Text Detection and Classification in Natural Images

Andrei Bajenov[1], Haisam Islam[2]
[1]Department of Electrical Engineering, [2]Department of Bioengineering
Stanford University, Stanford, CA

***Abstract*—Text detection and classification in natural images is important for many computer vision applications. Here, to detect text, we find connected components based on consistency in stroke width, chain them based on their relative spatial positions, and use a text classification engine to filter chains with low classification confidence scores. The algorithm performs well using the ICDAR 2015 competition images, with 72.4% precision and 71.0% recall, but struggles under certain imaging conditions or types of text. Finally, we discuss ways to improve the algorithm to yield better performance.**

Introduction

Text detection in natural images is an important step in extracting the textual information, which has applications that include text image search, assistance for the visually impaired, and landmark identification. However, it is also a difficult problem due to the variability in imaging conditions, such as lighting, specular reflections, noise, blur, and presence of obstructions over the text, and in the variability of the text itself, such as its scale, orientation, font, and style [1]. Good text detection algorithms should thus be robust against such variabilities.

Various approaches have been developed to address this problem, and can be classified into either texture-based methods, which identify text based on its difference in texture from the background, or connected component (CC)-based methods, which identify text based on its geometric properties [2]. Recently, Ephstein et al developed a CC-based text detection algorithm using the Stroke Width Transform (SWT), which computes for each pixel the stroke width, i.e. the width of the most likely stroke that generated that pixel [1]. The reasoning is that stroke width generally varies less in text than background, and thus may be used as a filter for text detection. The SWT is robust to scale changes and invariant to rotation, and has shown promising results for this application [1-3].

Here, we employ the method developed by Ephstein for text detection. First, we find candidate text components using the SWT, then chain them based on their relative spatial positions. We filter the chains based on consistency between their components, and pass them to an Optical Character Recognition (OCR) engine, Tesseract [4], which we use to remove chains with low classification confidence scores. The filtered chains are reinput into Tesseract for classification. In the rest of the paper, we describe the algorithm in greater detail, show results that highlight both the strengths and weaknesses of the algorithm, and discuss the performance and ways to potentially improve it.

Methods

Figure 1 shows a flowchart of the algorithm, and Figure 2 shows intermediate images at different stages of the algorithm. Briefly, we compute an edge map of the input image, and use it to calculate the stroke widths. Adjacent pixels of similar stroke width are merged into CCs, which are chained together based on their relative spatial positions. The chains are classified using Tesseract, and those with low classification confidence scores are filtered. The remaining chains are then reinput into Tesseract for final classification.

Various filtering operations are performed throughout the algorithm to improve the robustness and performance. Because the SWT requires the text "polarity", i.e. whether the text is bright on a dark background or dark on a bright background, be known, the algorithm consists of two passes of the operations listed above, one for each text polarity, and returns the union of the chains detected in each pass.

Stroke Width Transform

The input image is first converted to grayscale. To compute the SWT, an edge map and the x-y gradients are required. Before calculating these, we blur the grayscale image to increase robustness against noise and fine detail. For the edge map, we use a 3×3 box filter and perform Canny edge detection [5] with hysteresis thresholds of 50 and 110. For the x-y gradients, we use a 5×5 Gaussian filter and apply the Scharr

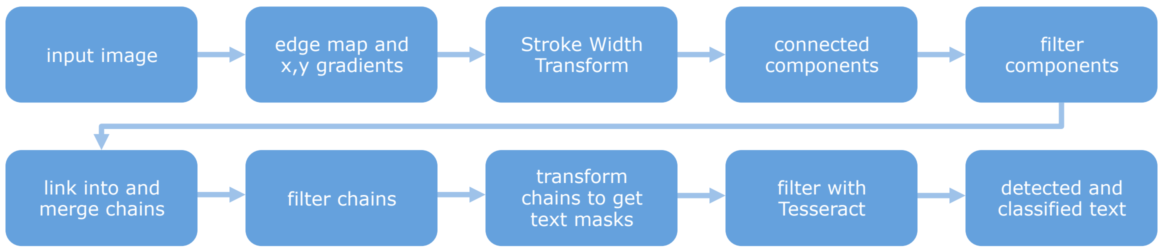

input image
edge map and x,y gradients
Stroke Width Transform
connected components
filter components
link into and merge chains
filter chains
transform chains to get text masks
filter with Tesseract
detected and classified text

Figure 1: Flow chart of one pass of the algorithm. Two passes are performed, one for each text polarity, and the union of the chains detected in each pass is returned.

operator for first derivatives [6]. The edge map and x-y gradients are shown in Figures 2b-d.

The SWT is computed as follows. At each edge pixel u, a ray is traced toward the interior of the text, either along or against the gradient g_x, depending on the text polarity. We stop if another edge pixel v is detected at which the gradient g_v points in the opposite direction, within some tolerance θ_t, i.e. $|\angle(g_u, -g_v)| < \theta_t$. We found a very conservative tolerance $\theta_t = \pi/2$ in combination with subsequent filtering operations to work best, as lower thresholds failed to detect some text CCs. All pixels intersected by the ray are assigned a stroke width value equal to the distance between the two edge pixels $||u - v||_2$. If multiple rays intersect a pixel, we use the minimum value.

After stroke widths are assigned, we perform a filtering operation that is necessary in certain parts of text where the stroke width would otherwise be inaccurate, such as near the corner of the letter “L”. This consists of first computing the median stroke width along each ray in the SWT, and then, for each pixel along the ray, if the stroke width is greater than the median value, setting it to the median value. The filtered SWT image is shown in Figure 2e.

Finding Connected Components

After we calculate the stroke widths, we group the pixels into CCs. We first apply an open operation to the stroke width image to improve separation between CCs. We then apply a flood-fill algorithm based on consistency in stroke width and color. Color consistency is measured using distance in the *Lab* color space, where distance and perceived color are linearly related [7]. Adjacent pixels are connected if the larger-to-smaller stroke width ratio is less than 3 and the *Lab* distance is less than 30, ensuring that each component is uniform in both stroke width and color. The input image masked by the current CCs is shown in Figure 2f.

Filtering Connected Components

After finding candidate CCs, we filter them based on various heuristics. Since most text has low variation in stroke width, we filter CCs whose stroke width standard deviation is greather than $1/2$ the mean stroke width. We also filter CCs with high variation in lightness and color in the *Lab* space. Since most classifiable text in images is not very small, we also filter CCs that are less than 10 pixels in width or height. Finally, we filter CCs with an aspect ratio greater than 10. The input image masked by the filtered CCs is shown in Figure 2g.

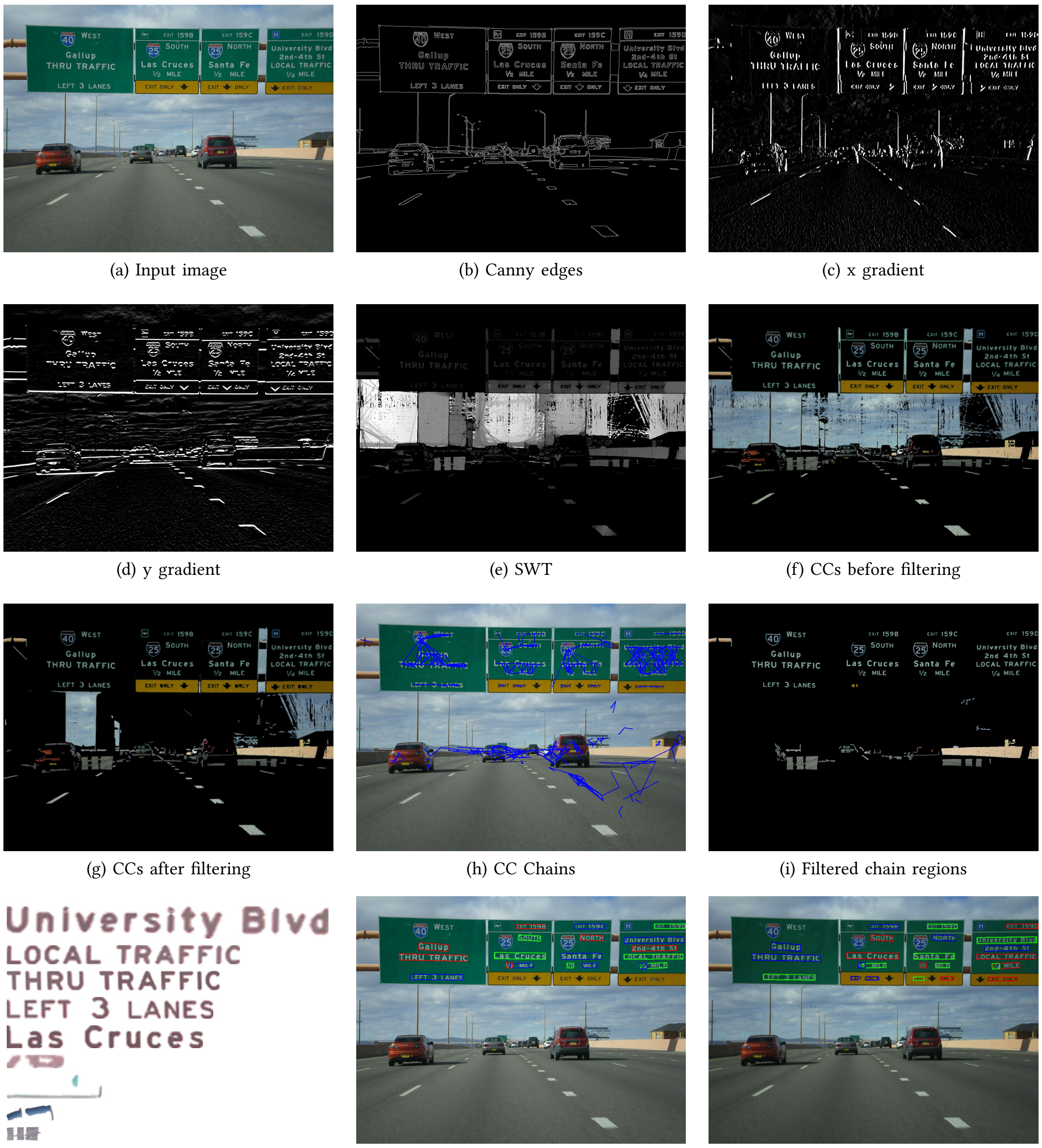
Creating and Filtering Component Chains

After filtering the CCs, we link them into chains, which are potentially words or lines of text. Adjacent CCs are linked if they have a *Lab* color distance less than 40, larger-to-smaller median stroke width ratio less than 2, and larger-to-smaller height ratio less than 2. Chains are iteratively merged if their difference in orientation is less than 15°, and then sorted based on length. The chains are shown in Figure 2h.

Chains are filtered based on the variance in stroke width within and between the components. To account for scale variations, we first normalize the stroke widths in each chain by the length of the diagonal of the chain’s bounding box plus an offset of 50, where the offset prevents overaggressive normalization of smaller chains. For filtering, we use a threshold of 1000 for within-component variance and 100 for between-component variance. The input image masked by the filtered chains is shown in Figure 2i.

Filtering and Classification using Tesseract

To yield optimal classification performance using Tesseract, we first preprocess the chains, which serve as masks over the input image. We extract the region under each mask, rotate it to be horizontal, and scale it to a height of 40 pixels. Tesseract determines



(a) Input image (b) Canny edges (c) x gradient

(d) y gradient (e) SWT (f) CCs before filtering

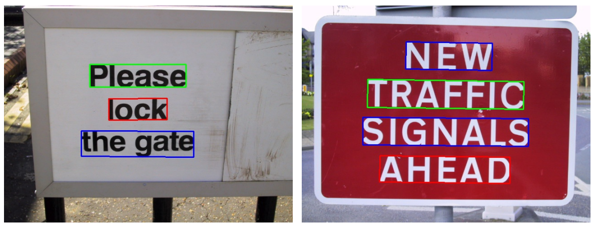
(g) CCs after filtering (h) CC Chains (i) Filtered chain regions

(j) Subset of detected chains (k) Detected text components, first pass (l) Detected components, both passes

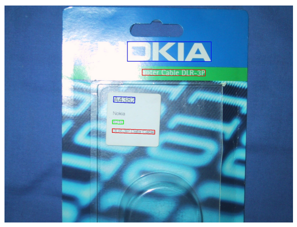
Figure 2: Input, intermediate, and final images obtained with the text detection algorithm.

foreground and background based on pixel majority, so each chain is padded with a sufficient number of background pixels. Figure 2j shows a subset of the chains detected for the example image, which include both text and non-text chains. The chains are then input into Tesseract, which provides a text classification as well as a classification confidence score. We filter the chains, favoring those with a higher confidence score and greater number of characters, in particular alphanumeric characters. The filtered chains are then reinput into Tesseract for final classification. The detected text after Tesseract filtering in the first pass of the algorithm, in which bright text on dark background is detected, is shown in Figure 2k, and after both passes in Figure 2l.Performance Measurement

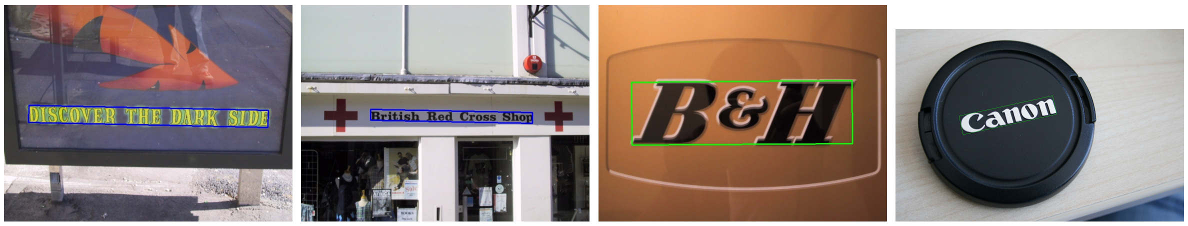
We measure the performance of the text detection



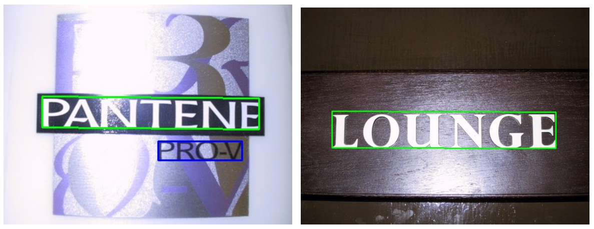
(a) Text polarity



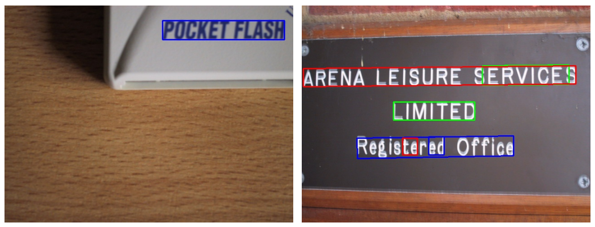
(b) Variation in scale of font



(c) Variations in fonts



(d) Variation in lighting or specular reflections



(e) Moderate amounts of blur

Figure 3: Example images for which the text is accurately detected.

Subfigure	Text classifications
(a)	"Please", "lock", "the gate"
	"NEW", "TRAFFIC", "SIGNALS", "AHEAD"
(b)	"ÏKIA", "A4382", "$210", "a tcr Cable DLR 3P"
(c)	"leC'®'ÉR THE DARK SIDJE"
	"Bntnsh Bod Shop"
	"Bl]"
	"Canon"
(d)	"PANTENE PRO V"
	"LOIINGE"
(e)	"POCKET FLASH"
	"ARENA LEISURE SERVICES", ""li-A'll'l"", "LIMITED", "Reg stared Office", "ta", "ed"

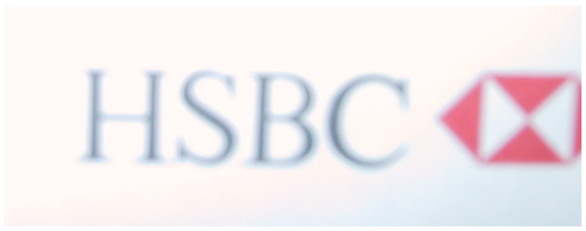
Table I: Text classifications for the examples in Figure 3. For each subfigure, each line lists the classified words for one image.

and classification on the ICDAR 2015 training data [8], which consists of 328 images. We trained our algorithm on images 100-139, and tested our algorithm on images 140-328. We used two standard metrics to assess performance, the mean precision and recall, where for each image, the precision is the fraction of detected chains that contain text, and the recall is the fraction of text elements detected.

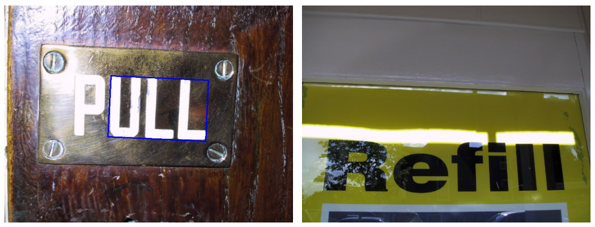
Results and Discussion

We obtained a mean precision of 72.4% and a mean recall of 71.0% on the test data. Figure 3 shows various images in which the text was correctly detected, and Figure 4 images in which the text was not detected. Images were selected to highlight both the strengths and weaknesses of the text detection algorithm. The performance on text classification was poorer. Table I shows the text classifications for the images in Figure 3.

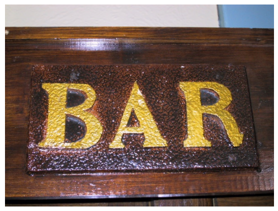
The text detection algorithm performs well for most text in natural images. Each subfigure in Figure 3 presents an imaging or text condition against which the algorithm shows some degree of robustness, at least in the range found in the examples shown. Figure 3a shows that the algorithm can detect both bright text on dark background and dark text on bright background,



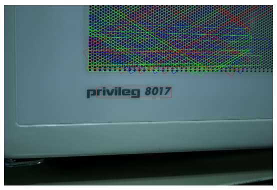
(a) Large amount of blur



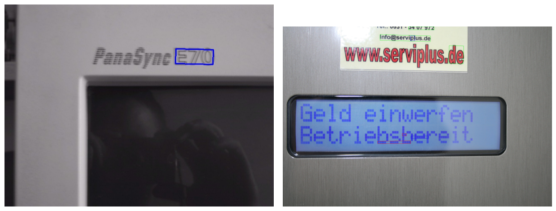
(b) Large variation in lighting or specular reflections



(c) Textured or non-flat text



(d) Repeating non-text patterns



(e) Segmented text

Figure 4: Example images for which the text is not accurately detected. Each subfigure presents an imaging or text condition for which the text detection algorithm fails.

which is expected since there are two passes in the algorithm, one to detect each polarity of text. Figure 3b shows robustness to scale, which is expected since the operations performed in the algorithm are, for the most part, scale-invariant.

Figure 3c shows robustness against the font of the text. We filtered CCs based on the assumption that the variation in stroke width is generally less for text than background, but the variation is larger for some fonts than others, e.g. serif fonts tend to have larger stroke width variation than sans-serif fonts. We used a conservative threshold for filtering so that almost no text CCs would be filtered based on stroke width variation, and relied on subsequent filtering operations to remove false positives.

Figures 3d-e show robustness against variation in lighting and specular reflections as well as moderate amounts of blur. These can affect the computed edge map, which is used to compute the SWT, as well as the variance in lightness, which is used for filtering CCs. Thus, we used conservative thresholds for both operations to ensure few text CCs would go undetected. The second image in Figure 3e also shows overlapping text detections. We do not filter overlapping chains since we do not know which one (if any) contains the "true" text and thus risk removing it.

The algorithm fails to detect text in the image under certain conditions. Figure 4a shows more severely blurred text that is undetected by the algorithm. As mentioned, blurring can cause edge detection to fail, which is the case here. Increasing the contrast or sharpening the image may mitigate the problem in some images, but may produce extra edges in others that result in incorrect stroke width values. One solution may be to estimate the blur in the image based on its autocorrelation or some other measure, and sharpen the image appropriately.

Figure 4b shows two images, one with a large variation in lighting and one in which the text is on a reflective surface with light shining on it. In the first image, the bright light blends with the "P" to produces one CC which is thus removed by filtering. One solution may be to high-pass filter the image to remove slowly varying lighting effects, or use other lighting correction methods [9]. In the second image, the light reflections produce edges inside the letters, which affects the SWT and causes certain CCs to be filtered out. This is a more difficult problem since the light reflections can split single regions into multiple regions with distinct intensities. Preprocessing the image to remove specular reflections [10] may mitigate this problem.

Figure 4c shows an image of textured or non-flat text. Such features can generate extra edges in the edge map, which as mentioned, can cause the CCs to be subsequently filtered out. The texture has much lower contrast than that found between the letters and the background, however, so an appropriate set of thresholds for the edge detection may capture only the salient edges around the letters. However, these values are not known *a priori*. Mosleh et al used a bandlet based edge-detector that generates fewer but more salient edges, yielding better results text detection [3].

Figure 4d shows an image with multiple almost identical holes, which are not filtered at any stage of the algorithm. This is unsurprising since they look similar to true text. Chen et al noted, however, that text rarely consists of almost identical repeating patterns, and used a template-matching procedure to filter such components [2].

Figure 4e shows two images which contain segmented text. In both images, each segment produces a separate CC, which is filtered out in a subsequent operation. One solution may be to dilate the CCs, thus connecting them and preventing their removal by filtering, though in certain cases, they may also cause merging with unwanted CCs, which would then be filtered out. Another solution may be to relax the constraints for linking chains so that segemented text still forms a chain that is input into Tesseract, and allow Tesseract to filter the non-text chains.

The text classification using Tesseract yielded poorer performance. For some of the sans-serif texts, thinner regions were removed, which fragmented certain letters, causing them to be misclassified. One solution may be to dilate the final detected masks, though this risks including additional non-text pixels in the Tesseract input. Another issue is that for some letters with more than one component, e.g. “i”, the smaller component was filtered out. One solution may be to use the mean color in the mask region and include any nearby pixels with similar color in the input to Tesseract. Text of non-standard font or style also pose issues for Tesseract, which performs classification based on a learned dictionary. Training Tesseract with images of various styles of text may help, but would be exhaustive given the range of text styles found in images, and may even decrease accuracy for standard text styles.

Using Tesseract for filtering requires knowledge of the language of the text since Tesseract classifies characters using a learned dictionary instead of with purely image processing operations. There are certain chains for which this is necessary since the chains may plausibly be text in some language, in which case image processing would not filter them out. Thus, a combination of image processing and dictionary learning is likely necessary to yield optimal performance in text detection.

Much of the optimization performed for the text detection algorithm involved tuning parameters to achieve a suitable compromise between the number of false positives and false negatives, which overlap for most of the operations performed. Thus, a more sophisticated algorithm that models the different imaging conditions may be required to achieve better performance. Additionally, there are also many imaging conditions not discussed, but we hope the examples shown here convey the degree of robustness of the algorithm.

Conclusion

We adapted the text detection algorithm by Ephstein et al that employs the SWT for filtering background from text, and used Tesseract for further filtering and text classification. Various preprocessing and filtering operations were performed to improve the robustness and performance of the detection algorithm, which performed well on most of the test data, but stuggled with some of the more severe variations in imaging conditions or unusual text types. Thus, further work on a more sophisticated algorithm that accounts for such variations remains to be done. The text classification gave poorer results, however, but can be improved using various methods.

Acknowledgements

We thank Dr. Bernd Girod and Dr. Gordon Wetzstein for teaching the Digital Image Processing course, which provided us with useful knowledge and the skills for not only this project but possibly future work as well. We also thank the teaching assistants Huizhong Chen and Jean-Baptiste Boin for their assistance throughout the quarter, with Huizhong providing useful suggestions for the project as well.

References

[1] B. Epshtein, E. Ofek, and Y. Wexler. "Detecting text in natural scenes with stroke width transform." In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pp. 2963-2970. IEEE, 2010.

[2] H. Chen, S.S. Tsai, G. Schroth, D.M. Chen, R. Grzeszczuk, and B. Girod. “Robust text detection in natural images with edge-enhanced maximally stable extremal regions”. In Image Processing (ICIP), 2011 18th IEEE International Conference on (pp. 2609-2612). IEEE.

[3] A. Mosleh, N. Bouguila, and A.B. Hamza. "Image Text Detection Using a Bandlet-Based Edge Detector and Stroke Width Transform." BMVC. 2012.

[4] R. Smith. "An Overview of the Tesseract OCR Engine." ICDAR. Vol. 7. No. 1. 2007.

[5] L. Ding, and A. Goshtasby. “On the Canny edge detector”. Pattern Recognition, 34(3), 721-725.

[6] B. Jähne, H. Scharr, and S. Körkel. Principles of filter design. In Handbook of Computer Vision and Applications. Academic Press, 1999.

[7] Wikipedia contributors, "Lab color space," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Lab_color_space (accessed June 4, 2015).

[8] D. Karatzas, S. Robles Mestre, J. Mas, F. Nourbakhsh, P. Pratim Roy , "ICDAR 2011 Robust Reading Competition - Challenge 1: Reading Text in Born-Digital Images (Web and Email)", In Proc. 11th International Conference of Document Analysis and Recognition, 2011, IEEE CPS, pp. 1485-1490

[9] Z. Hou, and W.Y. Yau. “Relative gradients for image lighting correction”. In Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on (pp. 1374-1377). IEEE.

[10] J. Wang., H.L. Eng, A.H. Kam, and W.Y. Yau. Specular reflection removal for human detection under aquatic environment. In Computer Vision and Pattern Recognition Workshop, 2004. CVPRW’04. Conference on (pp. 130-130). IEEE.

Appendix

Note: Code was adapted and heavily modified from https://github.com/aperrau/detecttext. Initially, Andrei was working with this code, written in C++, and Haisam was working on a Matlab implementation. It was decided that once one team member made sufficient progress, to continue the project with his code, which was Andrei's. The work contribution from each member is shown below.

Andrei initially ported the "detecttext" code, which did not work correctly, and got it to work. This includes code in the `Swt.*` and `SwtComponents.*` files. He performed filtering and classification using Tesseract (code in the `TessUtils.*` files), and wrote the Methods section of the paper.

Haisam implemented the computation of stats and filtering operations on the CCs and chains, found in the `SwtComponents.*` and `SwtUtils.*` files, and optimized the algorithm's parameters using the training images. He computed the results, i.e. precision and recall scores, on the testing images, and wrote the non-Methods sections of the paper.