Technical Report TR-05-002, Mathematics and Computer Science Department, University of Mannheim, April 2005
by Stephan Kopf, Thomas Haenselmann, Wolfgang Effelsberg

REIHE INFORMATIK
TR-05-002

Robust Character Recognition in Low-Resolution Images and Videos

Stephan Kopf, Thomas Haenselmann, Wolfgang Effelsberg
University of Mannheim
– Fakultät für Mathematik und Informatik –
Praktische Informatik IV
A5, 6
D-68159 Mannheim, Germany

Robust Character Recognition in Low-Resolution Images and Videos

Stephan Kopf, Thomas Haenselmann, Wolfgang Effelsberg
Dept. of Computer Science IV, University of Mannheim, Germany
{kopf,haenselmann,effelsberg}@informatik.uni-mannheim.de

ABSTRACT

Although OCR techniques work very reliably for high-resolution documents, the recognition of superimposed text in low-resolution images or videos with a complex background is still a challenge. Three major parts characterize our system for recognition of superimposed text in images and videos: localization of text regions, segmentation (binarization) of characters, and recognition.

We use standard approaches to locate text regions and focus in this paper on the last two steps. Many approaches (e.g., projection profiles, k-mean clustering) do not work very well for separating characters with very small font sizes. We apply in a vertical direction a shortest-path algorithm to separate the characters in a text line. The recognition of characters is based on the curvature scale space (CSS) approach which smoothes the contour of a character with a Gaussian kernel and tracks its inflection points. A major drawback of the CSS method is its poor representation of convex segments: Convex objects cannot be represented at all due to missing inflection points. We have extended the CSS approach to generate feature points for concave and convex segments of a contour. This generic approach is not only applicable to text characters but to arbitrary objects as well.

In the experimental results, we compare our approach against a pattern matching algorithm, two classification algorithms based on contour analysis, and a commercial OCR system. The overall recognition results are good enough even for the indexing of low resolution images and videos.

1. INTRODUCTION

Many efforts have been made in recent years to recognize text in images and videos. Even though the recognition of scanned text *documents* works very reliably, the recognition of text in low-resolution photos and videos with complex backgrounds is still a challenge.

Text in images or videos can be divided into two classes, *scene text* – like street signs or writing on shirts – or *superimposed text.* Superimposed text provides additional information that is usually not available in the image or – in the case of videos – in the audio track. Both scene text and superimposed text are rich in semantics, and thus a generally applicable and reliable OCR could be very useful for the indexing and searching of large collections of images or videos.

Several specialized OCR applications are already available as commercial products, such as the recognition of high-resolution scanned text documents, recognition systems for license plates or handwritten characters. These specialized applications are generally not applicable for videos or images. Major difficulties to recognize text in videos are caused by:

- the low image resolution. Often, the downscaling of images aliases text and background pixels.
- high compression rates of the images that blur the borders of characters and merge adjacent characters,
- complex and textured background so that a reliable distinction between characters and background is not possible, and
- unknown text fonts and text sizes.

We have implemented a text recognition system that performs the detection of text regions, the segmentation of text, and the recognition of characters in images and videos. We compare our new approach against a simple pattern matching algorithm, two classification algorithms based on contour analysis (zoning and contour profiles), and a commercial OCR system.

The remainder of this paper is organized as follows: Section 2 describes related work in the area of text segmentation and recognition. Section 3 gives a short overview of the text recognition application. The following sections describe the automatic recognition of text regions (Section 4), the segmentation of characters (Section 5) and the recognition (Section 6). We then present experimental results in Section 7 and conclude with Section 8.

2. RELATED WORK

Text segmentation approaches can be classified into two major categories based either on texture or connected component analysis. The first category applies filters to identify significant edges, corners or pixels with a high contrast. A major disadvantage of this approach is the large number of false characters pixels in images with a complex background. Alternative approaches identify and aggregate regions of similar colors and select text pixels by applying heuristics to the regions.

Hua et al. [8, 9] apply a texture-based approach to identify candidates for text regions and analyze the position of significant corners in an image. Corners that are close to each other are merged, and edge filters verify the detected text regions. To improve the results for videos they propose a combination of four methods that aggregate specific information of text regions in consecutive frames. Only those text frames are selected that have a high quality, e.g., the contrast in the neighborhood of text regions should be very low.

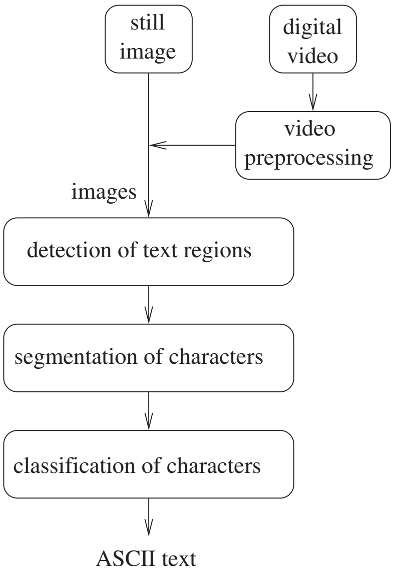

still image
digital video
video preprocessing
images
detection of text regions
segmentation of characters
classification of characters
ASCII text

Figure 1: Overview of the recognition process

Lienhart et al. have implemented algorithms in both categories. A rule-based approach [13] analyzes regions and verifies the contrast, color and size of candidate text regions. In a second approach a multilayer feed-forward network is used at different scales to detect text pixels [14]. Li et al. [12] use a neural network that is trained with wavelet-based features to identify text regions. They propose a tracking mechanism based on a text motion model and multi-resolution analysis.

Many techniques have been proposed to recognize segmented characters based on grayscale images, binary images, contours or skeletons. Approaches like the Karhunen-Loeve transform, zoning, fourier descriptors or contour profiles require good segmentation results or at least an exact estimation of the bounding box of a character. Several surveys and comparisons of standard recognition techniques have been published [4, 7, 15, 22].

Several systems were proposed that extract text information from news videos. Xi et al. [24] use edge maps and morphological filters to identify text regions. Sato et al. [20] improve the image quality of video frames by means of sub-pixel interpolation and multi-frame integration. Four specialized character recognition filters identify text pixels. The exact boundaries of characters are specified with projection profiles.

The detection of text regions in the *textfinder* [23] system is based on texture analysis and a K-means clustering algorithm. The characters are detected by analysis of strokes (significant edges) that are aggregated to regions. Other more specialized methods have been proposed, such as the recognition and automatic translation of street signs and company names [6, 25] or the recognition of vehicle license plates [3].

Most text recognition systems assume simple backgrounds, so that the separation of the character works quite well. On the other hand, the background in real-world images and videos can be very complex. We present a new approach that works much more reliably for low-resolution text. Furthermore, most systems did not implement a recognition component but send the segmented binary images to commercially available OCR systems. The fonts in videos and images are usually quite different from fonts in printed documents, and the recognition rates of these systems drop significantly with unknown fonts.

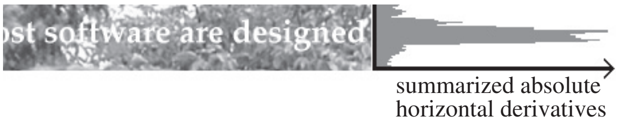


Figure 2: Horizontal projection profile

3. SYSTEM OVERVIEW

We use a three-step approach to recognize characters in images resp. frames of videos (see Figure 1). In case of videos, a *preprocessing* step improves the quality of the video frame. Frames are aggregated and only one frame in each shot (or group of frames in case of moving text) is processed by the recognition system. The video preprocessing and detection of text regions is based on well-known techniques. We briefly introduce the main concept of these steps in Section 4.

The recognition system analyzes the image and identifies text regions first. The segmentation step then locates the position of separators that segregate characters, and creates a binary image characterizing text and non-text pixels. The last step analyzes the contours of the segmented characters and classifies them.

4. PREPROCESSING AND RECOGNITION OF TEXT REGIONS

In the preprocessing step, we use sub-pixel interpolation and projection profiles on video frames as presented in [20] and get an inaccurate estimation of the position of text regions. A camera model with six degrees of freedom to describe the motion of text regions (horizontal, vertical, rotation and zoom) is applied. To calculate the camera parameters we identify corners in the text region that are tracked in consecutive frames. Correlations are established between these corners in successive frames. In order to estimate the motion of the text region reliably we apply a robust regression method. Details of the estimation of precise motion parameters have been published in [5].

The standard deviation of the pixels in the temporal direction of the aligned text regions gives an indication to a text or background region: Low values indicate text pixels or static background pixels. A median filter is applied in the temporal direction to define the color of these pixels. In the case of large standard deviation values (background pixels), a pixel at the image position is selected that maximizes its distance to already selected pixels.

We assume that each text line contains at least several characters. To locate a text region in an image we use the techniques presented by Sato and Smith [19, 21]: The idea is to identify regions with high contrast and sharp edges. A 3x3 horizontal filter with binary thresholding is applied to the image, and connected text blocks are identified. If this region suffices certain constrains like minimum size or fill factor, the bounding box of this region is classified as text region.

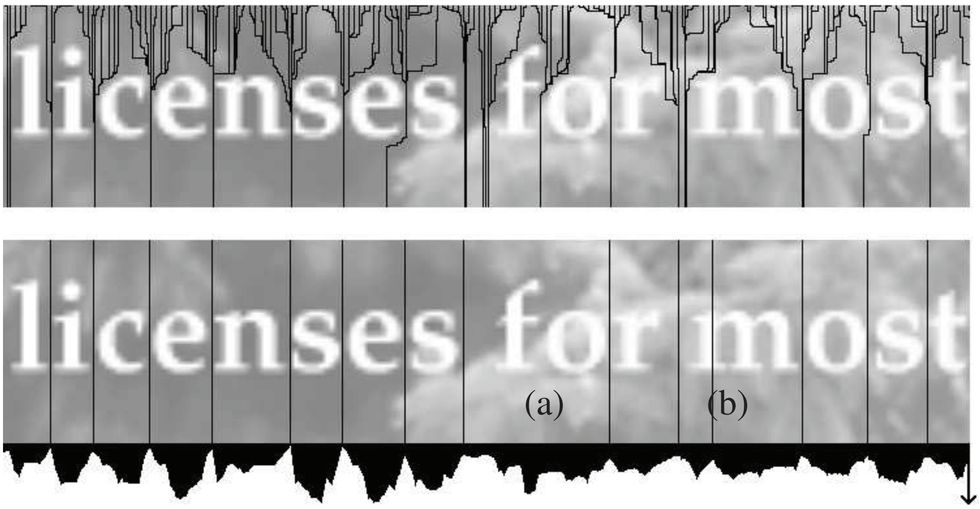


Figure 3: Top: Character separators based on cheapest paths. Bottom: Vertical projection profile with (a) missed separators and (b) split characters.

In images with complex backgrounds the bounding box may include more than one text line. Based on the approach presented by Lienhart et al. [14], the next step locates the borders of the text lines by analyzing horizontal projection profiles (see Figure 2). The profile is generated by summarizing the absolute horizontal derivatives. High values ($> 2\mu$) indicate the position of a text line, low values ($< \mu$) background regions. μ is defined as average value of the horizontal profile. The top and bottom of a text line is identified as the peak of a horizontal projection profile that was generated with derivatives in the vertical direction. Figure 13 displays an example of the detected bounding boxes in an image with a complex background.

5. SEGMENTATION OF CHARACTERS

The segmentation step is very essential in our approach, because classification techniques based on contour analysis fail without a reliable segmentation. Each pixel in each text region is classified as text or background pixel. The distinction is not trivial because the luminance and chrominance values of text pixels of one character can vary significantly. We propose a three-step approach: First, we locate separators for individual characters. Next, we estimate the dominant text colors based on histograms. Finally we classify text and background pixels by applying a region-growing algorithm.

5.1 Character separators

We use linear interpolation as presented in [20] to scale the text region by a factor four and thus improve the segmentation results. The separation of characters does not work very well with vertical projection profiles that summarize edge values for each column (see bottom of Figure 3). Many characters are split and separators are missed.

Usually, the contrast between text pixels and background pixels is high, whereas the average difference between adjacent background pixels is much lower. We take advantage of this fact and search a path from the top to the bottom of the text region. Different starting positions in the top row are selected, and the paths with the lowest costs are stored. The costs of a path are defined as the summarized pixel differences between adjacent path pixels. The path with the minimum cost which we call *cheapest path* rarely crosses character pixels and defines a good separator for characters.

We use the *Dijkstra* shortest-path algorithm for graphs to identify the separators. Each pixel is defined as a node that is connected to three neighbor pixels (left, right, down). The costs of travel from one node to another are defined as the absolute luminance differences between these two pixels. We start at different positions in the top row and calculate the path from each position to the bottom row. Results of the minimum paths (cheapest paths) are depicted in the top of Figure 3. A major advantage of this approach is the fact that no threshold is required to locate the separators of characters.

The computational effort is very high if the cheapest path is calculated for all pixels. We use the following algorithm for optimization:

1. Estimate the minimum width W of a character based on the height of the text region.

2. Initialize every $\frac{W}{2}$ pixel as a candidate *start pixel* in the top row of the text region (Figure 4 (a)).

3. Calculate the cheapest path for start pixels at the left and right border of the text region (Figure 4 (b)). Both start pixels are labeled *path pixels.*

4. Select a start pixel in the center of two path pixels (Figure 4 (c)). Its cheapest path is calculated and it is relabeled as path pixel.

5. If two cheapest paths end at the same position, all start pixels between these two paths are marked as path pixels. In Figure 4 the paths of (c) and (d) end at the same position. Therefore, the start pixels at the position (e) are relabeled.

6. If another start pixel is available, continue with 4).

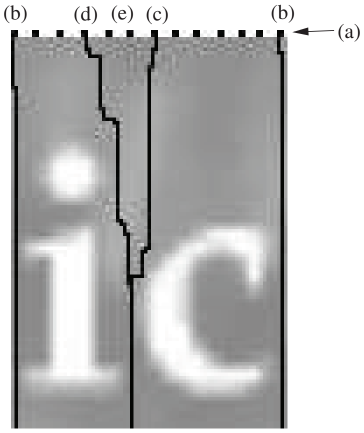


Figure 4: Optimization of the cheapest path algorithm

5.2 Estimation of text colors

We assume that text and background colors are significantly different. Two regions are defined: the approximated text region from Section 3 and an adjacent background region. Histograms based on eight-bit YUV images (four luminance bits, two bits for each chrominance component) are compared to estimate the dominant text color. Bins in the histogram of the text region that are larger than the corresponding bin in the adjacent region are defined as text color.

The text color is only a very rough approximation of the text pixels since due to significant variance in color and luminance a reliable segmentation based on color values alone is not possible. A binary image that was generated from the histogram analysis is depicted in Figure 5.

5.3 Selection of text pixels

The third step analyzes each block between two separators and classifies the pixel as text or background. We use a region-growing algorithm to initialize the regions based on the 8-bit YUV image. The following algorithm classifies pixels as text or background:

1. A region can be classified as *text region*, *background region*, or *undefined region*. All regions are initialized as undefined.

2. If the color of a region coincides with an estimated text color it is classified as text.

3. Undefined regions that adjoin the top or bottom border of the block are set to background regions.

4. The distance $D_{i,j}$ between each undefined region i and defined (text or background) region j is calculated:

   $$D_{i,j} = |C_i - C_j| + |G_i - G_j|. \tag{1}$$

   Each region is classified by its color C_i and its center of gravity G_i.

5. The minimum of $D_{i,j}$ is selected. Region i is classified as text or background (depending on region j).

6. If another undefined region is available, continue with 4.

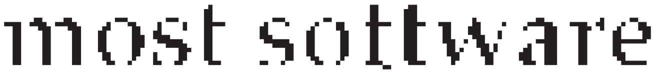


Figure 5: Rough segmentation of text pixels based on dominant text colors in histograms

An example of this algorithm after initialization of the regions (step 3) and segmentation (step 6) is depicted in Figure 6. The high compression rates in images and videos smooth and blur the edges of characters. The combination of color and spatial information in the distance measure D increases the quality of the segmentation.

Additionally, we have tested other algorithms that separate text and background pixels. A major disadvantage of the K-means algorithm is the fixed number of cluster centers. Many pixels are attached to the wrong cluster if only two cluster centers are used. To get good segmentation results the number of cluster centers must be adapted to the complexity and colors of the background.

6. CLASSIFICATION OF CHARACTERS

We analyze the contour of a character and derive features for classification. The features are based on the curvature scale space (CSS) technique that is presented in the following section. A major drawback of the standard CSS approach is its poor representation of the convex segments of a contour. We propose an approach to solve this problem.

6.1 Standard CSS Technique

The curvature scale space (CSS) technique [16, 17] is based on the idea of curve evolution. A CSS image provides a multi-scale representation of the curvature zero crossings of a closed planar contour. Consider a closed planar curve $\Gamma(u)$,

$$\Gamma(u) = \{(x(u), y(u)) | u \in [0, 1]\}, \tag{2}$$

with the normalized arc length parameter u. The curve is smoothed by a one-dimensional Gaussian kernel $g(u, \sigma)$ of width σ. The deformation of the closed planar curve is represented by

$$\Gamma(u, \sigma) = \{(X(u, \sigma), Y(u, \sigma)) | u \in [0, 1]\}, \tag{3}$$

where $X(u, \sigma)$ and $Y(u, \sigma)$ denote the components $x(u)$ and $y(u)$ after convolution with $g(u, \sigma)$.

The curvature $\kappa(u, \sigma)$ of an evolved curve can be computed using the derivatives $X_u(u, \sigma)$, $X_{uu}(u, \sigma)$, $Y_u(u, \sigma)$, and $Y_{uu}(u, \sigma)$:

$$\kappa(u, \sigma) = \frac{X_u(u, \sigma) \cdot Y_{uu}(u, \sigma) - X_{uu}(u, \sigma) \cdot Y_u(u, \sigma)}{(X_u(u, \sigma)^2 + Y_u(u, \sigma)^2)^{3/2}}. \tag{4}$$

A CSS image $I(u, \sigma)$ is defined by

$$I(u, \sigma) = \{(u, \sigma) | \kappa(u, \sigma) = 0\}. \tag{5}$$

The CSS image shows the zero crossings with respect to their positions on the contour and the width of the Gaussian kernel (or the number of iterations). An example of smoothed contours and the CSS image is depicted in Figure 7.

During the deformation process, zero crossings merge as the transitions between contour segments of different curvature are equal-

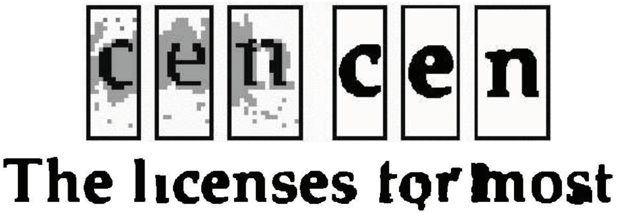


Figure 6: Top left: Initialization of text (black), background (white), and undefined regions (gray). Top right: Segmented characters. Bottom: Final segmentation of a text line.

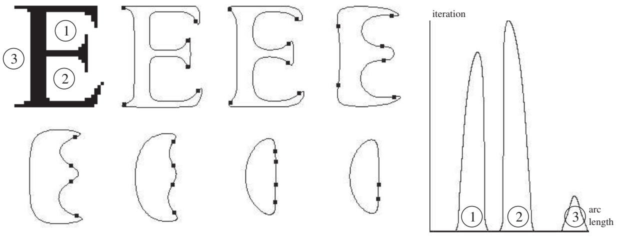


Figure 7: Original character and smoothed contours with inflection points after 5, 20, 100, 250, 500, 1000 and 1150 iterations. The corresponding CSS image is depicted on the right side. Three major concave segments are labeled.

ized. Consequently, after a certain number of iterations, inflection points cease to exist, and the shape of the closed curve becomes convex. Significant contour properties that are visible during a large number of iterations result in high peaks in the CSS image. However, areas with rapidly changing curvatures caused by noise produce only small local maxima.

In many cases, the peaks in the CSS image provide a robust and compact representation of a contour. We use a fixed number of samples to describe the contour of a character and get a CSS image that is invariant to scaling of the contour. Hence, no special consideration of the font size is required for the recognition process. A rotation of the character in the image plane corresponds to shifting the CSS image left or right. We limit the rotation of the characters to approximately +/- 20 degrees. A slight rotation enables the recognition of italic characters, even if they are not stored in a database for comparison.

6.2 Extended CSS Features

Certain contours that differ significantly in their visual appearance nevertheless have similar CSS images. One reason is that deep and shallow concavities on a contour may result in peaks of the same height in the CSS image. Abbasi [1] and Richter [18] presented approaches to avoid these ambiguities.

A major drawback is still the inadequate representation of convex segments on the contour. A CSS image represents the position of the inflection points, so concave segments on the contour are required. Contours of characters without concave segments (e.g., the ”I” and ”O”) cannot be distinguished.

We apply the standard CSS approach first to get characteristic feature vectors that classify concave parts of the contour very well. The general idea is now to create a second contour that will provide additional features for the convex segments of the original contour. The original contour is mapped to a new contour with an inverted curvature. Strong convex segments of the original contour become concave segments of the mapped contour. Significant curvatures in the original contour are still significant in the mapped contour [11].

To create a *mapped contour*, we enclose the contour of the character by a circle of radius R and identify the point P of the circle closest to each contour pixel. The contour pixels are mirrored on the tangent of the circle in P. Two mapped contours are depicted in Figure 8. Segments of the contour that have a strong convex curvature are mapped to concave segments. The strength of the curvature of the mapped contour depends on the radius R of the circle and on the curvature of the original contour. If a convex curvature is stronger than the curvature of the circle, the segment in the mapped contour will be concave.

The calculation of the mapped contour is quite fast. Each contour pixel at position u of the closed planar curve $(x(u), y(u))$ is mapped to a curve $(x'(u), y'(u))$. The center of the circle (M_x, M_y) with radius R is calculated as average position of contour pixels $(M_x = \frac{1}{N}\sum_{u=1}^{N} x(u), M_y = \frac{1}{N}\sum_{u=1}^{N} y(u))$.

$$D_{x(u),y(u)} = \sqrt{(M_x - x(u))^2 + (M_y - y(u))^2} \qquad (6)$$

$$x'(u) = (x(u) - M_x) \cdot \frac{2 \cdot R - D_{x(u),y(u)}}{D_{x(u),y(u)}} + M_x \qquad (7)$$

$$y'(u) = (y(u) - M_y) \cdot \frac{2 \cdot R - D_{x(u),y(u)}}{D_{x(u),y(u)}} + M_y \qquad (8)$$

$D_{x(u),y(u)}$ specifies the distance between the center of the circle and the current contour pixel. If the positions of a contour pixel and the center of the circle are the same, a mapping is not possible. In this case, the contour pixel is interpolated from adjacent contour pixels of the mapped contour.

In principle, the mirroring of contours is not limited to enclosing circles. Although other shapes could be used as well, some difficulties would arise. Angular shapes like rectangles would create discontinuous contours. Ellipses have the disadvantage that the point P (where the contour pixel is mirrored) is not always unique. E.g., in the case of ellipses that are parallel to the X- and Y-axis, the mirroring is undefined for all points on these axes.

We apply the standard CSS approach to the mapped contour. To indicate the classification of convex segments in the original contour we represent this new CSS image with negative values. In Figure 9 extended CSS images for the characters ”I”,”O”,”K” and ”X” are depicted. Positive values represent the original CSS images, negative values the CSS images of the mapped contours. The convex characters ”I” and ”O” cannot be classified with the standard CSS approach, but the dual CSS representations differ significantly. On the other hand, the convex segments of the characters ”K” and ”X” are very similar and generate comparable (negative) CSS peaks. We get twice the number of feature points (peaks in the CSS image) for each contour with our extended CSS approach.

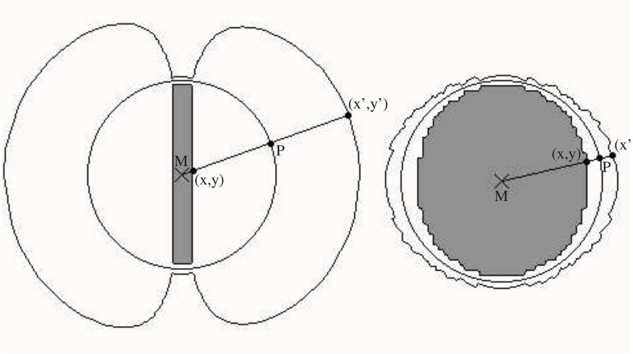


Figure 8: The contours of two characters (gray color) are "mirrored" at the circle.

6.3 CSS Matching

The matching of an unknown character is done in the following steps. The contour of the character is sampled with a fixed predefined number of samples. The CSS image is calculated, and peaks are extracted. It is sufficient to extract the significant maxima (above a certain noise level). The position on the contour and the value (iteration or Gaussian kernel width) are stored for each peak. For instance, assuming a noise level of five iterations in the example depicted in Figure 7, only three data pairs have to be stored.

These peaks characterize convex regions. The sampled contour pixels are transformed to the mapped (dual) contour, and a second CSS image is created. Up to ten feature vectors are stored for each CSS image (original and mapped). The mapped feature vectors are stored as negative values. An unknown character is matched by comparing the feature vectors (CSS peaks) to those of the characters that are stored in a database.

The general idea of the matching algorithm is to compare the peaks in the CSS images based on the height and position of the arc. This is done by first determining the best position to compare the peaks. It might be necessary to slightly rotate one of the CSS images to best align the peaks. As mentioned before, shifting the CSS image left or right corresponds to a rotation of the original object in the image. Each character is stored only once in the database, and the horizontal moves compensate small rotations of italic character.

A matching peak is determined for each peak in the CSS image of the unknown character. If a matching peak is found, the Euclidean distance of the height and position of each peak is calculated and added to the difference between the CSS images. Otherwise, the height of the peak in the first image is multiplied by a penalty factor and is added to the total difference. It is not possible to match negative and positive CSS peaks (the concave segments in the original and mapped contour).

If no adequate rotation can be found (+/- 20 degrees) or if the highest maxima in the CSS images do not match within a given tolerance range, a matching is not possible. If this is the case, the two objects are significantly different. This rejection helps to improve the overall results because noise or incorrectly segmented characters are rejected in the matching step. Details of the matching algorithm of CSS images are published in [18].

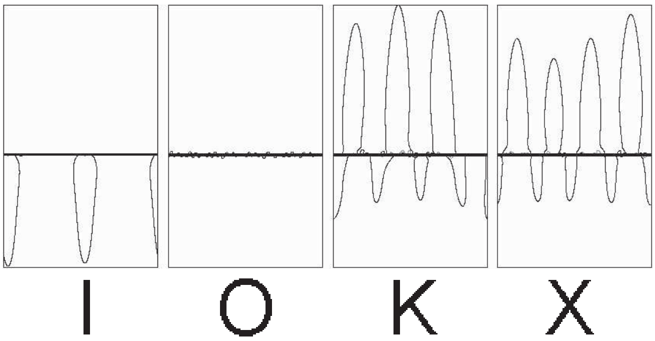


Figure 9: Four examples of extended CSS images are depicted. Positive values represent the original CSS images, negative values the dual images.

7. EXPERIMENTAL RESULTS

We have implemented the extended CSS algorithm, a simple pattern matching, a zoning algorithm based on contours, and a matching based on contour profiles. Additionally, we compare our new approach against a commercial OCR system that is part of a scanner software.

The *pattern matching* approach compares two segmented characters (binary images) and counts the number of pixel differences in these binary images. The segmented binary images are used as query images. The height and width of the query character is scaled to the default size ($n_x \times n_y$ pixels) of each character in the database. The distance $D_{q,i}$ of two characters is defined as:

$$D_{q,j} = \frac{1}{n_x \cdot n_y} \cdot \sum_{x=1,}^{n_x} \sum_{y=1,}^{n_y} \begin{cases} 0 & \text{if} \quad Q_{x,y} = J_{x,y} \\ 1 & \text{else} \end{cases} . \tag{9}$$

Q is the query image and J is one selected image from the database. Distance D is the normalized number of different pixels.

The second distance measure applies a *zoning* algorithm [10, 22] based on contour pixels. The idea is to superimpose a n × m grid on the character. The number of edge pixels in each block is used as feature vector. Improvements of this algorithm classify the orientation (horizontal, vertical and two diagonal directions) of line segments of neighboring contour pixels. We have selected the zoning technique because the commercial OCR system CALERA [2] used this approach and reported good recognition results with severely degraded characters.

We have implemented a third distance measure that is based on contour profiles [10, 22]. To calculate a horizontal profile we select the outermost (lowest and highest) pixel of the contour in each position. An example of a horizontal profile is depicted in Figure 10. The contour is rotated by 90 degrees to calculate the vertical profile. The values of the four profiles are used as feature vector.

7.1 Perfectly segmented characters

We compare the matching results of the different techniques. Slight modifications of characters such as a different font size or a small

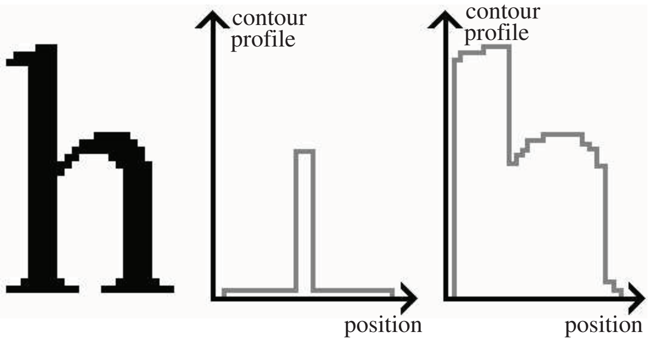
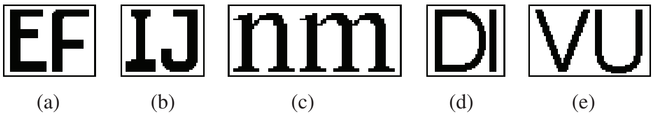


Figure 10: Horizontal contour profile.



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

Figure 11: Two examples of the license plate font (a)(b) illustrate the large minimum distance value of the pattern matching. The distance is very low in other fonts (c). The standard CSS approach cannot characterize convex characters (d). Even some characters (e) are very similar in the extended CSS approach.

rotation of a character should have no significant impact on the matching results. Even characters of different fonts should be comparable.

We have selected binary images of four different fonts (Arial, Times, Gothic, and the European license plate font). 52 characters of each font (26 characters for the European license plate font) are stored in the database. Each character has a font size of 36.

Figure 11 depicts sample characters of different fonts. All approaches works quite well with the European license plate font that was specially designed for easy recognition (compare the differences of characters like ”E” and ”F”, or ”I” and ”J”). Characters of other fonts are much more similar, like the character ”n” that is very close to ”m”. Convex characters (e.g., ”D” and ”I”) cannot be distinguished with the standard CSS approach. Some characters (e.g., ”V” and ”U”) are quite similar for the extended CSS method as well.

A good algorithm should even be robust when characters of different fonts are compared. We take a character of one font and identify the three best matches in the other fonts. Table 1 lists the percentages of correct matches for the best match, the two best matches and the three best matches. Only one match is counted if more than one character is correct.

	Best match is correct	One of two best matches is correct	One of three best matches is correct
Pattern matching	72.1 %	81.2 %	88.4 %
Zoning	63.2 %	69,5 %	74.1 %
Contour profiles	69.3 %	84.2 %	88.7 %
Standard CSS	69.8 %	83.5 %	88.1 %
Extended CSS	77.3 %	90.8 %	93.7 %

Table 1: Recognition rates between different fonts

We have also compared the recognition rates for characters of different sizes. The recognition rate drops significantly for characters with a height of eight pixels or less (the actual matching is performed on the images that are scaled by a factor of four). Large characters do not increase the recognition rate.

7.2 Artificially deteriorated characters

We have also investigated the effect of segmentation errors on the matching results. For each character several noisy variations of this character were generated. Up to twenty pixels on the contour were randomly selected and modified. A local dilatation or erosion operation with a radius of three is applied to these pixel positions. Some examples of heavily deteriorated characters are depicted in Figure 12. The recognition rates drop to 67.4 percent (pattern matching), 62.2 percent (zoning), 66.0 percent (contour profiles), 63.9 percent (CSS), and 71.2 percent (extended CSS). The CSS approaches are no longer applicable if a contour is incoherent. This is the case for the characters ”D”, ”w” and ”x” in Figure 12.

7.3 Recognition in images and videos

In the second part of our experimental results we have analyzed the recognition results for images and videos. We match the segmented characters against all characters stored in the database. Twenty images with complex backgrounds and ten video segments from different genres with a total length of 19 minutes were matched against the database.

We define precision and recall for the *localization of text lines*:

$$\text{precision} = \frac{\text{number of correctly retrieved text lines}}{\text{total number of retrieved text lines}} \quad (10)$$

$$\text{recall} = \frac{\text{number of correctly retrieved text lines}}{\text{actual number of text lines}} \quad (11)$$

Nearly no text line was missed (recall $>$ 97 percent), but in images with a complex background, many background regions were classified as text regions (precision $\approx$ 63 percent). Post-processing steps (like the analysis of the dominant text color or the minimum height of characters) improved the precision to approximately 91 percent without deteriorating the recall. The position of the upper and lower border of the text regions was always correct. Several characters were missed at the left and right border of the text regions if the first or last word was very short (6 percent of the characters).

A reliable *segmentation* is much more challenging. We define that a character is correctly segmented if it is not split or merged with other characters. Two characters often merge if text and background colors are similar. We have analyzed the quality of the

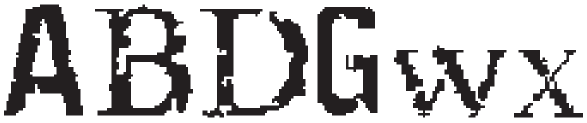


Figure 12: Example of deteriorated characters

segmentation of characters by comparing projection profiles and the optimum path approach as explained in Section 5.1. The results in Table 2 indicate that the optimum path algorithm is much more reliable (errors rates drop from 17,4 to 9,2 percent).

To calculate the recognition rate of the different approaches we manually built two groups of characters: correctly segmented characters and merged or split characters. The recognition rates of all approaches are very poor for characters in the second group (less than 8 percent and good matches seem to be random). Therefore, we analyze the correctly segmented characters separately. Table 3 lists the recognition results for images and video sequences. The results are significantly better in videos due to the additional preprocessing step.

	Optimum path	Projection profile
characters split	3.8 %	9.9 %
characters merged	5.4 %	7.5 %
total error rate	9.2 %	17.4 %

Table 2: Reliability of segmentation based on optimum path and projection profiles

The commercial OCR system could not recognize any characters in the original images. Therefore, we have computed the segmented binary images and manually removed merged or split characters in the images. An accurate comparison of the recognition rates is not possible due to the dictionary lookup in the commercial system. The quality of the segmentation is higher in video sequences, but the commercial OCR systems cannot benefit that much. We assume that the fonts in videos and text documents are very different and a dictionary lookup is less efficient with typical words in videos. The recognition rates drop significantly if only a few pixels change in the segmented image. E.g., only nine characters (50 percent) could be recognized in the text line in Figure 6 (bottom).

	Images	Video sequences
Number of characters	2986	1211 (after preprocessing)
Pattern matching	69.1 %	77.7 %
Zoning	64.2 %	69.7 %
Contour profiles	71.2 %	82.0 %
Standard CSS	66.9 %	78.8 %
Extended CSS	75.6 %	88.1 %
Commercial OCR and dictionary lookup	75.2 %	76.7 %
Localisation of text lines	96.6 %	97.1 %
Segmentation based on cheapest paths	90.8 %	91.0 %
Overall recognition rate with extended CSS approach	66.3 %	77.8 %

Table 3: Overview of the recognition results of correctly segmented characters

Figure 13 depicts the major recognition steps in an image with a complex background. The image includes characters with different fonts and sizes. The analysis of an image or video segment takes several seconds on a Pentium III processor with 1.8 GHz. Especially the median filter that smoothes the pixels in consecutive video frames, the estimation of the optimum path for each text line, and the evolution of the contour with the Gaussian kernel are complex operations.

We intentionally left out the last step in a typical OCR process: the matching of the recognized characters with a dictionary. It is a different subject and goes beyond the scope of this paper.

8. CONCLUSION AND OUTLOOK

We have presented an approach to automatically detect, segment and recognize text in low-resolution images and videos. The results of the segmentation can be significantly improved if the separators of characters are located with our cheapest path approach. Our extension of the CSS method classifies concave and convex segments of a contour and proves to be very powerful for the recognition of characters. As future work, we plan to evaluate the recognition results of the extended CSS method for deformable non-text objects. We have performed experiments and got some promising results for the recognition of postures and gestures of people.

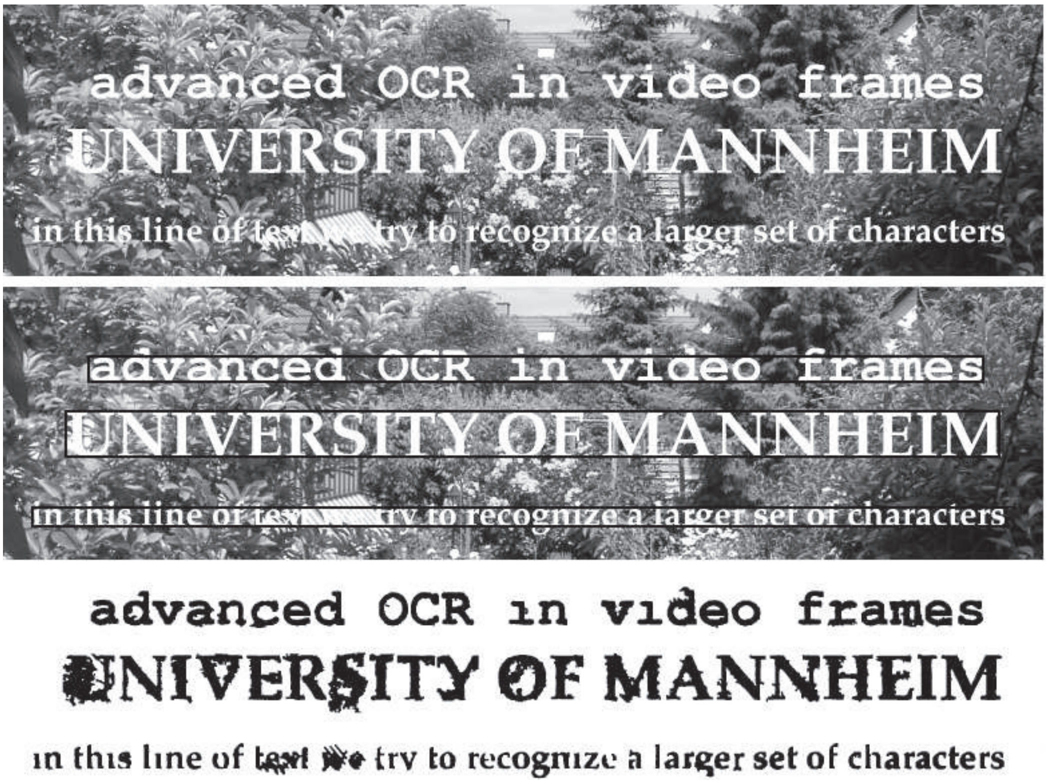

advanced OCR in video frames
UNIVERSITY OF MANNHEIM
in this line of text we try to recognize a larger set of characters

Figure 13: Original image (top), automatically detected text regions (center) and segmented text (bottom)

9. REFERENCES

[1] S. Abbasi, F. Mokhtarian, and J. Kittler. Enhancing CSS-based shape retrieval for objects with shallow concavities. In *Image and Vision Computing*, volume 18(3), pages 199–211, 2000.

[2] M. Bokser. Omnidocument technologies. In *Proceedings of the IEEE*, volume 80(7), pages 1066–1078, July 1992.

[3] Y. Cui and Q. Huang. Extracting characters of license plates from video sequences. In *Machine Vision and Applications*, volume 10, pages 308–320, April 1998.

[4] D. G. Elliman and I. T. Lancaster. A review of segmentation and contextual analysis techniques for text recognition. In *Pattern Recognition*, volume 23 (3-4), pages 337 – 346, March 1990.

[5] D. Farin, T. Haenselmann, S. Kopf, G. Kühne, and W. Effelsberg. Segmentation and classification of moving video objects. In B. Furht and O. Marques, editors, *Handbook of Video Databases: Design and Applications*, volume 8 of *Internet and Communications Series*, pages 561–591. CRC Press, Boca Raton, FL, USA, September 2003.

[6] J. Gao and J. Yang. An adaptive algorithm for text detection from natural scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 84–89, December 2001.

[7] V. K. Govindan and A. P. Shivaprasad. Character recognition - a review. In *Pattern Recognition*, volume 23 (7), pages 671 – 683, July 1990.

[8] X.-S. Hua, X.-R. Chen, L. Wenyin, and H.-J. Zhang. Automatic location of text in video frames. In *Intl. Workshop on Multimedia Information Retrieval (MIR)*, 2001.

[9] X.-S. Hua, P. Yin, and H.-J. Zhang. Efficient video text recognition using multiple frame integration. In *International Conference on Image Processing (ICIP)*, 2002.

[10] F. Kimura and M. Shridhar. Handwritten numerical recognition based on multiple algorithms. In *Pattern Recognition*, volume 24 (10), pages 969–983, 1991.

[11] S. Kopf, T. Haenselmann, and W. Effelsberg. Shape-based posture and gesture recognition in videos. In *Electronic Imaging*, volume 5682, pages 114–124. IS&T, SPIE, January 2005.

[12] H. Li, D. Doermann, and O. Kia. Automatic text detection and tracking in digital videos. In *IEEE Transactions on Image Processing*, volume 9, pages 147–156, January 2000.

[13] R. Lienhart and W. Effelsberg. Automatic text segmentation and text recognition for video indexing. In *ACM/Springer Multimedia Systems*, volume 8, pages 69–81. ACM Press, Jan. 2000.

[14] R. Lienhart and A. Wernicke. Localizing and segmenting text in images and videos. In *IEEE Transactions on Circuits and*

Systems for Video Technology, volume 12, pages 256–258, April 2002.

[15] J. Mantas. An overview of character recognition methodologies. In *Pattern Recognition*, volume 19, pages 425–430, 1986.

[16] F. Mokhtarian. Silhouette-based isolated object recognition through curvature scale space. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 17(5), pages 539–544, 1995.

[17] F. Mokhtarian, S. Abbasi, and J. Kittler. Robust and efficient shape indexing through curvature scale space. In *British Machine Vision Conference*, 1996.

[18] S. Richter, G. Kühne, and O. Schuster. Contour-based classification of video objects. In *Proceedings of IS&T/SPIE conference on Storage and Retrieval for Media Databases*, volume 4315, pages 608–618, January 2001.

[19] T. Sato, T. Kanade, E. K. Hughes, and M. A. Smith. Video OCR for digital news archives. In *IEEE International Workshop on Content-Based Access of Image and Video Databases (CAIVD)*, pages 52–60, 1998.

[20] T. Sato, T. Kanade, E. K. Hughes, M. A. Smith, and S. Satoh. Video OCR: Indexing digital news libraries by recognition of superimposed captions. In *ACM/Springer Multimedia Systems*, volume 7, pages 385 – 395. ACM Press, 1999.

[21] M. Smith and T. Kanade. Video skimming and characterization through the combination of image and language understanding. In *IEEE Intl. Workshop on Content-Based Access of Image and Video Databases*, pages 61 – 70, January 1998.

[22] Ø. Trier, A. Jain, and T. Taxt. Feature extraction methods for character recognition - a survey. In *Pattern Recognition*, volume 29 (4), pages 641–662, 1996.

[23] V.Wu, R.Manmatha, and E.M.Riseman. TextFinder: An automatic system to detect and recognize text in images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 21, pages 1224–1229, Nov. 1999.

[24] J. Xi, X.-S. Hua, X.-R. Chen, L. Wenyin, and H.-J. Zhang. A video text detection and recognition system. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, pages 873–876, 2001.

[25] J. Yang, X. Chen, J. Zhang, Y. Zhang, and A. Waibel. Automatic detection and translation of text from natural scenes. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 2101–2104, May 2002.