Recognition of Thai Characters and Text from Document Templates

Nattapoom Asavareongchai | Evan Giarta
nasavare@stanford.edu | evgiarta@stanford.edu
Department of Electrical Engineering, Stanford University
EE 368 Digital Image Processing, Autumn 2016

***Abstract*—We created an image processing system that takes in an input image of a text document, with Thai language as the primary text language, taken from a phone camera and outputs a matching between each Thai character detected in the document to a particular Thai template character. The output could then be used to input into a translation software to translate the Thai sentences out automatically without the need of a human translator. We implemented three different methods and compared their accuracy and consistency.**

I. Introduction

With the availability of books, reports, and journals from different regions of the world through either the internet or hard copies, many people would want to have access to these information to be able to read. However, not all these documents are translated into languages like English for people to understand. Many important documents in other parts of the world are written in the languages in those parts and will usually require professional people fluent in those languages to read and translate them manually. This could require a very long time to do so and will require man-power. Would it not be more efficient to take a picture of a page from books, documents, etc. and then automatically detects the language characters then print them out for translation through a computer program? This report aims to give an image processing system that is able to detect and classify one specific language text from an image of a document page. The language that this report will consider is Thai.

In the field of image processing and computer vision, many people have tackle the problem of text recognition and classification. Many of these literature however tackle the problem of detecting other languages other than Thai. For example, Chinese characters [1]. Some literature that works with Thai language requires sophisticated algorithm that requires machine learning, such as neural networks to tackle the classification problem [2]. However, in this report, we propose and analyze more simply methods that will only require image processing techniques to classify Thai characters. This will include using the SIFT and RANSAC algorithm, the SVD and Principle Component Analysis technique, and XOR template matching method. Not only will the report concentrate on the classification part of the problem, we will also propose a pre-processing pipeline that will clean up documents with Thai languages by utilizing the structure of the language to detect where texts are within an image.

II. Dataset

A. Font Templates

Obtaining a clean reference font and template images was a critical component of this project. This was achieved by having all of the 44 consonants and 22 vowels of the Thai alphabet typed out on a Google Doc and scaled up to size 72 font. Zoomed-in screenshots of each character were then taken, binarized using Otsu's method, and cropped the bounding box of the results. It is also relevant to note that the default Thai alphabet font on Mac OS X is noticeably different to the default Thai alphabet font in Windows 7.

B. Document Images

We used two particular types of documents with the different Windows and Mac fonts to quantify the quality of our methods and results.

Clean documents images were created by taking a direct screenshot of Thai text from Google Docs and saving the image as a Portable Network Graphic (`.png`), as opposed to a JPEG, so as not to lose any detail along character edges from compression.

Test documents images were produced by snapping a picture of document printout of Thai text at an angle, in uneven lighting conditions, and against dark background. Test documents images were also produces with both fonts. The text contained in the clean and test documents were copied from real-world Thai newspapers.

III. Methods

The image processing pipeline that defines our system can be categorized into two big sections. These sections include the pre-processing part and the text recognition part. We combine many image processing techniques within each sections to produce a desired output.

The input image into our system is an image of a text document page that contains Thai characters as the only text in the document. The image can either be a clean document (a correctly aligned document with no blur, noise, or variable lighting) or an image taken from a phone camera, which may not be clean. This input is then passed into our pre-processor to output a clean binarized image of the text. Sentences and characters are then separated from this output and input into our text recognition pipeline that matches these characters to a database of character templates.

A. Pre-process

In our pre-processing pipeline, we have assumed that the input image is not a clean document image. The four steps are as follows: (Example output of each sub methods can be found in the results section)

1) Locally Adaptive Thresholding: The first step in pre-processing the image is to binarize the image. We assumed that the image taken from a phone is an RGB image that contains noise, regions that may not correspond to text, may be rotated, and may have different lighting.

The image is initially converted into a gray-scale image to remove the RGB channels and reduce the size to a 2D matrix. The image is then filtered using a rank filter to accentuate regions of the text. This process is similar to dilation on a gray-scale image. We utilized the ordfilt2 function in MATLAB to do so. We then run the both the filtered gray-scaled image and the unfiltered gray-scale image through locally adaptive thresholding to binarize both images. The two images are then AND together to remove small noises that may be removed from the filtered image but not through the unfiltered one. Note that we have to convert the text regions into the foreground before doing the AND process.

2) Noise Removal with Region Labeling: The second step in the pipeline is to further filter the image to remove noise and unwanted artifacts.

To remove these artifacts, we tried to identify foreground regions that correspond to text and regions that do not correspond to text. We assumed that regions that are text are similar in size and area, not too large or too small. To implement this, we utilized region labeling. For each region we calculated the number of pixels covering the area of the region. We then found the mean and standard deviation of all the regions found. Regions with area larger than or smaller than one standard deviation above and below the mean are then removed.

Furthermore, we also assumed that the ratio of the area filled in by the region within its bounding box (the smallest box that is able to include all pixels of a region) is consistent for the Thai characters as well. Thai characters contains many holes in their structure and cover approximately around 50% of their bounding box. Therefore, we also decided to remove any regions with a ratio of area filled in its bounding box two standard deviation higher or lower than the average ratio of all the regions. A similar technique using region labeling is utilized.

3) Hough Transform: The input images may not necessary be correctly oriented and may be rotated at a certain degree. Therefore in our third step, to tackle this issue, the Hough transform is used to find the angle of rotation of the text sentences. The top 20 Hough peaks were recorded and then the lines plotted. The angles of the top 20 Hough peaks are however not consistent and angles at +-45 degrees and +- 90 degrees keep showing up consistently with any actual rotation angle of the text. From plotting the lines that correspond to these Hough peaks, we found out that many outlier angles are due to very short lines detected, which are anomalies. Thus to remove these angles from consideration, we removed any lines detected by the Hough transform with a length shorter than half the longest line. This was able to correctly removed all the anomalies. The modal Hough angle after the anomalies are removed is then used as the rotation angle to rotate the image to the correct orientation.

4) Sentence and Character Segmentation: The fourth and last step in the pre-processing pipeline is to detect the regions that are text and separate out the sentences and characters for detection and characterization.

Since we are assuming that the text image is an image with a document format, we therefore assumed that the text will be typed/written within a text box with margins around it. To find this text box, we summed up the pixels of the columns to get a 1D vector of the x axis of the image. This is in order to figure out the left and right boundaries of the text box. We then plotted the cumulative sum of the pixels from left to right. Since the text should reside in the text box, the cumulative sum graph should show a flat region before the text box with low number of pixels, then the graph should rise linearly within the region of the text box (since pixels corresponding to text are added up), then it will plateau and flatten out when it leaves the text box and enters the right margin. We calculated the 10% and 90% point of the cumulative sum graph and interpolate it to estimate the x-axis location of the 0% to 100% pixel counts that will correspond to the text box region. This is to remove any pixel noise in the margin regions outside the text box.

This is then repeated for the rows or the y-axis of the image to find the top and bottom boundaries of the text box, using the same technique. The overall resulting text box is then found.

After finding the text box, we moved on the separate the sentences and text into their individual characters. To do this we assumed that each sentences are separated by empty regions and so we sectioned out the regions of the rows with non-zero pixels and regions with zero pixels. This is done by first summing up the pixels for each row. If the sum of the rows are less than a certain threshold, we say that the rows do not correspond to part of a sentence and is an empty region between the sentences. We therefore zero out these rows.

Then we used region labeling in 1D to label the location of the sentences.

One difficulty that arises with region labeling of each sentences is that the Thai language contains vowels that exists on top and below a consonant character. These vowels will show up as their own sentence with the region labeling technique we used. Therefore to solve this problem, we detected these vowel regions by find the regions with a sentence height (thickness in the y-axis direction), smaller than the half the average sentence height. We classify these regions as top/bottom vowel regions and do not consider them as a sentence. We then merged these regions with the nearest sentence region either above or below it to create a complete Thai sentence region.

Then for each sentences, we next separate out the characters one by one. This is done by region labeling the x-axis regions of the sentence (similar to the sentences case) where each region will correspond to a section of the characters. Further region labeling is done for each x-axis region to section out vowels that may exist above or below a consonant but will be part of the same x-axis region as the consonant.

One problem we encountered was that blurring and noise can cause vowels to be connected to the consonants, either above or below it. This will cause a mis-classify if we leave it as it is. Therefore to solve this we eroded the character to separate them.

B. Text Recognition

1) SIFT and RANSAC: SIFT feature descriptors and frames of each font template image are calculated using the specified peak, `peak_thresh`, and edge thresholds `edge_thresh`. After a padding, `padding` and an increased scaling,`scale`, is applied, the SIFT features of each extracted character from the document image are calculated with the same peak and edge thresholds as the font template images. The padding and scaling helped increase the number of features detected in the extracted character.

A match and score computation is done between this character-to-be-identified and each template image. The frames of the matches are then ran through RANSAC to calculate a best-fit homography matrix and identify inlier features between the character and template. RANSAC sets, $\mathtt{k}$, the number of matching pixels needed to compute the homography and samples for the best homography $\mathtt{S}$ times. The average score of these inlying features is taken, and the template which achieves the lowest average inlier score is selected as the character's identity and match.

The user has the ability to change the values of `peak_thresh`, `edge_thresh`, `scale`. The default values for these parameters are $\mathtt{peak\_thresh} = 0, \mathtt{edge\_thresh} = 50, \mathtt{padding} = 10, \mathtt{scale} = 3, \mathtt{k} = 3, \mathtt{S} = 2000$.

2) SVD and PCA: Before SVD and PCA analysis can be applied to the font template images and extracted document characters, the images must be resized to the same matrix dimensions. SVD is then applied to the resized templates and extracted characters images and the normalized left or right eigenvectors and associated eigenvalues are obtained. The principle eigenvector, that is the eigenvector associated with the largest magnitude eigenvalue, of each template image is recorded and compared with the principle eigenvector of each extracted character by calculating the dot product of these normalized eigenvectors. The larger the dot product, the more similar these vectors, and therefore by extensions these characters, are to one another. Thus, the template image that whose principle eigenvector is produces the largest dot product with the extracted character-to-be-identified is selected as the character's identity and match.

By default, the method employed uses the right eigenvectors from SVD, since no significant difference between the results of the left and right eigenvectors were found.

3) XOR + Resize + WDS: Each extracted document character is resized to match the dimensions of the candidate font template image. Matching the sizes allows these binary images to be directly XOR'ed together. The resulting image is a visualization of where the images matched, and more importantly, where they mismatched. Matches occur if the values in both the extracted character and the template image are the same at corresponding pixels; in this case, the XOR result of the pixels is 0. Mismatches occur if the values in both the extracted character and the template image are the different at corresponding pixels; in this case, the XOR result of the pixels is 1.

The resulting image is given a weighted distance score (WDS). This is computed by locating the mismatches in the XOR'ed image, and determining how far in pixels the mismatch was from a match. This distance value is then raised to the power of a desired weighting penalty $\mathtt{p}$ and summed to a running total that is then normalized by dividing it by the number of pixels in the template. The resulting value is the distance score. The idea behind this method was to place large penalties on mismatched pixels that were far away from a match. Interestingly, as $\mathtt{p} \to \infty$, the distance score asymptotically becomes the maximum distance of any mismatch. And when $\mathtt{p} = 0$, the distance score is the normalized count of the number of mismatches. It was expected that the optimal $\mathtt{p}$ would lie between 0 and 5, but ultimately, the best results came about by simply adding the mismatch count, that is when $\mathtt{p} = 0$

IV. Results

A. Pre-processing Results

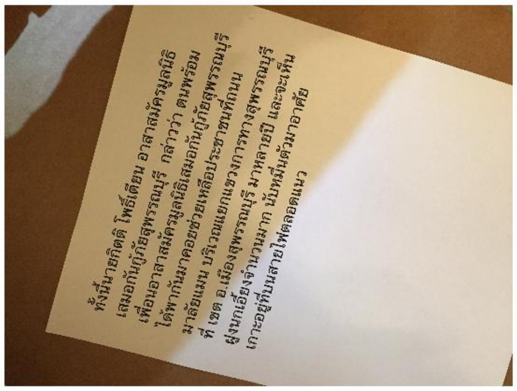


Fig. 1. Input Camera photo image

1) Locally Adaptive Thresholding:

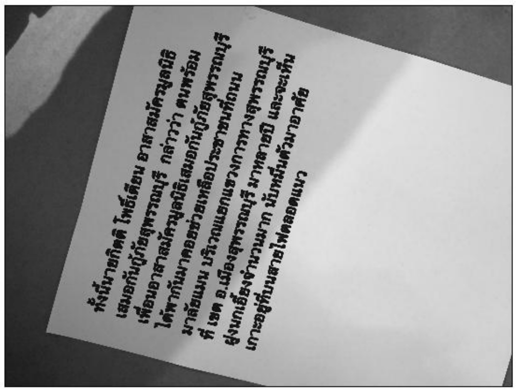


Fig. 2. Grayscaled rank filtered image to accentuate text

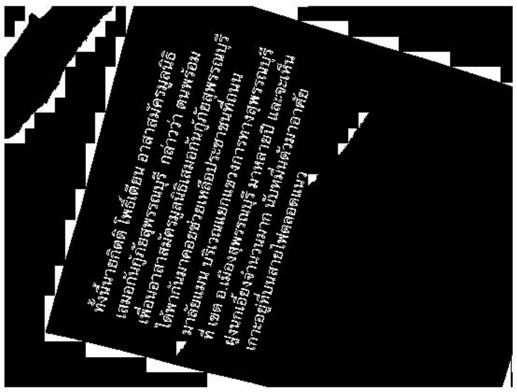


Fig. 3. Output of Locally Adaptive thresholding image after AND filtered and unfiltered grayscale image

2) Noise Removal with Region Labeling:

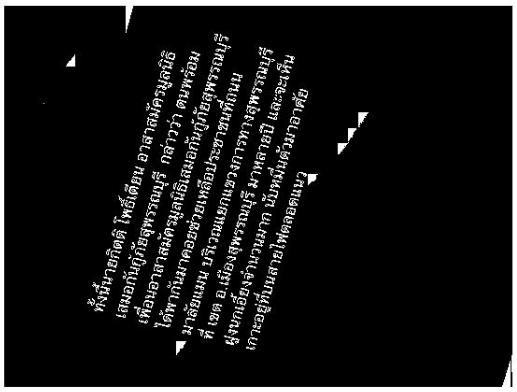


Fig. 4. Result after removing noise using Region Labeling

3) Hough Transform:

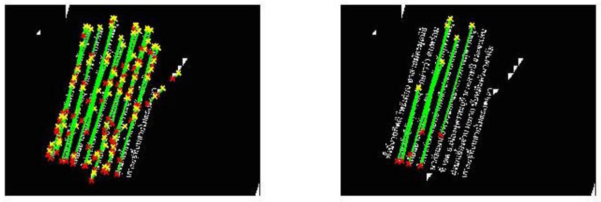


Fig. 5. Hough line peaks before anomaly removal (left), after anomaly removal (right)

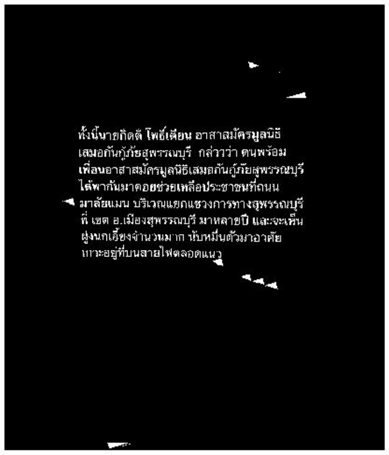


Fig. 6. Image after rotation

4) Sentence and Character Segmentation:

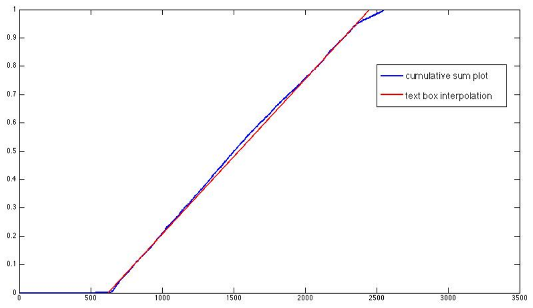
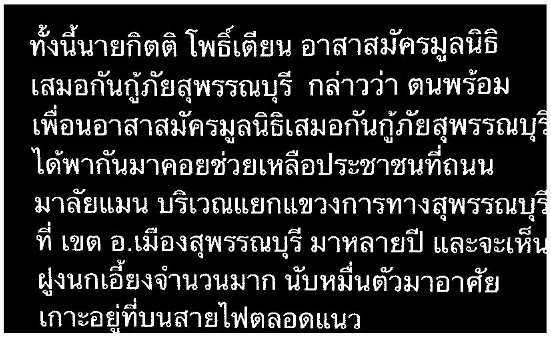
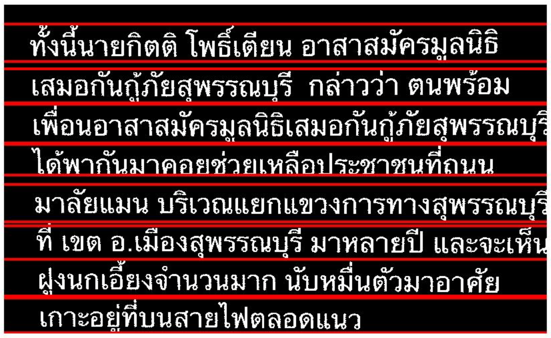


Fig. 7. Plot of cumulative sum of column pixels normalized vs. x-axis image columns



ทั้งนี้นายกิตติ โพธิ์เตียน อาสาสมัครมูลนิธิ
เสมอกันกู้ภัยสุพรรณบุรี กล่าวว่า ตนพร้อม
เพื่อนอาสาสมัครมูลนิธิเสมอกันกู้ภัยสุพรรณบุ
ได้พากันมาคอยช่วยเหลือประชาชนที่ถนน
มาลัยแมน บริเวณแยกแขวงการทางสุพรรณบุ
ที่ เขต อ.เมืองสุพรรณบุรี มาหลายปี และจะเห็น
ฝูงนกเอี้ยงจำนวนมาก นับหมื่นตัวมาอาศัย
เกาะอยู่ที่บนสายไฟตลอดแนว

Fig. 8. Image after text box recognition



ทั้งนี้นายกิตติ โพธิ์เตียน อาสาสมัครมูลนิธิ
เสมอกันกู้ภัยสุพรรณบุรี กล่าวว่า ตนพร้อม
เพื่อนอาสาสมัครมูลนิธิเสมอกันกู้ภัยสุพรรณบุ
ได้พากันมาคอยช่วยเหลือประชาชนที่ถนน
มาลัยแมน บริเวณแยกแขวงการทางสุพรรณบุ
ที่ เขต อ.เมืองสุพรรณบุรี มาหลายปี และจะเห็น
ฝูงนกเอี้ยงจำนวนมาก นับหมื่นตัวมาอาศัย
เกาะอยู่ที่บนสายไฟตลอดแนว

Fig. 9. Sentence Segmentation

B. Character Identification Results

1) SIFT + RANSAC:

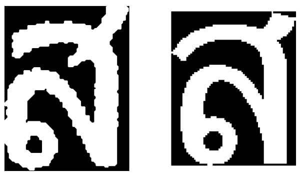


Fig. 10. SIFT + RANSAC match

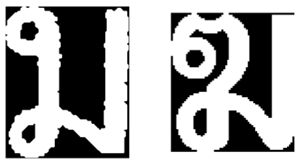


Fig. 11. SIFT + RANSAC mismatch

2) SVD + PCA:

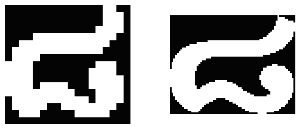


Fig. 12. SVD + PCA match

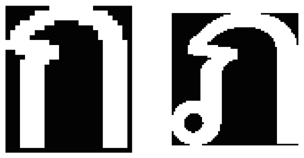


Fig. 13. SVD + PCA mismatch

3) *XOR + Resize + WDS:*

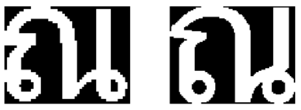


Fig. 14. XOR + Resize + WDS match

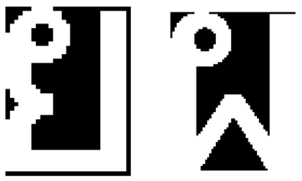


Fig. 15. XOR + Resize + WDS mismatch

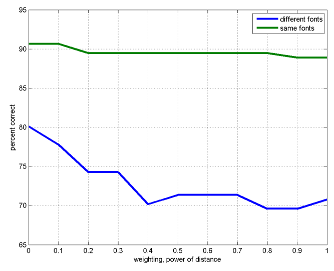


Fig. 16. XOR penalty weight percentage correct classification graph

TABLE I
PERCENTAGE CORRECT CLASSIFICATION

Document Type	XOR + Resize	SVD + PCA	SIFT + RANSAC
Clean (same font)	91%	73%	64%
Clean (different font)	80%	36%	36%
Test (same font)	95%	69%	59%
Test (different font)	83%	36%	18%

C. Result Discussion

Included here as samples of some of the matches and mismatches produced by the three different methods we implemented. It can seen that the methods can handle a noticeable margin of differences and still produce the correct. At the same time, due to the similarity of various Thai characters, some mismatches are pretty close to matches. All in all, the most method with the most favorable result is our custom XOR + Resize + WDS comparison algorithm.

V. CONCLUSION

The methods that we have implemented in this report are far from perfect and do have limitations to their successes in many cases. There are many future improvements that can be added on top of the existing pipeline to make the system more robust.

A. Limitations

There are many limitations to our system and many cases where our system will very likely produce bad results. These limitations are as follows:

1) One limitation of our implementation during the pre-processing pipeline is if there are too much noise outside the text box, regions outside the document page. This will cause the algorithm to find the wrong boundaries for the text box as well as section out the wrong regions for the sentences and characters. Some characters may be unintentionally removed by the region labeling step (step 2) in the pre-process pipeline. A better and more robust system against noise in the background outside the page documents (e.g. images of table behind the document) would be a beneficial addition.

2) Another limitation in the pre-processing step is if the image of the document is warped. This means that the page is not correctly aligned in 2D but instead is rotated or folded in the third dimension (in and out of the page). This will produce characters in the image with different sizes and well as have sentences that are not straight, but curved. With this image setup, the image segmentation step that segments out sentences and characters will not be able to separate the right sentences. Parts of some sentences may be merged with other parts of other sentences.

3) For our SIFT + RANSAC method, we notice that the bad result was due to the lack of SIFT features and descriptors for each characters. Each character template contains on average around 30-40 features and do not reach the 100 feature point. Characters detected from the camera images have even fewer features (around 20) and so the matching features when run through RANSAC reduces even more. With too few features, we could not use the maximum inliers as a representation of the best match, but needed to use the average score of the matches instead.

4) SVD + PCA method requires large first principle eigenvalue component compared to the other eigenvalues. This is the case with the template characters, with the first principle eigenvalue being about 3 times larger than the next largest. However, the noisy characters detected from the image did not have this trait and had the principle eigenvalue around 1.5-1.7 times larger than the next one. This removes many of the features that can be captured by the principle eigenvectors for the detected characters and produced poor results.

5) An interesting result we found with our XOR template

matching method is that it will consistently mis-classify the same character over and over. It will on the other hand also correctly classify the same character. Thus, the errors resulting from this method is consistent and are pretty robust with noise. This is different from the other two methods where the same characters in two different locations may be matched with two different characters.

B. Future Improvements

1) Pre-process: Pre-process could be improved to be more robust against background noise. Also a method to fix the perspective warp of the characters could be implemented within the pipeline to straighten out the sentences and have the characters be resized to the same size, for better segmentation.

2) Character detection and Classification: One big improvement to add to our classification implementation is to create a confidence score for each classification. Since currently we are only noting if the classification is either correct or incorrect, if a confidence score is incorporated, we will be able to identify classifications that we are not confident about and re-process these characters with better algorithms.

With a confidence score, we can then combine the three methods we mentioned for text/character recognition. We would first run XOR template matching since it will consistently correctly classify 80-90% of the characters. Characters that are incorrectly matched will be detected using the confidence score and then we run using the two other methods and confidence score of the three methods combined can be used to find the best match (i.e. the matching character with the highest confidence out of the three methods).

One last improvement we can make to the SVD + PCA method is to create a database of the 66 Thai characters with many fonts. Using these templates, we can create an eigen-image or fisher-image of each character and then classify the characters according to these eigen/fisher-images.

References

[1] Jin, Michelle, Ling Xiao Wang, and Boyang Zhang. Poster: Text to Image Translation for Restaurant Menus. EE 368/CS 232, Department of Electrical Engineering, Spring 2014.

[2] Phokharatkul, Pisit, and Chom Kimpan. ”Recognition of handprinted Thai characters using the cavity features of character based on neural network.”Circuits and Systems, 1998. IEEE APCCAS 1998. The 1998 IEEE Asia-Pacific Conference on. IEEE, 1998.

[3] Hochberg, Judith, et al. ”Automatic script identification from images using cluster-based templates.” Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on. Vol. 1. IEEE, 1995.

Appendix

Project Idea, Data collection, Poster, Report, SVD + PCA, RANSAC: Nattapoom Asavareongchai, Evan Giarta

Preprocessing: Nattapoom Asavareongchai

SIFT, XOR, WDS: Evan Giarta