it moves or changes over time. Together, the detection, localization, and tracking modules determine the temporal and spatial locations and extents of text events.
- **Binarization:** The *text binarization* problem involves separating text strokes from the background in a localized text region.[1] The output of a binarization module is a binary image, with pixels corresponding to text strokes marked as one binary level and background pixels marked as the other.
- **Recognition:** The final stage is the *text recognition* problem, in which the text appearing in the binarized text image is recognized. I do not discuss the recognition problem in this thesis. It is assumed that once text has been binarized, any of the many commercial document image OCR systems could be used for the recognition stage.

[1]In previous publications (e.g. [3, 11]) we used the term *segmentation* to refer to the binarization problem. We used it in the context of segmenting individual text pixels from background pixels. Unfortunately this term is used inconsistently in the text extraction literature. Some authors (e.g. [6]) use this term to refer to the text *region* segmentation problem. Others (e.g. [22]) use it to refer to the *character* segmentation problem, in which individual characters are located. To avoid confusion, I will avoid the term *segmentation* in this thesis.

This thesis discusses the text detection, tracking, and binarization problems, and presents the results of work toward their solutions. Chapter 2 describes the text detection and localization problems. After a review of previous work in this area, two algorithms are presented for detecting and localizing text in video frames. One of the algorithms assumes that text is horizontal and within a size range; the other removes both of these restrictions. A quantitative performance evaluation is performed to compare these algorithms with others in the literature. In Chapter 3, the text tracking problem is discussed. A tracking algorithm is presented that tracks rigid text events using MPEG motion vectors for speed and robustness. A second tracking algorithm is presented that removes the rigidity constraint, allowing for growing, shrinking, and rotating text to be tracked. In Chapter 4, the binarization problem is discussed. A binarization algorithm is presented that makes few assumptions about the nature of the text. It is designed to work with text of arbitrary color appearing against complex backgrounds. Outputs from this algorithm are compared to outputs from another binarization algorithm in the literature. Finally, conclusions are drawn and areas for future work are identified in Chapter 5.

Chapter 2

Detection and localization of unconstrained caption text

2.1 Introduction

A digital video is a sequence of still images, displayed rapidly to give the illusion of continuous motion. Locating text in video therefore begins with locating text in images. This chapter considers the problem of identifying text regions in images and video frames.

The process of identifying text regions can be split into two subproblems: detection and localization. In the detection step, general regions of the frame are classified as text or non-text. The size and shape of these regions differ from algorithm to algorithm. For example, some algorithms classify 8×8 pixel blocks, while others classify individual scan lines. In the localization step, the results of detection are grouped together to form one or more text instances. This is usually represented as a bounding box around each text instance.

The remainder of the chapter discusses the detection and localization problems. In Section 2.2, I give a survey of related work in the literature to date. In Section 2.3, I present a fast algorithm for detecting and localizing horizontal caption text in MPEG video. Section 2.4 extends this work to allow detection and localization of oriented text and to improve accuracy. Finally, the results of a performance evaluation of this and other detection algorithms in the literature are presented in Section 2.5.

2.2 Review of prior text detection and localization work

This section reviews past work in locating text in individual images and video frames. I have classified these existing algorithms into five categories according to their basic underlying approaches. Each type of approach is reviewed in the following sections.

2.2.1 Edge-based text localization

Text tends to have complex shapes and high contrast with the background. The algorithms in this category exploit this by looking for edges in the image. Alignment, size, and orientation features of the edges are used to discriminate text regions from other "edgy" portions of an image.

- LeBourgeois [22] localizes text in complex grayscale images. After pre-processing, image gradients are smeared in the horizontal direction. Connected components are found in the resulting image to localize text regions into text lines. Text lines are further segmented into individual characters by locating valleys in the horizontal and vertical projection profiles.

- Sato *et al* [45] localize caption text in news broadcasts by looking for areas of edge pixels that satisfy aspect ratio and other criteria. Text is assumed to be light-colored, appear over a dark background, and have horizontal orientation.

- Agnihotri and Dimitrova [1] detect horizontal white, yellow, and black caption text in video frames. A pre-processing step enhances edges and removes salt-and-pepper noise. Edge pixels are found using a kernel and a fixed threshold. Frame regions with very high edge density are considered too noisy for text extraction and are disregarded. Connected components are found in the edge pixels of remaining regions. Edge components are merged based on size, spacing, and alignment heuristics to produce the localization result. This algorithm relies on many fixed thresholds. It appears too restrictive and fragile for use in general-purpose video.

- Garcia and Apostolidis [9] locate horizontal text in color images. Edge pixel magnitudes and locations are determined in each color plane. Text regions are selected by identifying areas with high edge density and high variance of edge orientation. This prevents incorrect identification of regions with "simple" edges uncharacteristic of text. Morphological operations are performed to remove singletons and non-horizontal regions. Localization is performed by finding connected components. Candidate text regions are joined together or split apart based on the geometric constraints of horizontal text.

- Qi *et al* [43] extract captions from news video sequences. Horizontal and vertical edge maps for a video frame are determined using a Sobel operator. Alignment of edges is analyzed to find horizontally-oriented text instances. Sample results shown in the paper are quite noisy, suggesting that the algorithm is unsuitable for general-purpose video.

2.2.2 Stroke-based text localization

Text is usually composed of strokes of uniform width and color. Algorithms in this category look for pixel runs of similar color that may correspond to character strokes.

The distinction between edge- and stroke-based localization techniques is similar to the distinction between edge- and region-based image segmentation [18].

- Ohya *et al* [38] threshold gray level images and localize text regions by looking for strokes of high contrast, uniform width, and uniform gray level. An OCR stage is used to validate the detection. If a localized region cannot be recognized with high confidence by the OCR module, it is discarded.

- Lee *et al* [24] locate vertical and horizontal runs of pixels in a quantized gray scale image. Runs having high contrast with neighboring pixels are assumed to lie on the boundary of a text instance. Connected segments are merged to form character candidate regions. Post-processing removes non-characters based on size, aspect ratio, and contrast heuristics. Special consideration is given to differentiate "1" and "l" characters from solid non-text connected components. The algorithm is tested on images of identification numbers appearing on the sides of railroad boxcars.

- Lienhart [28] applies the split-and-merge image segmentation technique [14] to locate text in video frames. Local color variance in the R'G'B' space [41] is used as the homogeneity criteria for segmentation. Segmented regions are chosen based on text-like size, spacing, and contrast heuristics. Example detection results shown in the paper show many false alarms. This is mitigated by a custom OCR module that discards candidate regions that cannot be recognized with a reasonable confidence. Inter-frame analysis is performed to eliminate regions of noise persisting for just a single frame.

- Shim *et al* [48, 49] propose a method to detect caption text in video frames. Regions with homogeneous intensity are identified, positive and negative images are formed by double thresholding, and heuristics are applied to eliminate non-text regions. Text is assumed to be either black or white. Inter-frame analysis is performed for added robustness.

- Gargi *et al* [10] describe an algorithm for locating horizontal text strings in video frames. Their method looks for horizontal streaks of similar color that may correspond to character strokes. Size and aspect ratio heuristics are applied to reduce false alarms.

2.2.3 Local texture-based localization

Algorithms in this category examine local texture features within small regions of an image. Text is assumed to have a distinct texture. If the texture features are consistent with the characteristics of text, all pixels in the region are marked as text.

- Wu *et al* [57] describe a scheme for finding text in images. Texture segmentation is used to locate potential text regions. Edge detection is then applied to find candidate text strokes, which are merged to form text regions. Their algorithm is tested against a dataset of images with text appearing on relatively simple backgrounds.

- Schaar-Mitrea *et al* [46] propose an algorithm to find overlaid text and graphics in video frames. Blocks of size 4×4 pixels are examined. The number of pixels within the block having similar gray levels is counted. If this count is greater than a threshold, and if the dynamic range of the block is found to be less than a threshold or greater than another threshold, the block is classified as text.

- Wong [56] locate text in the luminance plane of a video frame. A 1×21 pixel window is passed over the image, and the difference between the maximum and minimum gradients within the window are determined. Gradient zero-crossings are found and the mean and variance between zero-crossings are computed. Pixels under the window are marked as text if the gradient difference is high, the variance is low, and the mean is within a reasonable range. These text lines are merged together into localized text regions.

2.2.4 Color clustering-based localization

Algorithms in this category try to simplify image content by performing color clustering. The assumption is that text pixels and the background will fall into separate color clusters. Features of the clustered image are examined to locate text regions.

- Jain and Yu [17] presents a method to locate text in pseudo-color images on the web, full color images, and color video frames. Quantization and color clustering are performed in the RGB color space. It is assumed that the largest color cluster is the background (non-text) region and the other clusters represent text. Connected components in the foreground colors are found and are grouped together into text lines using alignment, spacing, and projection profile heuristics. The example video images shown in the paper are relatively simple, yet the algorithm inexplicably misses several prominent text instances. It is not clear that the assumption that all background pixels are clustered together is true for unconstrained video.

- Mariano [31] performs simultaneous detection and binarization of horizontal text regions by performing color clustering on individual scan lines. Streaks on adjacent scanlines belonging to the same color cluster are assumed to be character strokes. The algorithm fails if text is even slightly oriented off of horizontal. It gives good results for horizontal text, but its large computation cost makes it prohibitive.

2.2.5 Neural-network based localization

Some researchers have applied neural networks to the problem of detection and localization of text regions. Two sample papers are mentioned here.

- Jeong *et al* [19] apply neural networks to find text captions in Korean news broadcasts. Detection is performed on sub-sampled images in a hierarchical fashion to detect text of different sizes. Character spacing, text line spacing, horizontal alignment, and aspect ratio heuristics are applied in post-processing.

- Li *et al* [27] apply wavelets and a neural network to find text. A window of 16×16 pixels is passed over the image. The wavelet transform of the pixels under the window is taken, and moments of it are used as input into a neural network classifier. If the classifier indicates a text region, all pixels under the window are marked as text. A horizontal bounding box is determined for each connected component of text pixels. This process is repeated on different scales to allow detection of text of different sizes.

2.2.6 Observations on text detection literature to date

Unfortunately, it is difficult to determine the performance of detection and localization algorithms presented in the literature just by reading the papers. Many of the "experimental results" sections of the above papers consist simply of the proposed algorithm applied to a few sample images. It is impossible to know whether the sample outputs represent the typical performance of the algorithm, or if carefully-selected result images have been presented. None of the above papers perform a comparative performance evaluation. Some present an absolute quantitative performance evaluation, but because no standard test dataset has been adopted, it is impossible to compare them. To address this issue, we have carried out a performance evaluation of several of the most promising algorithms [5]. This evaluation is discussed in detail in Section 2.5.

Based on this evaluation and information presented in the papers, I observe the following about the work in the literature to date. Many algorithms make *a priori* assumptions about the text to be extracted (e.g. strong restrictions on text color, size, location, etc.). This makes them unsuitable for use on general-purpose, unconstrained video. Other algorithms work well on images with relatively simple backgrounds, but give high false alarm rates when applied to complex images. Most algorithms have high computation costs. No algorithm so far detects oriented (non-horizontal) text.

The observation that different algorithms make different assumptions about the nature of text in video sparked the idea that outputs of multiple algorithms could be combined to give a more accurate output than any individual algorithm. Please see [4] and [11] for a discussion of our work in this area. Detection algorithm fusion is not further discussed in this thesis.

In the next section, a computationally-efficient algorithm is presented that uses the DCT coefficients of MPEG video frames to detect text. In Section 2.4, another DCT-based algorithm is presented that can detect text with non-horizontal orientation. In Section 2.5, these algorithms are compared with others in the literature using a quantitative performance evaluation.

2.3 Algorithm A: A DCT-based algorithm for caption text detection

2.3.1 Chaddha94

Chaddha [6] proposed the following simple algorithm for discriminating text regions from non-text regions in document images. First, the block-wise Discrete Cosine Transform (DCT) is performed on the image. A block size of 8x8 pixels was used. In each block, the sum of the absolute values of a subset of DCT coefficients is computed to give an energy value for the block. The optimal subset of DCT coefficients were empirically determined to be coefficients 3, 4, 5, 11, 12, 13, 19, 20, 21, 43, 44, 45, 51, 52, 53, 59, 60, and 61, in row-major order. This energy is a simple measure of local texture. If the energy is greater than a threshold, the block is declared to contain text. Otherwise, it is marked as a non-text block.

Chaddha's application was detecting text regions in JPEG-compressed images of documents. In a performance comparison with detection schemes using other image features, the DCT-based method was found to give the most accurate results. It was also found to be the most computationally efficient. Explicitly performing the DCT transform is not required because the coefficients are already available in JPEG-compressed images.

The method uses a fixed threshold parameter. This is possible in a document image application because it is known *a priori* that the non-text regions are relatively smooth and have low texture energy. There is a wide gap between the texture energy of non-text blocks and text blocks in document images. The algorithm is therefore not very sensitive to the threshold value. An optimal threshold empirically determined on one document image dataset is likely to give good results on another dataset of document images.

2.3.2 Application to video frames

I applied the DCT-based text detection approach described above to intra-coded (I) frames of MPEG-1 video sequences. Like JPEG images, I-frames are encoded using the block-wise DCT transform, so the DCT coefficients are available in the bit stream.

Detecting text in broadcast video frames is much more difficult than in images of documents. In unconstrained video frames, non-text regions may be quite complex and have high texture energy. The gap between the texture energy of text and non-text blocks is small.

Experimentation showed that the DCT text detection method gave acceptable results on video frames once an appropriate threshold was chosen. Unfortunately, the optimal threshold value varied widely from frame to frame. Figure 2.1 illustrates this sensitivity to the choice of threshold. Two sample video frames are shown in images (a) and (b). The optimal threshold for each image was determined empirically by exhaustively testing all possible thresholds and choosing the one that minimized the difference between precision and recall (see Section 2.5 for details on the evaluation criteria). The optimal thresholds were found to be 310 for image (a) and 102 for image (b). Acceptable results are produced when the optimal thresholds are used, as shown in images (c) and (f). But the lower threshold produced many false alarms for image (a), and the higher threshold caused many missed text blocks when applied to image (b). Further, even in the outputs obtained using the optimal thresholds, a high false alarm rate is observed.

2.3.3 Region-growing to increase detection accuracy

I observed that the blocks with the highest DCT texture energy in a frame usually belong to text regions. Also, at least one block in each text region has very high energy. These observations inspired a region-growing scheme that decreases the algorithm's reliance on fixed thresholds.

Region growing is carried out in the following manner. First, blocks with DCT energy above some threshold T_h are marked as text. A threshold variable T is initialized to T_h. Then the following is performed iteratively. T is decremented by some step value ΔT. Blocks with thresholds above T are marked as text if at least one of their 8-neighbors has already been marked as text in an earlier iteration. Iteration continues until T reaches some low threshold T_l. By experimentation, I found $T_h = 150$, $T_l = 30$, and $\Delta T = 10$ worked well.

Region-growing improves detection accuracy in two ways. First, it suppresses false alarms, because regions may only grow around "seed" blocks of high energy that

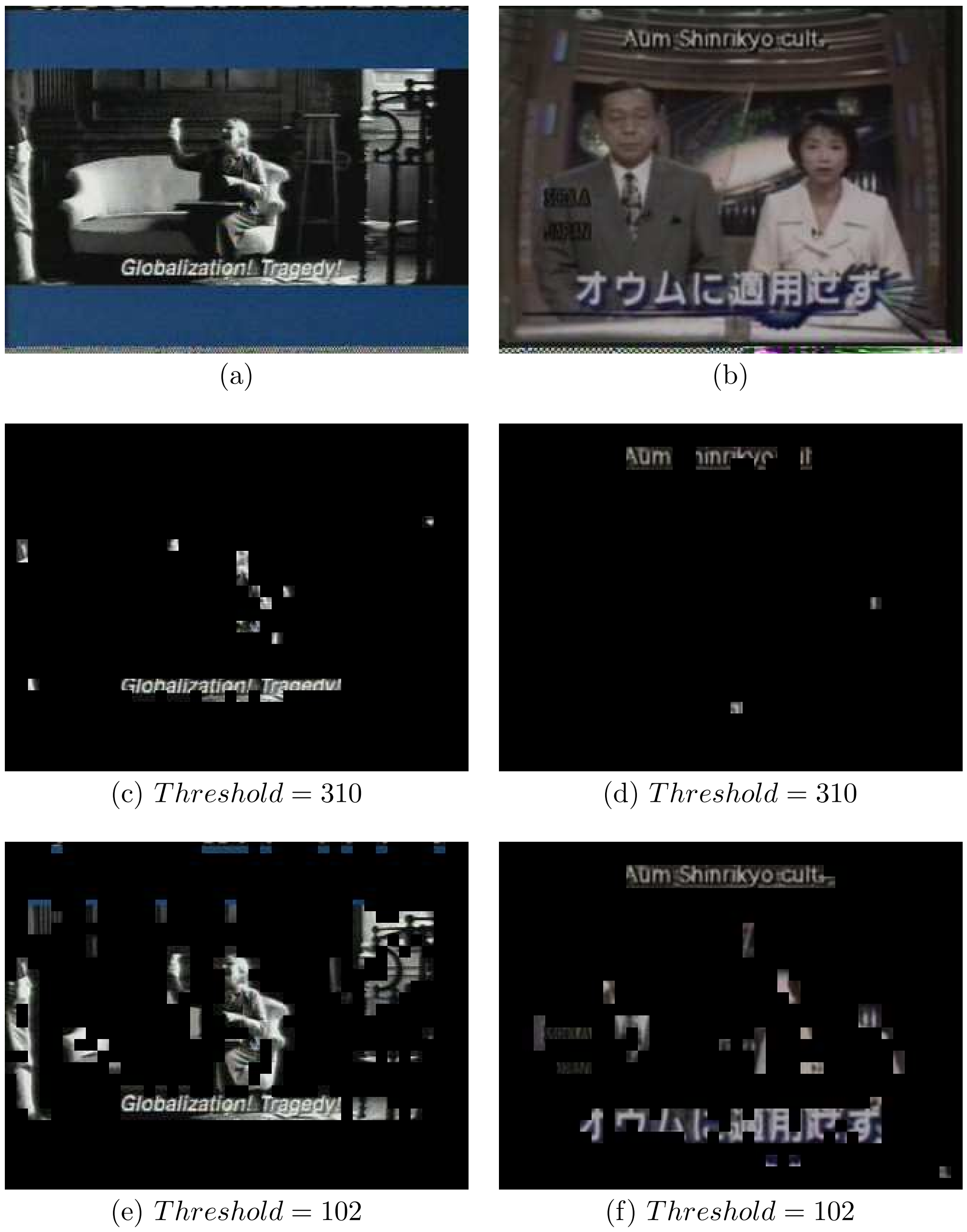


Fig. 2.1. Text block detection by thresholding DCT coefficient energies. *(a) and (b):* Two sample video frames. *(c) and (d):* Results of detection with threshold at 310. *(e) and (f):* Results of detection with threshold at 102.

are very probably text. Second, fewer missed-detect blocks are observed because lower thresholds are reached during iteration.

2.3.4 Heuristic filtering

Heuristics are applied to reduce blocks incorrectly identified as text. Since horizontal text orientation is assumed, it is reasonable that a text instance should be more than 8 pixels wide. Therefore, candidate text blocks with neither a left nor a right text block neighbor are marked as non-text. Candidate text blocks without any 8-neighbors are also discarded as noise.

I observed that many false alarms are due to steep luminance “cliffs” in the image. The cliffs are edges whose gradients are so high that they cause very high DCT energies. This effect is visible in Figure 2.1(e), in which blocks along the boundary between the blue background and the scene have been incorrectly marked as text. The following heuristic is used to remove such false alarms. A block marked as text is checked if its coefficient energy is above some threshold T_c. Then, the average of the DC coefficients[1] of the three blocks to the left are computed. Similar averages are found for the three blocks to the left, top, and bottom. If the absolute difference between the averages to the left and right are greater than some threshold T_d, or if the absolute difference between the averages to the top and bottom are greater than T_d, then the block's high DCT energy is probably due to an image cliff. The block is marked as non-text. Empirically, I determined that $T_c = 100$ and $T_d = 300$ give good results.

[1]The DC coefficient of a block is its first DCT coefficient. It indicates the average intensity of the pixels in the block.

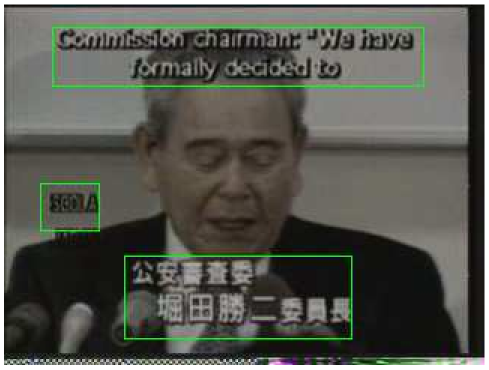
2.3.5 Text box localization

Localization is performed by finding connected components of detected text blocks. Bounding text boxes are found around each connected component. Spatial heuristics are then applied to remove boxes corresponding to non-text regions. Boxes with non-horizontal aspect ratios are discarded.

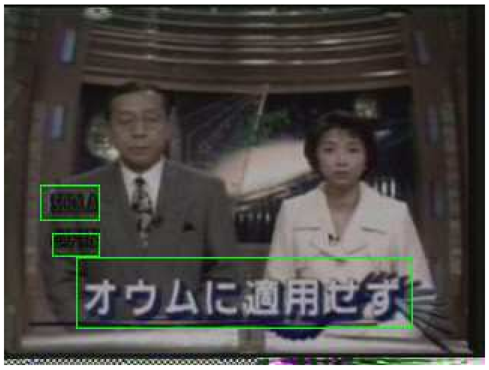
2.3.6 Results and Discussion

In this section I present output of this algorithm applied to sample video frames. A quantitative evaluation of this algorithm is presented in Section 2.5. All frames shown in this section were extracted from MPEG-1 videos with spatial resolution of 320×240 pixels. Videos were captured from a variety of television channels, including CNN and foreign news broadcasts. Note that the foreign news broadcasts are a challenging dataset for detection algorithms, because of their lower quality and contrast. Refer to Section 2.5.1 for further details about our video dataset.

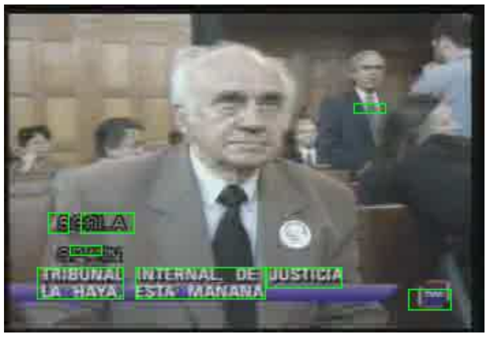
Figure 2.2 presents example localization results on a variety of video frames. The algorithm can detect text of different scripts, as demonstrated by image (a). It is even able to detect the very low-contrast "SCOLA" text appearing in images (a), (b), and (c). There are some false alarms along the top edge of image (d). These false alarms are caused by noise along this edge due to imperfect video capture. The algorithm is able to detect the text in a very small font size in image (e), although the localized text rectangle is a bit loose. Some small false alarms appear in image (f), but these could be easily suppressed by adding a minimum text box size heuristic.



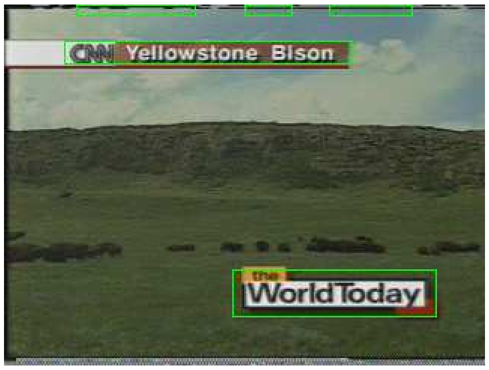
(a)



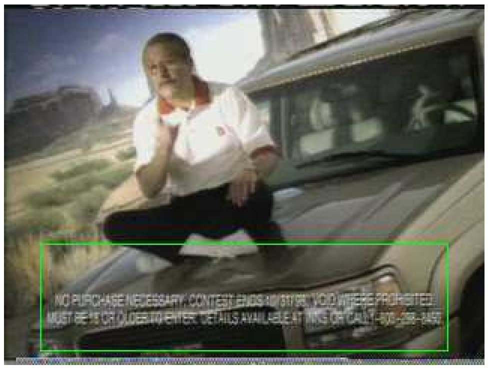
(b)



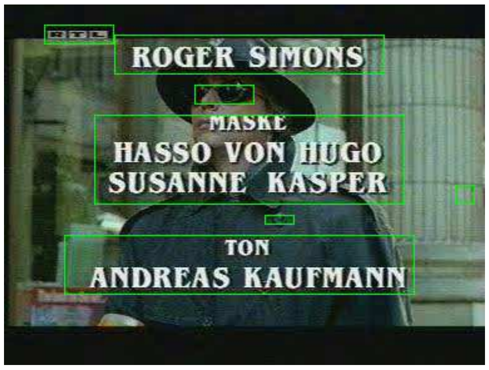
(c)



(d)


NO PURCHASE NECESSARY. CONTEST ENDS 10/31/98. VOID WHERE PROHIBITED.
MUST BE 18 OR OLDER TO ENTER. DETAILS AVAILABLE AT

(e)



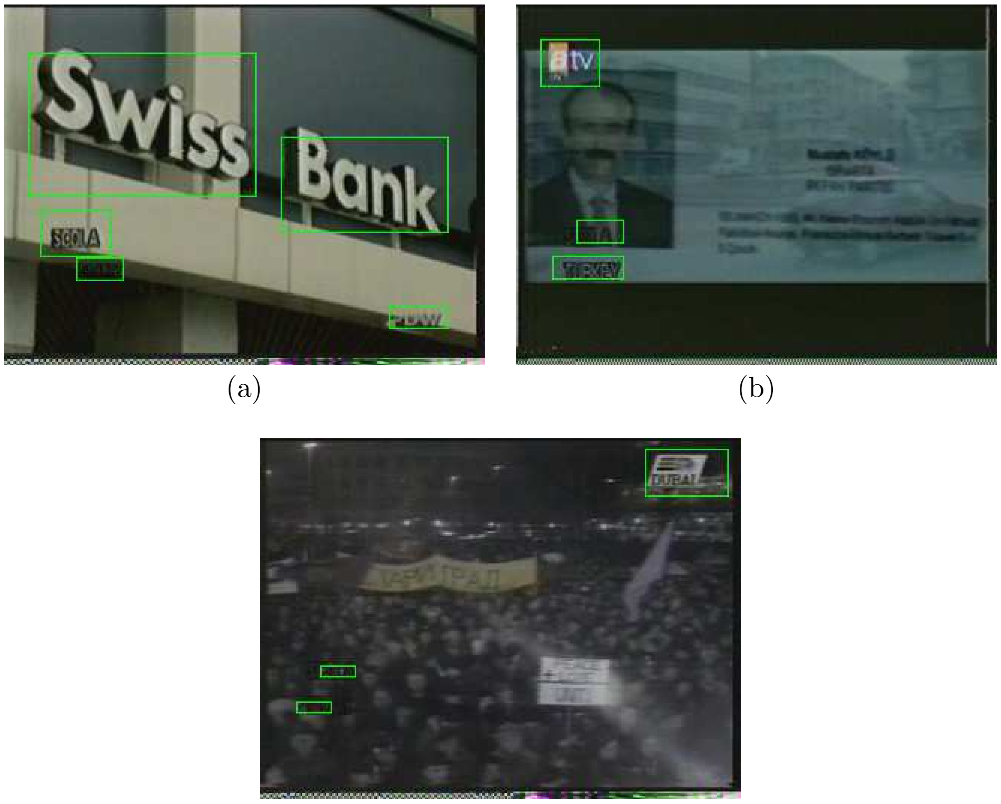
(f)

Fig. 2.2. Output of Algorithm A on video frames with caption text.

Although my focus is on caption text, the algorithm correctly detects some scene text as well. Figure 2.3 shows this. In image (a), the scene text instance “Swiss Bank” has been properly localized. But the low-contrast scene text instances in images (b) and (c) have been missed. Note that in all three frames, all caption text has been detected, and there are no false alarms.

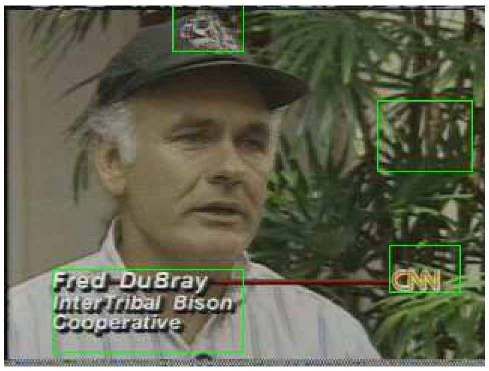
Figure 2.4 demonstrates some typical failures of the detection and localization algorithm. The algorithm exhibits two false alarms in image (a). The texture of the plant in this case is similar to that of text. Image (b) demonstrates a more severe case of false alarms. The high texture energy of the crowd scene has caused the algorithm to find the entire frame as one large text box. This could be solved by raising the T_h and T_l thresholds. This example demonstrates that, while the region-growing thresholding scheme reduces the sensitivity to fixed thresholds, it does not eliminate the sensitivity entirely. Images (c) and (d) show two examples of text missed due to limitations of the algorithm. The text region in image (c) is not localized properly because it violates the assumption of horizontal text. The text in image (d) causes difficulty because of its large stroke width. A consequence of using an 8×8 pixel block size is that the text stroke width must be less than about 8 pixels for proper detection. This explains the incorrect localization observed in image (d), where the character strokes are about 20 pixels wide.

I have shown that Algorithm A performs well on a variety of text instances in a variety of types of video. Another advantage to this algorithm is its relatively low computation cost. In fact, the algorithm requires only the DCT coefficients of a video frame. Since the DCT coefficients of I-frames are immediately available in the MPEG bit

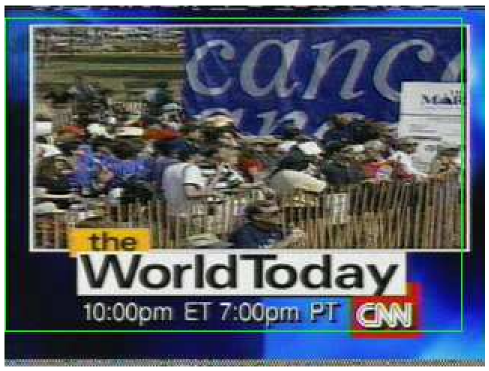


(a) (b) (c)

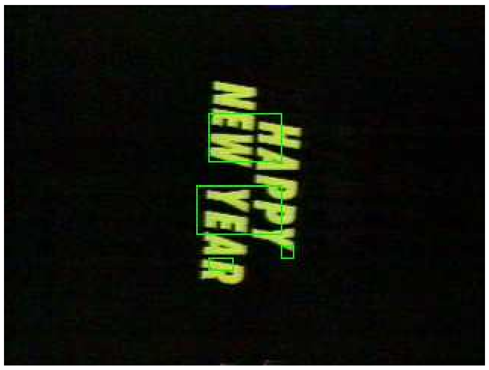
Fig. 2.3. Output of Algorithm A on video frames with caption text.



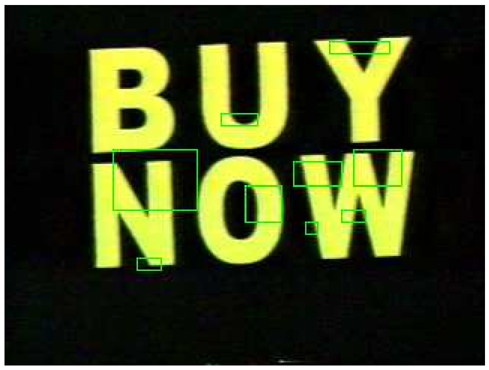
(a)



(b)



(c)



(d)

Fig. 2.4. Examples of poor-quality detection results from Algorithm A.

stream, the video file need not be fully decompressed. My unoptimized implementation runs at a real-time rate of over 30 I-frames per second on an SGI Octane workstation.

In some applications, it may be sufficient to perform text detection on just I-frames, since they occur relatively frequently in the MPEG stream (usually about three times a second) and text events tend to persist for at least several seconds. To process predictive (P- and B-) frames, it is necessary to reconstruct the DCT coefficients after motion compensation. The simple approach used in my implementation completely decodes each frame, then uses a fast DCT algorithm [47] to compute the coefficients. This is a naive implementation, but it still achieves a speed of 10 frames per second. For faster performance it is possible to perform motion vector compensation directly in the frequency domain [32].

2.4 Algorithm B: An algorithm for detecting caption text of arbitrary size and orientation

In the previous section, an efficient DCT-based text detection and localization algorithm (Algorithm A) was presented. While it gave good results on many video frames, I also noted the algorithm's limitations. It is unable to detect text having non-horizontal orientation. It relies on fixed energy thresholds that cause many false alarms in some scenes. It is unable to detect text with large stroke widths. In this section, modifications to the above algorithm are presented that circumvent these limitations.

2.4.1 Choice of DCT coefficients

Algorithm A used the coefficients found to be optimal by Chaddha [6]. However, Chaddha's application was document images, and a dataset of just five images was used for the optimization. The coefficients found in this manner may not be optimal for general-purpose video frames.

Unfortunately, finding the optimal coefficients is non-trivial. An exhaustive search would require trying all combinations of between 1 and 64 coefficients, or $\sum_{i=1}^{64}\binom{64}{i} \approx 1.8 \times 10^{19}$ possibilities. An alternative is suggested in [6]. The average absolute value of each coefficient for both text and non-text blocks is determined. Coefficients are sorted by the difference between text and non-text sums. Coefficients are then added one-by-one in the sorted order until the optimal choice of coefficients is found. This procedure requires trying at most 64 combinations of coefficients.

I performed the optimization in this manner using 9,329 frames of video from our ground-truthed dataset described in Section 2.5. This represents a much larger dataset than in Chaddha's optimization. His dataset had 4,800 blocks total; I used 9,122,279 blocks (539,941 text blocks, 8,582,338 non-text blocks). Figure 2.5 compares the average absolute value of each coefficient for text and non-text blocks. Using the procedure described above, the optimal coefficients were determined to be 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 16, 17, 18, 19, 24, 25, 26, 32, and 40, in row-major order.

2.4.2 Detection of text blocks

The above coefficient choice optimization was performed for horizontal text. Because the 2-D Discrete Cosine Transform is separable, transposing the matrix of pixel

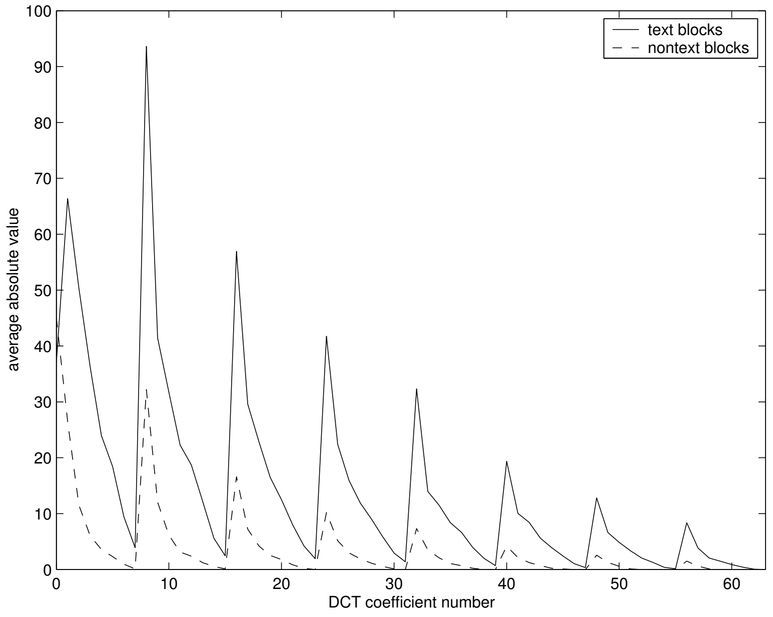


Fig. 2.5. Average DCT coefficient energy for text and non-text DCT blocks.

values of a block in the spatial domain corresponds with the transpose of the DCT coefficient matrix as well. It follows that vertical text can be detected by first taking the transpose of a block's DCT coefficient matrix, and then using the same coefficients determined during the optimization for horizontal text. I have observed that text oriented between horizontal and vertical has a combination of horizontal and vertical DCT text energy.

These observations motivate the following method for detecting text blocks. For each DCT block, the horizontal text texture energy TTE_h is computed by summing the coefficients listed above. Similarly, the vertical text texture energy TTE_v is computed by transposing the DCT coefficient matrix, and then summing the above coefficients. Instead of thresholding individual blocks, horizontal and vertical groups of three blocks are examined. This encourages blocks with high TTE_h to grow into horizontal text boxes, and blocks with high TTE_v to grow vertically. This is accomplished in the following way. The average of the TTE_h values for a block and its two horizontal neighbors is computed. This is added to the average of the TTE_v for the block and its vertical neighbors. If the total is greater than a threshold, the block is marked as text. That is, the block at row i and column j in an image is marked as text if

$$\frac{TTE_h(i,j-1)+TTE_h(i,j)+TTE_h(i,j+1)}{3}+$$
$$\frac{TTE_v(i-1,j)+TTE_v(i,j)+TTE_v(i+1,j)}{3}>T$$

2.4.3 Choice of threshold

Next we consider how to choose the threshold T. As mentioned earlier, while using a fixed threshold may be possible for document images, I found that the optimal threshold varies widely from one video sequence to the next. Algorithm A employed a region-growing scheme that reduced the reliance on the thresholds. However, fixed thresholds were still used, and this caused poor results on some video frames.

In informal experimentation, I have observed that the optimal threshold is fairly uniform across all frames of the same video sequence. Also, different video sequences of the same general type have similar optimal thresholds. This suggests that the optimal threshold depends on general characteristics of the video that could be computed or known *a priori*. For example, perhaps one threshold is best suited for news broadcasts, while another is better for commercials. The genre of video may be known *a priori*, or an algorithm could be used to automatically determine the genre (e.g. [15]). I also observed that low-level image features could also be used to predict optimal threshold. Specifically, I hypothesized that video contrast could be used.

This hypothesis was tested as follows. First, the optimal threshold for each video sequence in our dataset was determined by exhaustively trying all possible thresholds within a reasonable range (again according to the experimental protocol and evaluation criteria discussed in Section 2.5). The average contrast per frame for each video sequence was also computed. The contrast measure used was the difference between the highest and lowest gray level in the luminance plane of a frame.

Figure 2.6 plots the optimal threshold versus the average frame contrast. The figure indicates that there is a strong linear correlation between contrast and ideal threshold. The best-fit line that minimizes least-square-error was found to be about $T(c) = 23.8c - 2018.5$ for a given average contrast c. This result suggests that it is possible to predict a good threshold based only on the general characteristics of a video sequence.

Note that this analysis was carried out on a relatively small dataset of 11 sequences and 11000 frames total. More experimentation would be necessary to determine that this simple linear relationship holds for a larger dataset. Also, the optimal threshold may be better correlated with video sequence features other than average frame contrast. For the work in this thesis, however, we use only contrast to predict the threshold. The threshold for each sequence is computed using the $T(c)$ expression given above.

2.4.4 Hierarchical subsampling to detect different sizes of text

It was noted earlier that Algorithm A is unable to detect text with stroke width larger than the DCT block size. This problem can be circumvented by analyzing a subsampled version of the frame. For example, text with strokes up to 16 pixels wide can be detected in a frame subsampled to half the original size.

Subsampling is incorporated into the algorithm as follows. The text block detection algorithm is applied to the image. Then, the image is subsampled to half its dimensions, and the 8×8 block classification algorithm is applied again. A block at this level corresponds to four blocks in the original image. For each block classified as text in the subsampled image, the corresponding four blocks in the original image are marked

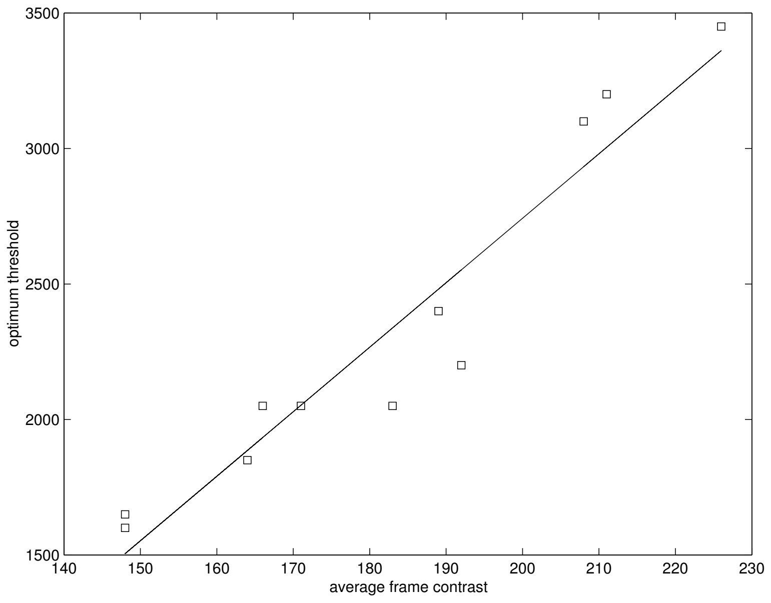


Fig. 2.6. Optimal threshold versus average video frame contrast.

as text. Subsampling is continued iteratively until some lower bound on the frame dimensions is reached. I have observed that proceeding to a resolution of 160×120 ensures that text of all reasonable sizes is found. This corresponds to two hierarchical levels for an original frame size of 320×240 and three levels for a frame size of 640×480.

My present implementation uses a naive approach to the subsampling. The original image is converted to the spatial domain, subsampled, and then the DCT transform is taken. A much more efficient approach is to perform subsampling directly in the DCT domain. A method for doing this is described in [8].

2.4.5 Localization of text with arbitrary orientation

Once blocks of a frame have been classified as text or non-text, we wish to group the blocks into text instances. This is done by determining tightly-fitting bounding rectangles for each text instance. In the case of oriented text, the bounding rectangle should be oriented at the appropriate angle. As was noted in Section 2.2, no work so far in the literature has considered the problem of localizing non-horizontal text.

We propose an iterative greedy algorithm for separating text instances and determining tight bounding boxes around them. First, connected component analysis is performed on the blocks marked as text. Orthogonal bounding rectangles are computed for each component. Then, the bounding rectangles are iteratively refined by moving, changing size, and changing orientation. Each iteration of the greedy algorithm attempts to increase the criteria

$$G = P_t \times (1 - P_{nt})$$

where P_t is the percentage of the detected text pixels that lie underneath the rectangle, and P_{nt} is the percentage of the rectangle's area covering non-text pixels. During each iteration, each bounding rectangle is visited. One of the following actions is taken, according to which produces the maximum G:

- Rectangle is left unchanged
- Rectangle height is incremented or decremented by one block
- Rectangle width is incremented or decremented by one block
- Rectangle is moved one block to the left or right
- Rectangle is moved one block up or down
- Rectangle is rotated by 15 degrees clockwise or counter-clockwise

Once all rectangles have been visited during an iteration, overlapping rectangles are merged together if doing so does not lower the overall value of G.

The iteration continues until convergence. Simple heuristics can then be applied to discard non-text regions based on rectangle dimensions. In my implementation, we discard rectangles whose length or width is less than 8 pixels. Very few text instances are less than this size, and even if present, it is doubtful that an OCR module could recognize them accurately.

2.4.6 Results

Figure 2.7 presents sample results of the algorithm applied to 320×240 pixel MPEG-1 video frames captured from television channels. The rectangles superimposed on the video frames indicate the results of the text localization. Image (a) demonstrates the algorithm's effectiveness on simple, horizontal caption text. Note that the algorithm works on a variety of language scripts. Images (b), (c), and (d) show examples of detection of both horizontal text and text oriented at different angles. Image (c) includes some text missed by the algorithm. This text is less than 8 pixels tall and thus was discarded by our size heuristic.

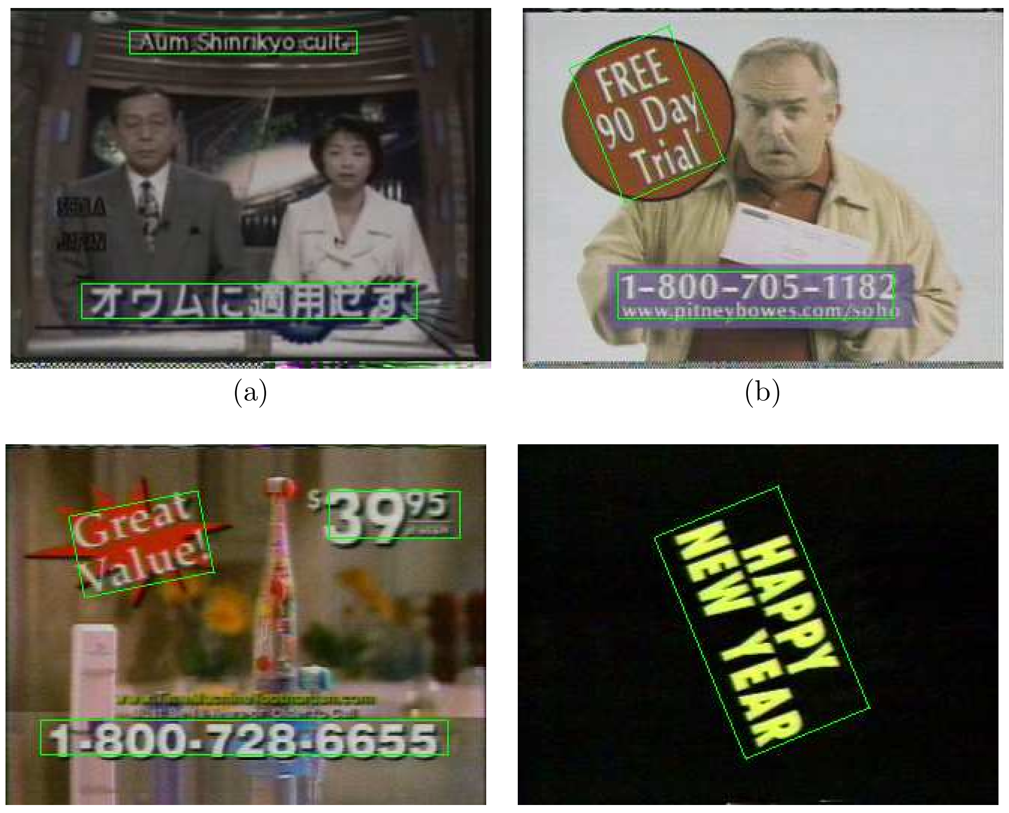
2.5 Performance Evaluation

As noted in section 2.2, there have been no quantitative, comparative performance evaluations of text detection algorithms presented in the literature. In this section, I present the results of a quantitative evaluation of the above two algorithms and four others from the literature. We have presented a similar evaluation in [5].

2.5.1 Video datasets

Two datasets were used in the evaluation. Dataset A contained mostly static caption text typical of news broadcasts. Dataset B consisted of commercials that included moving, rotating, growing, and shrinking text. Details of the datasets are as follows:

- **Dataset A** consisted of 15 MPEG-1 video sequences with 320×240 pixel resolution. There were a total of 10299 frames (about 175 megabytes of data). There were 156 caption text events and 144 scene text events in the video data. The dataset represented a wide variety of video captured from television broadcast channels. Video clips included newscasts from Turkey, United Arab Emirates, Japan, and Germany, CNN's The World Today program, CNN's Business Unusual program,



(a) (b)

(d) (e)

Fig. 2.7. Examples of detected text of various orientations.

ABC's World News Tonight, and commercials from various channels. This diverse collection of video contained a wide variety of text fonts, colors, placements, languages, and scripts.

- **Dataset B** consisted of 1 MPEG-1 video sequence with 320×240 pixel resolution. There were a total of 916 frames (about 26 megabytes of video data), and 25 caption text events. The dataset consisted of portions of commercials captured from various television channels. A wide variety of text sizes and colors was included in the dataset. All captions were in English. In addition to static text, text events undergoing rotation and size changes were included.

Video sequences for both datasets were captured at 30 frames per second by either a CosmoCompress motion-JPEG hardware compression board on an SGI Indy workstation, or by an ICE motion-JPEG hardware compression board on an SGI O2 workstation. The movies were converted to MPEG-1 using SGI's dmconvert software encoder. The compressed bit rate was 4.15 megabits per second. The group of pictures (GOP) size (i.e. distance between adjacent I-frames) was 12 frames.

Dataset A was ground-truthed by Jin Hyeong Park, Vladimir Mariano, Sameer Antani, and me using the ViPER tool from the University of Maryland [7]. Dataset B was ground-truthed my me. In each frame, tight bounding rectangles were drawn around any text regions (regardless of whether the text could actually be read).

2.5.2 Evaluation criteria

It is not obvious how to design a good evaluation criteria for text detection and localization algorithms. Most evaluations presented in the literature (e.g. [17]) give evaluation results as a single percentage called "accuracy." This indicates the percentage of text instances detected by the algorithm. However, this accuracy statistic is misleading, because it does not capture the false alarm rate. For example, using this evaluation strategy, an algorithm that simply places text boxes around the entire area of every frame would achieve 100% accuracy. It is also not clear how to decide whether an algorithm has detected a text event or not. For example, has the algorithm detected the text in Figure 2.4(d)? It has marked parts of the text, but not all of it. Unfortunately these details are rarely specified in papers in the literature. From communications with authors, it appears that usually a human's subjective judgment is used to determine whether the algorithm reasonably detected a given text instance or not.

We desire an evaluation criteria that rewards algorithms for tightly localizing text events. Algorithms should be penalized for failing to detect text or for detecting only a portion of text. They should also be penalized for false alarms, or for loose localization of text. Further, the criteria should be objective and automatically computable by a program.

We perform a pixel-by-pixel match of the ground truth against the output of a localization algorithm. A pixel is counted as a *correct detect* if it is marked as text in the ground truth and in the algorithm's output. A *false alarm* appears in the algorithm output but not the ground truth. A *missed detect* appears in the ground truth but not the algorithm output. To perform the evaluation, the number of correct detect, false alarm, and missed detect pixels are counted. The results are expressed as recall and precision, where:

$$\begin{aligned} Recall &= \frac{correct\ detects}{correct\ detects + missed\ detects} \\ Precision &= \frac{correct\ detects}{correct\ detects + false\ alarms} \end{aligned}$$

Intuitively, recall expresses the ability of an algorithm to detect text. A recall of 100% indicates that the algorithm found all text in the dataset. Precision is a measure of the tightness of the localization. A precision of 100% indicates that the algorithm's output exhibited no false alarm pixels. Note that there is a trade-off between recall and precision. For example, an algorithm's parameters can be adjusted to increase recall, but this will generally cause precision to decrease.

The relative importance of recall and precision depends on the application. For this evaluation, I will assume that recall and precision are equally important. Therefore I will compare algorithms at the point where parameters have been adjusted such that recall and precision are equal.

2.5.3 Experimental protocol

Algorithms A and B presented in this chapter were evaluated along with four other promising algorithms from the literature. This included the work of Gargi *et al* [10], LeBourgeois [22], Mariano *et al* [31], and Mitrea *et al* [46]. Source code provided by the authors was used for the Gargi and Mariano algorithms. The LeBourgeois and

Mitrea algorithms were implemented by Ryan Keener, Albert Roberts, and me based on the algorithm descriptions in the papers.

The evaluation was carried out as follows. Each algorithm requires one or more fixed parameters. The parameters were optimized on Dataset A by varying each parameter over a reasonable range. For each combination of parameters, the evaluation was performed on the full 10299 frames. The combination of parameters that gave the highest recall and precision under the constraint $recall = precision$ was declared optimal. The recall and precision obtained using this set of parameter values were used to represent the performance of the algorithm.

Our ground truth contains localization data for both caption and scene text. Since the focus of this thesis is caption text, algorithms were tested only on caption text. Algorithm output was ignored for regions marked as scene text in the ground truth. Therefore algorithms were neither penalized nor rewarded for missing or finding scene text.

2.5.4 Results

Table 2.1 presents the results of the evaluation for Dataset A. It is observed that Algorithm B exhibits the best performance, followed very closely by Algorithm A. The Mariano algorithm was next best, followed closely by the Mitrea algorithm.

Table 2.2 presents the evaluation results for Dataset B. Two sets of recall and precision statistics are given. The evaluation was first performed using the parameter values determined as optimal over Dataset A. These results are shown in the second and third columns of Table 2.2. The parameter values for each algorithm were then varied to find the optimal parameter sets for Dataset B. These results are shown in the fourth and fifth columns of the table. Note that Mariano's algorithm was not included in these runs, because our dataset violated its assumption that all text is perfectly horizontal.

It is observed that Algorithm B gives by far the best performance on Dataset B, with an optimal precision and recall of 74%. Further, the results indicate that the optimal parameters for Algorithm B on Dataset A are very close to optimal on Dataset B. This suggests that Algorithm B is relatively insensitive to the value of its parameters. The LeBourgeois, Mitrea, and Gargi algorithms exhibit optimal precision and recalls of around 49%, about 25 percentage points lower than those of Algorithm B. The results also suggest that these algorithms are more sensitive to the values of their parameters. Algorithm A gives poor performance on this dataset. This is because much of the text in Dataset B is relatively large, violating Algorithm A's assumption that the text size is comparable to the 8×8 pixel block size.

I observe that Algorithm B has shown the best performance on both datasets. It performs slightly better than other algorithms in the literature on a dataset containing mostly static, horizontal text. It performs significantly better than other algorithms on a dataset including non-horizontal text that rotates, changes size, and moves over time. This is an encouraging observation, because it demonstrates that it is possible to design text detection algorithms that make fewer assumptions about text in video while maintaining the accuracy typical of algorithms found in the literature. It is hoped that in the future, other researchers will attempt to reduce the restrictions their algorithms place on the types of text that can be detected.

The results of the quantitative performance evaluation indicate that precision and recall of state-of-the-art detection and localization algorithms are quite low. It is disappointing to see precision and recall values under 50%, when we would like values close to 100%. This highlights the need for further research in designing more accurate algorithms that can detect and localize text in general-purpose video. However, there are two caveats to our performance evaluation that should be kept in mind. First, our evaluation criteria is very strict. An algorithm must generate output that exactly matches the ground truth in order to achieve perfect precision and recall. In an actual application, it probably does not matter if a localization algorithm's output is off by a few pixels. Second, our dataset is extremely challenging. The ground truth has been marked with all caption events that could be detected by a human, even if they could not be read. Such text may not even be useful to an application.

Algorithm	Recall	Precision
Algorithm A	46%	45%
Algorithm B	46%	48%
Gargi	29%	30%
LeBourgeois	33%	34%
Mariano	40%	39%
Mitrea	37%	37%

Table 2.1. Detection/localization algorithm performance for caption text on Dataset A. Dataset A contains mostly horizontal, static text events.

Algorithm	Preset parameter set		Optimal parameter set	
	Recall	Precision	Recall	Precision
Algorithm A	36%	36%	36%	36%
Algorithm B	73%	75%	74%	74%
Gargi	37%	62%	46%	48%
LeBourgeois	25%	73%	49%	49%
Mitrea	37%	58%	47%	48%

Table 2.2. Detection/localization algorithm performance for caption text on Dataset B. Dataset B includes text that moves, rotates, grows, and shrinks over time. Results are shown both for when the parameters were set to values found optimal for Dataset A, and when set to those found optimal for Dataset B.

Chapter 3

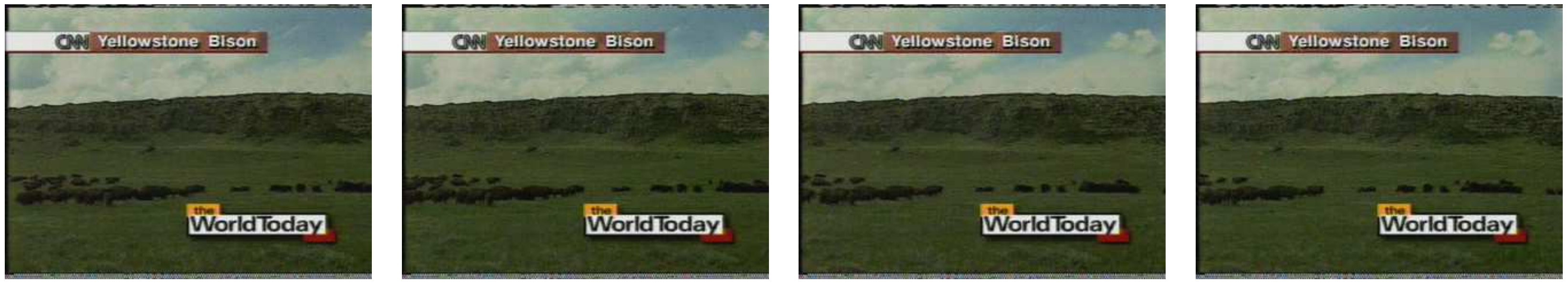
Text tracking

3.1 Introduction

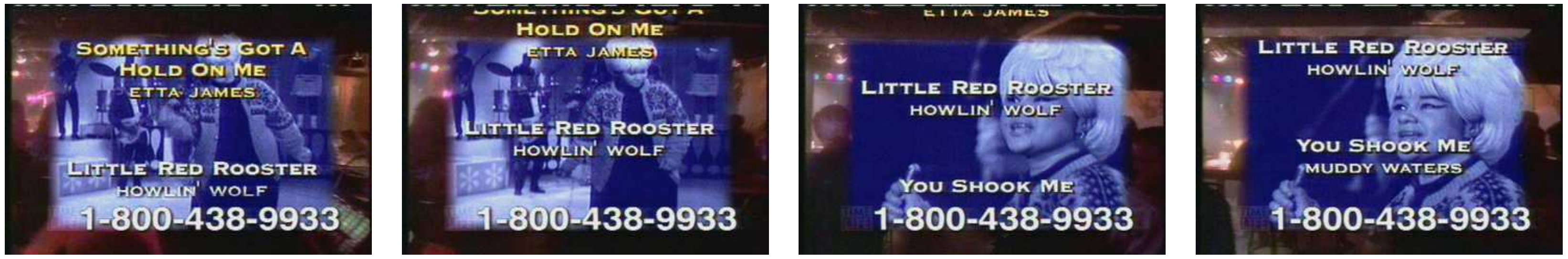
Text in video persists for multiple frames. A typical text event lasts for at least a second to allow human viewers adequate time to read it. At the NTSC frame rate of about 30 frames per second, even a one-second text event appears in 30 video frames. The text event may remain stationary, in which case the spatial location of the text is the same in all frames. It may exhibit a slow, linear motion, as typified by scrolling movie credits. Or it may move quickly in a complex trajectory, it may change size or shape, it may undergo perspective distortion, it may rotate, or it may exhibit a combination of these behaviors. Figure 3.1 shows examples of text events and their behaviors over time.

A tracker is necessary to follow text as it moves. A text tracker could have several purposes in a video indexing system:

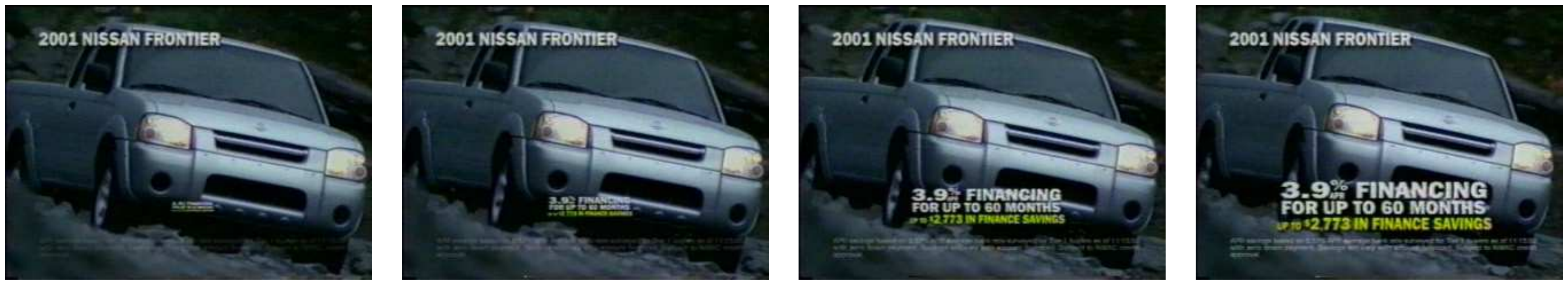
- **Determination of text events:** We would like to build an index of the text occurring in video for content-based retrieval purposes. The index would not include entries for individual frames, but instead for each text *event* that appears, persists for some time, and then disappears. That is, we would like to find the temporal location and extent of a text event, as well as the spatial location and extent in each frame. The tracker can be used to combine the localized text regions of individual frames into text events.



(a)



(b)



(c)

Fig. 3.1. Examples of caption text behavior over time. *(a):* Stationary text, *(b):* Scrolling rigid text exhibiting simple, linear motion, *(c):* Text changing size with time.

- **Verification of text localization:** Since it is assumed that text persists for multiple frames, a region localized as text in one frame but not in neighboring frames indicates that it is a false alarm and should be discarded. Assuming all text is stationary, a candidate region could be discarded if no region exists at the same location in the neighboring frames. But this would fail for moving text. A text tracker is required to verify that motion in the localization output is consistent with motion in the video.

- **Human-assisted text event indexing:** State-of-the-art text detection algorithms may not perform well on certain datasets (*e.g.* very noisy video data). In these cases, a human operator could mark a text region in the first frame in which it appears. The tracker could then automatically determine the location of the text in subsequent frames.

This chapter considers the text tracking problem. In Section 3.2, prior work related to text tracking is presented. Then, two algorithms representing two different approaches to text detection are described. In Section 3.3, I describe a fast algorithm for tracking rigid text events exhibiting linear motion in MPEG video. This algorithm can operate independently to support an operator-assisted environment as described above. In Section 3.4, an algorithm is presented for tracking text whose size, position, and orientation may be changing over time. This algorithm requires tight integration with a text detection algorithm, like those presented in Chapter 2.

3.2 Review of prior work

While there has been a significant amount of work on the extraction of text in images and video frames, very little text detection work is found in the literature that considers the temporal nature of video. This section surveys the few approaches that do include temporal analysis.

Shim *et al* [48, 49] uses a simple inter-frame analysis technique to reduce false alarms. Individual frames are first processed by finding regions with homogeneous intensity, forming positive and negative images by double thresholding, and applying heuristics to remove non-text regions. Then, the candidate text regions in groups of five adjacent frames are considered. Text is assumed to be stationary. A candidate text region is discarded if regions of similar position, intensity, and shape do not appear in the other four frames. Note that this approach would incorrectly discard moving text regions.

Lienhart [29, 28] takes a similar approach, but allows text motion. Individual frames are segmented using properties of local color histograms and choosing text candidate regions using heuristics. Temporal analysis is used to refine detection results. For each potential text region detected in a frame, the text candidate regions in the next frame are searched for one of identical size, color, and shape. If such an area is not found, the region is discarded as non-text. This approach assumes text remains rigid. It also requires that text detection is applied to each frame, so it is not applicable to the operator-assisted application mentioned above.

Li and Doermann [25, 26] describe a simple algorithm for tracking rigid, moving text in video. A simple pixel-level template matching scheme is used. It is assumed that the text is moving with constant velocity. A record is kept of this velocity. Given the known location of a text region in a frame, its location in the next frame is predicted using this velocity. A simple least-squared-error search is performed around a neighborhood of the predicted location to find the precise location. Note that the pattern matching is performed on both text pixels and background pixels alike. This can be problematic when text occurs on complex backgrounds, or when text moves over backgrounds of different gray level intensity. This approach also fails for text exhibiting a non-linear velocity.

A recent extension to Li and Doermann's work [27] adds a post-processing step to correct tracking results in the case that text grows or shrinks very slightly from frame to frame. The text region is enlarged, and a Canny edge detector is applied to find character edges. A tight bounding rectangle is found around these characters, and is used as the final tracking result. The authors found that this extension failed for text moving over complex backgrounds. To overcome this problem, the least-squared-error value computed during the neighborhood search is monitored. If a spike in the error occurs, it is assumed that the text is moving over a complex background, and the post-processing step is disabled. Once tracking begins, their tracker continues until the end of the video sequence. They do not consider the problem of determining when a text event has disappeared, or when a text event ends and another begins. Their algorithm simply follows some "edgy" region in the video; it does not ensure that it is following the same text from frame to frame.

3.3 Method for tracking rigid text using MPEG motion vectors

As discussed in the previous section, Li and Doermann's work represents the state-of-the-art in text tracking. Unfortunately, this approach has several limitations. First, it assumes that text moves with constant velocity in a linear trajectory. This assumption could be relaxed by using a more sophisticated trajectory model; however, even this would fail for random, erratic motion. Another approach would be to remove the predictive stage altogether and increase the size of the template search window. Unfortunately this increases the computation cost prohibitively. A least-squared-error search for an $m \times n$ text region over a $w \times w$ pixel search window requires $m \times n \times w^2$ pixel comparisons. Therefore it becomes very expensive to increase the search window because the search operation is of order $O(w^2)$. A second limitation of their algorithm is that it compares all pixels within the localized text region, including background pixels. This can cause the algorithm to track the background instead of the text if the background changes or if text moves over backgrounds of different intensities.

In this section, I present an algorithm for efficiently tracking rigid text in MPEG video. I use the motion vectors present in the MPEG-compressed video bit stream to predict text motion with very little computation cost to the tracker. In effect, the computation cost has already been paid by the MPEG encoder. This idea was inspired by papers by Nakajima *et al* [36], who used motion vectors to detect moving objects in MPEG video, and by Pilu [40], who used them to detect camera motion.

3.3.1 Review of MPEG motion vectors

A brief overview of motion compensation in the MPEG-1 video coding standard is presented here. The reader is referred to [34] for a detailed treatment of the standard.

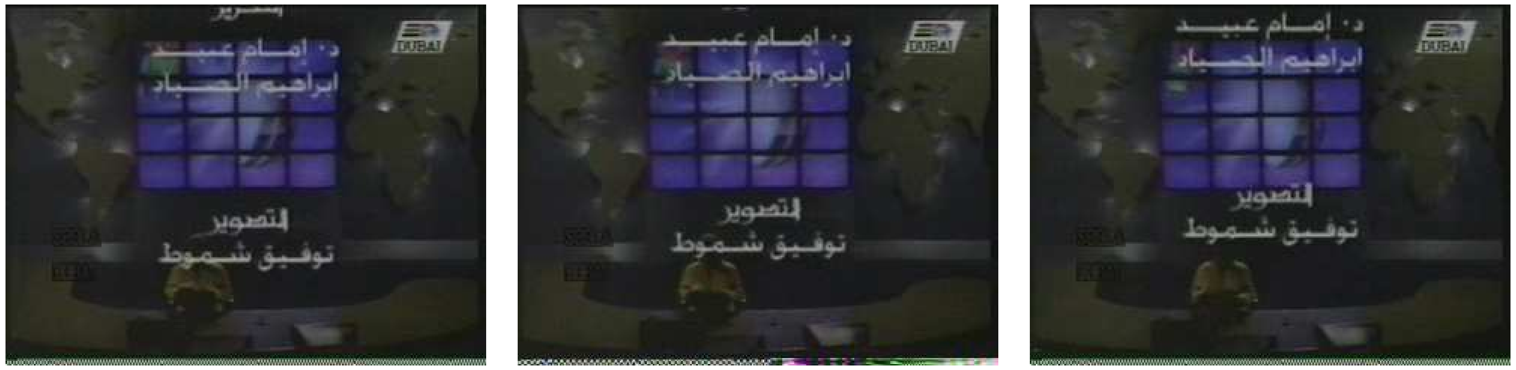
The MPEG video standard uses motion compensation to reduce temporal redundancy in the compressed video stream. MPEG defines three types of frames: intra-coded (I) frames, predictive (P) frames, and bidirectional predictive (B) frames. An I-frame is self-contained in that it has all the information required to reconstruct the frame. P- and B-frames are split into non-overlapping 16×16 pixel regions called *macroblocks*. Each macroblock in a P-frame includes a motion vector indicating an x and y pixel displacement from the last I- or P-frame. It also includes DCT error correction coefficients. To reconstruct a given P-frame macroblock, the MPEG decoder begins with the 16×16 pixel area pointed to by the motion vector, and adds the IDCT of the correction coefficients. A B-frame is similar to a P-frame except that it can include both a motion vector to the previous I- or P- frame and a motion vector to the next I- or P- frame. Reconstruction of B-frames is accomplished by averaging the two macroblocks pointed to by the motion vectors and adding the error correction.

3.3.2 Motion prediction using MPEG motion vectors

At first it may seem trivial to apply MPEG motion vectors to the problem of tracking text in video. Unfortunately, MPEG motion vectors are usually too noisy for direct use in a tracker. This is explained by the following observation. Given a region of one frame, a tracker wishes to find the precise location of that region in the next frame. On the other hand, the goal of the MPEG encoder is to achieve minimal coding requirements in a minimum amount of time. MPEG encoders are willing to trade off motion vector accuracy for a decrease in the encoding time.

Figure 3.2 illustrates typical motion vectors found in MPEG video. Three consecutive P-frames of an MPEG encoded video are shown in (a). The video contains upward-scrolling text. Graphical representations of the macroblock boundaries and motion vectors found in the MPEG bit stream for these three frames are shown in (b). The white grid indicates the macroblock boundaries. Macroblocks marked with an "X" are I-coded macroblocks, meaning that they are self-contained and do not require motion compensation. For macroblocks that are motion-compensated, the motion vectors are drawn from the macroblock center to the center of the region used for motion compensation in the last frame. Macroblocks drawn with neither an "X" nor a vector have a motion vector of length zero. It is observed in this figure that many of the macroblocks corresponding to the text have motion vectors that accurately indicate the text's motion. However, some of the motion vectors point in random directions. In particular, I observed that macroblocks containing few edges tend to have incorrect motion vectors. Macroblocks containing strong edges tend to be reliable.

My algorithm deals with these issues as follows. Given a localized text region in one frame, search the next frame for all macroblocks whose motion vectors point back to any part of the text region. Extract the motion vectors from these macroblocks. Several constraints are then applied to the motion vectors to determine those that are likely to be reliable. Very small motion vectors (less than 2 pixels in magnitude) are probably noisy and are removed from consideration. Motion vectors from relatively featureless macroblocks are also discarded, because they are not likely to be accurate. This is


(a)

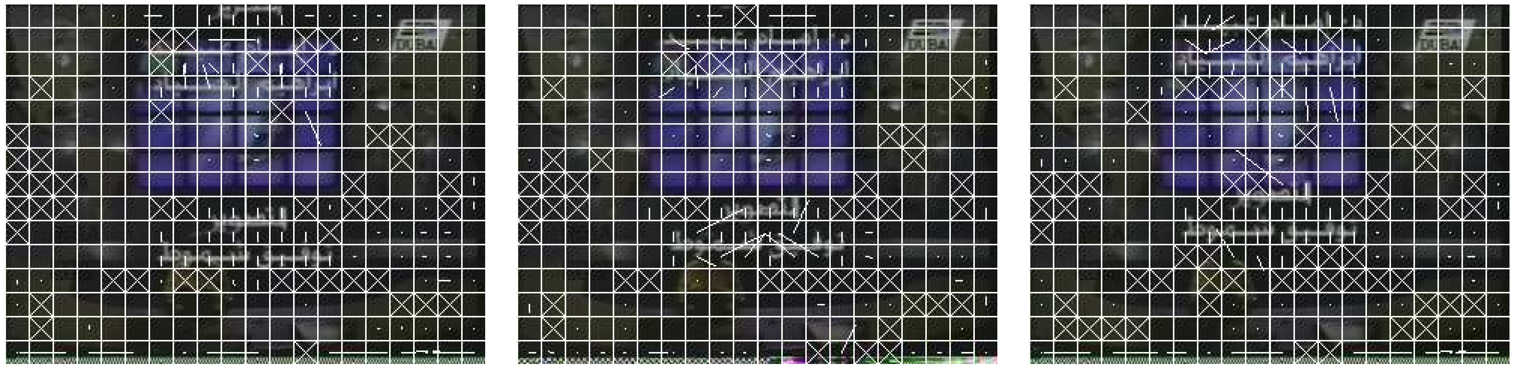

(b)

Fig. 3.2. MPEG motion vectors indicate object motion but are noisy. *(a):* Three consecutive P-frames in an MPEG-1 video. *(b):* The same three frames overlaid with graphical representations of the macroblock boundaries and motion vectors.

determined by applying a Sobel edge detector on each macroblock, and eliminating macroblocks that contain less than four edge pixels.

The magnitude and direction of the remaining motion vectors are then clustered. It is assumed that the largest cluster corresponds to the approximate motion of the text block. Note that this clustering process implicitly ignores noisy motion vectors. The vectors in this cluster are then averaged to yield a single motion vector for the text region.

It is clear that a text event cannot be tracked in an I-frame in this way, because I-frames do not contain motion vectors. Fortunately, I-frames are relatively rare in an MPEG stream. Typically I-frames occur only once every ten or twelve frames. Tracking through an I-frame is handled by averaging the motion vectors determined for the region in the frame before the I-frame and the frame after.

3.3.3 Refinement using gradient-based correspondence

I have found that the motion vector determined using the simple process above is generally of very high quality. In fact, for many text events it is possible to track a moving text region using the MPEG motion vectors alone. However, any small errors made in the tracking from one frame to the next propagate through the entire lifetime of the text event. For a long video sequence, the tracking location is usually inaccurate by several pixels after tracking text through several seconds of video.

I therefore employ a least-squared-error correspondence search around a very small neighborhood of the location predicted by the MPEG motion vector analysis.

Instead of comparing pixel gray levels directly, as in Li & Doermann's method, I perform the correspondence search only on edge pixels (pixels with high gradient). This encourages the algorithm to match only on the text pixels and not on the background pixels. Matching on edges implies that the text can move across backgrounds of different colors without affecting tracking reliability.

MPEG encoders generally use a search window of 32 pixels in each direction during motion compensation searches [34]. This creates a large computation cost and accounts for the slow performance of MPEG encoding. The tracker algorithm so far, however, is able to take advantage of this wide search window "for free." Unlike Li & Doermann's algorithm, this algorithm makes no assumptions about the trajectory of text, and therefore can handle a greater variety of text motion. Assuming a 32 pixel search window during MPEG encoding, a text event would have to move at a speed greater than 32 pixels per frame in order for the tracking algorithm to fail. Text moving this fast is very unlikely, as it would travel from one edge of a 320×240 pixel video to the other edge in a third of a second.

Unfortunately, successful use of motion vectors is highly dependent on the MPEG encoder used to encode the video. It is possible, for example, to encode a video using only I-frames,[1] or using a very small search window during motion compensation. To handle these cases, I also include a simple trajectory-based prediction similar to Li and Doermann's algorithm. A record of the current trajectory of the text region is kept. After performing the motion vector-based tracking approach described above, text motion is

[1]Note that MPEG videos are rarely encoded in this manner, because bypassing motion compensation significantly increases the MPEG file size.

separately predicted using the past trajectory. A least-square-error search is performed around a neighborhood of the predicted location. The lowest error of this search is compared to the lowest error found during the motion vector-based search. Of these two choices, the location with the lowest error is chosen.

3.3.4 Text entering or leaving the video

Text in video sometimes scrolls on or off the screen, such that in some frames only a portion of the text event is visible. I include special cases in the algorithm for handling this type of motion. Text exiting the frame is the easier case. The motion determination steps discussed above are applied only on the portion of the text event that is visible. If the computed motion indicates that the text event is exiting the frame, the tracked text box is clipped at the video frame boundary.

Text scrolling into the video is more difficult, because the spatial extent of the text event is not known. For example, in the operator-assisted indexing application described above, the human may mark the visible portion of a text event occurring on the edge of the frame. In subsequent frames, the tracker determines that the text event is moving towards the center of the frame. We would like the tracker to be able to automatically resize the tracking box as more text enters the frame. This case is handled in the following way. The number of edge pixels occurring in the known text region is counted and used as a texture measure. When the tracker detects that the tracking box is moving from the edge of the frame towards the center, the number of edge pixels in the region near the edge is also counted. If the density of edges between the two regions is comparable, the tracking box is expanded to accommodate the incoming text.

3.3.5 Results

In this section, I present the results of running the algorithm on a variety of video sequences. Except for “poster.mpg”, all video sequences have a spatial resolution of 320×240 pixels and a frame rate of 30 frames per second. They were captured using the ICE hardware JPEG compression module on an SGI O2 and converted to MPEG-1 format using SGI’s dmconvert utility. “Poster.mpg” was captured at 15 frames per second directly to MPEG-1 format using a hand-held camera connected to a Sun ULTRA-1 workstation equipped with a SunVideo hardware MPEG compression card. Text regions were marked by hand in the first frame of each sequence, and the algorithm automatically tracked the regions for the remainder of the sequence.

Figure 3.3 shows tracking results on typical moving caption text events in a commercial video sequence. Note that the tracking continued successfully through the sudden change in background. Figure 3.4 shows the algorithm tracking caption text scrolling horizontally. Text is entering and exiting the frame. The algorithm does a good job of determining the bounding boxes on incoming text using the texture similarity method described in Section 3.3.4, although the left boundary of the “HOWLIN’ WOLF” text event is somewhat loose. Figure 3.5 illustrates tracker performance on vertically-scrolling text in an Arabic script. Note that tracking text in this script is challenging because the text has fewer edges than text in Latin script.

Although the tracking algorithm was designed for caption text, I have found that it works for quasi-rigid scene text events as well. Figure 3.6 demonstrates this. The algorithm tracks successfully despite the changing text size due to camera motion. The algorithm's robustness to erratic, fast motion is demonstrated in Figure 3.7. A tracker assuming a simple linear trajectory model would fail in this case.

3.4 Shape-based method for tracking unconstrained caption text

The work presented in the last section focused on tracking rigid text. However, caption text events can change over time. Text can grow, shrink, or rotate. In this section, I describe a method for tracking text that changes in these ways over time.

Instead of a stand-alone tracking algorithm, I propose tightly coupling the detection and tracking modules. The detection and localization algorithm identifies text instances in each frame. It is the responsibility of the tracker to determine which text instances (if any) in adjacent frames correspond to the same text event.

Two text instances belong to the same text event if the *content* of the text is the same, regardless of changes in size, location, etc. Therefore it follows that although some characteristics of a text event may change over time, the basic shape of the characters remains constant. This property can be exploited to determine whether two text boxes correspond to the same text event.

I propose analyzing two consecutive frames at a time. First, the text box localization algorithm described in Section 2.4 is applied to each frame. Oriented text instances are made horizontal by applying a simple rotation transformation. A text binarization algorithm is next applied on each text instance. My implementation uses the binarization algorithm presented in Chapter 4, although another binarization algorithm could be substituted.

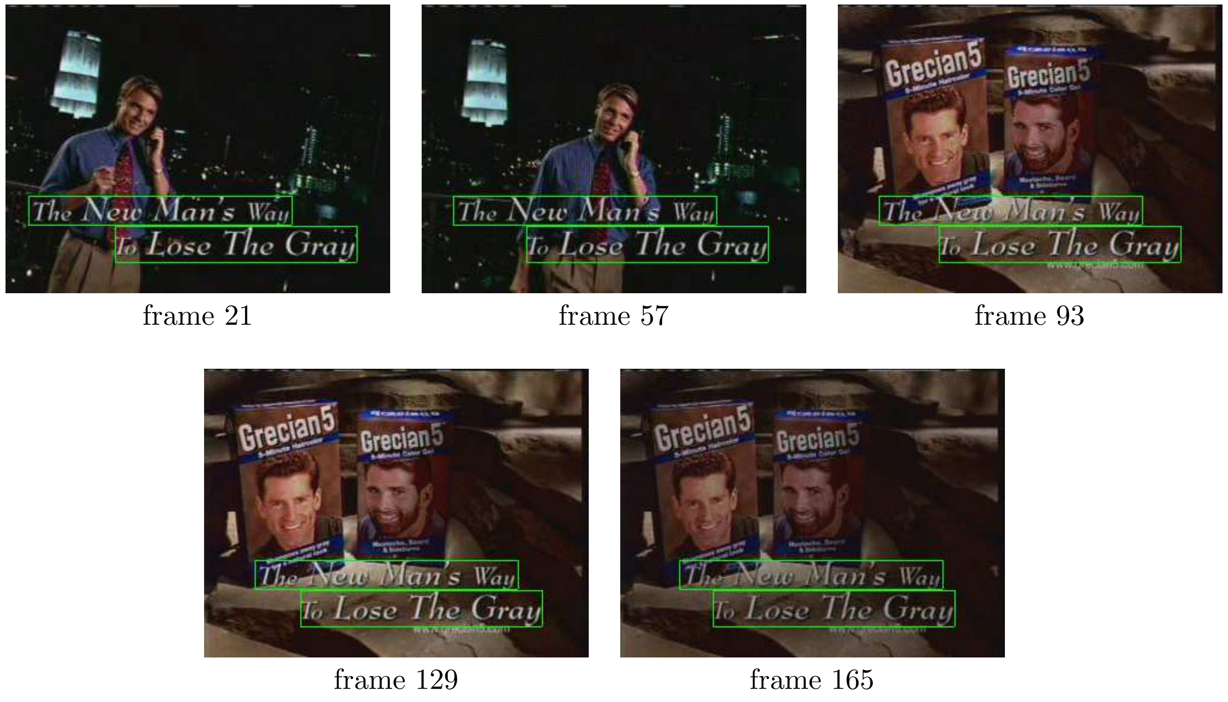


Fig. 3.3. Tracking algorithm applied to "losethegray.mpg" video sequence with a changing background.

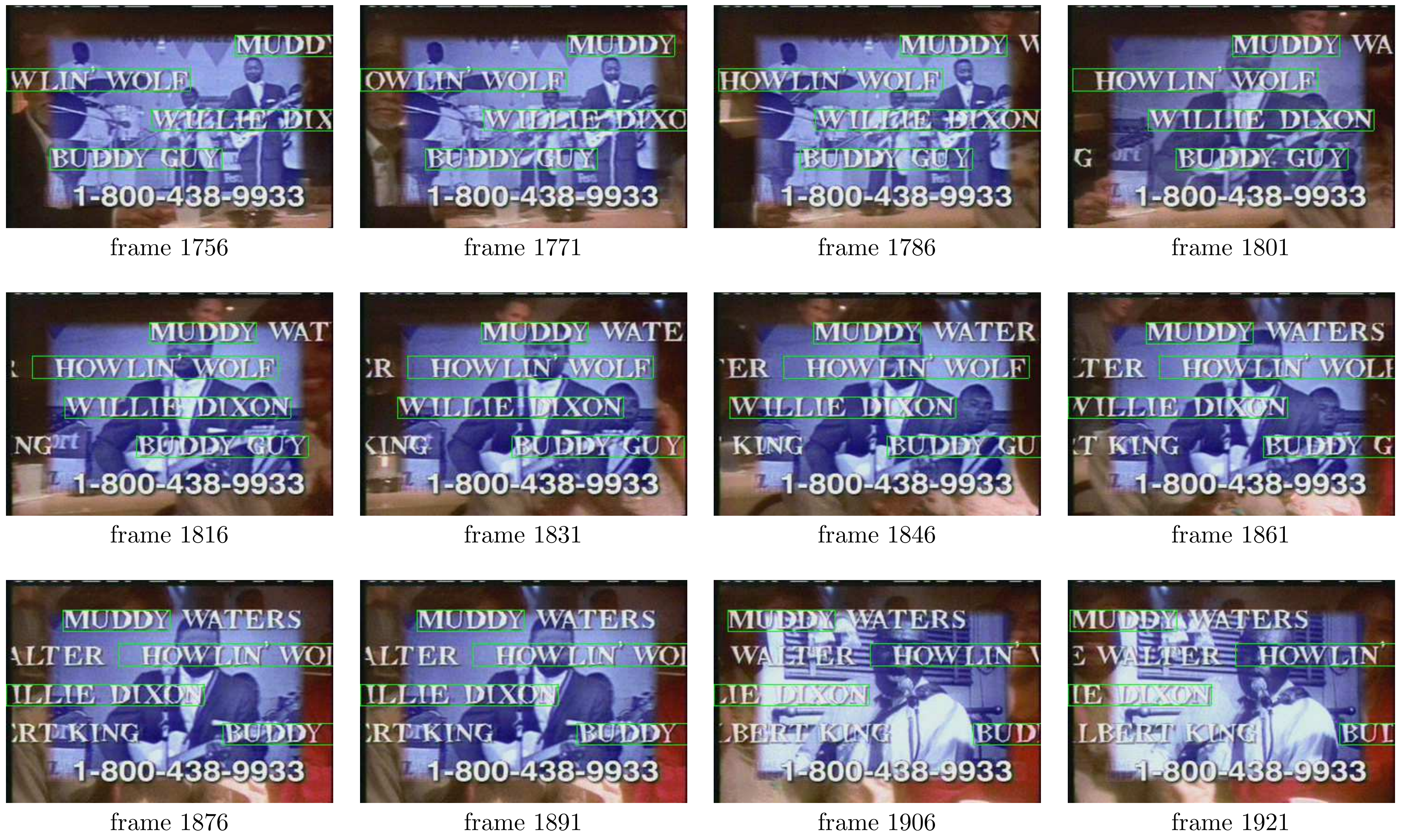


Fig. 3.4. Tracking algorithm applied to “scrolling7.mpg” video sequence with horizontally-scrolling text entering and exiting the frame.

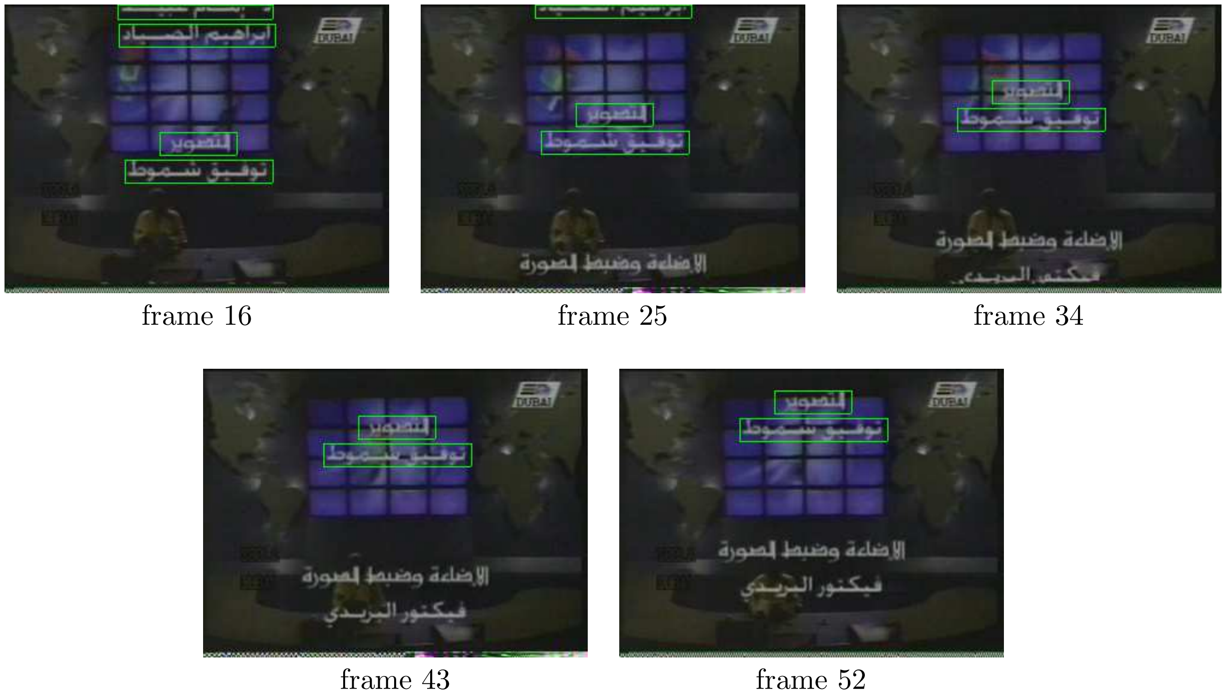


Fig. 3.5. Tracking algorithm applied to “t4.mpg” video sequence with vertically-scrolling caption text.

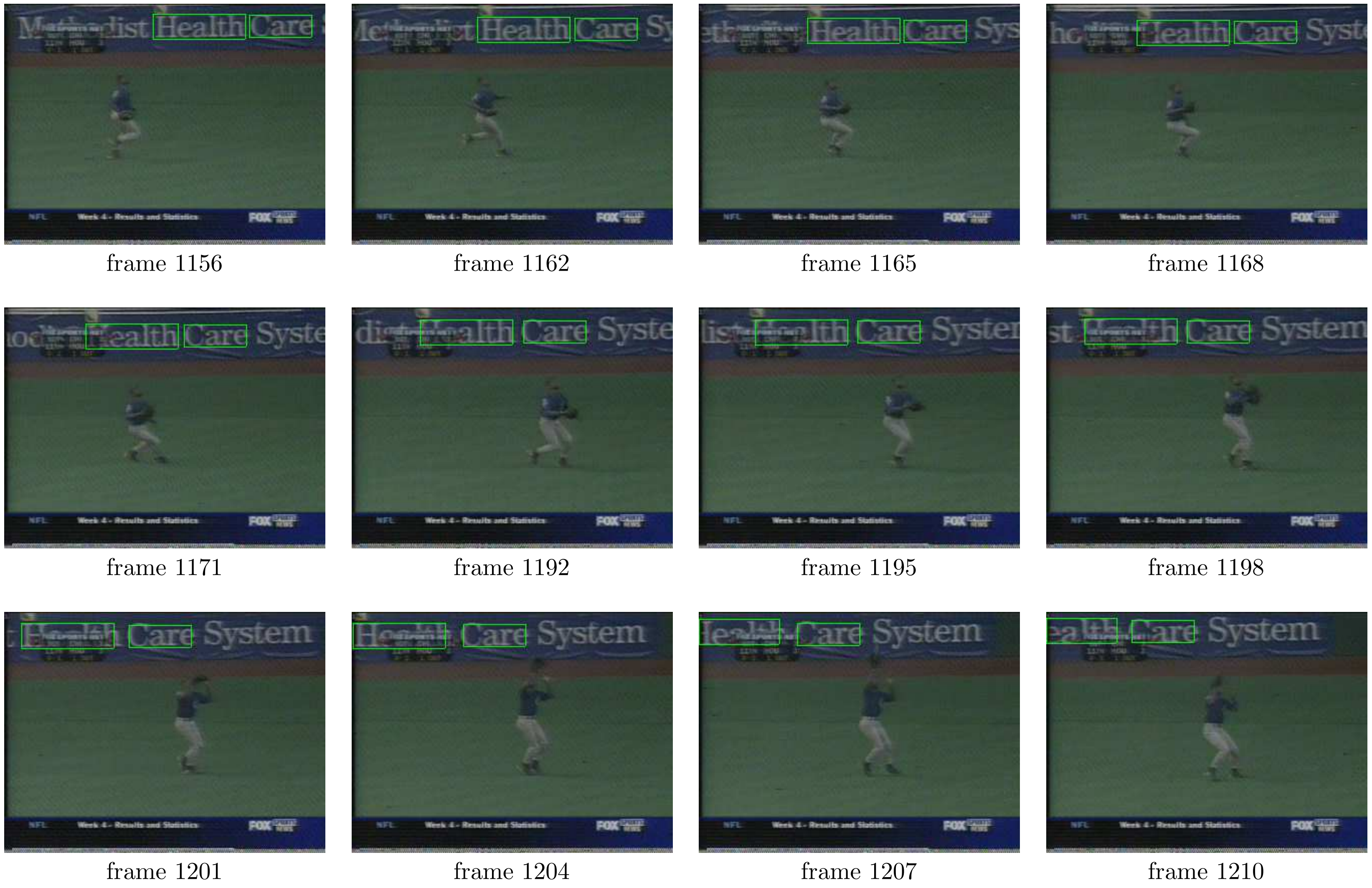


Fig. 3.6. Tracking algorithm applied to scene text in “foxsports.mpg” video sequence.

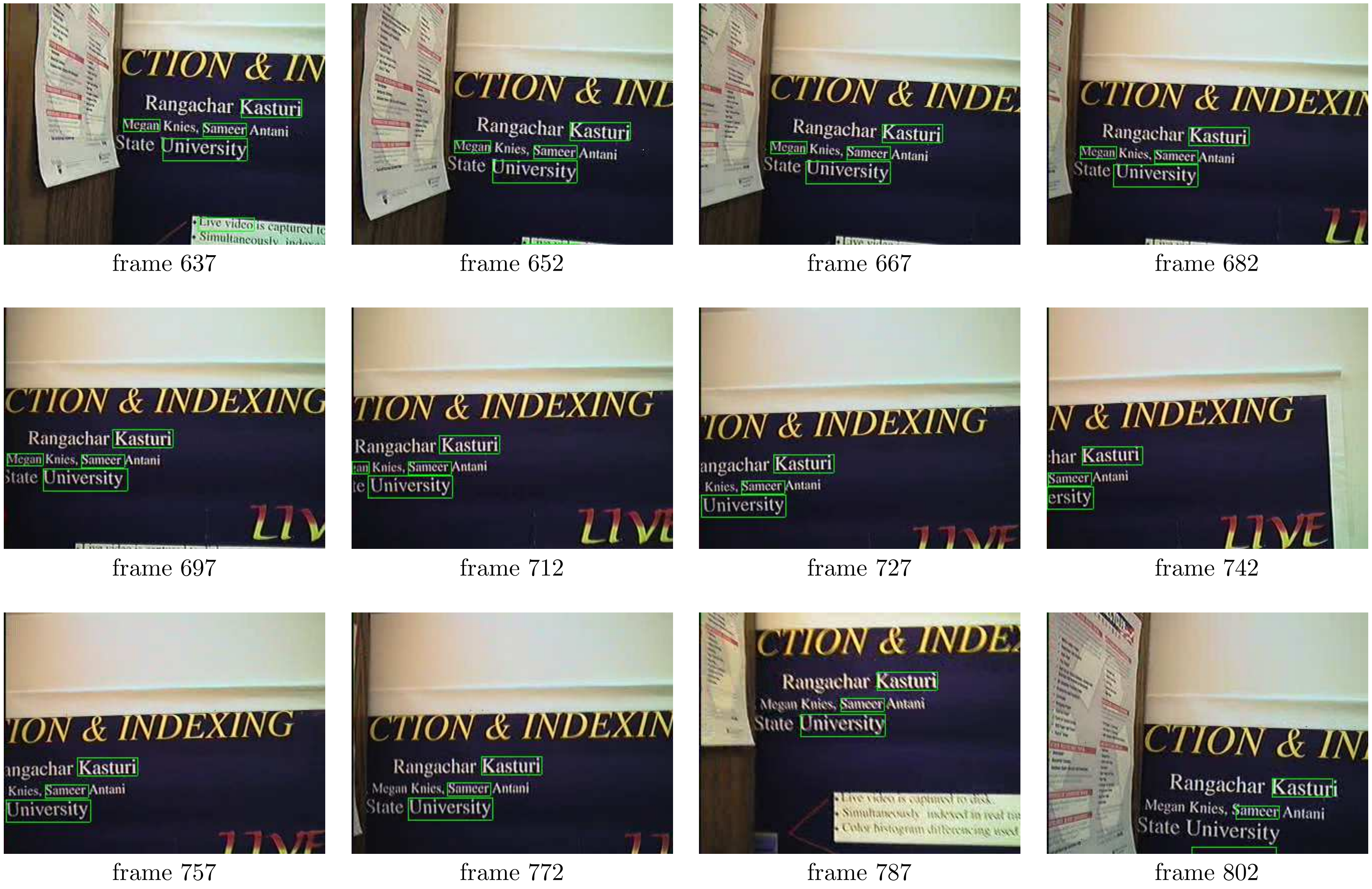


Fig. 3.7. Tracking algorithm applied to scene text with erratic motion in “poster.mpg” video sequence.

Connected component analysis is performed on the binarized text to locate individual characters. The contour of each connected component is traversed and stored as a chain code [12]. Each chain code is then parameterized as two 1-D functions $\theta(t)$ and $r(t)$, using the usual definitions

$$\begin{aligned} \theta(t) &= \tan\left(\frac{y(t)-y_0}{x(t)-x_0}\right) \\ r(t) &= \sqrt{(x(t)-x_0)^2+(y(t)-y_0)^2} \end{aligned}$$

where $(x(t), y(t))$ is a point on the contour, and (x_0, y_0) is some reference point for the connected component. To smooth out noise introduced by imprecise binarization, a low-pass filter is then applied to both functions by convolving with a Gaussian:

$$\begin{aligned} \theta_s(t) &= \theta(t) * G(t) \\ r_s(t) &= r(t) * G(t) \end{aligned}$$

I found empirically that $\sigma = 0.1$ for the Gaussian function worked well.

The resulting smoothed functions $\theta_s(t)$ and $r_s(t)$ represent a signature of the shape of a given character. From this shape, feature points are extracted. I use the points of maximum curvature (critical points) as the features. Zhu & Chirlian's critical point detection algorithm [58] was used in my implementation. The result is a set of points P for each localized text box, indicating the coordinates of each feature point with respect to the upper-left corner of the text box. The coordinates of P are then normalized by text rectangle height and width to give values between 0 and 1.

To decide whether two text boxes A and B in two adjacent frames belong to the same text event, the normalized feature point sets P_A and P_B are examined. For each point p_i in P_A, the point q_j in P_B having the smallest Euclidean distance from p_i is identified. The sum of the distances over all i is calculated. Then the process is repeated in the reverse direction. More formally:

$$D(A,B) \quad = \quad \sum_i \min_j \left(\text{dist}\left(p_i, q_j\right)\right) + \sum_j \min_i \left(\text{dist}\left(p_i, q_j\right)\right)$$

where dist (r, s) is the Euclidean distance between points r and s.

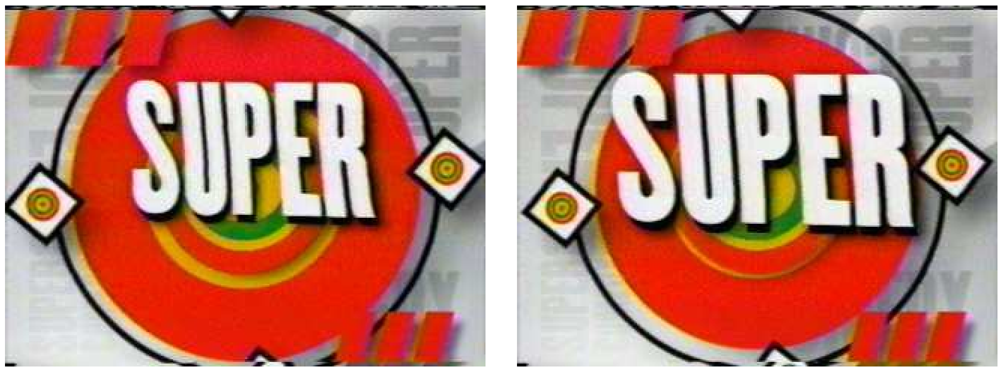
The resulting value $D(A, B)$ for two text boxes A and B is a measure of the difference between the shapes of the two text instances. A low value indicates that A and B likely contain the same text, and hence belong to the same text event. A high $D(A, B)$ value indicates that they probably contain different text. Therefore, the two text boxes are declared to belong to the same text event if $D(A, B)$ is below some threshold T_D.

Figures 3.8 and 3.9 illustrate the process of determining critical point features and comparing them between frames. In Figure 3.8, the images in (a) show two consecutive frames from a video sequence with growing text. The proposed algorithm is applied to these frames to determine whether they contain the same text event. The frames are binarized, as shown in (b). Note that due to imperfect binarization, there are a few small connected components that do not correspond to characters. In (c), the contour of each character has been found, and critical points have been identified. The diagram in D shows the normalized feature points of both the first frame (small, green squares) overlaid on those of the second frame (larger, blue squares). Vectors are drawn between each feature point in each frame and its nearest neighbor in the other frame. It is observed that the lines between feature points are, in general, relatively short, causing a low value for the shape difference $D(A, B)$. This is expected because the text regions in this case correspond to the same text event. Most of the longer vectors in the diagram are caused by the non-character connected components introduced by the imperfect binarization algorithm.

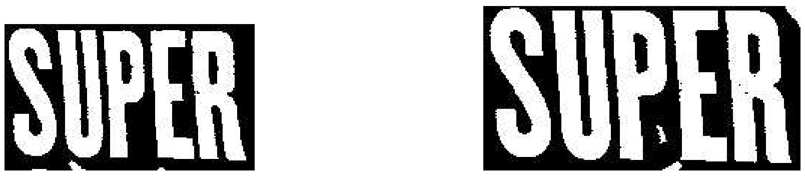
Figure 3.9 is similar, but shows two adjacent frames that have text boxes that do not correspond to the same text event. The binarization and feature point extraction steps are presented in images (b) and (c), respectively. The normalized feature points for both frames are presented in image (d). We observe qualitatively that the vectors are longer and appear more random than those in Figure 3.8(d). This causes the $D(A, B)$ shape difference metric to be high, indicating that the two text boxes are from different text events.

3.4.1 Results

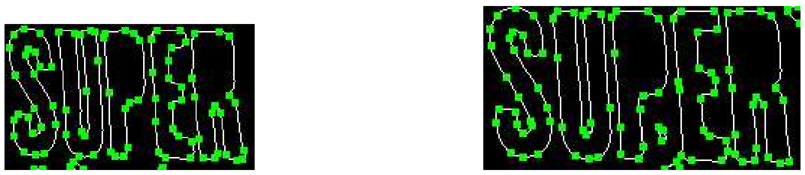
Experimentation was performed to investigate the proposed method's accuracy. A dataset of 27 video sequences, each containing one caption text event, was captured from television commercials. The data was captured in the same manner described in Section 2.5.1. There were a total of 1005 frames in the dataset. A variety of growing, shrinking, moving, and rotating text events were included. The text in each frame was localized manually by me, again using the ViPER ground-truth tool [7]. The 27 individual video sequences were combined into a single video sequence by appending



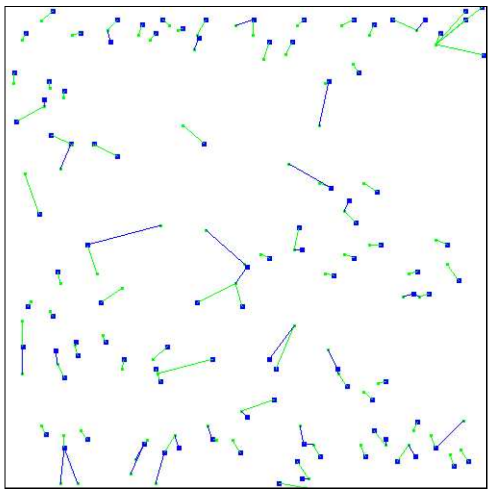
(a)



(b)

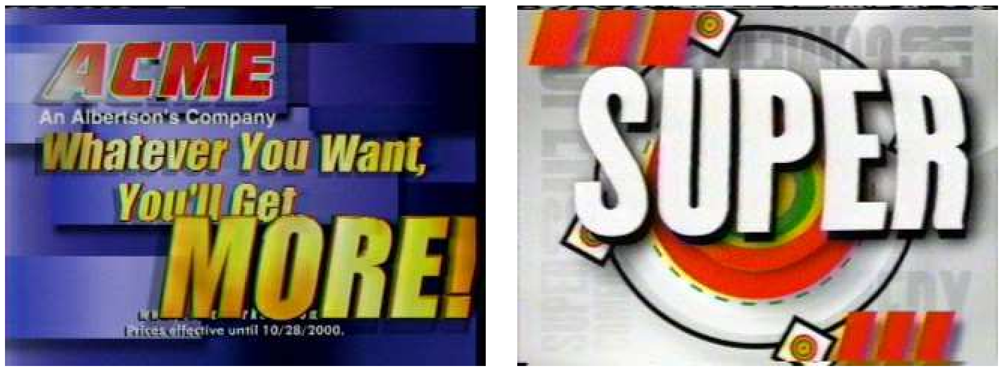


(c)

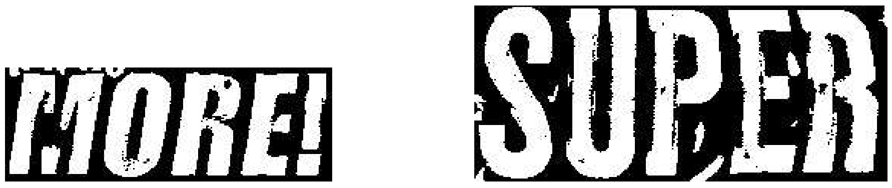


(d)

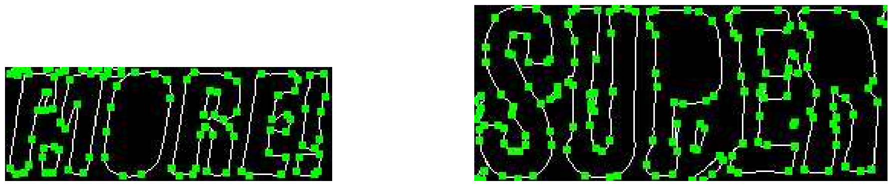
Fig. 3.8. Text feature point extraction and comparison for two consecutive video frames containing the same text event.



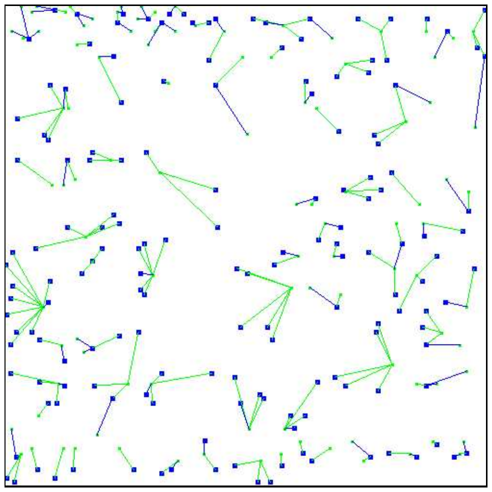
(a)



(b)



(c)



(d)

Fig. 3.9. Text feature point extraction and comparison for two consecutive video frames containing different text events.

together randomly-selected groups of adjacent frames of random lengths from the video sequences. The result was a single 1005 frame video sequence with 111 text events.

The evaluation was carried out as follows. The algorithm was run on the 1005-frame video sequence. For each pair of consecutive frames, the algorithm decided whether the text in the two frames belonged to the same text event or not. If the algorithm correctly determined that the text belonged to the same text event, a *correct detect* was recorded. If the algorithm incorrectly concluded that the two frames shared a text event, a *missed detect* was tallied. If the algorithm incorrectly concluded that the two frames had different text events, a *false alarm* was recorded. Precision and recall statistics were then computed using the definitions presented in Section 2.5.2.

Figure 3.10 presents the results of the experimentation as an ROC curve. Each point on the curve shows the precision and recall achieved for one value of threshold T_D. It is observed from the ROC curve that very good precision and recall can be achieved. The optimal threshold value depends on the needs of the application. For example, for an application in which precision and recall are equally important, a threshold of $T_D = 0.55$ is optimal, at which precision and recall are both 97.5%. In a video indexing application, however, it is likely that a high recall would be more important than a high precision. This is because it is very important that all text events are entered at least once into the index. While it is preferable that each event is entered exactly once, duplicate entries are not harmful. It is observed from the ROC curve that it is possible to obtain a recall of 100% with a precision of 96% at $T_D = 400$.

Figure 3.11 shows sample qualitative results of the combined text detection, localization, and tracking steps. Images (a) through (h) show eight consecutive frames from a commercial featuring rotating, shrinking, and growing text, and the boxes localized by our algorithm. The tracking algorithm concluded that the text boxes in images (a), (b), (c) and (d) correspond to the same text event. Similarly the algorithm determined that images (f) through (h) belong to a separate text event. Image (e) confused the algorithm somewhat due to the overlapping text. The tracking algorithm concluded that image (e) belonged to its own, one-frame text event.

Figure 3.12 demonstrates the algorithm's effectiveness on text occurring against complex, unconstrained backgrounds. The tracking algorithm correctly identified the text in image (a) as one text event, and the text occurring in images (b) through (f) as another text event.

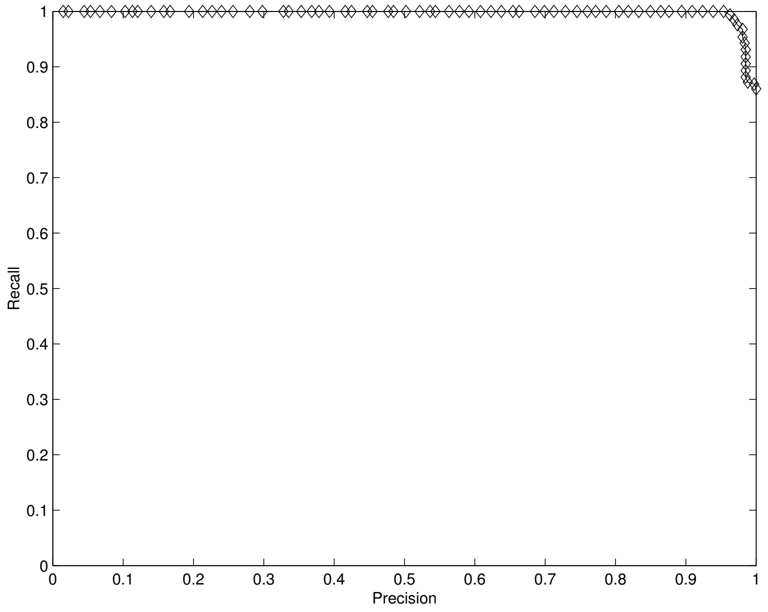


Fig. 3.10. ROC curve of tracker performance.

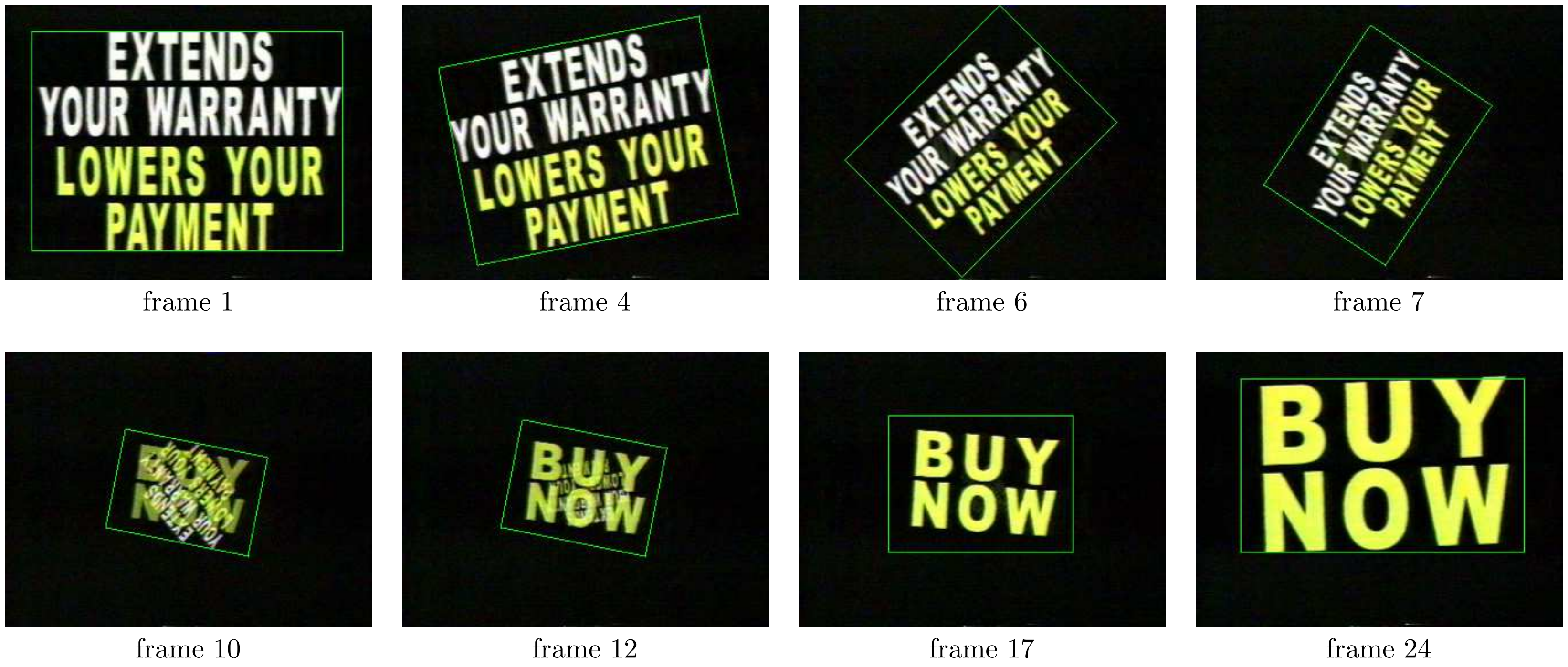
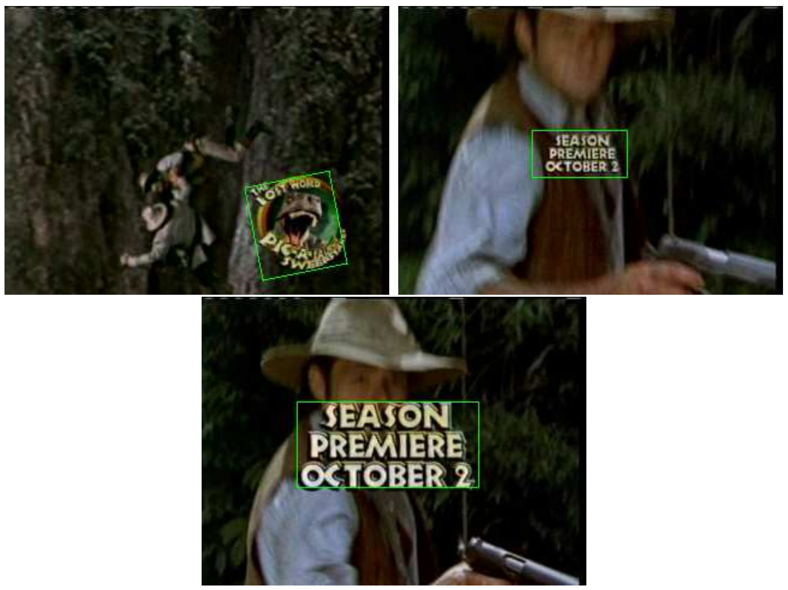
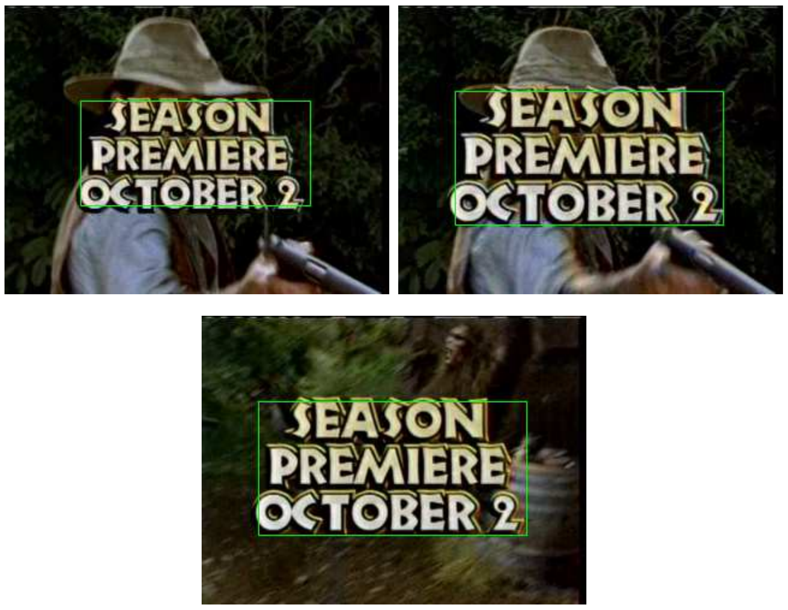


Fig. 3.11. Sample results of combined text detection and tracking on growing, shrinking, and rotating text.



(a) (b) (c)



(d) (e) (f)

Fig. 3.12. Sample results of combined text detection and tracking on growing text against an unconstrained background.

Chapter 4

Binarization of caption text

4.1 Introduction

Binarization is the process of separating character strokes from the background. That is, given a localized region of the color video frame known to contain text, binarization produces a binary image of the text. The detection problem studied in Chapter 2 concerns classifying video frame regions as text or background; the binarization problem concerns classifying individual pixels as text or background.

Binarization is necessary to bridge the gap between localization and recognition. The eventual goal of a text extraction system is to recognize the text appearing in video. Optical character recognition (OCR) in the context of document images has been extensively studied [35]. If possible, we would like to apply these extensively-studied, highly-refined OCR algorithms to the text-in-video extraction problem. However, most recognition algorithms expect images resembling documents, with text strokes in black ink against a white background. In contrast, text occurring in video can be of any color and can appear against complex backgrounds. It is the responsibility of a binarization algorithm to convert the complicated text regions occurring in video frames to the simple binary images required by OCR.

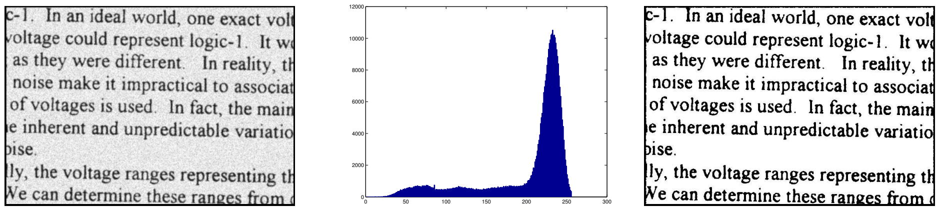
4.2 Challenges of binarization of video frames

Binarization of text has been studied in the document image analysis domain. In most applications, images of documents are obtained by a grayscale optical scanner. The original document usually has text in black ink appearing against a white background. Noise introduced during the scanning process may cause the grayscale image to have more than two gray levels. However, the histogram of the grayscale image is still strongly bimodal, with one peak corresponding to text pixels and the other corresponding to background pixels. Thresholding (either locally or globally) can be performed at a well-chosen gray level in the valley of the two peaks to give accurate binarization results. Many techniques have been studied for finding the ideal threshold (e.g. [21, 55, 44]).

Figure 4.1 illustrates this binarization process for a document image. The histogram for the fragment of a document image in (a) is shown in (b). The strong peak at about 225 corresponds to background pixels, while the weaker peaks around 75 and 125 corresponds to text pixels. A threshold at 160 gives good binarization results, as shown in image (c).

It may seem that a similar technique can be applied to the binarization of caption text. Like text in documents, caption text in video is usually designed to be easily readable by human viewers. Contrast with the background appears high. Text stroke color usually appears relatively uniform. It seems logical that simple thresholding could be applicable to video frames.

Unfortunately, I have found that it usually is not. Upon closer examination, it is found that text stroke color actually varies widely, even when it appears uniform to the



c-1. In an ideal world, one exact volt
voltage could represent logic-1. It wo
as they were different. In reality, th
noise make it impractical to associat
of voltages is used. In fact, the main
e inherent and unpredictable variatio
bise.
lly, the voltage ranges representing th
We can determine these ranges from c

c-1. In an ideal world, one exact volt
voltage could represent logic-1. It wo
as they were different. In reality, th
noise make it impractical to associat
of voltages is used. In fact, the main
e inherent and unpredictable variatio
bise.
lly, the voltage ranges representing th
We can determine these ranges from c

(a) (b) (c)

Fig. 4.1. Histogram-based thresholding of a document image. *(a):* A portion of a document image, *(b):* Its histogram, *(c):* Results of thresholding at gray level 160.

human eye. Because the background in a video frame is unconstrained, the same colors making up a text stroke may also occur in background objects. Figure 4.2 illustrates these problems with the sample video frame shown in (a). A localized text region is shown in (b), and its histogram is presented in (c). The histogram is bimodal with a valley at about gray level 75. But thresholding at this gray level gives the poor results in (d). This is because the peaks of the histogram are due to variation in background color instead of the separation between text and non-text pixels. The binarization was repeated at a higher threshold (gray level 130), but this also led to the disappointing results in (e). Even the result of double thresholding, shown in (f), failed to give good results. In fact, any histogram-based thresholding scheme will fail in this case, because many pixels in the text strokes share the same color as pixels in the background.

The challenges of text binarization in video are summarized as follows:

- **Low resolution:** Video frames are typically captured at resolutions of 320×240 or 640×480 pixels. In contrast, document images are typically digitized at resolutions

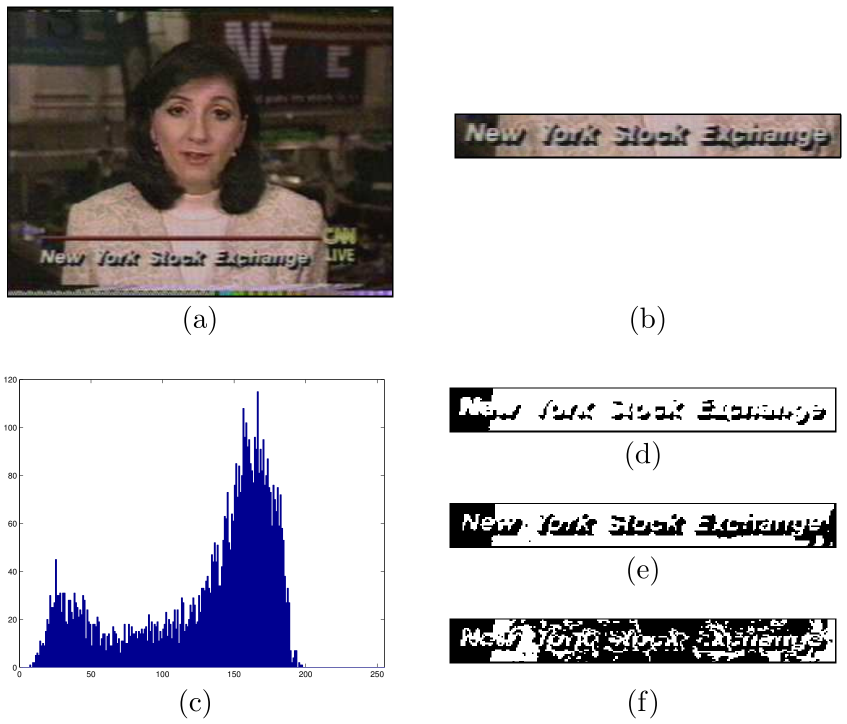


Fig. 4.2. Histogram-based thresholding gives poor results on unconstrained video frames. *(a):* original video frame; *(b):* localized text region; *(c):* histogram of text region; *(d), (e):* results of thresholding on gray levels 75 and 130, respectively; *(f):* result of double thresholding at gray levels 100 and 175.

of 300 dots per inch or greater. For example, a lowercase letter "e" in the document image in Figure 4.1(a) is about 21 pixels wide and 26 pixels tall. A lowercase letter "e" in the localized video text box in Figure 4.2(b) is just 9 pixels wide and 8 pixels tall.

- **Unknown text color:** Text can have arbitrary color.
- **Unconstrained background:** The background can have colors similar to the text color. The background may include streaks that appear very similar to character strokes.
- **Color bleeding:** Lossy video compression may cause colors to run together. This blurs the edges between text strokes and background pixels.
- **Low contrast:** Low bit-rate video compression can cause loss of contrast between character strokes and the background.

4.3 Review of prior binarization work

This section summarizes some of the past approaches to binarization of text in images and video frames. I have classified these algorithms into four main approaches to binarization: global thresholding, local thresholding, color clustering, and neural networks. The algorithms belonging to each approach are now discussed.

4.3.1 Global thresholding

Global thresholding is a common approach. Algorithms in this category determine some grayscale threshold, and apply it to all pixels in a localized text region. The methods differ in the strategy for choosing the threshold, and in the preprocessing and post-processing steps.

- Wu *et al* [57] binarize text found in web page images by smoothing the grayscale image and then thresholding at the valleys on either end of the grayscale histogram. This allows for both light text and dark text to be binarized. The algorithm does not determine whether the text is lighter or darker but instead generates two outputs, one for each case.

- LeBourgeois [22] assumes that the dominant portion of the image histogram is the background. The global threshold is found by an entropy-maximizing scheme [55]. A post-processing stage splits characters inadvertently connected by the thresholding. Characters are assumed to be of a fixed font size.

- Sato *et al* [45] apply filters designed to detect vertical, horizontal, and diagonal line elements to localized text regions in videos of newscasts. The union of the outputs of all filters is taken. Final binarization is performed by thresholding at a fixed, pre-set threshold. It is assumed that text is white.

- Messelodi and Modena [33] present a system for extracting text from book covers with plain backgrounds. They use a simple global thresholding scheme at the tails of each side of the histogram. Their method considers binarization of oriented text.

- Agnihotri and Dimitrova [1] binarize caption text appearing in video. They process only the red plane of an image, under the assumption that text of interest is white, yellow, or black. Thresholding is performed at the average pixel value of the localized text region. The average of the pixels on the border of the text region is also computed and assumed to approximate the background color.

Global thresholding has been found to be useful in some applications. However, as discussed in Section 4.2, global thresholding is not able to perform well on caption text occurring in general-purpose, unconstrained video.

4.3.2 Local thresholding

Local thresholding passes a small window over a localized text region. A threshold is computed based on the pixels underneath the window. The pixel at the center of the window is then binarized based on this threshold.

- Ohya *et al* [38] use a combined detection/binarization stage to extract characters from scene images. Text is assumed to be either black or white. Regions of the image with bimodal histograms are assumed to be text regions. Local thresholding is performed on these regions using the threshold selection algorithm described in [39]. Shape and size heuristics are applied to filter out non-text strokes.

- Lee and Kankanhalli [23] also use a combined detection/binarization stage. After quantizing the gray levels in the image, detection is performed by searching for strokes with uniform gray level. Each potential character is thresholded using the gray level of its boundary. Post-processing removes components with suspicious aspect ratios, contrast, and fill ratios.

- Winger *et al* [54] use a modified form of Niblack's Multiple and Variable Thresholding scheme [37], which employs variable thresholds based on mean local pixel intensity. After calculating the variance, the modified scheme uses a different multiplier and exponent. Our implementation of the method (by Ryan Keener) did not produce good results. Subsequent correspondence with the author suggested that good results are possible only when algorithm parameters are manually tuned to appropriate values for a given image.

- Shim [48, 49] thresholds each character stroke component box individually by analyzing the grayscale histogram. The threshold is selected using the iterative method described in [44].

4.3.3 Color clustering

Most of the work discussed so far operates only on the luminance plane of images and video frames. The algorithms in this section incorporate color information. It is assumed that the strokes in a text instance had uniform color, although in the compressed video stream color bleeding and quantization may have introduced noise. Color clustering can then be applied to group together pixels of nearly the same color.

- Garcia *et al* [9] quantizes and clusters color pixels in localized text regions in the HSV color space. It is assumed that after clustering, all text pixels will correspond to a single cluster. That cluster is identified by choosing the cluster with the most periodic vertical profile.

- Wong *et al* [56] continue to perform color clustering on localized text regions until two clusters are obtained. The two clusters are assumed to correspond with text pixels and background pixels. My experimentation with color clustering has indicated that this assumption rarely holds due to complex backgrounds that contain colors similar to the text color.

- Mariano *et al* [31] performs text region detection and binarization in one step. Color clustering in the L*a*b* color space [42] is performed on individual scan lines of a video frame. The patterns of clusters occurring in neighboring scan lines are analyzed to find regularly-spaced streaks corresponding to text strokes. It is assumed that text is precisely horizontal.

Color clustering seems to be a promising approach. Unfortunately, the computation demands of color clustering seem to be prohibitive on today's systems. An implementation of Mariano's algorithm obtained from the author and optimized for speed by me required about 50 minutes to process a 1-second video clip on an SGI Octane workstation.

4.3.4 Neural Networks

Some work has applied neural networks to the problem of video text binarization. For example, Shin *et al.* [50] perform detection and binarization in one step by applying a support vector machine (SVM) to classify each pixel as text or non-text. The features used as input to the SVM are the grayscale pixel values in a local neighborhood. A hierarchical strategy is employed to handle text of various sizes.

4.3.5 No binarization

A final approach is to skip the binarization step altogether. There has been recent interest in OCR algorithms that operate directly on grayscale images without binarization [35]. Proponents of this system say that information is inherently lost in the binarization process. Lienhart [28, 30] describes a custom OCR package for recognizing text in video frames without binarization. Unfortunately it does not work well when text appears against complex backgrounds. The accuracy of their grayscale OCR package was unable to compete against the accuracy typical of commercial OCR packages. As the state-of-the-art in grayscale image recognition improves, circumventing the binarization stage may be a viable option.

4.3.6 Remarks on the state-of-the-art

From this survey of recent approaches to the binarization of text in video, I observe the following. Most existing binarization approaches make assumptions that are valid for document images but not for text appearing in general-purpose video. Many approaches use simple histogram thresholding methods which assume that the background is simple. They create noisy binarizations when applied to text occurring on complex backgrounds. Some algorithms work only for text of a certain color, *etc.* which severely limits their usefulness for general-purpose video.

In contrast, I observe that most of these text binarization algorithms do not take advantage of reasonable assumptions about text in video that can improve performance. It can be assumed that text in video usually persists for more than one frame. Multiple frames can be integrated to give better binarization results. Characters in a text instance usually have uniform stroke width and color. Characters are evenly-spaced, aligned, and have roughly uniform size. Reasonable upper and lower bounds exist on the size of text characters possible in a video frame.

In the following section, a binarization algorithm is presented that takes advantage of these additional assumptions. Unlike previous work, it is designed to work well with text appearing against complex backgrounds, and does not make *a priori* assumptions about text color.

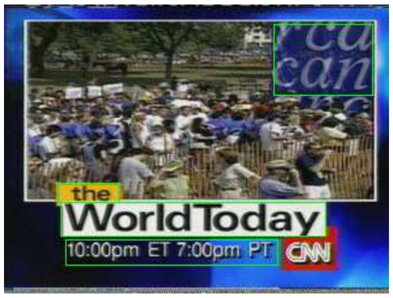
4.4 An algorithm for text binarization in video frames

An algorithm is now presented for binarization of text in video. Each step is explained in detail in the following sections. Figure 4.3 illustrates each of the steps on a sample video frame.

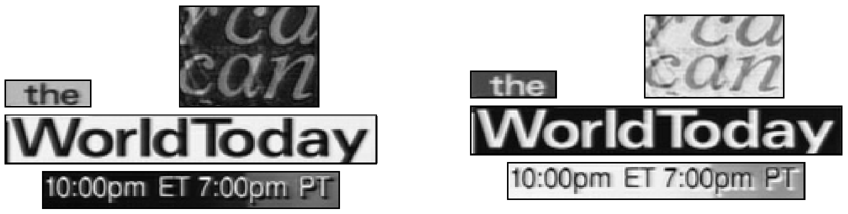
4.4.1 Temporal integration

There are several motivations for analyzing more than one frame during the binarization process. Lossy video compression methods introduce noise, but the noise varies from frame to frame. Simple temporal averaging can reduce such noise. Temporal integration is also helpful for background removal. Caption text often remains stationary while the background behind it changes or moves. Or the text may move, causing the background behind the text to change. In either case, temporal averaging can be used to smooth out the background.

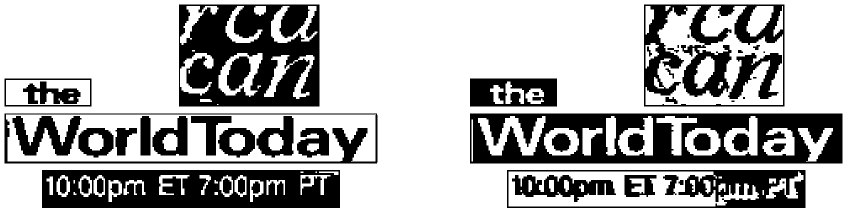
I have developed a text tracking module (discussed in Chapter 3) to determine the pixel-accurate location of a text event in each frame. During temporal integration,



(a)Video frame with localized text regions

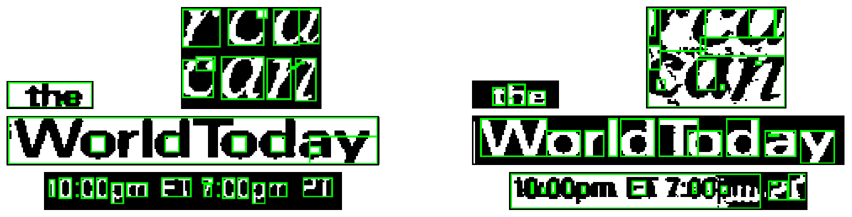


(b) Temporal, resolution, and contrast enhancement is applied on each text box. The inverse of each text box is obtained. (§4.4.2)

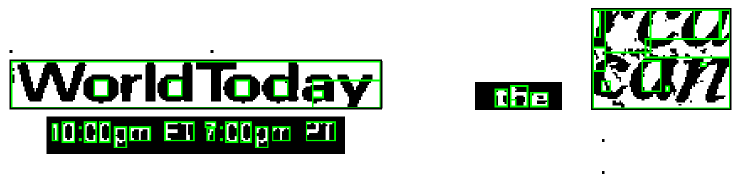


(c) Logical level thresholding is applied to both polarities of each text box. (§4.4.3)

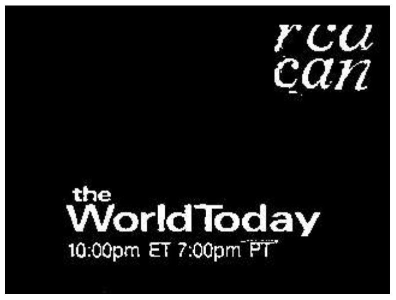
Fig. 4.3. Steps of the binarization algorithm (continued on next page).



(d) Connected components are found. Heuristics are applied to remove non-character-like components. (§4.4.4)



(e) Alignment and size of components are used to choose the polarity for each text region. (§4.4.5)



(f) Final segmentation result.

Figure 4.3 (continued)

the region location in each frame identified by the tracker are averaged. Note that this assumes that the text remains rigid as it moves. If it does not remain rigid, the temporal averaging procedure will blur the text strokes in addition to the background. To prevent this, the confidence of the text tracker is monitored. If the confidence falls below a threshold, it is likely that the text is changing over time, and temporal averaging is disabled for processing the text instance.

Figure 4.4 shows an example of temporal averaging applied to a localized text region. Image (a) shows a sample frame from a sequence of 60 frames having a stationary text event appearing on a moving background. Images (b) and (c) show the localized text region from two frames in the sequence. Note that the complex background is quite prominent in both images. Image (d) shows the result after performing temporal averaging on the text region over its 60-frame lifetime. The background complexity has been reduced, and the contrast of the text against the background has been substantially improved.

4.4.2 Resolution and contrast enhancement

A simple linear interpolation step is used to double the resolution of the image. Although this resolution enhancement step cannot truly recreate lot resolution, I have found that even simple linear interpolation improves the binarization results. More sophisticated resolution enhancement schemes could be investigated. For example, there has been some work in using motion information in video to improve resolution [52]. This approach could be applied to resolution enhancement of moving text.

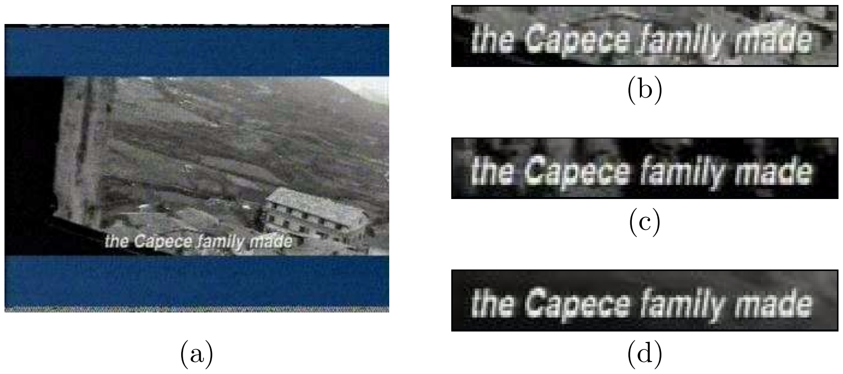


Fig. 4.4. Temporal averaging reduces background noise and improves contrast. *(a):* sample frame from a video sequence; *(b) and (c):* localized text region in two frames of the sequence; *(d):* result of temporal averaging.

The contrast between the text and the background in a localized text region may be quite low. To improve the contrast, simple grayscale histogram stretching [12] is performed.

4.4.3 Logical level thresholding

Some document analysis work has considered the problem of binarizing text occurring in noisy document images. This problem shares similarities with our problem of extracting text occurring against complex backgrounds. Kamel and Zhao [20] evaluate seven binarization techniques on noisy bank check images. Their novel method, logical level thresholding, was shown to perform the best.

Logical level thresholding works as follows. A maximum stroke width W is assumed. Then for every pixel p in the grayscale image, the eight pixels P_i at radius W and angles $\frac{i\pi}{4}$ with $i = 0, 1, ..., 7$ from p are considered. The local average avg_i of the

$(2W+1) \times (2W+1)$ neighborhood around each P_i is computed. Pixel p is declared to be text if for some $j = 0, 1, 2, 3$, all of avg_j, $avg_{(j+1)mod8}$, $avg_{(j+4)mod8}$, and $avg_{(j+5)mod8}$ are greater than p by some threshold T.

The algorithm's strength over other binarization techniques is that it enforces restrictions on uniformity of stroke grayscale level, uniformity of stroke width, and bounds on stroke width. This leads to less noise in the binarized output.

I applied the algorithm to binarization of localized text regions in video frames. After the temporal averaging, resolution enhancement, and contrast stretching steps described above, the image region is converted to the L*a*b* color space [42]. This color space mimics the human visual system's perception of luminance and color, so that text that appears to be high-contrast by a human has numerically high contrast in L*a*b* space. Logical level thresholding is then applied on the luminance plane. Logical level thresholding requires two parameters, the maximum stroke width W and the contrast threshold T. However I observed that the algorithm's performance is relatively insensitive to the choice of parameters. In my implementation, I use $T = 5$, which is a good compromise between allowing binarization of low-contrast text and preventing noise. My choice of stroke width W is proportional to the size of the input video frame. For a frame resolution of 320×240, $W = 10$ works well. This does not limit the algorithm's practical ability to binarize text of different sizes, because a stroke width of 10 pixels corresponds to text that nearly fills the video frame.

Logical level thresholding requires that the text stroke color is darker than the background. In our application this is not an acceptable assumption because we wish to extract text of any color. I tried to modify the logical level algorithm to allow strokes darker and lighter than the background by modifying the thresholding step. Unfortunately, this relaxes the restriction of stroke color consistency and creates noise. Instead, I assume that all text strokes within a localized text region are either lighter than or darker than the background. Logical level thresholding is then applied to both the original region and its inverse to produce two independent binarized outputs. The choice of correct polarity is delayed until step 4.4.5.

4.4.4 Character candidate filtering

Connected component analysis is performed on both output images of logical level thresholding. These connected components are either characters or noise. Heuristics are applied to preserve characters while removing noise. These heuristics are:

- **Minimum character size:** Components having height less than 5 pixels or area less than 12 pixels are removed. Connected components this small are unlikely to be characters. Even if they are characters, it would probably not be possible for the OCR module to recognize them. Note that a minimum character width is not enforced because lowercase "l" characters are often only one pixel wide.

- **Aspect ratio bounds:** A component whose aspect ratio $\frac{width}{height}$ is very large or very small is discarded. These components are often horizontal or vertical lines, or other noise. We currently use the range $[0.1, 1.0]$ as acceptable aspect ratios.

4.4.5 Choice of binarization polarity

As noted earlier, the logical level thresholding was applied on both the original localized text region and its inverse. In one of the polarities, the text is lighter than the background; in the other, it is darker. Logical level thresholding applied to the dark-text image will result in a binarization of the text. When applied to the light-text image, the algorithm will attempt to binarize the background. The correct binarization will have connected components with spacing, size, and alignment consistent with text characters. The incorrect binarization has irregular components due to its attempt to binarize the background.

My algorithm chooses the correct binarization by analyzing several statistics about the connected components in each binarization polarity. A voting strategy is used. For each statistic, a vote is cast for the binarization that demonstrates the more text-like quality. The binarization with the most votes is chosen as the final binarization output. The criteria used in my implementation are:

- **Height similarity:** Low standard deviation of connected component heights
- **Width similarity:** Low standard deviation of connected component widths
- **Spacing consistency:** Low standard deviation of horizontal distance between adjacent component centers
- **Horizontal alignment:** High number of pairs of components whose bottoms share roughly the same vertical scan line
- **Character-like aspect ratio:** Low difference between average component aspect ratio and 1.0
- **Clean spacing:** Low number of pixels that occur within the bounding box of more than one connected component

- **Periodicity of vertical projection:** The even spacing of text characters should cause the vertical projection to be roughly periodic. The more periodic of the two polarities is chosen using the method presented in [9]

Note that the number of votes for the winner is a confidence measure. In most cases, I have observed that the voting results in a clear majority, indicating a high confidence that the correct binarization was chosen. A close vote indicates a lower confidence. In these cases, it may be appropriate to pass both binarizations to the OCR module, and choose the one with the higher recognition confidence. A close vote may also indicate that the localized region does not actually contain text.

4.5 Results

Figure 4.5 presents results of the binarization algorithm on localized text boxes in sample video frames. For comparative purposes, the outputs from my implementation of the binarization method proposed by Agnihotri *et al* [1] are also presented. This algorithm was selected for comparison because it is the most recent complete text extraction system found in the literature designed for general-purpose video.

Column (a) in Figure 4.5 shows the localized text regions used as input to the binarization algorithms. Column (b) presents the output of Agnihotri's binarization method. Column (c) presents the output of my method. It is observed that the binarizations produced by my algorithm are significantly cleaner than those produced by Agnihotri. This is especially apparent in the middle row of images in the figure. This is an example of how Agnihotri's method suffers from the inherent problems with global thresholding discussed in Section 4.2. Also, the algorithm's assumption that the background color can be determined by averaging the pixels on the border of the localized text region is violated in this case. The background color varies from very dark in the lower-left corner of the text region, to bright in the upper-right corner.

Figure 4.6 shows some examples of the binarization algorithms applied on very challenging video frames. These examples highlight some of the problems with my binarization algorithm. The first row of images shows the output of the binarization algorithms on Arabic caption text. The output of my algorithm, shown in column (c), has given reasonable binarizations for three of the text boxes, but it has failed to binarize the top text box accurately. The problem is that the algorithm has selected the incorrect binarization polarity for this text box. This can be explained by reviewing the polarity selection criteria described in Section 4.4.5. Many of the criteria assume that connected components in the binarization correspond to text characters. This assumption is not compatible with the Arabic script in this example, in which characters are connected together. I conclude that my polarity selection criteria will give accurate results only for scripts with separated characters. Alternative selection criteria could be devised to handle other scripts. Agnihotri's algorithm does not suffer from this restriction, and has chosen the correct polarities. However, their binarization is still quite noisy.

The second row of images in Figure 4.6 shows the algorithms applied to very small text. The average character size of this text is about 8 pixels high by 5 pixels wide, with a sub-pixel stroke width. Agnihotri's algorithm generates illegible binarization in this case. My binarization algorithm's results are reasonable, but are still probably not clean enough to be accurately recognized by a standard OCR module. Binarizing text of such

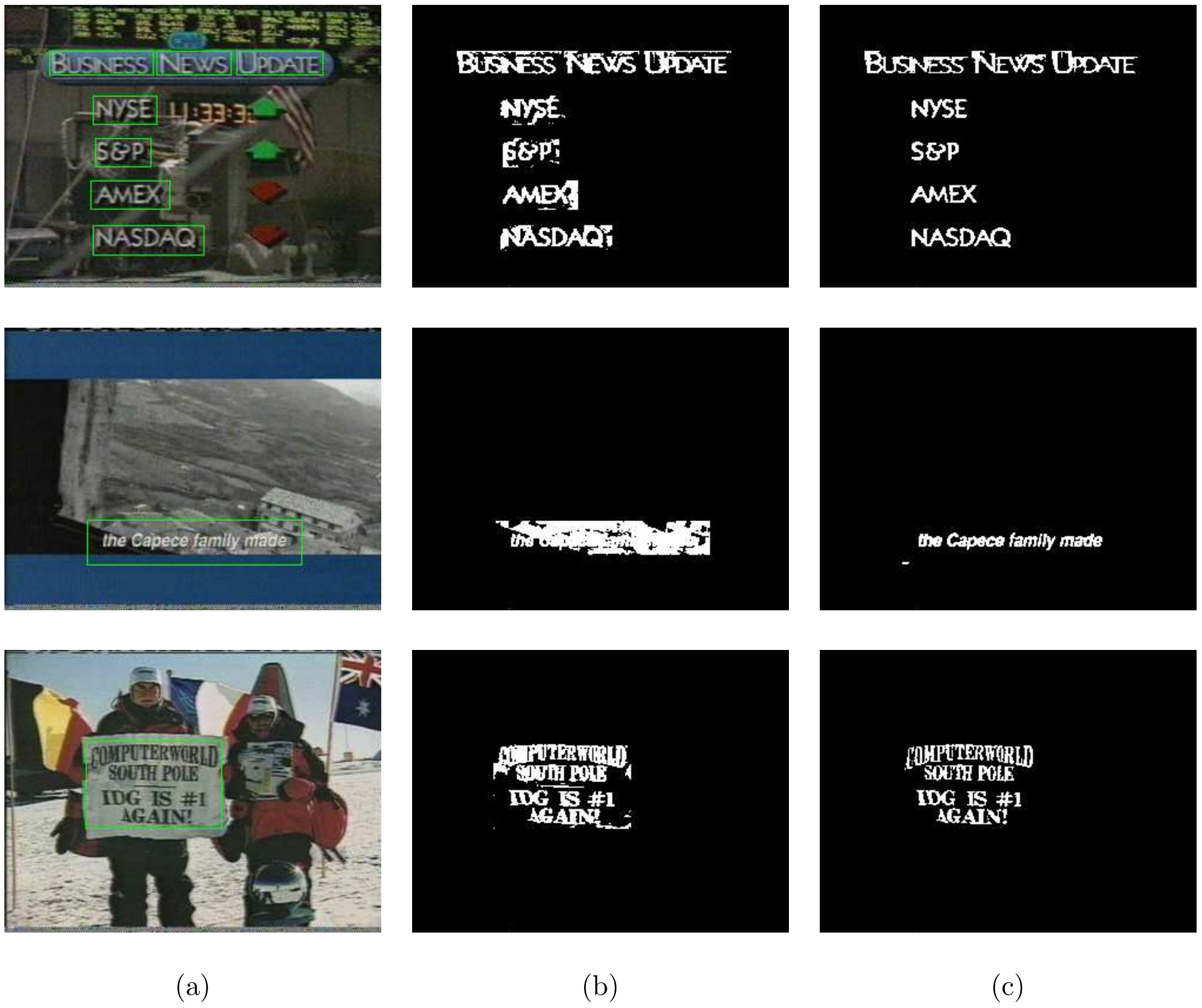


Fig. 4.5. Binarization results for sample video frames. *(a):* localized text regions; *(b):* output of binarization algorithm by Agnihotri *et al* [1]; *(c):* output of my binarization algorithm.

a small size is extremely challenging, and further research will be necessary to develop algorithms that can do it accurately.

The third row of images in Figure 4.6 presents the results of binarization on text with very low contrast with the background. Agnihotri's algorithm incorrectly chooses the polarity of the text, and attempts to binarization the shadows behind the characters. The incorrect selection of polarity is due to their assumption that the background color can be determined by averaging the pixels along the text region border. The contrast between background and foreground is so low that this assumption fails in this case. My algorithm produces better results, but there is still much noise that would probably cause recognition to fail. Binarization of low-contrast text is another area that requires further research.

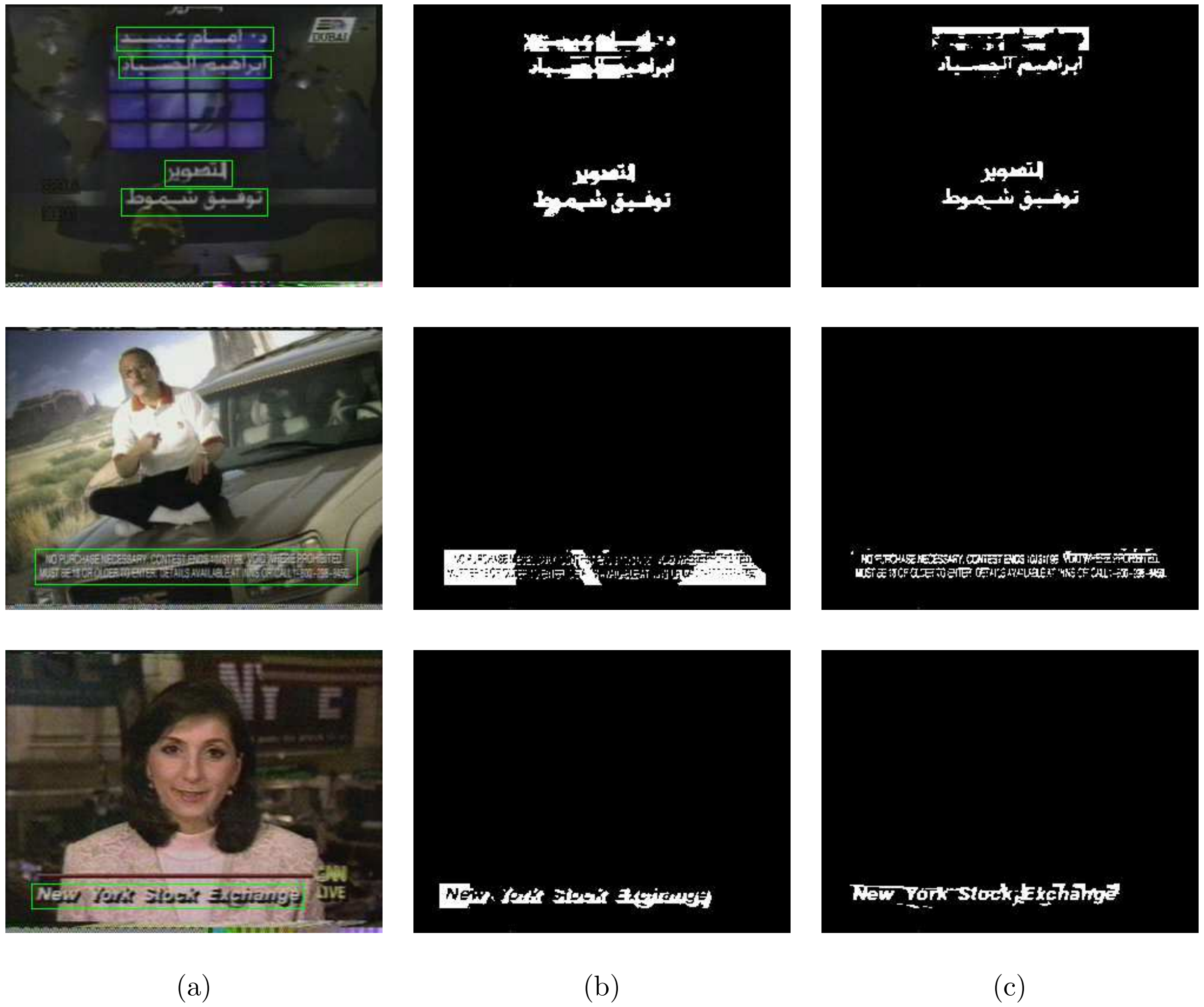


Fig. 4.6. Binarization results for very challenging video frames. *(a):* localized text regions; *(b):* output of binarization algorithm by Agnihotri *et al* [1]; *(c):* output of my binarization algorithm.

Chapter 5

Summary and Conclusions

This thesis has discussed the extraction of text events from general-purpose video. Text appearing in video is one feature that gives insight into a video's content. Automatic extraction of text would therefore be useful in video indexing applications. I have discussed the several sub-problems of text extraction, including detection, localization, tracking, and binarization. These are significantly harder than the corresponding problems in document analysis.

The detection and localization problems involve finding tight bounding boxes around any text in a given frame. I have presented two detection and localization algorithms. Algorithm A detects and localizes horizontal text of constrained size and horizontal orientation. Algorithm B detects non-horizontal text and text of arbitrary size. Both run directly on MPEG-compressed video bit streams. These algorithms have been evaluated on challenging datasets against other algorithms presented in the literature. It was found that Algorithm B gave better results than other algorithms.

The tracking problem involves locating text regions as they move or change over time. I have presented two tracking algorithms. The first works on rigid text exhibiting simple, linear motion. It uses MPEG motion vectors for speed and robustness. I have also presented a tracking algorithm that handles text events that grow, shrink, and rotate over time. This algorithm was experimentally evaluated on a challenging dataset.

The binarization problem involves converting a color image of a text string into a binary image suitable for OCR. This is difficult because the background may be quite complex and have colors similar to the text color. I have proposed a binarization algorithm that works with arbitrary text color and background complexity.

5.1 Opportunities for future work

As with any research, many dead ends and blind alleys were encountered during the work described in this thesis. I believe that many of these unsuccessful ideas were good in theory, but I was unable to solve the necessary details needed to implement them. In this section, I describe some of the avenues of the text extraction problems that remain unexplored.

My evaluation of state-of-the-art detection and localization algorithms showed that no algorithm could achieve greater than 50% recall and precision simultaneously on a challenging dataset of general-purpose video. For application in a video indexing system, algorithms with better accuracy are needed. Current algorithms are confused by image regions having texture similar to text. Research is needed to explore other features that can robustly distinguish between text and non-text regions. One possibility is to combine the outputs of multiple detection and localization algorithms in an intelligent way to produce a single, better output. Another possibility is to use an OCR module to assist in text localization. The confidence of an OCR module could be used to discard image regions that cannot be recognized.

I explored the idea of analyzing the shapes within a candidate text region to verify that it contains text. For example, the frequency of corners and edges of the shapes in a region could be used to remove very simple shapes unlikely to be text characters. Unfortunately, some characters in some scripts have very simple shapes. I abandoned this idea because I was unwilling to impose constraints on the script of the text to be detected. However I believe the idea of analyzing shapes within a candidate text region deserves further investigation.

The binarization algorithm presented in this thesis works well with large font sizes. However, there is a significant amount of text in video that has very small size, sometimes with stroke widths less than one pixel. Connected component labeling on such small fonts often gives inaccurate results, causing my binarization algorithms to fail. I explored the use of topographical analysis [24] to binarize small text. Unfortunately, I found that such an approach was very susceptible to noise. More research is needed into the accurate binarization of small text.

I have also tried to incorporate color features into the binarization algorithm to improve the results. It is desirable to consider color during binarization because text may have little contrast with the background in the luminance image plane, but have high contrast in a color plane. I tried using color clustering [16] to separate text strokes from the background. This approach worked well once the parameters of the color clustering algorithm were manually adjusted for a given text instance. Unfortunately, I was unable to find a mechanism for automatically setting these parameters. More research is needed to find a way to incorporate color information into the binarization process.

In addition to growing, shrinking, and rotating text, other types of “stylized” text can be found in general-purpose video. For example, text can break into pieces, or morph between fonts, or undergo perspective distortion. The tracking algorithm presented in this thesis works for some of these cases, but further research is required to extend the algorithm to handle more types of stylized text.

The recognition problem has not been covered in this thesis. Several researchers [57, 53, 13] have attempted recognition from images and video. Unfortunately, even with constraints on the video dataset and application-specific text dictionaries available *a priori*, recognition accuracy has been low. More research is needed to design OCR modules geared specifically for the unique challenges of text in video.

References

[1] L. Agnihotri and N. Dimitrova. Text Detection for Video Analysis. In *Proceedings of the IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 109–113, 1999.

[2] AltaVista Company, Inc. Altavista. http://www.altavista.com/.

[3] S. Antani, D. Crandall, and R. Kasturi. Robust extraction of text in video. In *Proc. International Conference on Pattern Recognition*, volume 3, pages 831–834, 2000.

[4] S. Antani, D. Crandall, V. Y. Mariano, A. Narasimhamurthy, and R. Kasturi. Reliable extraction of text in video. Technical Report CSE-00-022, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16801, November 2000.

[5] S. Antani, D. Crandall, A. Narasimamurthy, Y. Mariano, and R. Kasturi. Evaluation of Methods for Extraction of Text from Video. In *IAPR International Workshop on Document Analysis Systems*, pages 507–514, 2000.

[6] N. Chaddha, R. Sharma, A. Agrawal, and A. Gupta. Text Segmentation in Mixed–Mode Images. In *28th Asilomar Conference on Signals, Systems and Computers*, pages 1356–1361, October 1995.

[7] D. Doermann and D. Mihalcik. Tools and techniques for video performance evaluation. In *Proc. International Conference on Pattern Recognition*, pages 167–170, 2000.

[8] R. Dugad and N. Ahuja. A Fast Scheme for Altering Resolution in the Compressed Domain. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 213–218, 1999.

[9] C. Garcia and X. Apostolidis. Text detection and segmentation in complex color images. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 2326–2329, 2000.

[10] U. Gargi, S. Antani, and R. Kasturi. Indexing text events in digital video databases. In *Proc. International Conference on Pattern Recognition*, volume 1, pages 916–918, 1998.

[11] U. Gargi, D. Crandall, S. Antani, T. Gandhi, R. Keener, and R. Kasturi. A system for automatic text detection in video. In *International Conference on Document Analysis and Recognition*, pages 29–32, 1999.

[12] R.C. Gonzalez and R.E. Woods. *Digital Image Processing.* Addison-Wesley Publishing Company, Inc., 1993.

[13] O. Hori. A video text extraction method for character recognition. In *International Conference on Document Analysis and Recognition*, pages 25–28, 1999.

[14] S.L. Horowitz and T. Pavlidis. Picture Segmentation by a Traversal Algorithm. *Computer Graphics and Image Processing*, 1:360–372, 1972.

[15] J. Huang, Z. Liu, Y. Wang, Y. Chen, and E.K. Wong. Integration of multimodal features for video classification based on hmm. In *Proc. IEEE Signal Processing Society Workshop on Multimedia Signal Processing*, 1999.

[16] A.K. Jain. *Algorithms for clustering data.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[17] A.K. Jain and B. Yu. Automatic Text Location in Images and Video Frames. *Pattern Recognition*, 31(12):2055–2076, 1998.

[18] R. Jain, R. Kasturi, and B.G. Schunck. *Machine Vision.* McGraw Hill, 1995.

[19] K.Y. Jeong, K. Jung, E.Y. Kim, and H.J Kim. Neural network-based text location for news video indexing. In *Proc. IEEE International Conference on Image Processing*, pages 319–323, 1999.

[20] M. Kamel and A. Zhao. Extraction of Binary Character/Graphics Images from Grayscale Document Images. *Computer Vision, Graphics, and Image Processing*, 55(3):203–217, May 1993.

[21] J.N. Kapur, P.K. Sahoo, and A.K.C. Wong. A New Method for Gray-Level Picture Thresholding Using the Entropy of the Histogram. *Computer Vision, Graphics, and Image Processing*, 29(3):273–285, March 1985.

[22] F. LeBourgeois. Robust Multifont OCR System from Gray Level Images. In *International Conference on Document Analysis and Recognition*, volume 1, pages 1–5, 1997.

[23] C.-M. Lee and A. Kankanhalli. Automatic Extraction of Characters in Complex Scene Images. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(1):67–82, February 1995.

[24] S.-W. Lee and Y.J. Kim. Direct Extraction of Topographical Features for Gray Scale Character Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):724–728, July 1995.

[25] H. Li and D. Doermann. Automatic text tracking in digital videos. In *IEEE Second Workshop on Multimedia Signal Processing*, pages 21–26, 1998.

[26] H. Li, D. Doermann, and O. Kia. Text extraction and recognition in digital video. In *IAPR International Workshop on Document Analysis Systems*, pages 119–128, 1998.

[27] H. Li, D. Doermann, and O Kia. Automatic Text Detection and Tracking in Digital Video. *IEEE Transactions on Image Processing*, 9(1):147–156, 2000.

[28] R. Lienhart and F. Stuber. Automatic Text Recognition for Video Indexing. In *Proceedings of the ACM International Multimedia Conference & Exhibition*, pages 11–20, 1996.

[29] R. Lienhart and F. Stuber. Automatic Text Recognition in Digital Videos. In *Proceedings of SPIE*, volume 2666, pages 180–188, 1996.

[30] R. Lienhart and F. Stuber. Indexing and Retrieval of Digital Video Sequences based on Automatic Text Recognition. In *Proceedings of the ACM International Multimedia Conference & Exhibition*, pages 419–420, 1996.

[31] V.Y. Mariano and R. Kasturi. Locating Uniform-Colored Text in Video Frames. In *Proc. International Conference on Pattern Recognition*, volume 4, pages 539–542, 2000.

[32] N. Merhav and V. Bhaskaran. Fast algorithms for dct-domain image down-sampling and for inverse motion compensation. *IEEE Transactions on Circuits and Systems for Video Technology*, 7:468–476, 1997.

[33] S. Messelodi and C.M. Modena. Automatic Identification and Skew Estimation of Text Lines in Real Scene Images. *Pattern Recognition*, 32(5):791–810, May 1999.

[34] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. LeGall. *MPEG Video Compression Standard*. Digital Multimedia Standards Series. Chapman and Hall, 1997.

[35] G. Nagy. Twenty Years of Document Image Analysis in PAMI. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):38–62, 2000.

[36] Y. Nakajima, A. Yoneyama, H. Yanagihara, and M. Sugano. Moving Object Detection from MPEG Coded Data. In *Proceedings of SPIE*, volume 3309, pages 988–996, 1998.

[37] W. Niblack. *An introduction to digital image processing.* Prentice-Hall International, 1986.

[38] J. Ohya, A. Shio, and S. Akamatsu. Recognizing Characters in Scene Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:214–224, 1994.

[39] N. Otsu. A threshold selection method from gray-scale histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, 1979.

[40] M. Pilu. On Using Raw MPEG Motion Vectors to Determine Global Camera Motion. In *Proceedings of SPIE*, volume 3309, pages 448–459, 1998.

[41] Charles A. Poynton. Frequently asked questions about colour. http://ftp.inforamp.net/pub/users/poynton/doc/colour/, 1995.

[42] W.K. Pratt. *Digital Image Processing, 2nd Ed.* John Wiley & Sons, 1991.

[43] W. Qi, L. Gu, H. Jiang, X.R. Chen, and H.J. Zhang. Integrating visual, audio and text analysis for news video. In *Proc. IEEE International Conference on Image Processing*, pages 520–523, 2000.

[44] T. Ridler and S. Calvard. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man and Cybernetics*, 8(8):630–632, 1978.

[45] T. Sato, T Kanade, E.K. Hughes, and M.A. Smith. Video OCR for Digital News Archive. In *IEEE International Workshop on Content–Based Access of Image and Video Databases CAIVD'98*, pages 52–60, January 1998.

[46] M.v.d. Schaar-Mitrea and P.H.N. de With. Compression of Mixed Video and Graphics Images for TV Systems. In *SPIE Visual Communications and Image Processing*, pages 213–221, 1998.

[47] B.G. Sherlock and D.M. Munro. Algorithm 749: Fast Discrete Cosine Transform. *ACM Transactions on Mathematical Software*, 21(4):372–378, 1995.

[48] J.C. Shim, C. Dorai, and R. Bolle. Automatic Text Extraction from Video for Content-Based Annotation and Retrieval. In *Proc. International Conference on Pattern Recognition*, pages 618–620, 1998.

[49] J.C. Shim, C. Dorai, and R. Bolle. Automatic Text Extraction from Video for Content-Based Annotation and Retrieval. Technical Report RC21087(94340), IBM T.J. Watson Research Division, Yorktown Heights, NY, 1998.

[50] C.S. Shin, K.I. Kim, M.H. Park, and H.J. Kim. Support vector machine-based text detection in digital video. In *Proc. IEEE Signal Processing Society Workshop*, pages 634–641, 2000.

[51] Terra Lycos, Inc. Lycos. http://www.lycos.com/.

[52] H.C. Tom and H.K. Katsaggelos. Resolution enhancement of monochrome and color video using motion compensation. *IEEE Transactions on Image Processing*, 10(2):278–287, 2001.

[53] Y. Watanabe, Y. Okada, K. Kaneji, and Y. Sakamoto. Retrieving related tv news reports and newspaper articles. *IEEE Intelligent Systems*, pages 40–44, September/October 1999.

[54] L.L. Winger, M.E. Jernigan, and J.A. Robinson. Character Segmentation and Thresholding in Low-Contrast Scene Images. In *Proceedings of SPIE*, volume 2660, pages 286–296, 1996.

[55] A.K.C. Wong and P.K. Shaoo. A Gray-Level Threshold Selection Method Based on Maximum Entropy Principle. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(4):866–871, July 1989.

[56] E.K. Wong and M. Chen. A robust algorithm for text extraction in color video. In *Proc. IEEE International Conference on Multimedia and Expo*, pages 797–800, 2000.

[57] V. Wu, R. Manmatha, and E.M. Riseman. Finding Text in Images. In *Second ACM International Conference on Digital Libraries*, 1997.

[58] P. Zhu and P.M. Chirlian. On Critical-Point Detection of Digital Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):737–748, August 1995.