Localizing Blurry and Low-Resolution Text in Natural Images

Pannag Sanketi Huiying Shen James M. Coughlan

The Smith-Kettlewell Eye Research Institute
San Francisco, CA 94115
{pannag, hshen, coughlan}@ski.org

Abstract

There is a growing body of work addressing the problem of localizing printed text regions occurring in natural scenes, all of it focused on images in which the text to be localized is resolved clearly enough to be read by OCR. This paper introduces an alternative approach to text localization based on the fact that it is often useful to localize text that is ***identifiable as text but too blurry or small to be read****, for two reasons. First, an image can be decimated and processed at a coarser resolution than usual, resulting in faster localization before OCR is performed (at full resolution, if needed). Second, in real-time applications such as a cell phone app to find and read text, text may initially be acquired from a lower-resolution video image in which it appears too small to be read; once the text's presence and location have been established, a higher-resolution image can be taken in order to resolve the text clearly enough to read it.*

We demonstrate proof of concept of this approach by describing a novel algorithm for binarizing the image and extracting candidate text features, called "blobs," and grouping and classifying the blobs into text and non-text categories. Experimental results are shown on a variety of images in which the text is resolved too poorly to be clearly read, but is still identifiable by our algorithm as text.

1. Introduction

OCR is a successful application of image processing and computer vision to the problem of reading printed text that is well resolved in good-quality images [7]. In such images, the text regions comprise the majority of pixels, and there is a minimum of non-text background clutter. A growing body of work addresses the more challenging complementary problem of finding and localizing text in natural images, which are dominated by clutter, so that the text regions may be segmented out and input to OCR. Such functionality is very useful for finding and reading printed signs, such as street, office and other informational signs, as well as other text in the environment. The text localization stage is necessary since standard OCR techniques are ill-equipped to sort through and discard the numerous non-text regions among the background clutter.

A common thread among past work on text localization is the assumption that any text to be detected is resolved clearly enough in the image to be read by OCR: images in this past work show individual text characters subtending heights of many (i.e. usually 20 or more) pixels, and adjacent characters in words are separated by enough space that they can be separately segmented.

However, we have found that this assumption may not be appropriate for our application of finding and reading text for blind and visually impaired persons, which poses two important challenges: the limited computational resources available on the cell phone platform that we are using, and the often poor visibility of text when seen from typical viewing distances and imaged at VGA video resolution. To meet these challenges, we devised a text localization algorithm that detects text even when it is not resolved large or clearly enough to be read.

Such an algorithm is useful for our application for two reasons. First, an image can be decimated and processed at a coarser resolution than usual, resulting in faster localization before OCR is performed (at full resolution, if needed). Second, in real-time applications such as a cell phone app to find and read text, text may initially be acquired from a lower-resolution video image in which it appears too small to be read; once the text's presence and location have been established, a higher-resolution image can be taken in order to resolve the text clearly enough to read it.

We demonstrate proof of concept of this approach by describing a novel algorithm for binarizing the image and extracting candidate text features, called "blobs," and grouping and classifying the blobs into text and non-text categories. Experimental results are shown on a variety of images in which the text is resolved too poorly to be clearly

978-1-4244-9497-2/10/$26.00 ©2010 IEEE

read, but is still identifiable by our algorithm as text.

2. Related Work

The literature on text localization/segmentation in cluttered images is too extensive to cover in detail, so we include here only a brief summary of some of this work that is relevant to this paper; see [7] for a survey of this research area.

Many text segmentation algorithms employ deterministic, bottom-up processes for grouping text features into candidate text regions using features such as edges, color or texture [9, 5, 6, 4]. Statistical methods have also been developed, such as an Adaboost-based algorithm [2] that uses a filter cascade to hone in on text regions.

Recent work on text localization [3], which outperforms the latest published algorithms, estimates the local width of stroke features in the image, and identifies text regions on the basis of local homogeneity of this stroke width (corresponding to consistent stroke widths in characters). However, it is important to note that the local stroke width is only well defined when the text is resolved clearly enough such that individual strokes are well delineated – which requires individual characters to be clearly visible.

Our approach was motivated by the simple observation that text regions form an identifiable texture [8], whether or not the text itself is readable. Under these conditions, letters (and groups of letters that have been merged together) in text regions have distinctive geometric properties that distinguish these regions from non-text regions. Accordingly, we extract text features (called "blobs") that fulfill these geometric properties, and use these properties as a basis for distinguishing between text and non-text (see Sec. 3.2).

3. Algorithm

The query image is binarized and the candidate text features, called "blobs", are extracted. The blobs are assigned a potential of being text based on histograms of geometric properties of blobs, learned from the training data. Neighboring blobs are grouped into "superblobs" based on a similarity measure. The resulting superblobs are classified into text and non-text categories.

One of the main advantages of our method is that we do not use a sliding-window approach. The text regions emerge automatically as a result of our algorithm.

3.1. Blob Feature Detection

First, the image is binarized using a technique very similar to Niblack's method. For each pixel, the mean value over a small neighborhood, say, 11x11 is calculated. If the pixel value is greater than the mean plus a threshold, say, 10, its binary value is set to 255; otherwise, it is set to 0. Then we extract connected components, called "blobs."

A blob is a collection of connected horizontal line segments. A horizontal line segment, L_{yi}, is defined by its y-coordinate, y, and starting and ending x-coordinates: x_1, and x_2. For a given binary image, one can find one array of horizontal line segments for each y-coordinate, AL_y. A blob is defined by a collection of connected L_{yi}, that is also organized into a group of AL_y according to their y-coordinates.

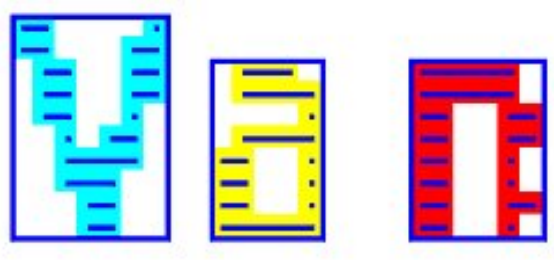

Figure 1. Blobs as a collection of horizontal line segments

In Figure 1, there are three blobs, represented by three colors and their bounding rectangles. In each blob, there are many horizontal line segments, L_{yi}, represented by blue lines, that are connected. For a given y, AL_y, the array of L_{yi} can have one or more L_{yi}.

The algorithm for blob construction is shown in Algorithm 1, with binary image as input.

Figure 2 shows the blobs extracted from an image, overlaid on the top of the original image. Notice that the image is negated. Hence, the blobs correspond to dark text on light background. Figure 3 shows a zoomed in shot of the blobs. When text is well resolved, each character in a word usually gives rise to one blob; when text is poorly resolved, one or more adjacent characters are merged into a single blob.

Either way, the shape of the blob provides evidence for whether the blob is text or non-text. In addition, alignment among nearby blobs provides additional evidence.

3.2. Blob Scoring

Once the blobs from the image are extracted, each of those is assigned a unitary potential of being text. We use histograms of geometric properties of the blobs learned from training images to determine the potential.

The blob is a bitmap, whose top left corner is (x_{left}, y_{top}), width W and height H; and the geometric center of the bounding box is given by (x_c, y_c). The geometric properties extracted from a blob are as follows.

1. Blob Height (H)
2. Area Ratio (AR) = $\frac{\text{Bitmap Area}(BA)}{\text{Area of Bounding Box}}$
3. Aspect Ratio (AspR) = $\frac{H}{W}$

Algorithm 1 Blob Detection Algorithm

1. For each row, find an array of horizontal line segments, indexed by row number (AL_y).

2. At the end of Step 1, we get an array of line segment arrays (ALA) representing the image.

3. Initiate a blob with one line, Ly_i from first AL_y of ALA, and remove L_{yi} from ALA.

4. Sweeping downward: for each y where the blob has any line segment, and for each line in the line array, search AL_{y+1} in ALA for connected lines. If a line segment is found, add it to the blob at $(y+1)$, and remove it from ALA.

5. Sweeping upward: for each y where the blob has any line segment, and for each line in the line array, search AL_{y-1} in ALA for connected lines. If a line segment is found, add it to the blob at $(y-1)$, and remove it from ALA.

6. Repeat steps 4 and 5 until no new line segment is added for either step 4 or 5. Store the blob.

7. Repeat steps 3 - 6 until ALA is exhausted.

Figure 2. Example of Blobs extracted from an image

4. X Moment (XM) = $\sum_{(i,j)\in \text{ blob}} (x_c - x_{ij})$

5. Y Moment (YM) = $\sum_{(i,j)\in \text{ blob}} (y_c - y_{ij})$

6. X Moment Second (XMS) = $\sum_i \left(\sum_j (y_c - y_{ij})^2\right) dA_i$

Figure 3. Example of Blobs extracted from an image: zoomed in

where, dA_i = sum of i^{th} row

7. Y Moment Second (YMS) = $\sum_j \left(\sum_i (x_c - x_{ij})^2\right) dA_j$
where, dA_i = sum of j^{th} column

8. X Moment Radius Gyration = $\frac{XMS}{BA}$

9. Y Moment Radius Gyration = $\frac{YMS}{BA}$

10. X Moment Two (XMT) = $\sum_i \sum_j (y_c - y_{ij})^2$

11. Y Moment Two (YMT) = $\sum_j \sum_i (x_c - x_{ij})^2$

12. XMoment Two Per Area = $\frac{XMT}{BA}$

13. YMoment Two Per Area = $\frac{YMT}{BA}$

14. Perimeter

15. Perimeter Sq Per Area = $\frac{Perimeter^2}{BA}$

As part of the training, around 1200 blobs in 40 images are manually labelled as text and non-text. Figures 4 through 7 show the geometric properties of a randomly chosen subset of labeled blobs. In these figures, the x-axis is just the blob index, and the y-axis shows the values (offset plus scaled) of the properties. In fact, the y-axis is inverted since Java considers y-axis as positive downwards by default. The actual scale of the index itself is not important; the figures just indicate the effectiveness of the geometric properties in distinguishing between text and non-text blobs.

The blobs are divided in to two groups, "long" and "not long" blobs, based on the aspect ratio. Blobs with aspect ratio ≤ 0.35 are treated as "long". Figure 8 shows the blobs from an image in the increasing order of aspect ratio.

Using the training blobs, a histogram of each of these properties is formed, separately for long and not long blobs, one each from positive and negative examples. The positive blobs' histogram gives the probability of a blob being

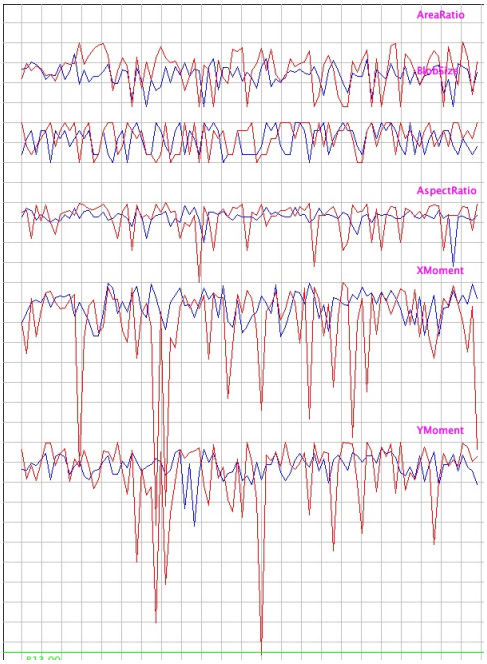

Figure 4. Basic properties of blobs (Blue = text, Red = non text)

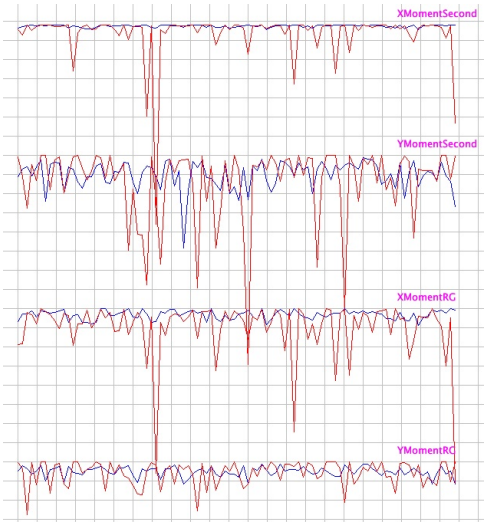

Figure 5. Second moments of blobs (Blue = text, Red = non text)

text, and the negative blobs' histogram gives the probability of a blob being not text. If a query blob is long, then the histogram set learned from long blobs is used.

A query blob's unitary potential of being text is defined as the ratio $\frac{P(\vec{f}|Text)}{P(\vec{f}|NonText)}$ as given by the histograms, where $\vec{f}$ denotes the vector of all 15 geometric properties for a given blob. A product is taken of the histograms across all

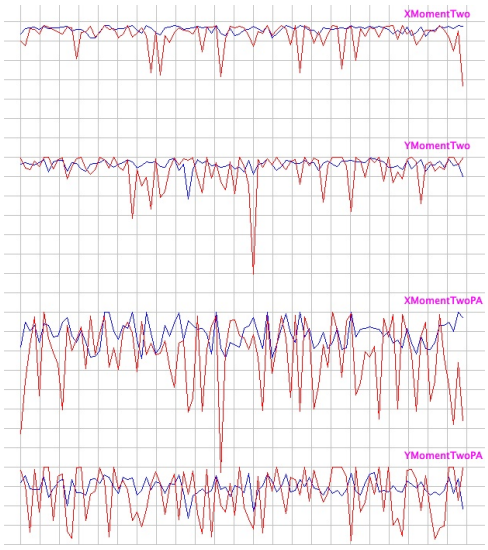

Figure 6. Second moments of blobs (Blue = text, Red = non text)

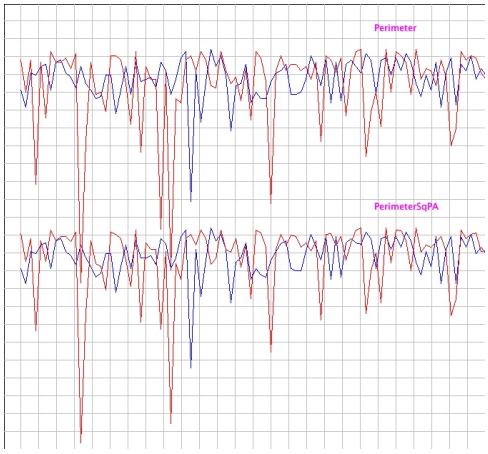

Figure 7. Perimeter related properties of blobs (Blue = text, Red = non text)

15 features, in a Naive Bayes fashion, i.e.

$$P(\vec{f}|Text) = \prod_{i=1}^{15} P(f_i|Text)$$
$$P(\vec{f}|NonText) = \prod_{i=1}^{15} P(f_i|NonText) \quad (1)$$

where $P(f_i|Text)$ and $P(f_i|NonText)$ come from the trained histograms of i^{th} feature. The query blob's unitary potential of being not text is set to a constant 1.

Figure 9 shows the blobs from an image color coded smoothly according to their potentials as obtained using Equation 1. A darker shade of blue indicates a higher potential of being text, and a darker shade of red indicates a

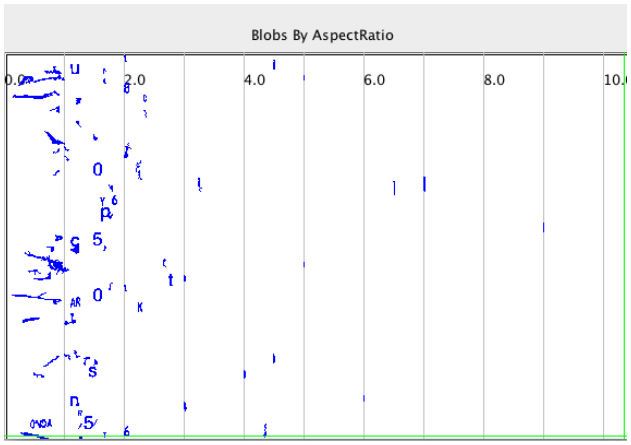

Figure 8. Showing Blobs in increasing order of Aspect Ratio

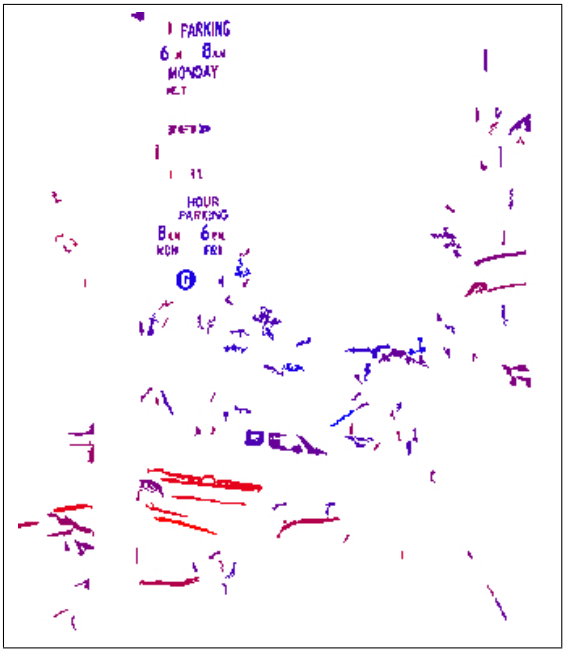

Figure 9. Blobs color graded according to their potentials (blue is highest, red is lowest)

lower potential of being text, or a higher potential of being non text. You can see from the figure that the unitary potentials provide a lot of information. They form an important basis to correctly classify and localize text. We performed experiments on blobs classification based on just unitary potentials. We used roughly 85% of our labeled blobs for training, and remaining (200) blobs for testing. Out of 200 blobs used for testing, our classifier gave 14 false negatives and 26 false positives.

Individual blobs may not represent a whole word. Along with unitary potentials, we further use context information to localize words. Context information also makes the localization robust. Hence, we group the blobs into larger chunks using a grouping algorithm.

3.3. Grouping Blobs

A graph is defined using the blobs in the image, with each blob representing a node in the graph. For each blob in the graph, we define a neighborhood criterion as follows. Two blobs are "neighbors" if they (a) overlap vertically and (b) are "close" horizontally. Vertical overlapping criterion is necessary since we are aiming to detect mostly horizontal text. The threshold for horizontal proximity is a function of the size of the blob. This is to accommodate the fact that larger texts have larger gaps in between letters. Using this neighborhood criterion, we calculate the connectivity of the blobs within the graph.

Since we are aiming to find words of text, we group the neighboring blobs which are "similar" into "superblobs". The similarity measure of two neighborhood blobs is based upon (a) 50% or more similarity in height and (b) 90% or more horizontal alignment either at the top or the bottom, i.e. the difference in the y_{top} (or the difference in the y_{bot}) values of the two blobs should be less than 10% of either blobs' height. Thus the emphasis is on horizontal alignment. Of course, two blobs have to be neighbors of each other to be considered similar. A blob is said to be similar to a superblob, if the blob is similar to any of the blobs in the superblob.

The grouping algorithm starts by creating the first superblob with the first blob. At each further iteration, for each blob in the image, the algorithm goes through all the superblobs so far. If it is similar to any of those superblobs, add the blob to that superblob. If the blob is similar to more than one superblobs, then, all those superblobs are merged in to one superblob. If none of the existing superblobs are similar to the current blob, then a new superblob is createed with the current blob alone. The iteration is repeated until all the blobs in the image are analyzed. Each resulting superblob corresponds roughly to a single word.

Figures 10 and 11 show the effectiveness of the grouping algorithm. Before grouping, the words were broken in to different blobs. After grouping, the blobs unite into superblobs, each superblob representing a word.

3.4. Final Classification

This step corresponds to classifying each resulting superblob as text or not text. The potential of a superblob being text is defined as the multiplication of the potentials of the individual blobs it contains. Each superblob is classified as text or not text based on (a) Its potential and (b) the number of the nodes contained in it, and (c) the overall area ratio of the supernode. The potential of the superblob has to be larger than a threshold. The threshold on the potential of a superblob is a function of the number of blobs contained

Figure 10. Before grouping: Color coded raw blobs

Figure 11. Post grouping: Color coded Superblobs

in it. We expect words to have more than one letter, hence two or more "similar" blobs next to each other in the image to have a high likelihood of being text as opposed to a single isolated blob. Naturally, the higher the number of blobs in a superblob, the smaller the threshold. Also, the overall area ratio of the superblob is taken in to account while clasifying. The overall area ratio of the superblob has to be larger than a certain threshold for the superblob to be classified as text.

4. Experimental Results

The algorithm was tuned to find text with height between 5 and 14 pixels. It was tested on various pictures, some of resolution 320 x 480 taken from an iPhone 3GS viewfinder (Figures 12 through 15), and some taken from an online dataset, down-sampled to approximately 200 pixels wide (figures 16 through 18). For an image of size 320 x 480, the current detection algorithm runs in around 100 ms on a standard iMac desktop computer. However, there have been no attempts at optimizing the algorithm yet.

The algorithm searches for text in one polarity, i.e. light text on a dark background. To find dark text on a light background, the algorithm is run with the image contrast reversed. Each of the results shown marks only light text in the given image. Hence, *dark text on light background will not be marked, and not considered as missed positives*. It can be seen from the results that the algorithm performs really well when the text can be seen clearly and the image has a good contrast, such as those in Figures 16 through 18. Even when the image does not have a good contrast, the algorithm performs reasonably well. In Figures 19 through 21, zoomed in regions of our results are shown. Note that the algorithm detects text even at a very low resolution (height $\leq$ 7), which may not be read even by humans. These are low quality pictures taken from an iPhone viewfinder (320 x 480). Though, not every low resolution text region is detected, this shows the effectiveness of the algorithm.

There are a few missed positives, most of which trace back to the Naive Bayes histograms based approach. Also, single letters are more likely to be missed in our algorithm since the potential threshold on a single blob is really high. With a better method to find the potential of a blob such as Adaboost [2] and random trees [1], the missed positives can be reduced. Also, more training data of different resolutions can reduce missed positives. In some cases, the word is broken down in two or more pieces. This is due to variation in the gap in between the letters or a very unclear letter in a word. These kind of cases can be greatly reduced by reiterating the grouping of superblobs. Sometimes, part of the word is seen missing such as the last letter "S" in the word "Sandwiches" in Figure 12. This is due to the fact the last letter S is not aligned with the rest of the word as per our stringent alignment criterion. If the criterion is relaxed, the word will be detected in its entirety, however, it will lead to more false positives. The false positives are only a concern with regards to the runtime speed of the overall algorithm (detection plus OCR recognition) and not with regards to the accuracy, as the false positives will be eliminated when

sent to the OCR engine.

Figure 12. Example: detected text regions (only light text on dark background)

Figure 13. Example: detected text regions (only light text on dark background)

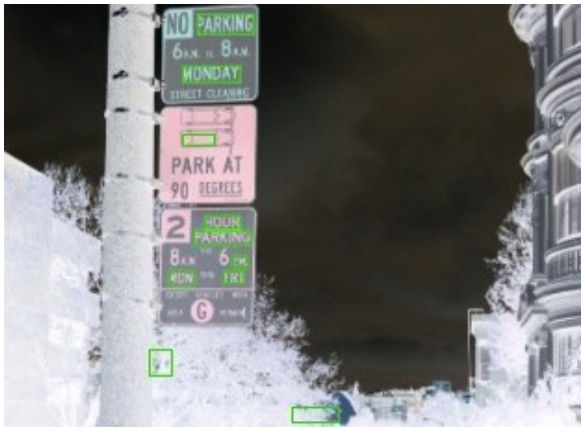

Figure 14. Example: detected text regions (only light text on dark background, image contrast reversed)

Figure 15. Example: detected text regions (only light text on dark background, image contrast reversed)

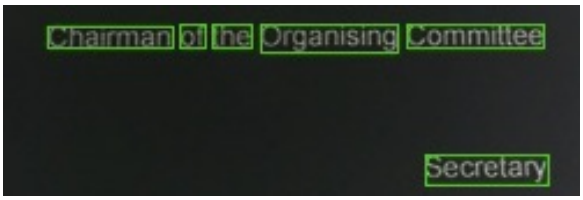

Figure 16. Example: detected text regions (only light text on dark background, image contrast reversed)

Figure 17. Example: detected text regions (only light text on dark background, image contrast reversed)

Figure 18. Example: detected text regions (only light text on dark background)

5. Conclusion

We have described a novel approach to text localization aimed at detecting text even when it is too blurry or small to be read accurately, based on the fact that such text may still form a recognizable texture. The motivation for the approach is two-fold, both to permit the processing of the image at a coarser resolution for efficiency, and to allow text to be detected in coarse video images before taking higher-

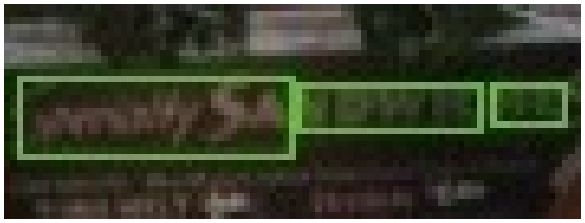

Figure 19. Example: detected low resolution text regions (zoomed portion of Figure 12)

Figure 20. Example: detected low resolution text regions (zoomed portion of Figure 13)

Figure 21. Example: detected low resolution text regions (zoomed portion of Figure 15)

resolution stills to attempt to scrutinize it, when necessary. We have demonstrated proof of concept of our approach on a variety of images, acquired both from standard text image datasets and from an iPhone at low resolution.

In the future, we plan to improve the algorithm to overcome some of its current limitations, including the requirement that text be oriented horizontally, the non-negligible incidence of false positives on text-like image regions containing structures such as trees and textured walls, and the fact that words are sometimes split into multiple pieces in the detection process. These improvements may be achieved by using better classifiers than the naive Bayes method we are currently using, including Adaboost [2] and random trees [1], and enlarging the set of training data. Finally, once the algorithm has been improved we plan to port it to the iPhone platform, and use it as the front end of a system we are developing that will enable blind and visually impaired users to find and read printed text signs.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1996. 6, 8

[2] X. Chen and A. Yuille. Adaboost learning for detecting and reading text in city scenes. In *CVPR*, 2004. 2, 6, 8

[3] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *Proc. CVPR*, 2010. 2

[4] J. Gao and J. Yang. An adaptive algorithm for text detection from natural scenes. In *Proc. CVPR*, pages 84–89, 2001. 2

[5] A. K. Jain and B. Yu. Automatic text location in images and video frames. *Pattern Recognition, International Conference on*, 2:1497, 1998. 2

[6] H. Li, H. Li, D. Doermann, D. Doermann, O. Kia, and O. Kia. Automatic text detection and tracking in digital video. *IEEE Transactions on Image Processing*, 1998. 2

[7] J. Liang, D. Doermann, and H. Li. Camera-based analysis of text and documents: a survey. *International Journal on Document Analysis and Recognition*, 7:83–200, 2005. 1, 2

[8] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vision*, 40(1):49–70, 2000. 2

[9] V. Wu, R. Manmatha, and E. M. Riseman. Finding text in images. In *DL '97: Proceedings of the second ACM international conference on Digital libraries*, pages 3–12, New York, NY, USA, 1997. ACM. 2