Mobile Visual Text Translator

Wei Wang
Department of Electrical Engineering
Stanford University
Email: wwang23@stanford.edu

Qiaojing Yan
Department of Electrical Engineering
Stanford University
Email: qiaojing@stanford.edu

***Abstract*—An Android mobile visual text translator is proposed and developed. The translator is able to extract text from images and perform in situ translation. Our app can automatically recognize texts in simple plain images such as book, sign, and map and overlays the translated text on top of the original one, while preserving information such as location, color, font size, etc. This is often needed when a user tries to know the meaning of a segment of text in another language. The image processing pipeline includes text detection, OCR, text erasure and translation overlay. We tested the app on different images that includes maps, signs, book covers, and white-board. A breakdown of accuracy for text detection and OCR is discussed.**

I. Introduction

The prevalence of mobile devices has given rise to a lot of images processing apps that benefit our lives. In this project, we try to build an app that helps translating words in images from one language to another. Currently, the most common way of translation on mobile device is typing the words into a dictionary or search engine to find the result. Typing is tiresome and some apps let users to take a picture on camera, select a region of interest, and then do OCR plus translation automatically, and finally output the translation results. There are two weakness of these apps that make them not convenient. The first one is that users have to explicitly inform the app where the texts are. When there are many texts scattered on the screen, pointing out every segment is tedious. Another weaknesses is that it lost the positional information of the original words. Imagine translating a map from English to Chinese, if the output is only the concatenated translations–a string of location names in this case–then it doesn't help us in knowing where each place is on the map.

We build a visual translation app that address these two weaknesses. We use image processing methods to automatically detect the text regions in an image. After doing OCR on the detected region and getting translation results, we erase the original text and overlay the translation results on top of the image at places where the origin results are. Fig. 1 shows an example of the functionality of the app.

Fig. 1. Detection Example, left: original image, middle: text region detection result, right: erasure and translation output

II. Related Works

There are some text detection methods that extract text from natural image scenes, most of which use maximal stable extremal regions [1]. However, MSER is relatively slow and needs to build machine learning models to train text regions vs non-text regions. For our own app, since we have a well defined input (maps or simple plain images), we can utilize other image processing techniques to achieve high recognition accuracy and fast processing speed at the same time.

There are many OCR libraries, such as Tesseract [2], ABBYY OCR [3] and Google Mobile Vision Text Recognition API [4]. The Tesseract library is an open source OCR library developed by HP and Google and is well suited for our task. It has an wrapper for android that we can directly use [5].

The Google Mobile Vision is a very powerful text recognition that can do both detection and recognition. However, it is not open sourced. We cannot customize it so we decided not to use it.

The authors are aware that Google Translate has the same functionality we are trying to achieve. It has a very robust text detection and OCR engine. However, Google Translate is not open source. Also, it doesn't detect rotation so it doesn't perform well when characters in an images is not upright.

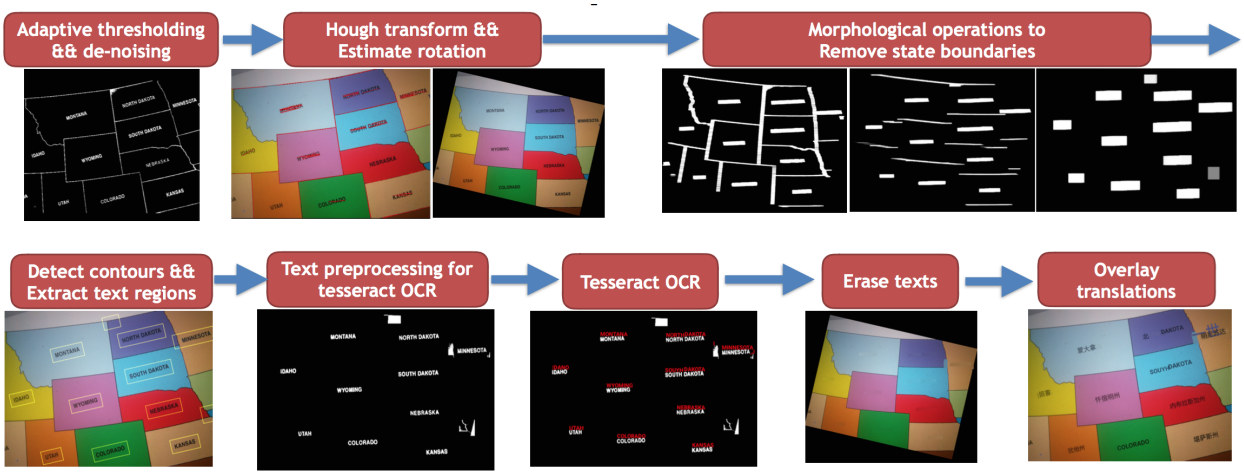


Fig. 2. Image processing pipeline.

III. Methods

Fig. 2 shows the pipeline of our text detection algorithm. It consists of text detection, OCR, text erasure and overlay.

A. Text Detection

The first step in text region detection is to perform an adaptive thresholding on the input image. Then a small region removal is performed to remove any insignificant portion resulting from the thresholding, assuming all text regions have certain size. Then Hough transform is used to estimate the rotation of the images such that all texts are aligned horizontally. Otherwise Tesseract is not able to recognize tilted texts. Since maps usually contain state boundaries, and none-map images may contain line-shaped structures which are either not in the text orientation or are thin enough that the enclosing region is not able to contain any meaningful words, we developed a three step morphological operations to get rid of such structures. First, image is dilated with a horizontal line structure element (SE) such that individual characters in the same text region are connected. Then a horizontal line SE with height smaller then the normal text regions and with width longer than the previous SE but shorter than text region width is used to perform the morphological open operation on the image and we could get rid of all the vertical structures. Similarly a vertical line SE with width and height smaller then the text region size is used to remove all the horizontal structures. SE sizes can be designed such that after the morphological operations, the objects left can be identified as text regions assuming a simple plain image/map as input. And the contours of those objects can be detected as text region contours. One might argue that this processing algorithm poses a limitation on the text size in the image. However, almost every mobile device supports perfect zooming techniques on images, so this is not an issue.

With the detected text region contours, we then extract the regions from the rotated color image. For each region, we perform a series of preprocessings including thresholding, noise removal, smoothing of the text body by a close operation. We do these because Tesseract has a extremely low recognition rate on natural images. Then the output is feed into Tesseract for OCR.

B. Optical Character Recognition

As discussed in section II, Tesseract is more suitable to be used as an OCR engine in our project. The text detection steps give us a list of rectangle bounding boxes, each contains a segment of text. We iterate through the bounding boxes and set the OCR region in Tesseract. Then we obtain the detected words and their positions.

There are two reason why we detect characters in each bounding box instead of on the whole images. First, it is more robust because we are feeding regions that actually contains text into Tesseract. Also, it is fast because we reduce the size of areas that need to be processed.

The Tesseract engine also give us a confidence score which indicates how confident it is about the recognition result. We set a threshold to this confidence score and discard the regions where the confidence score is below the threshold. This helps removing false positives in the result of text detection.

C. Text Erasure and Translation Overlay

In this step we remove the original text in the image and overlay the translated words at places where the original text were.

For removal, Tesseract give us a binarized image as a mask that indicates where the characters are in the image. Each word has such a mask and we use this mask to get the color of the text and the color of the background. The method we use is:

$$TextColor = MeanColor(ImageSegment * erode(mask))$$
$$BackgroundColor = MeanColor(ImageSegment * (\sim dilate(mask)))$$

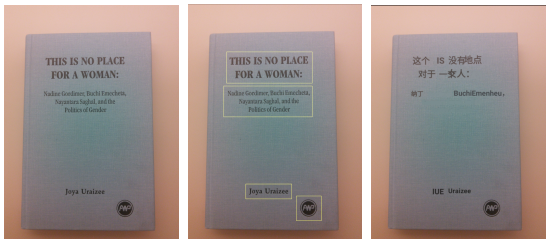
We use erosion and dilation in the above equations to avoid the uncertainties of the boundaries between text and background. We erase the original text by setting the text regions' color to be that of the background.

For translation, we use Google Cloud Translation API [6]. We currently set it to translate from English to Chinese. In the future we can let the users to select the language they use.

After we get the translation result, we overlay the new text at each original words' position using their text color.

IV. Results

We tested the proposed algorithms on a collection of simple images each with about 10 words, such as books, signs, plain maps, and another collection of some complicated natural images such as scenes with a large number of text where text is not the main object in the image. Fig. 3 shows some samples of the former collection results. The algorithm worked pretty well on the this collection, achieving a successful text region detection rate of 89.3% and a successful Tesseract recognition rate of 71.6%.



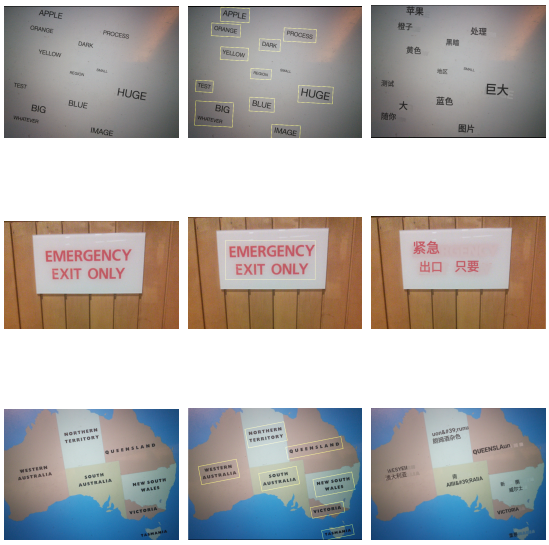


Fig. 3. Sample testing results from collection 1, left: original image, middle: text region detection result, right: erasure and translation output

Image	Detection Acc	Detection Rec	OCR Acc	Overall Acc
Map 1	100%	100%	45%	45%
Map 2	100%	100%	60%	60%
Map 3	75%	100%	92%	92%
Book 1	75%	100%	70%	70%
Sign 1	100%	100%	67%	67%
Sign 2	100%	100%	100%	100%
Board 1	100%	92%	90%	77%
Average	89%	96%	74%	72%

TABLE I
DETECTION ACCURACY: PERCENTAGE OF THE TEXT REGIONS THAT WE FOUND ARE CORRECT;
DETECTION RECALL: PERCENTAGE OF TEXT REGIONS THAT ARE IDENTIFIED;
OCR ACCURACY: PERCENTAGE OF WORDS THAT ARE CORRECTLY RECOGNIZED IF THEY ARE IDENTIFIED;
OVERALL ACCURACY: PERCENTAGE OF WORDS THAT ARE CORRECTLY RECOGNIZED FOR ALL WORDS IN THE IMAGE

Table I lists the detailed results on our simple test images. We can see that in general the text detection has a high recall and can detect all the text regions in the image. It has some false negatives in images with complex background and thus have a lower detection accuracy. The majority of the error is from the OCR engine. There are at least three things that contributes to this error: 1. After thresholding, the boundaries of characters become obsecure. This is more severe for small letters. 2. Even though the text detection has a recall of nearly 100%, it sometimes find a bounding box that is too large and contains contents other than text. 3. The common practice for using Tesseract is to first specify the input font and train the engine using that

font. However, in our app, it is not realistic to specify a single font to be allowed in the input.

On the latter collection of complex images, however, our app is not working so well. The app usually crashes because the OCR workload is too high. For images that the app output results, both the detection and recognition rate are below 50% . The reason for this is because natural images have more completed structures and keypoints which we didn't take into consideration. Therefore, slower and more complicated algorithms such as MSER feature detector with machine learning text filtering might need to be used in those cases.

V. Conclusion

We built an android app that can do visual image translation. It can detect texts in an image, recognize the words, do translation, and overlay the translated results. The app relieves users burdens of searching word one at a time. We believe this app would be useful for foreign visitors and foreign language learners.

We achieved a good accuracy and recall on detecting text in relatively simple images. The OCR has not very high accuracy which should be further improved in the future.

The main challenge we encountered in this project is developing a good algorithm for detecting text regions and calculate bounding boxes. Another challenge is OCR. Given the current status, Tesseract may be the most suitable engine that we can use. However, its accuracy is not good enough for this app to be fully functional.

For future work, we can try to improve OCR accuracy by incorporating mechanism similar to spelling correction. Currently, single character error in a word would lead to failure of the whole word. Instead, we can try to correct this error by comparing the result to a dictionary. Also, currently our work uses Java, Tesseract OCR and Google online translator. The processing speed is not promising. In the future we can integrate all the modules in C and with local dictionary to make this app faster and achieve real time detection.

References

[1] https://www.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html.

[2] R. Smith, *An overview of the Tesseract OCR engine*, Google Research, 2007.

[3] https://www.abbyy.com/en-us/.

[4] https://developers.google.com/vision/text-overview.

[5] https://github.com/rmtheis/tess-two.

[6] https://cloud.google.com/translate/.

APPENDIX

Qiaojing wrote the project proposal, worked on the Tesseract OCR, implemented the text recognition, erasure, translation and overlay. He also wrote the structure of the Android app code. Wei worked on the poster and implemented the text detection pipeline, which includes thresholding, Hough transform, morphological operation and contour detection. They both worked on testing and the final report.