Reagent Label Text Detection Using the Stroke Width Transform

Alexandre Colavin and Jiongyi Tan

Abstract—We implemented an algorithm to extract text from reagent labels to facilitate the retrieval of safety information in a laboratory using an Android mobile device. Our algorithm combined the stroke width transform and a variety of connected component filters to detect text candidates. These were then processed using Tesseract, an open-source optical character recognition engine. We concluded that Tesseract benefits from our implementation when reagent labels contain many non-text objects.

I. INTRODUCTION

As scientists, we often must look up reagent safety and or ordering information. To do so, we currently use desktop or laptop computers, which are frequently out of reach when doing laboratory work [1]. Hence, we were motivated to write a mobile application that could retrieve text from photos of reagent labels. The application would then perform a text search and return any information of interest to the user. For the first implementation of this application, we focused on retrieving material safety data sheets, which are publicly available by law. Our application, which we named findMSDS, would help laboratory personnel access these data sheets using their mobile devices, which are already ubiquitous outside the laboratory.

We anticipated several challenges to our goals. Many computer vision applications must detect text in natural scenes, but this task is difficult imaging conditions vary greatly and text in these scenes is interspersed with clutter. As a result, most optical character recognition engines perform poorly when detecting text in natural scenes. Our main challenge was to design processing algorithms to detect text-rich regions in a given photo before performing optical character recognition.

Text in reagent labels have some common characteristics that defined the processing steps we needed to perform and the features that we could exploit. First, the text in these labels is usually printed using typefaces that have very uniform stroke widths. Second, most text is arranged in lines. And last, most text is printed in a single color, which is frequently black. Thus, our application was designed to filter objects based on these defining properties. However, these properties are often skewed on reagent labels because many containers on which these are printed are cylindrical. A curved surface distorts lines and changes the perceived stroke width of an object. As a result, we also explored algorithms that would attempt to correct these distortions.

We chose to use the optical character recognition engine Tesseract, an open source project that began at Hewlett-Packard in the 1990s but has been in active development under Google [2]. Tesseract was used because it is open-source and reportedly one of the more accurate engines available. Tesseract was used to detect text in the processed images, and the resulting text was sent as a Google search query, the results of which would then be displayed back to the user.

II. RELATED WORK

Our work uses existing approaches that were designed to detect text in natural scenes. We first explored the stroke width transform implemented by Epshtein et al [3]. The intensity of a pixel in the stroke width transform of an image indicates the width of the stroke that contains that pixel. This transform can provide a means of filtering objects based on stroke width. In Epshtein’s implementation, a valid stroke spans two edge pixels that have nearly opposing intensity gradients. During testing, we observed that pixels near the corners of a stroke did not indicate the expected stroke width, and we believed that this was a result of how strokes were defined based on parallel edges (Data not shown).

We adopted a more recent approach developed by Chen et al [4]. In this approach, the stroke width transform is generated from the distance transform of an image containing text candidates. The distance transform generates an image where the intensity of a pixel is equal to its distance to the nearest background pixel. We found this method to be more robust because the orientation of opposing edges do not have to be compared, and nearest distances are always found along lines perpendicular to the edge. In Chen’s approach, the text candidates are detected in a first pass as maximally stable extremal regions (MSER). We adopted this step as well because text regions have areas that should not change appreciably as intensity thresholds change. The method we implement does not follow all of the subsequent steps described by Chen et al, however, because we do not separate words, and we form text lines differently.

III. IMPLEMENTATION

A. Perspective Correction

The vast majority of reagents in a laboratory are in cylindrical bottles. The reagent name is usually printed across the label on the cylinder. When a photo of the cylinder is taken, the two-dimensional projection warps the image. We implemented two algorithms to attempt correction; in the end, however, neither was robust enough to improve results significantly.

In the first method, we modeled the camera as a pinhole. We defined an analytic transformation to convert the three
dimensional coordinates of the points on a cylindrical surface into the two dimensional coordinates of the same points image through the pinhole. The equation for the \((u, v)\) position of a pixel with real space coordinates \((x, y, z)\) is as follows:

\[
\begin{align*}
    u &= \frac{f}{z} x \\
    v &= \frac{f}{z} y
\end{align*}
\]

In the equations above, \(f\) is the camera’s focal length, and \(z\) is the distance between the pinhole and the object. Unfortunately, this approach was fundamentally flawed since the resulting system of equations is underdefined. As an alternate approach, we attempted to correct for the depth distortion separately. To do so, we first used a Hough transform to find the edges of the given cylinder. Second, we interpolated lines between the edges of the cylinder to create a new image, where each interpolated line corresponded to a vertical line in the new image, whose pixel values were interpolated from the values under the interpolated lines in the original image. The lines were spaced as a function of distance between the cylinder edges, such that lines were spaced evenly in the cylindrical coordinate space of the label. (i.e. \(-\sin(\pi/2) : \sin(\pi/2)\)).

In the end, this latter algorithm did not merit inclusion in our final implementation because we could not correct out-of-plane distortion.

B. Maximally Stable Extremal Regions Detection and Cropping

To take advantage of the fact that most text in reagent labels lies in regions with areas robust against varying intensity thresholds, we first detected maximally stable extremal regions (MSER) in each photo after conversion to grayscale colorspace.

Existing natural scene text detection algorithms often include a step to form cluster MSER into text lines based on uniformity of distance between regions and their geometric similarities. Similar strategies were explored, but the best strategy was to perform a dilation of MSER in order to find blocks of text candidates. This step worked because text characters are often uniformly spaced. The MSER were dilated using a structural element with dimensions derived from twice the average size of the ellipses bounding the detected MSER. We then selected for the largest rectangle bounding the dilated MSER, and cropped the MSER mask using this rectangular contour.

C. Pruning

We noticed that many of the detected MSER corresponding to separate text characters were fused. Moreover, since the MSER mask was generated using a union of all the detected regions, candidates corresponding to letters with holes (e.g. "o" and "d") did not have them. Thus, we adopted a pruning strategy proposed by Chen et al. To prune the detected MSER, we first detected Canny edges on the grayscale image. We then followed the gradient at each gray pixel and set each subsequent pixel to background value until we ran into another edge pixel. This approach is similar to the means by which Epshtein et al used to find valid stroke rays in a natural scene.

D. Geometric Filtering

Once the MSER mask was pruned, we then computed the connected components and filtered them by applying geometric constraints. Based on the documentation available for Tesseract, we chose to remove connected components whose bounding rectangles were shorter than 20 pixels. In addition, we noticed that most letter shapes have aspect ratios (width to height ratio) close to 1, so we discarded connected components with very small or very large aspect ratios. These thresholds were relaxed enough to keep "i" and "m" characters, which frequently have the smallest and largest aspect ratio respectively.

E. Stroke Width Filtering

We then computed the stroke width transform of the filtered image. Since we expected most characters to have uniform stroke widths, we removed connected components whose standard deviation to mean stroke width ratio was greater than 0.5, as done in Chen et al. Lastly, we computed the median stroke width for each connected component, and calculated a stroke width median threshold using Otsu’s method to remove connected components with narrow strokes. This final mask was submitted to Tesseract for character recognition.

Our application was developed on the Android smart phone operating system due to its open-source framework and its growing market dominance. Initially, we planned to perform all text detection and recognition on the phone since Android can interface with OpenCV, an open-source computer vision library, and Tesseract. This was ideal since it would not have required the user to have a data plan to decipher text. However, we ran into many complications with the Tesseract engine and OpenCV. Tesseract requires a training dictionary that is several megabytes large, beyond the limit imposed on older phones—including the Motorola Droid used for the project.

We implemented part of the algorithm in OpenCV, but due to our lack of experience in programming in C++, we were unable to write all the necessary steps. Thus, we decided to set up a web server that could perform all the image analysis and text recognition in MATLAB and Tesseract respectively using a PHP script. The output text was then streamed back to the Android device, which could then be used to retrieve the relevant MSDS.

IV. RESULTS

We used the provided Motorola Droid to capture various images of reagent labels under various imaging conditions. The raw images were uploaded to a web server we had set up to perform the image analysis and text recognition. Below, we display the original images, and the final binary masks that are submitted to Tesseract for character recognition.

The results above show that our application can select regions likely to contain text objects, and the first example
Figure 1. Representative results of text region selection process of our application. Images on the left are raw photos, and the ones on the right are the binary masks produced by our algorithm.

(calcium chloride) represents an ideal result. The second example shows a case where our algorithm detected a region that did not necessarily contain the most informative text (in an ideal case, the algorithm would have cropped to the region reading “Sodium Phosphate Dibasic”). We believe this issue arises during dilation of MSER and cropping based on the largest bounding box. If an image contains large blocks of small text such as in the second example, the algorithm will crop to that region and all subsequent processing will be done to this area. A possible strategy to resolve this issue is to weight the size of each bounding box in the dilated MSER mask by the stroke width on the include objects, but this step would have to be performed after computing the stroke width transform.

The figure below shows the output of Tesseract performed on the raw image and the final binary mask.

The results shown in Table 1 suggest that Tesseract can perform well in situations where it is provided with an image containing very few non-text objects. In fact, we realized during incremental testing that simply cropping the original image to the most text-rich region was sufficient to improve Tesseract’s performance to an acceptable level—we deemed this to be acceptable if the name of the reagent came up on the first few search results. Moreover, in some cases Tesseract’s performance was better on the cropped color image than it was on the final binary mask because the processed MSER had many holes that eroded the original shape. This problem resulted from the pruning process because the edges containing the MSER were not always continuous. Since pruning did not stop until an edge pixel was found, in some cases this procedure continued until the border of the image if the opposite edge pixel was missing. This problem is evident in the first example.

Table I

<table>
<thead>
<tr>
<th>Raw</th>
<th>Masked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calcium Chloride</td>
<td>Calcium Chloridil</td>
</tr>
<tr>
<td>4-20 Mesh</td>
<td>4 20 Mesh</td>
</tr>
<tr>
<td>Anhydrous</td>
<td>Anhydrous</td>
</tr>
</tbody>
</table>

Table 1

REPRESENTATIVE RESULTS OF THE CHARACTER RECOGNITION PART OF OUR APPLICATION.

The character recognition results from the second photo are not comparable because they are performed on different regions. However, it is important to note that Tesseract uses a specific dictionary during recognition, and thus many of the mismatches in the Tesseract output on the binary mask may have been due to the variety of languages the text is in.

In the third example, Tesseract’s performance was better on the binary mask than it was on the raw photo. In this case, the image contained a lot of non-text clutter such as label graphics, bar codes, and a combination of white-on-black and black-on-white text. Our algorithm was able to remove much of this clutter, and most of the objects in the binary mask are valid text candidates. Presumably, the bar codes were not removed because they were uniformly spaced. The presence of repeated instances of similar objects was an issue dealt by Chen et al by removing them based on template matching, but we did not address this issue in our work.

V. Future Work

The processing algorithm takes several seconds to run on the web server, which is equipped with a 2.6 GHz Intel Core 2 Duo processor and 2 GB of RAM. The pixel size of each image was 2592 × 1936. We believe that the algorithm would have been significantly faster had the processing algorithm been written in native C++ code using the OpenCV library. Once this is accomplished, the efficacy of our algorithm could be tested against a large database.

Even though Tesseract does detect text found in the given labels, a Google search query does not always yield the desired results because there is a lot of text clutter such as non-alphanumeric characters. A possible way to address this issue would be to compute the Levenshtein or edit distance line-wise. The results could be used to discard lines of text which are unlikely to be meaningful dictionary text before Tesseract’s output is sent out as a search query.

Given that in many cases cropping the raw photo to the region most likely to contain text objects is sufficient to
improve Tesseract’s performance, future work could insert a decision-making step before subsequent MSER pruning, geometric filtering and stroke-width filtering. The application could be trained to distinguish cases where cropped images can be sent directly to Tesseract or where they must be processed through the remainder of the algorithm. This decision could be made based on the sparsity of detected MSER and or object size uniformity.

VI. Conclusion

The performance of our application depends greatly on imaging conditions. The application works well when text is in focus and sparse in the reagent label. Also, Tesseract’s text recognition is somewhat robust against distortion due to curvature as long as the photo is taken close enough to reduce the perceived curvature, and there is not a lot of clutter (small text and non-text) in the field of view. In situations where there is such clutter, our algorithm does improve Tesseract’s detection capabilities.

Acknowledgments

We would like to acknowledge that J.T. wrote the flat image processing algorithm in MATLAB, set up the web server for image analysis and text recognition, and drafted this report. A.C. implemented perspective correction in MATLAB, wrote the Android application, and drafted the project poster. Finally, the authors would like to thank Professor Bernd Girod, David Chen, and Sam Tsai from the Department of Electrical Engineering at Stanford University.

References