Computer Vision and Image Understanding xxx (2013) xxx–xxx

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

SnooperText: A text detection system for automatic indexing of urban scenes ☆

Rodrigo Minetto [a,*], Nicolas Thome [b], Matthieu Cord [b], Neucimar J. Leite [c], Jorge Stolfi [c]

[a] *DAINF, Federal University of Technology – Paraná, Curitiba, Brazil*
[b] *Laboratoire d'Informatique Paris 6 (LIP6), Université Pierre et Marie Curie, Paris, France*
[c] *Institute of Computing, University of Campinas, Campinas, Brazil*

ARTICLE INFO

Article history:
Received 11 July 2012
Accepted 8 October 2013
Available online xxxx

Keywords:
Text detection
Text region classification
Histogram of oriented gradients for text
Text descriptor
Textual indexing in urban scene images

ABSTRACT

We describe SnooperText, an original detector for textual information embedded in photos of building façades (such as names of stores, products and services) that we developed for the iTowns urban geographic information project. SnooperText locates candidate characters by using toggle-mapping image segmentation and character/non-character classification based on shape descriptors. The candidate characters are then grouped to form either candidate words or candidate text lines. These candidate regions are then validated by a text/non-text classifier using a HOG-based descriptor specifically tuned to single-line text regions. These operations are applied at multiple image scales in order to suppress irrelevant detail in character shapes and to avoid the use of overly large kernels in the segmentation. We show that SnooperText outperforms other published state-of-the-art text detection algorithms on standard image benchmarks. We also describe two metrics to evaluate the end-to-end performance of text extraction systems, and show that the use of SnooperText as a pre-filter significantly improves the performance of a general-purpose OCR algorithm when applied to photos of urban scenes.

1. Introduction

Here we describe SnooperText, an algorithm for the detection of text embedded in images or videos of urban scenes. This is a challenging problem in computer vision [1] with many potential applications, such as traffic monitoring, geographic information systems, road navigation, and scene understanding.

SnooperText was developed specifically for use in iTowns [2], a pilot project for resource location and immersive navigation in urban environments, similar to Google's Street View [3]. The main raw data for the iTowns project is a collection of GPS-tagged high-resolution digital photos of the city, including building façades, taken with a set of car-mounted cameras. The mean viewpoint spacing between photo sets is about 1 m. See Fig. 1.

The frontal images of the building façades are processed offline to extract any legible textual information, such as street and traffic signs, store names, and building numbers. The extracted strings are then stored in a geo-referenced database, which is used to answer textual queries by users—for example, to locate the addresses of stores with a specified name or selling a specified product. The user is then offered a navigable 3D view of the location, created by suitable projection of pre-stitched image mosaics. See Fig. 2.

A project like iTowns could easily generate hundreds of thousands of such mosaics in a single city. The manual annotation of all these images with the visible textual information would be very time consuming and probably impractical. Clearly, automated algorithms for this task are highly desirable.

The difficulties in this task mainly come from the diversity of the texts (including extreme text size and font variations, and tilted or curved baselines), the complexity of the backgrounds (including many vaguely text-like objects such as fences, windows, and cobblestones) and difficult illumination conditions. OCR algorithms designed for scanned documents perform very poorly on such photos. See Fig. 3(a). Much better results are obtained by applying an OCR algorithm to the output of a text detector designed specifically for such images, as illustrated in Fig. 3(b).

The SnooperText detector initially locates candidate characters on the images by using image segmentation and shape-based character/non-character binary classification. The candidate characters found in this step, represented by their bounding boxes, are grouped by simple geometric criteria to form either candidate words or candidate text lines. These candidate text regions are then validated by a binary text/non-text region classifier that rejects any candidate region that does not appear to contain a single line of text. This classifier uses the T-HOG descriptor [4], which is based on the multi-cell *histogram of oriented gradients* (HOG) of

☆ This paper has been recommended for acceptance by Daniel Lopresti, Ph.D.
* Corresponding author.
E-mail address: rodrigo.minetto@gmail.com (R. Minetto).

1077-3142/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.cviu.2013.10.004

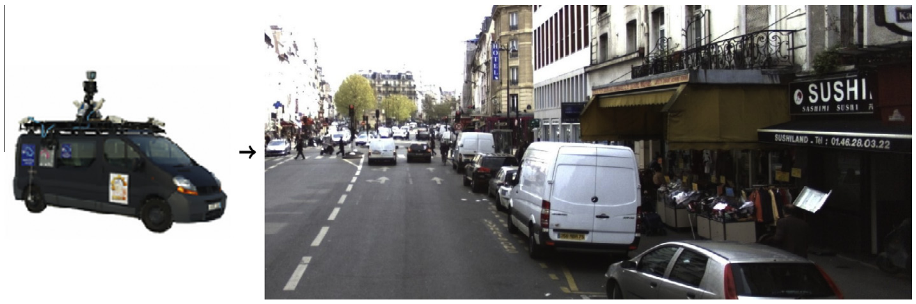


Fig. 1. The iTowns imaging vehicle and a captured image.

Dalal and Triggs [5]. These steps are performed in a multi-scale fashion, in order to efficiently handle widely different character sizes and to suppress irrelevant texture details inside the characters. Finally, the regions found by SnooperText are fed to Tesseract's back-end for OCR processing [6]. See Fig. 4.

Tests show that the accuracy of SnooperText on street images is comparable to that of the best text detectors described in the literature [1,7–11], and better than Tesseract's own text detector.

The SnooperText detector described and tested here is an improved version of the detector presented at ICIP 2010 [12]. The improvements include the use of the T-HOG descriptor for text-region validation, and the tuning of various internal parameters of the algorithm, such as the number and spacing of levels of the multi-scale pyramid and the range of character sizes considered at each level. SnooperText is implemented in Java and its source is available at the project's site [13].

This paper is organized as follows. In Section 2 we review the literature on text detectors and text/non-text region classification, with emphasis on urban photos. The SnooperText detector is described in Section 3, and its experimental evaluation is reported in Sections 4 and 5. The limitations of SnooperText are described in Section 6.

2. Previous work

2.1. Text detection

There is an extensive literature on text detection. The surveys of Jung et al. [14] and Liang et al. [15] cover some systems up to 2005. Many published text detectors are devoted to specific contexts, such as postal addresses on envelopes [16], cursive handwriting [17], and license plates [18]. Only a few systems have been designed specifically for photos of outdoor scenes [1,7,8,19–21].

Text detection algorithms can be classified into two categories. A *bottom-up* algorithm first attempts to identify probable characters, which are then grouped into words or texts. A *top-down* algorithm first attempts to find regions of the image that appear to contain text, and then tries to split those regions into characters.

The system of Hinnerk Becker [22] (winner of the 2005 ICDAR challenge) is an example of bottom-up solution. It uses an adaptive binarization scheme to extract character regions which are then combined into text lines according to certain geometrical constraints. The detector of Alex Chen et al. [22] (second place in the 2005 ICDAR challenge) follows the top-down approach: it identifies probable text regions of the image by their statistical properties, which are then segmented into presumed characters.

In 2007, Mancas-Thillou and Gosselin [19] proposed another top-down approach. They focused on the segmentation and extraction of characters, assuming that the text-containing regions were previously identified. They used pixel clustering by color similarity and log-Gabor filters to segment characters. This approach is prone to fail in those texts with similar colors for foreground and background.

In 2010, Epshtein et al. [7] proposed a bottom-up approach, that they called Stroke Width Transform (SWT), to detect characters in images. They used the pixel gradients orientation over image edges to determine a "local stroke width", and gathered pixels with similar stroke widths into candidate characters. (They also provided one of the image datasets [23] that we used in our tests.)

In 2011 and 2012, Chen et al. [8] and Neumann et al. [10], proposed bottom-up methods based on Extremal Regions (ER). Chen et al. used Maximally Stable Extremal Regions (MSER), which are a subset of ER, for edge-enhancement in order to candidate character detection. The letter candidates were then filtered out using stroke width information computed by the SWT. However, as observed by Neumann et al., MSER detectors have problems with

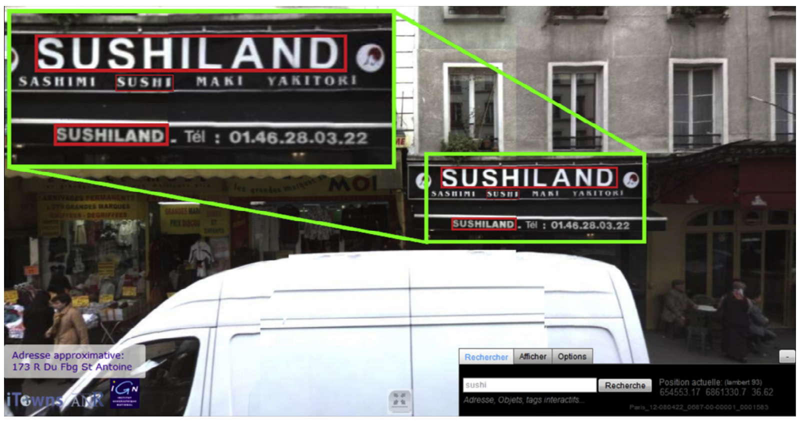
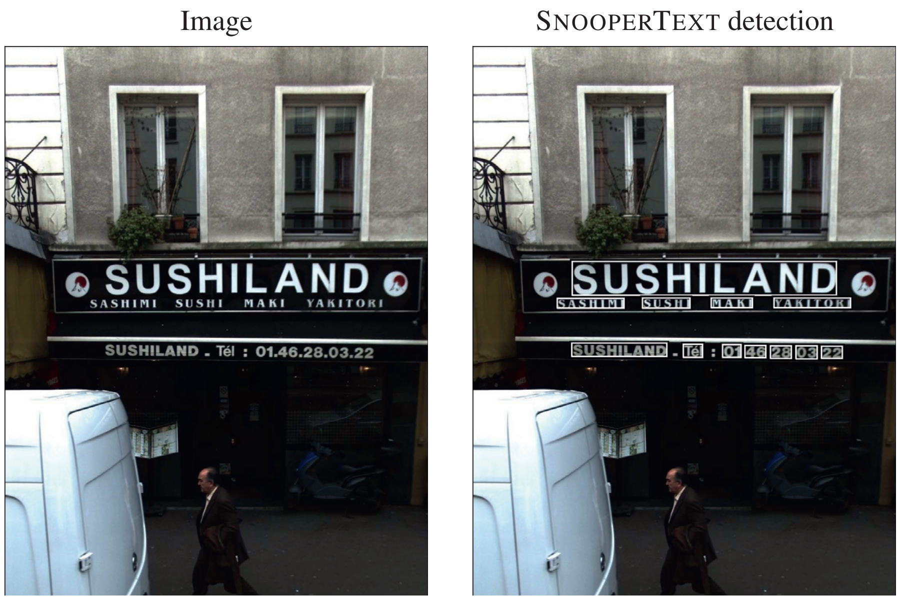


Fig. 2. Result of a search for the query string "sushi" through the iTowns user interface. The textual information was automatically extracted from the photographs.



TESSERACT

```
. . . . .  _ _ -   P V
.H '!=N!.QK.?
SUSHILAND . Tel : 01.46.28.03.22
M
,2
5
\-
E.
\
```

SNOOPERTEXT+TESSBACK

```
SUSHILAND
MAKI
YAKITORI
SUSHI
SASHIMI
SUSHILAND
Tel
01
46
28
03
22
```

Fig. 3. Top left: a store-front photo from the iTowns image base. Bottom left: output of the Tesseract OCR software applied to that whole image. Top right: text regions identified by SnooperText on that image. Bottom right: output of Tesseract's back-end OCR module (TessBack) applied to those regions.

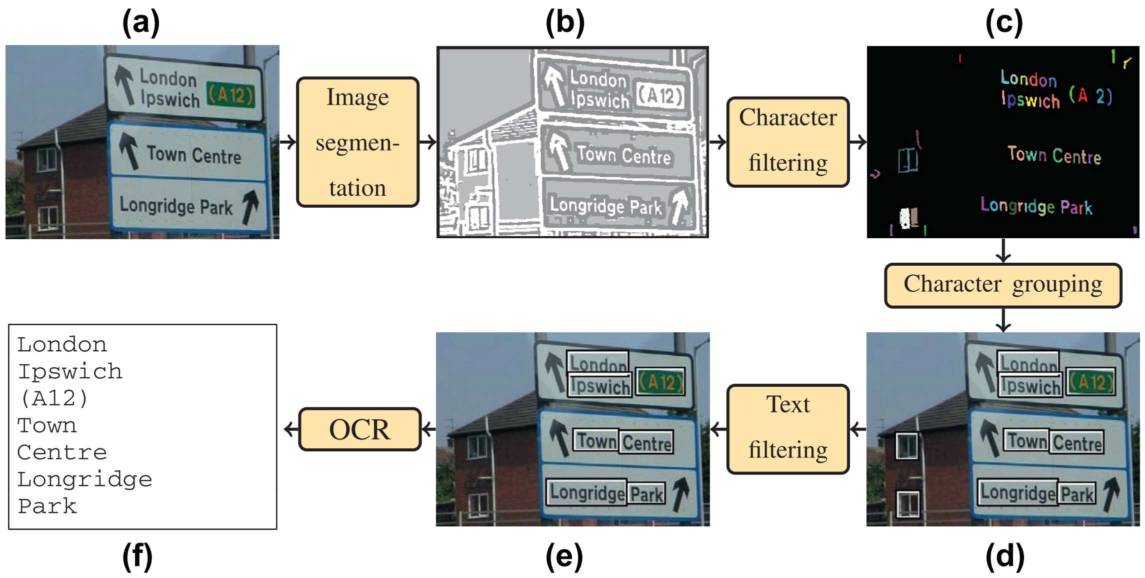


Fig. 4. Overall diagram of the SnooperText detector (a)–(e) and the OCR step (e) and (f). These steps are repeated at several scales of image resolution.

blurry images and characters with low contrast. Therefore, Neumann et al. used all ERs and classified them as being characters or not by developing original features.

Also in 2011, Pan et al. [9] proposed an hybrid multi-scale approach for text detection. They used a sliding window detector, in each scale of the pyramid, composed by a cascade of classifiers trained over HOG features, to initially estimate the text positions. The estimates were then used to enhance the original image in order to help the character segmentation. Then, the authors performed character classification and grouping.

In 2012, Yi et al. [11] and Yao et al. [1] proposed bottom-up methods that use the stroke width information to character identification. As observed by Yi et al., the letter boundary, used by the SWT, may be broken or connected to a non-text object due to background interference. To avoid this problem, the authors proposed to combine edge pixel clustering and the structural analysis of the stroke boundary with a color assignment procedure. Characters were grouped by geometric criteria. Yao et al. used the SWT for character extraction and two layers of filters based on geometric and statistical properties, as well as a classifier trained with scale and rotation invariant features to reject non-text characters found by the SWT. The character grouping was done by considering the stroke properties, geometric and color features of nearby characters. A greedy hierarchical agglomerative clustering method was also applied to aggregate character pairs into candidate chains.

2.2. Text/non-text region classification

Comparatively little has been published about text/non-text region *classification* algorithms, although they are often present as post-filters in many text detectors.

Text/non-text region classification is often cast as a texture classification problem, and several texture descriptors have been considered in the literature. For instance, in 2004, Kim et al. [24] described a text recognizer that decomposes the candidate sub-image into a multi-scale 16×16 cell grid and computes wavelet moments for each block. Then each block is classified as text or not using an SVM. The ratio of text to non-text outcomes is used to decide if the entire sub-region is text or non-text. In 2005, Ye et al. [25] described a similar text recognizer based on multi-scale wavelet decomposition; however, they used more elaborate features, including moments, energy, and entropy. In 2004, Chen and Yuille [26] proposed a descriptor that combines several features, including 2D histograms of image intensity and gradient, computed separately for the top, middle and bottom of the text region, as well as for more complex slices subdivisions of the image—89 features in total.

Other text detectors, such as the one described by Anthimopoulos et al. [27] in 2010, have used descriptors based on multi-scale *local binary patterns* (LBP) introduced by Ojala et al. [28]. Their descriptor has 256 features.

In 2012, Yi et al. [11] proposed a text line descriptor that combines the Gabor filter with gradient and stroke information. They used the block patterns proposed by Chen and Yuille. Their descriptor has 98 features.

The use of gradient orientation histograms (HOGs) as texture descriptors was introduced by Dalal and Triggs in 2005 [5], for human recognition. HOG-based descriptors have since been used for other object recognition and tracking problems [29]. They have been used in some recent text recognizers. The classifier described in 2008 by Pan et al. [30] partitions the candidate sub-image into 14 cells, as proposed by Chen and Yuille, but computes for each cell a 4-bin HOG complemented by a 2×3 array of LBP features. Their resulting descriptor has 140 features.

Other HOG-based text recognizers have been proposed in 2009 by Hanif and Prevost [31] for single-line text, and Wang et al. [32] for isolated Chinese and Roman characters as well as single-line text. Hanif and Prevost's descriptor has 151 features (16 cells, each with an 8-bin HOG and a standard deviation, plus 7 cell mean differences). The descriptor of Wang et al. has 80 features (8 cells, each with a 8-bin HOG, 1 mean difference and 1 standard deviation).

2.3. Optical character reading

The last stage of a text extraction system is to parse the candidate text-containing regions of the images to yield the text strings.

Robust OCR algorithms especially designed for images of urban scenes is an active area of research, and some recently advances were described by Wang et al. [33], Mishra et al. [34] and Neumann and Matas [10]. However, an evaluation of OCR algorithms is beyond the scope of this paper, since our contribution is limited to the text detection part.

3. Description of the text detector

As shown in Fig. 4, the SnooperText detector consists of four main modules: *image segmentation*, *character filtering*, *character grouping*, and *text region filtering*. These modules are applied at various scales of resolution, as described in Section 3.5.

3.1. Image segmentation module

The segmentation algorithm used in SnooperText was developed by Fabrizio et al. [35]. It is a modified version of Serra's *toggle mapping* [36], a morphological operator for local contrast enhancement and thresholding, using morphological erosions and dilations [37] to define the local foreground and background levels.

Specifically, in order to segment an input image $\mathbb{I}$, we first compute a local background image $\mathbb{B}$ by gray-scale erosion (neighborhood minimum) and a local foreground image $\mathbb{F}$ by gray-scale dilation (neighborhood maximum), using an 11×11 square structuring element. Note that $\mathbb{B}(p) \leqslant \mathbb{I}(p) \leqslant \mathbb{F}(p)$ for every pixel p. Then each sample $\mathbb{I}(p)$ is mapped to a ternary class value $\mathbb{D}(p) \in \{0, 1, 2\}$ as follows. If $|\mathbb{F}(p) - \mathbb{B}(p)|$ is less than a fixed threshold $c_{\min}$, then $\mathbb{D}(p)$ is set to 1 (indeterminate). Otherwise, $\mathbb{D}(p)$ is set to 0 (presumed background) or 2 (presumed foreground) depending on whether the relative brightness $|\mathbb{I}(p) - \mathbb{B}(p)|/|\mathbb{F}(p) - \mathbb{B}(p)|$ is less than or greater than another threshold c_{med}.

Since the thresholding is not symmetrical between dark and light regions, and target scenes often have light text on dark background, the segmentation is repeated on the negative (pixel-wise complemented) image. See Fig. 5.

3.2. Character filtering module

The segmented foreground regions of the positive and negative images are then screened to identify plausible characters. First, the module checks simple size and aspect ratio constraints

$$h_{\min} \leqslant h \leqslant h_{\max}$$
$$r_{\min} \leqslant h/w \leqslant r_{\max}$$

where w and h are the width and height of the segment's axes-aligned bounding box, and $h_{\min}$, $h_{\max}$, $r_{\min}$ and $r_{\max}$ are parameters of the module. See Fig. 6(a). Each segment that satisfies these constraints is then tested with a more elaborate character/non-character classifier based on the shape of the segmented region; see Fig. 6(b).

The shape classifier is based on three scale- and rotation-invariant shape descriptors extracted from the segmented region: Fourier moments, pseudo-Zernike moments, and an original polar encoding [35]. These descriptors are fed to three separate SVM classifiers, whose numeric outputs are packed as a three-dimensional vector and fed to a final SVM classifier [38]. The output of the final SVM is then thresholded to yield a binary character/non-character decision. See Fig. 7.

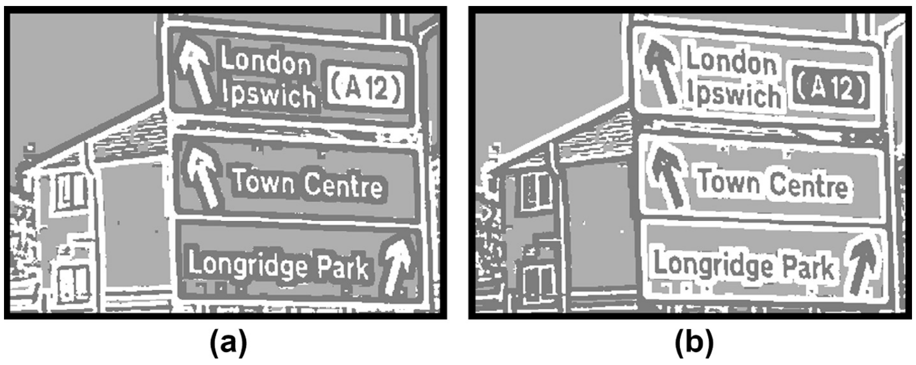


Fig. 5. (a) Toggle segmentation of the original image of Fig. 4 into background (dark gray), foreground (white) and indeterminate (light gray) pixels. (b) Toggle segmentation of the negative image.

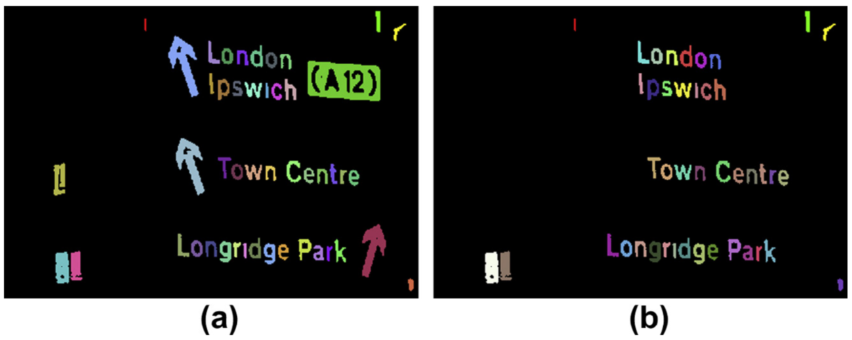


Fig. 6. (a) The foreground segments of Fig. 5(a) that satisfy the height and aspect ratio constraints with h_{min} = 13 px, h_{max} = 78 px, r_{min} = 0.5, and r_{max} = 8.0. (b) The subset of the segments in (a) that pass the character/non-character shape classifier.

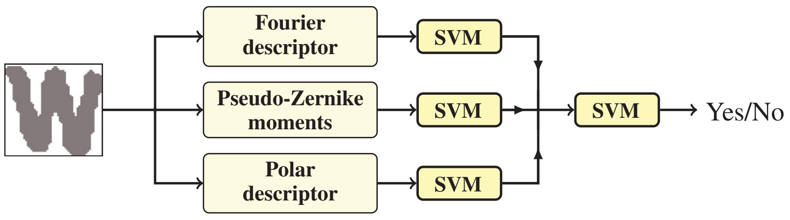


Fig. 7. The SVM-based character/non-character shape classifier.

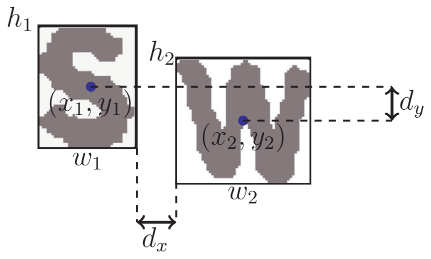


Fig. 8. Geometric parameters used for character grouping.

3.3. Character grouping module

SnooperText's character grouping module joins the candidate characters found by the character detector into text regions – which may be either words or text lines – according to geometric criteria defined by Retornaz and Marcotegui [39]. These criteria take into account the heights h_1, h_2 and widths w_1, w_2 of the two bounding boxes, as well as the coordinates (x_1,y_1) and (x_2,y_2) of their centers. See Fig. 8.

Specifically, let $h = \min(h_1, h_2), d_x = |x_1 - x_2| - (w_1 + w_2)/2$ and $d_y = |y_1 - y_2|$. Note that d_x is negative if and only if the two boxes overlap in the x direction. Then the two boxes are said to be *compatible*—that is, assumed to belong to the same text word or line—if and only if

$$|h_1 - h_2| < t_1 h$$
$$d_x < t_2 h$$
$$d_y < t_3 h$$

where t_1, t_2 and t_3 are parameters of the module. The parameter t_2, in particular, determines whether the groups will be words or text lines.

These criteria are applied to all pairs of detected characters. The groups are the equivalence classes of the transitive closure of this compatibility relation.

In the iTowns application we found that grouping characters into words often failed for store names because of extra-wide spaces used between characters. Increasing t_2 to cover those cases was not feasible because it would cause words to be joined in texts with normal inter-character and inter-word spaces. To get around that problem we run the character grouping module twice, with increasing values t_2' and t_2'' for the relative separation limit, but considering in the second pass only those character candidates that were not joined to any group by the first pass. Characters candidates that remain ungrouped after both passes are discarded; this requirement normally eliminates a large fraction of the false positives (non-character regions classified as characters by the previous steps).

Each group is then summarized by a single axis-aligned rectangle, which is the bounding box of its component characters. See Fig. 9.

The grouping module is applied separately to the candidate characters found in the segmentation of each version of the image, positive and negative. Each version produces a list of rectangles, each rectangle being a candidate text line region. These two lists are then merged, and any two regions that have significant overlap (70% of the area of the smaller box) are fused into a single candidate text line region.

3.4. Text filtering module

The task of SnooperText's text filtering module is to examine the image contents of each candidate region that is output by the character grouping module, and discard those that do not seem to contain a text line. Specifically, this stage is intended to eliminate those spurious text line candidates that result from two or more non-character image segments that passed the character filtering module and were accidentally grouped together. See Fig. 10.

The text filtering module is basically a texture classifier based on the T-HOG descriptor [4]. The latter is a variant of Dalal and Triggs's R-HOG descriptor [5], specialized to capture the gradient distribution characteristic of character strokes in occidental-like scripts. The T-HOG descriptor is fed to an SVM classifier, whose output is thresholded to give a binary text/non-text region classification.

The T-HOG descriptor is based on the observation by Chen and Yuille (2004) that different areas of a text have distinctive distributions of gradient orientations [26]. The reason is that the strongest gradients are generally perpendicular to the strokes that form the characters.

The HOG-based text/non-text discriminators described in the literature, generally divide the image into a two-dimensional array of $n_x \times n_y$ cells and compute a separate histogram of gradient orientations with a fixed number n_b of bins within each cell, as Dalal and Triggs did for human body recognition [5]. The resulting multi-hog descriptors, that are often complemented with other statistics, typically have more than 100 features.

However, while a two-dimensional cell array may be justifiable for isolated characters, it does not seem to be useful for multi-character texts of variable width. In such texts, the gradient distribution is largely independent of horizontal position; therefore, a cell layout with vertical cuts increases the size of the descriptor without providing any additional relevant information. Indeed, through extensive experiments [4] we confirmed that, for any descriptor length, partitioning the image into a small number of horizontal stripes (between 3 and 7) was generally more effective than a two-dimensional cell arrangement. Moreover, near-optimal results could be obtained with relatively small descriptors.

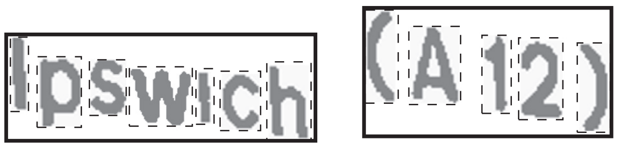


Fig. 9. Grouping characters into text words.

We also found that pre-scaling the given text region to a small fixed height H (between 20 and 30 pixels), preserving its aspect ratio, was more effective than computing the HOGs at the original image resolution. This resizing step seems to provide a good balance between preservation of useful detail and removal of noise and spurious texture.

The detailed description of the T-HOG descriptor and its experimental analysis have been published separately [4]. In brief, the sub-image delimited by the candidate rectangle is extracted from the input image $\mathbb{I}$, converted to gray-scale, scaled to the fixed height H, and normalized with a Gaussian weight window to compensate for local variations of brightness and contrast. This normalized image is then divided into n_y horizontal stripes, the image gradient is computed at each pixel within the stripe, its direction is quantized into a small number n_b of equal angular ranges, and the corresponding bins of the histogram are incremented. Opposite directions are identified, so each bin is π/n_b radians wide. The T-HOG descriptor is the concatenation of those n_y histograms.

The contribution of each pixel to the histogram is weighted by the gradient's norm, so that the small gradients that result from camera and quantization noise are largely ignored. Both the stripes and the histogram bins have gradual boundaries in order to minimize the impact of small vertical shifts and rotations of the text inside the bounding box.

3.5. Multi-scale processing

The basic SnooperText algorithm as described above performs rather poorly on images that contain characters of widely different font sizes and styles, as usually happens in photos of urban scenes. In particular, characters that are much larger than the structuring element used in the morphological thresholding are often over-segmented. To overcome this problem, the basic SnooperText detector is applied in a multi-scale fashion [40].

More precisely, for each image $\mathbb{I}$, SnooperText first builds a multi-scale *image pyramid* $\mathbb{I}^{(0)}, \mathbb{I}^{(1)}, \ldots, \mathbb{I}^{(m)}$. The base $\mathbb{I}^{(0)}$ of the pyramid is the original image $\mathbb{I}$, and each subsequent image (*level*) $\mathbb{I}^{(k)}$ is a copy of the preceding one $\mathbb{I}^{(k-1)}$, reduced in width and height by a factor $1/\mu$, for some real parameter μ greater than 1. Therefore level $\mathbb{I}^{(k)}$ has $1/\mu^{2k}$ as many pixels as level $\mathbb{I}^{(0)}$. The maximum level m depends on the size of the original image and the minimum size of the characters to be detected.

The character detection and character grouping modules are applied separately to each level of the pyramid. As described in Section 3.2, at each level k the algorithm only looks for characters whose heights lie in a bounded range $[h_{\min} \ldots h_{\max}]$ which corresponds to the range $[\mu^k h_{\min} \ldots \mu^k h_{\max}]$ in the original image. The parameters $h_{\min}$ and $h_{\max}$ should be chosen so that there is some overlap between two consecutive scales k and $k+1$, namely $h_{\max} > \mu h_{\min}$. Segmented regions whose height fall outside the interval $[h_{\min} \ldots h_{\max}]$ are ignored, since they are expected to be found at other scales. See Fig. 11. One advantage of the multi-scale approach is that we can use a structuring element of fixed (and modest) size in each morphological operation, with significant speed gains. Note that the cost of processing the whole image pyramid, for characters of any size, is only $\sum_{i=0}^{m} 1/\mu^{2i} \approx \mu^2/(\mu^2-1)$ times the cost of processing the original image for characters with the height range $[h_{\min} \ldots h_{\max}]$.

Another advantage of the multi-scale approach is that it makes the segmentation algorithm insensitive to character texture – high frequency details that are much smaller than the characters themselves. Those details may cause each character to be split into several separate segments, and will tend to confuse the character/non-character classifier. With the multi-scale approach, these problems are largely avoided when the segmentation procedure

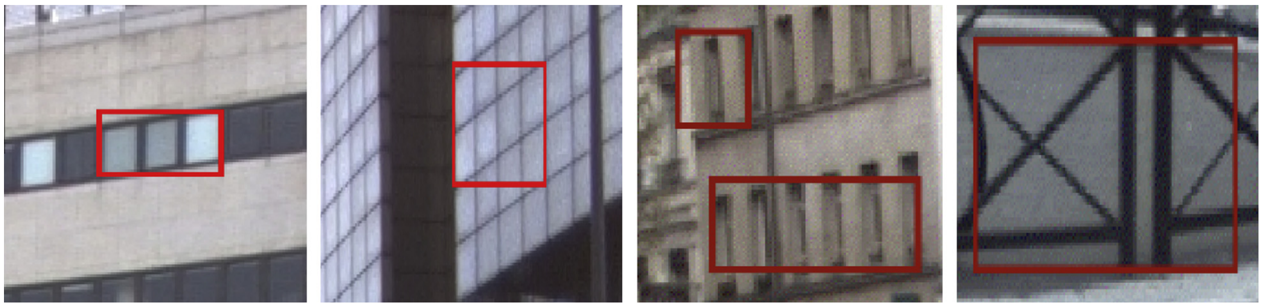


Fig. 10. Examples of false text candidates produced by the grouping module.

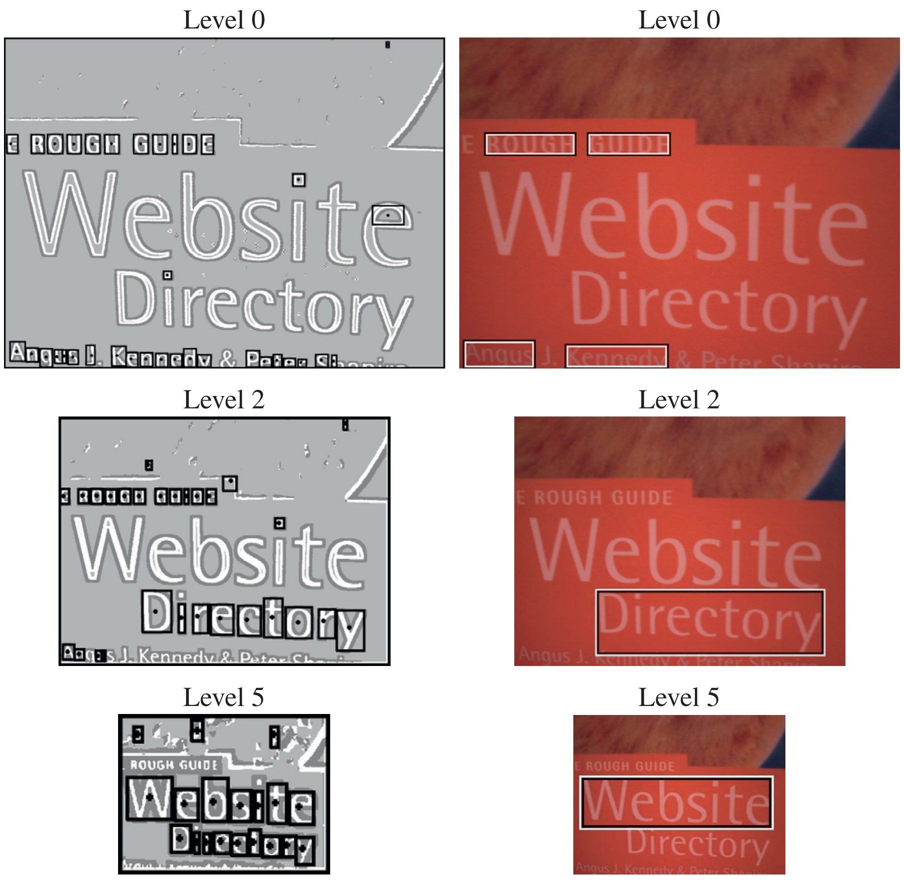


Fig. 11. Example of multi-scale segmentation and character detection, showing accepted candidate characters (left) and accepted text lines (right) at each level.

is applied at the scale where the characters are still legible but those finer details have been blurred away. See Fig. 12.

3.6. Asymptotic worst-case analysis

The asymptotic running time of the image segmentation step is $O(N)$ where N is the number of pixels in the input image. Since the size of each segment that passes the size/aspect criteria is bounded, the cost of computing the shape invariants for each segment is also bounded, and therefore the running time of the character filter is proportional to the number M_1 of segments that pass those criteria. The cost of the T-HOG text region validation module is at worst linear on the total area of the text regions, which is bounded by a multiple of the number M_2 of characters that enter the grouping step. Since $M_2 \leqslant M_1 \leqslant N$, these parts of SnooperText run in $O(N)$ time.

The naive implementation of the character grouping module enumerates all pairs of candidate characters that pass through the character filter module, and therefore has cost $\Theta(M_2^2)$. In practice this is usually not a problem, since very little computation is spent on each pair, and M_2 is almost always much smaller than N (typically, hundreds of character candidates against millions of pixels). However, some peculiar images—such as a wall covered with many round dots—may generate tens of thousands of character candidates, in which case the naive grouping algorithm may become the bottleneck. Fortunately, since the characters have bounded size, it is possible to implement the grouping step to run in $O(M_2)$ time with a proper data structure.

4. Performance of the text detector

In this section we compare the performance of SnooperText with that of other text detectors published in the literature in their ability to locate text lines on images of building façades.

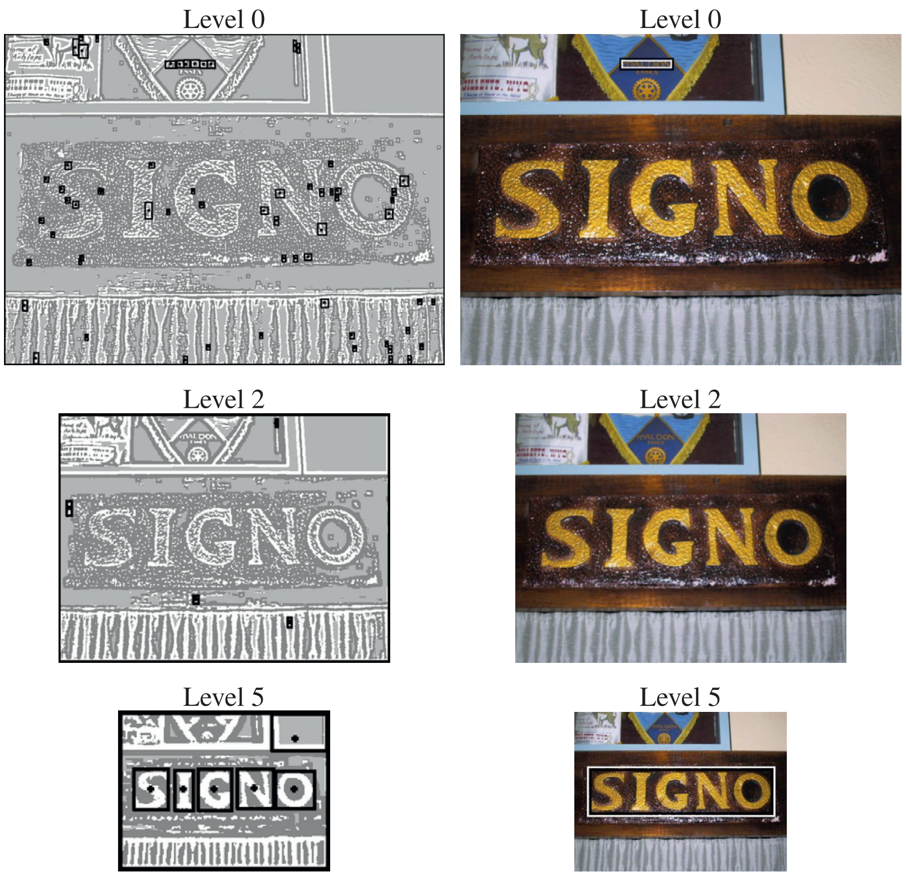


Fig. 12. Example of multi-scale segmentation and character filtering with reduction factor $\mu = \sqrt{2}$, illustrating suppression of spurious texture at the proper scale. Note that the characters of 'SIGNO' are over-segmented in levels 0 and 2 but correctly segmented and recognized (black boxes) in level 5.

4.1. Image collections

For our tests, we used four image collections, described below. For each of these datasets we have a *reference file* containing the bounding boxes of the text regions visible in each image, and the corresponding text contents, in a simple XML format. This data was obtained by human inspection of the images, as opposed to examination the actual scene.

1. *ITW*: a subset of the iTowns Project's image collection [2], consisting of 100 frontal high-resolution color photos of Parisian façades, with 1080 × 1920 pixels, as taken by the iTowns vehicle. The reference file, containing 848 readable text words, is available on-line [13].
2. *SVT*: a public benchmark of 249 urban photos selected from the Google Street View images by Wang et al., ranging from 1024 × 768 to 1918 × 898 pixels. The reference file contains 647 readable words [41].
3. *EPS*: the benchmark used by Epshtein et al. [7], with 307 color images of urban scenes, ranging from 1024 × 768 to 1024 × 1360 pixels, taken with hand-held cameras. The reference file, containing 1981 readable text lines, was provided by the authors [23] and converted to XML by us [13].
4. *ICD*: the "testing" half of the 2005 ICDAR Challenge collection [42], consisting of 249 color images, ranging from 307 × 93 to 1280 × 960 pixels, captured with various digital cameras, of book covers, road signs, posters, etc. [42]. The reference file has 1107 readable text words.

The ICD collection is not very appropriate for our purposes, since it includes images that differ substantially from photos of storefront signage in many ways (such as sharpness, brightness, contrast, angle of view, font variety, and the frequency of occlusions). We included it because it is a popular benchmark for text detection, and it is the only one for which we have reliable data on the performance of several other detectors.

4.2. Parameter settings

In all these tests, the SnooperText parameters were set as follows. In the image segmentation module: minimum contrast $c_{\min} = 30/255$, relative segmentation threshold $c_{\text{med}} = 0.79$. In the character filtering module: character height limits $h_{\min} = 13$ px and $h_{\max} = 78$ px, aspect ratio limits $r_{\min} = 0.5$ and $r_{\max} = 8$. In the character grouping module: max relative height difference $t_1 = 0.70$, max relative y offset $t_3 = 0.40$. In the text filtering module: extracted image height $H = 24$ px, number of T-HOG cells (horizontal strips) $n_y = 7$, bins per histogram $n_b = 9$. The detector was applied on an image pyramid with reduction factor $\mu = \sqrt{2}$ and maximum level $m = 9$ (10 levels).

The t_2 parameter (maximum relative letter spacing) of the character grouping module was set differently for each database, so that characters would be grouped in the same way (words or lines) as in the corresponding reference file. Thus, for the EPS dataset we used a single grouping pass with $t_2 = 1.4$ to get the characters grouped into lines. For the ITW, ICD and SVT we performed two grouping passes, with relative spacing limits $t_2' = 0.38$ (to get isolated words of normal text) and $t_2'' = 1.1$ (to get isolated words of store names and other texts with extra-wide inter-character spacing).

In the text filtering module, T-HOG descriptors with $n_y = 7$ stripes and $n_b = 9$ bins provided the best scores overall. However, in separate tests we found that the settings $n_y = 4$ and $n_b = 5$, that reduce the descriptor size from 63 to 20, would have lowered the scores by only 1–2% on average. The text filtering SVM was configured to use a Gaussian χ^2 kernel, whose standard deviation was found by cross-validation.

4.3. SVM training

The four SVM classifiers in the character filtering module (Section 3.2) were trained on a dataset of bi-level images prepared by Fabrizio et al. [35,43]. The set contains 16,200 instances of uppercase and lowercase letters (positive samples) and 16,200 non-letter shapes (negative samples). See Fig. 13.

To train the SVM of the text filtering module we used true and false examples produced by SnooperText's character detection and grouping modules. Since these databases are fairly small, instead of splitting each database into training and evaluation halves, we used the whole database for evaluation and a combination of the other databases for training. Specifically, for the tests with the `ICD` and `SVT` datasets we trained the SVM with the "training" subset of the ICDAR Challenge dataset (with 258 images and 1157 hand-annotated words), together with all of the `ITW` and `EPS` datasets. For the tests with the `EPS` collection we trained with the ICDAR "training" subset together with the whole `ITW` dataset. Finally, for the tests with the `ITW` collection we trained with the `ICD` "training" subset and the whole `EPS` dataset.

In each case, we ran SnooperText's character detection and grouping modules on the chosen collection of training images. The resulting set X of rectangles was compared with the set Y of rectangles in the corresponding reference files, and separated into true positives (X^+) and false positives (X^-). Note that the rectangles in X^+ were similar, but not identical, to the corresponding reference rectangles. We also identified the set Y^- of the false negatives, that is, rectangles in the reference files that did not match any rectangle in X. The text filter was then trained with the T-HOG descriptors of $X^+ \cup Y^-$ as the positive samples, and of X^- as the negative samples.

4.4. Competing detectors

We compared SnooperText against several state-of-the-art text detectors described in the literature. Specifically, we compared it with the contestants of the ICDAR Challenge [22], and also with the detectors of Epshtein et al. [7], H. Chen et al. [8], Pan et al. [9], Neumann et al. [10], Yi et al. [11], and Yao et al. [1]. (The system of Mancas-Thillou and Gosselin [19] uses the text detector of Alex Chen, which is included in our set.)

We added to the list of competing text detectors the front-end module of the popular open-source Tesseract OCR software, denoted here TessFront. The front-end's task is to locate the candidate text regions in the input image before calling the back-end

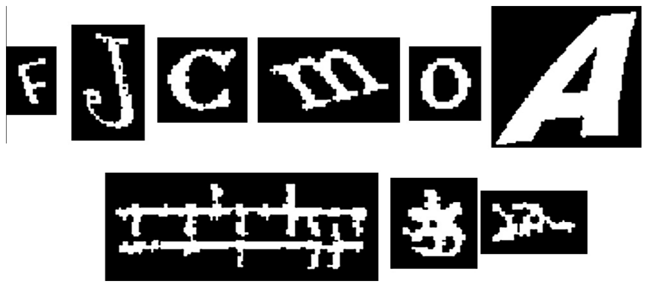


Fig. 13. Some of the positive samples (top) and negative samples (bottom) used to train the SVM shape classifiers for character/non-character discrimination. (Provided by Fabrizio [35].)

(TessBack) to parse them. Tesseract is considered one of the best OCRs publicly available today [6]; however, its front-end was designed for scanned documents, and it usually reports a large number of false positives when applied to photos of 3D scenes. Finally, we also added to the list of competing detectors the combination of TessFront with the T-HOG as an output filter (TessFront + T-HOG).

4.5. Detection performance metrics

The quantitative criteria we used to compare these text detector systems are based on the ICDAR 2005 measure of similarity [22] between two rectangles r, s, defined as

$$m(r,s) = \frac{A(r \cap s)}{A(r \cup s)} \tag{1}$$

where $A(t)$ is the area of the smallest rectangle enclosing the set t. The function $m(r,s)$ ranges between 0 (if the rectangles are disjoint) and 1 (if they are identical). See Fig. 14.

The function m is extended to a set of rectangles S by the formula

$$m(r,S) = \max_{s \in S} m(r,s) \tag{2}$$

From this indicator one derives the ICDAR *precision* P and *recall* R scores [22],

$$P = \frac{\sum_{r \in E} m(r,T)}{\#E} \qquad R = \frac{\sum_{r \in T} m(r,E)}{\#T} \tag{3}$$

where T is the set of rectangles in the reference file, and E is the set of rectangles reported by the detector.

For ranking purposes, the ICDAR 2005 Committee used the *F-measure* [22], which is the harmonic mean of precision and recall: $F = 2/(1/P + 1/R)$.

4.6. Computing average scores

There are several ways of averaging the P, R, and F scores over a multi-image database. The approach used by the ICDAR 2005 scoring program (method I) is to evaluate P, R and F separately for each image, and then compute the arithmetic mean of all three scores over all images. Another approach (method II) is to compute P and R for each image, then take the arithmetic means of all P and R values, and compute F from these means. Yet another approach (method III) is to compute the precision and recall formulas (3) by taking E and T as the union of all text regions in all images.

We note that averaging method I suffers from higher sampling noise and a negative bias compared to the other two, because it gives equal weight to each image irrespective of the number of recoverable text objects in it, and because the F-score is a non-linear function of the P and R ratios. In particular, the averaged F score (method I) tends to be lower than the harmonic mean of averaged P and R (method II). This point must be considered when comparing F values reported by different authors, since it is not always clear how they were averaged.

4.7. Detection test results

The text detection scores on the four image collections are shown in Tables 1–4. In most cases, neither the program nor the detected regions were available; therefore, we had to rely on the scores reported by the authors. For the reasons above, we recomputed the overall F score ourselves (according to the averaging method II), for all detectors, from the global P and R scores reported by the authors. We could not use method III because, for most competitors, the required data (the set of rectangles detected in each image) was not available.

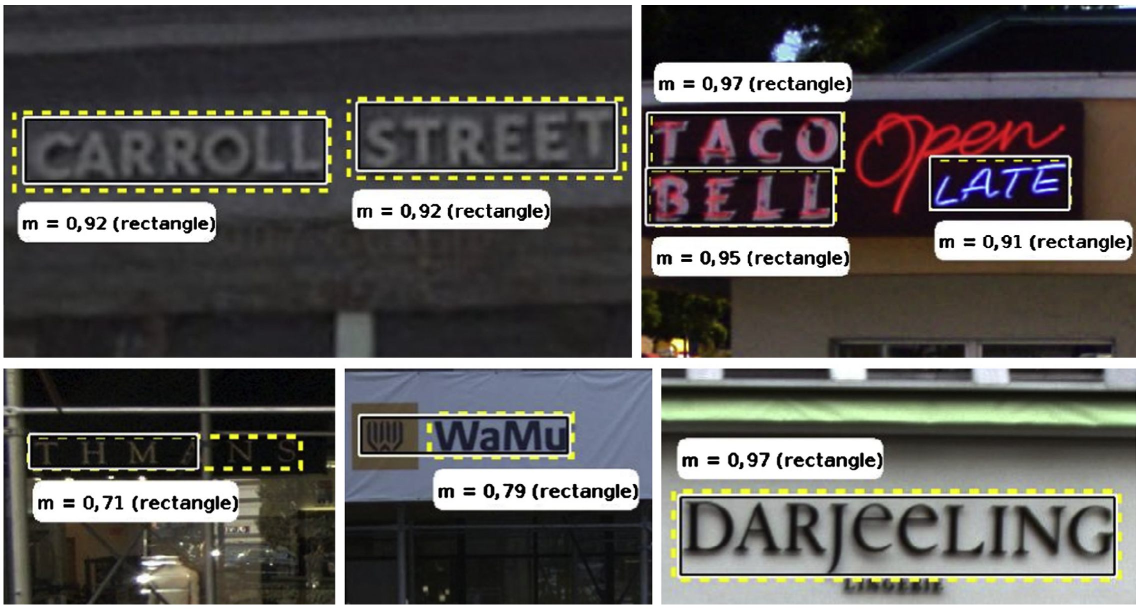


Fig. 14. The rectangle similarity score $m(r,s)$ for various text regions s detected by SnooperText (solid outlines) and the best-matching regions r from the reference file (dashed).

Table 1
Text detection scores of SnooperText and other detectors on the ITW dataset.

Detector	Detection scores		
	P	*R*	*F*
SnooperText	**0.71**	**0.51**	**0.59**
TessFront + T-HOG	0.30	0.13	0.18
TessFront	0.05	0.15	0.07

Table 2
Text detection scores of SnooperText and other detectors on the SVT dataset.

Detector	Detection scores		
	P	*R*	*F*
SnooperText	**0.36**	**0.54**	**0.43**
Neumann et al. [10]	0.19	0.33	0.26
TessFront + T-HOG	0.15	0.15	0.15
TessFront	0.04	0.18	0.06

Table 3
Text detection scores of SnooperText and other detectors on the EPS dataset.

Detector	Detection scores		
	P	*R*	*F*
SnooperText	**0.60**	**0.51**	**0.55**
Epshtein et al. [7]	0.54	0.42	0.47
TessFront + T-HOG	0.21	0.10	0.13
TessFront	0.02	0.14	0.04

As can be seen from Tables 1–4 the performance of SnooperText is comparable to that of the best detectors described in the literature. In particular, Table 3 shows that it outperformed the Stroke Width Transform detector [7] of Epshtein et al. on its own dataset.

4.8. Time analysis

The average execution time of SnooperText for each dataset is shown in Table 5. The tests were carried out on an Intel Core i7 machine (3.4 GHz) with 32 GB of RAM running Linux and Java.

Table 4
Text detection scores of SnooperText and other detectors on the ICD dataset. The competitors of the ICDAR 2003 and 2005 challenges are marked with †.

Detector	Detection scores		
	P	*R*	*F*
Yi et al. [11]	0.73	0.67	**0.70**
Pan et al. [9]	0.67	**0.70**	0.69
SnooperText	**0.74**	0.63	0.68
Yao et al. [1]	0.69	0.66	0.67
Chen et al. [8]	0.73	0.60	0.66
Epshtein et al. [7]	0.73	0.60	0.66
Hinnerk Becker†	0.62	0.67	0.64
Alex Chen†	0.60	0.60	0.60
Ashida†	0.55	0.46	0.50
HWDavid†	0.44	0.46	0.45
Wolf†	0.30	0.44	0.36
Qiang Zhu†	0.33	0.40	0.36
TessFront + T-HOG	0.35	0.27	0.30
Jisoo Kim†	0.22	0.28	0.25
Nobuo Ezaki†	0.18	0.36	0.24
TessFront	0.18	0.29	0.22
Todoran†	0.19	0.18	0.19

5. End-to-end performance

To conclude this article, we report on the effectiveness of SnooperText as the front end for its motivating application, the iTowns application.

In order to maximize the portability of their software, the iTowns project chose the open-source Tesseract package as the OCR engine. As we observed in Section 1, Tesseract on its own performs rather poorly on the iTowns images. We obtained better results after replacing its built-in text detector (TessFront) by SnooperText.

Table 5
The average execution time of SnooperText.

Dataset	Average speed (s)
ITW	14.9
SVT	8.9
EPS	12.3
ICD	7.7

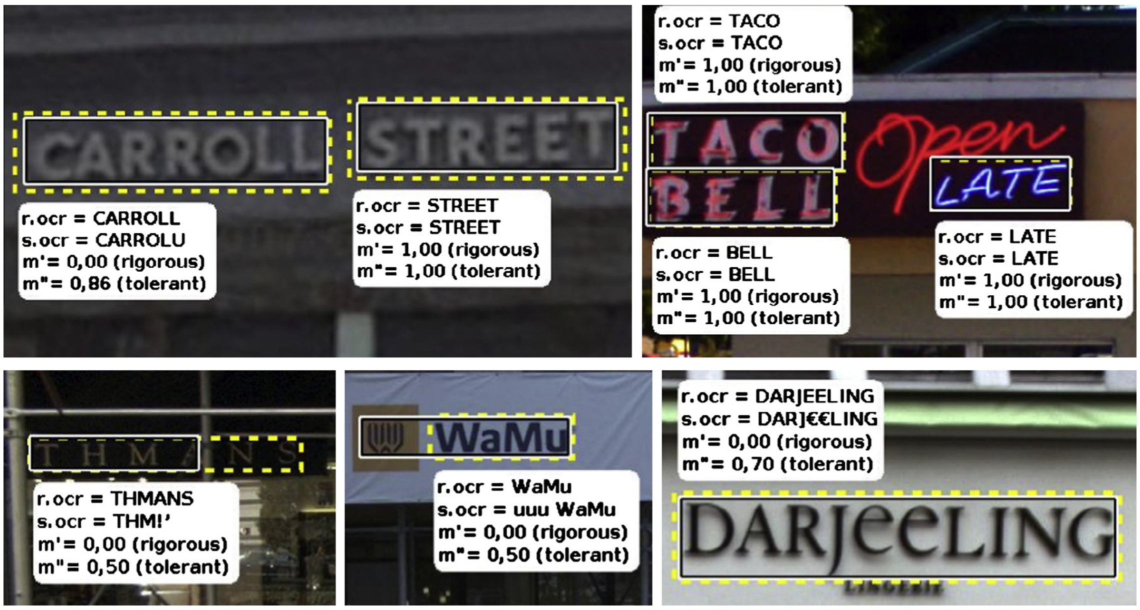


Fig. 15. The OCR similarity scores $m'(r,s)$ and $m''(r,s)$ for various text regions r extracted by SnooperText + TessBack (solid outlines), and the best-matching regions s from the reference file (dashed).

Table 6
OCR performance scores of the Tesseract back-end with the three text detectors on the ITW dataset.

Detector	OCR scores					
	Rigorous			Tolerant		
	P'	R'	F'	P''	R''	F''
HandCrop	0.29	0.29	0.29	0.50	0.50	0.50
SnooperText	0.24	0.21	0.23	0.44	0.37	0.40
TessFront + T-HOG	0.28	0.03	0.06	0.45	0.07	0.12
TessFront	0.01	0.05	0.01	0.01	0.10	0.03

Besides SnooperText and TessFront, we evaluated two other detector algorithms, namely: the "best possible" detector (HandCrop), that returns the manually annotated text regions as defined in the reference file; and the TessFront module with its output filtered by SnooperText's text region validation module (TessFront + T-HOG). Each of these detectors has a similar output, namely a list of rectangular sub-images that are presumed to contain the individual words in each image. Each of these rectangular sub-images were fed to the TessBack module with option 8 ("parse single word"). Tesseract can use a list of valid words to improve its accuracy, but we disabled that feature in order to make the test language-independent.

5.1. OCR performance metrics

For these comparisons, we used two scoring functions that take into account the correctness of the OCR-extracted text. Both functions assume that the strings are converted to lower case, because it is often impossible to tell whether a text in urban signage is in upper or lower case (for example, any text that uses only the letters C, O, S, U, V, X, W and Z).

We assume that the OCR algorithm attaches to each rectangle r reported by the detector the corresponding text string, denoted by $r.ocr$. We define the *rigorous OCR similarity score* m' for two rectangles r and s as

$$m'(r,s) = \begin{cases} 1 & \text{if } m(r,s) \geqslant \lambda \text{ and } r.ocr = s.ocr \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where m is the rectangle similarity function defined by formula (1), and λ is a fixed threshold (0.2 in our tests).

The scoring function m' may be considered too rigorous, because at the current state of the art one cannot expect that an OCR algorithm will correctly read store and product names which are missing from its spell-checking dictionary. Therefore, we also defined a *tolerant OCR similarity score* m'' that gives credit for partially correct OCR readings; namely,

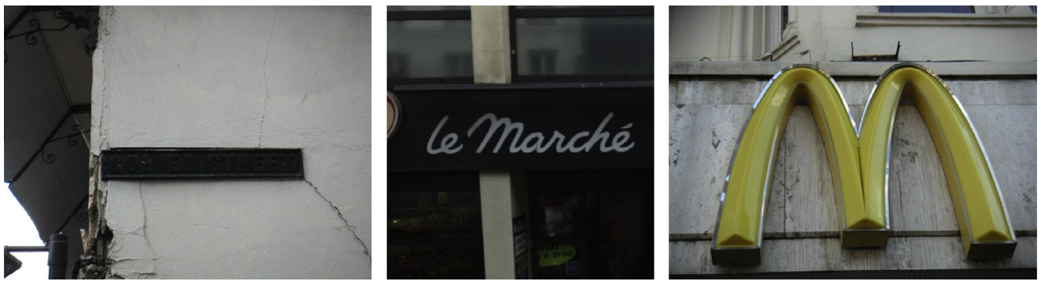


Fig. 16. Some text regions missed by SnooperText.

$$m''(r,s) = \begin{cases} 1 - \frac{\mathrm{dist}(r.ocr,s.ocr)}{\max(|r.ocr|,|s.ocr|)} & \text{if } m(r,s) \geqslant \lambda \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Here $|u|$ denotes the length of string u, and dist denotes the Levenshtein distance between strings [44]. The latter is defined as the minimum number of edit operations needed to transform one string into the other, where each operation is the insertion, deletion, or substitution of a single character. Since the Levenshtein distance does not exceed the length of the longest string, the metric $m''(r,s)$ ranges between 0 (when the strings have no characters in common) and 1 (when the strings are equal).

As in Section 4.5, we extend the scoring function m' to a set of OCR-scanned rectangles S by the formula

$$m'(r,S) = \max_{s \in S} m'(r,s) \quad (6)$$

We then define the *rigorous OCR performance scores* P' (precision) and R' (recall) by the formulas

$$P' = \frac{\sum_{r \in E} m'(r,T)}{\#E} \quad R' = \frac{\sum_{r \in T} m'(r,E)}{\#T} \quad (7)$$

where T is the set of manually identified text regions in all input images, with the fields *ocr* set to the visually extracted text values, as recorded in the reference file; and E is the set of text regions reported by the detector, with the TessBack-computed *.ocr* fields. As before, we combine the OCR precision and recall into a single OCR score $F' = 2/(1/P' + 1/R')$. The *tolerant OCR performance scores* P'' R'', and F'' are defined in the same way, using m'' instead of m' in formulas (6) and (7). See Fig. 15.

5.2. End-to-end test results

Table 6 shows the end-to-end scores obtained with the various front-ends on the iTowns dataset. We note in Table 6 that the rigorous OCR score F' obtained with SnooperText (23%), while low in absolute terms, is 79% of the score obtained with TessBack on hand-cropped word images (29%). The tolerant OCR score F'' of SnooperText (40%) is 80% of the hand-cropped score (50%). In both aspects, SnooperText is significantly better that Tesseract's front end, even when the latter is combined with the T-HOG text region validation module. Therefore, we can say that the OCR algorithm, not the text detector, is the main bottleneck of the iTowns system at present.

6. Limitations

SnooperText's errors seem to be mostly due to texts that are near the low limit of legibility (small in size, blurred, partly obscured by noise); to groups of two or more characters that cannot be separated by the segmentation phase; to cursive or excessively distorted fonts; and to isolated characters that are discarded by the grouping module. Also, SnooperText currently does not attempt to detect vertical or extremely tilted text. See Fig. 16.

7. Conclusions

The experiments reported in Section 4 show that SnooperText is comparable to state-of-the-art text detection algorithms for images of building façades. In particular, it can accurately locate more than 60% of the text regions present the ICDAR Challenge benchmark, with less than 30% of false positives, even though that benchmark is not representative of the intended application. We attribute SnooperText's good performance mainly to its use of multi-scale processing for segmentation and character detection, and to its effective text region validation module based on the T-HOG descriptor.

The SnooperText detector was very effective also in the iTowns project. On a sample of the iTowns images (which are somewhat more difficult than those of the ICDAR Challenge) SnooperText was able to accurately locate about 50% of the legible text regions, with less than 30% of false positives. As reported in Section 5, the end-to-end performance of the system (including the external TessBack OCR module) was rather low, with only 21% of those texts being successfully parsed. However, that result was still 4 times the success rate of the unaided Tesseract reader, and more than 70% of the success rate obtained by applying TessBack on the hand-cropped texts.

Acknowledgments

This work was partly supported by FAPESP (Grants 07/54201-6 and 07/52015-0), CNPq (301016/92-5), CAPES/COFECUB (592/08), and ANR 07-MDCO-007-03.

References

[1] C. Yao, X. Bai, W. Liu, Y. Ma, Z. Tu, Detecting texts of arbitrary orientations in natural images, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2012, pp. 1083–1090.
[2] A.N. de Recherche, The iTowns Project <http://www.itowns.fr>.
[3] Google, Google Street View <http://maps.google.com/help/maps/streetview>.
[4] R. Minetto, N. Thome, M. Cord, N.J. Leite, J. Stolfi, T-HOG: an effective gradient-based descriptor for single line text regions, Pattern Recognition (PR) 46 (3) (2013) 1078–1090 (Elsevier).
[5] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2005, pp. 886–893.
[6] Google, Tesseract-OCR <http://code.google.com/p/tesseract-ocr/>.
[7] B. Epshtein, E. Ofek, Y. Wexler, Detecting text in natural scenes with stroke width transform, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2010, pp. 886–893.
[8] H. Chen, S. Tsai, G. Schroth, D. Chen, R. Grzeszczuk, B. Girod, Robust text detection in natural images with edge-enhanced maximally stable extremal regions, in: IEEE International Conference on Image Processing (ICIP), 2011, pp. 2609–2612.
[9] Y.-F. Pan, X. Hou, C.-L. Liu, A hybrid approach to detect and localize texts in natural scene images, IEEE Transactions on Image Processing (TIP) 20 (3) (2011) 800–813.
[10] L. Neumann, J. Matas, Real-time scene text localization and recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 3538–3545.
[11] C. Yi, Y. Tian, Localizing text in scene images by boundary clustering, stroke segmentation, and string fragment classification, IEEE Transactions on Image Processing (TIP) 21 (9) (2012) 4256–4268.
[12] R. Minetto, N. Thome, M. Cord, J. Fabrizio, B. Marcotegui, Snoopertext: a multiresolution system for text detection in complex visual scenes, in: IEEE International Conference on Image Processing (ICIP), 2010, pp. 3861–3864.
[13] Rodrigo Minetto, SnooperText Source Code, 2013 <http://www.dainf.ct.utfpr.edu.br/~rminetto/projects/snoopertext/>.
[14] K. Jung, K. Kim, A. Jain, Text information extraction in images and video: a survey, Pattern Recognition (PR) 37 (5) (2004) 977–997 (Elsevier).
[15] J. Liang, D. Doermann, H. Li, Camera-based analysis of text and documents: a survey, International Journal on Document Analysis and Recognition (IJDAR) 7 (2) (2005) 84–104.
[16] P.W. Palumbo, S.N. Srihari, J. Soh, R. Sridhar, V. Demjanenko, Postal address block location in real time, Computer 25 (7) (1992) 34–42.
[17] I.S.I. Abuhaiba, M.J.J. Holt, S. Datta, Recognition of off-line cursive handwriting, Computer Vision and Image Understanding (CVIU) 71 (1) (1998) 19–38 (Elsevier).
[18] N. Thome, A. Vacavant, L. Robinault, S. Miguet, A cognitive and video-based approach for multinational license plate recognition, Machine Vision and Applications (MVA) (2) (2011) 389–407 (Springer).
[19] C. Mancas-Thillou, B. Gosselin, Color text extraction with selective metric-based clustering, Computer Vision and Image Understanding (CVIU) 107 (1–2) (2007) 97–107. Elsevier.
[20] R. Minetto, N. Thome, M. Cord, J. Stolfi, F. Precioso, J. Guyomard, N.J. Leite, Text detection and recognition in urban scenes, in: IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011, pp. 227–234. http://dx.doi.org/10.1109/ICCVW.2011.6130247
[21] R. Minetto, N. Thome, M. Cord, N.J. Leite, J. Stolfi, Snoopertrack: Text detection and tracking for outdoor videos, in: 18th IEEE International Conference on Image Processing (ICIP), 2011, pp. 505–508. http://dx.doi.org/10.1109/ICIP.2011.6116563.
[22] S.M. Lucas, Icdar 2005 text locating competition results, in: International Conference on Document Analysis and Recognition (ICDAR), vol. 1, 2005, pp. 80–84.

[23] Epshtein et al., Text Detection Database <http://research.microsoft.com/enus/um/people/eyalofek/text_detection_database.zip>.
[24] K.C. Kim, H.R. Byun, Y.J. Song, Y.W. Choi, S.Y. Chi, K.K. Kim, Y.K. Chung, Scene text extraction in natural scene images using hierarchical feature combining and verification, in: International Conference on Pattern Recognition (ICPR), vol. 2, 2004, pp. 679–682.
[25] Q. Ye, Q. Huang, W. Gao, D. Zhao, Fast and robust text detection in images, and video frames, Image and Vision Computing (IVC) 23 (6) (2005) 565–576. Elsevier.
[26] X. Chen, A.L. Yuille, Detecting and reading text in natural scenes, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2, IEEE Computer Society, Los Alamitos, CA, USA, 2004, pp. 366–373.
[27] M. Anthimopoulos, B. Gatos, I. Pratikakis, A two-stage scheme for text detection in video images, Image and Vision Computing (IVC) 28 (9) (2010) 1413–1426. Elsevier.
[28] T. Ojala, M. Pietikäinen, T. Mäenpää, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 24 (2002) 971–987.
[29] Z. Han, Q. Ye, J. Jiao, Combined feature evaluation for adaptive visual object tracking, Computer Vision and Image Understanding (CVIU) 115 (1) (2011) 69–80. Elsevier.
[30] Y.-F. Pan, X. Hou, C.-L. Liu, A robust system to detect and localize texts in natural scene images, in: IAPR International Workshop on Document Analysis Systems, IEEE Computer Society, Washington, DC, USA, 2008, pp. 35–42.
[31] S.M. Hanif, L. Prevost, Text detection and localization in complex scene images using constrained adaboost algorithm, in: International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society, Washington, DC, USA, 2009, pp. 1–5.
[32] X. Wang, L. Huang, C. Liu, A new block partitioned text feature for text verification, in: International Conference on Document Analysis and Recognition (ICDAR), Los Alamitos, CA, USA, 2009, pp. 366–370.
[33] K. Wang, B. Babenko, S. Belongie, End-to-end scene text recognition, in: International Conference on Computer Vision (ICCV), 2011, pp. 1457–1464.
[34] A. Mishra, K. Alahari, C.V. Jawahar, Top-down and bottom-up cues for scene text recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012, pp. 2687–2694.
[35] J. Fabrizio, M. Cord, B. Marcotegui, Text extraction from street level images, in: ISPRS Workshop on City Models, Roads and Traffic (CMRT), vol. 38, 2009, pp. 199–204.
[36] J. Serra, in: J.C. Simon (Ed.), Toggle Mappings: From Pixels to Features, Elsevier, 1989, pp. 61–72.
[37] J. Serra, Image Analysis and Mathematical Morphology, Academic Press, Inc., Orlando, FL, USA, 1983.
[38] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (1995) 273–297.
[39] T. Retornaz, B. Marcotegui, Scene text localization based on the ultimate opening, in: International Symposium on Mathematical Morphology (ISMM), vol. 1, 2007, pp. 177–188.
[40] J.-M. Jolion, A. Rosenfeld, A Pyramid Framework for Early Vision: Multiresolutional Computer Vision, Kluwer Academic Publishers, Norwell, MA, USA,, 1994.
[41] Kai Wang, The Street View Text Dataset, 2011 <http://vision.ucsd.edu/~kai/svt/>.
[42] S.M. Lucas, Text Locating Competition – International Conference on Document Analysis and Recognition (ICDAR), 2003–2005 <http://algoval.essex.ac.uk/icdar/Datasets.html>.
[43] J. Fabrizio, B. Marcotegui, M. Cord, Text segmentation in natural scenes using toggle-mapping, in: IEEE International Conference on Image Processing (ICIP), IEEE Press, 2009, pp. 2373–2376.
[44] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Soviet Physics Doklady 10 (8) (1966) 707–710.