Event Info Extraction from Flyers

Yang Zhang*
Department of Geophyiscs
Stanford University
Stanford,CA
zyang03@stanford.edu

Hao Zhang*, HaoranLi*
Department of Electrical Engineering
Stanford University
Stanford, CA
{hzhang22, aimeeli}@stanford.edu

Abstract—**In this study, we developed an Android app to add poster dates and venue information to Google calendar. Several digital image processing methods are used to preprocess the captured poster's image, which might have been taken from a difficult perspective, bad lightening conditions or partial occlusions. And we also coped with the complex background of the poster to make it easier to locate the text area in the poster. After locating the text bounding boxes, the text would be extracted by an optical character recognition (OCR) engine, Tesseract, and parsed to isolate locations and venues. The image processing part is done in a server by several heuristic methods and the resulting concatenated text would be pushed back to the phone. Natural language processing (NLP) technique then would be utilized to analyze the text string and extract the date and venue to e automatically add to Google calendar. Test results show that our algorithm performs well to locate the text area and extract the key information.**

Keywords—Android, Image Processing, OCR, event info extraction

I. INTRODUCTION

Every day we encounter a lot of events information from the paper flyers posted on the hallway walls, entrance doors and elevators in our buildings. Will it be a presentation, a forum discussion or a concert; they all come with a date, time and venue. People might want to keep a record of interested events in their calendar when they come across those posters. But since the information are not digitized, it is very inconvenient for us to manually input such information into our smartphone calendars. Therefore, we aim to build an OCR-based mobile image processing system that can automatically extract event information from flyer pictures and directly integrate then imto the users' diigital calendars. Our goal is to design a small tool on Android platform that enables the user to put the event info (time and location) into his personal calendar by simply snapping a picture on the related flyer/poster Figure1.

Previous EE368 students have worked on a project of importing contact from business cards automatically [1]. Here we are targetting posters, which have more variations and involve more complex image components. Therefore, it would be more challenging for accurate text detection of various posters. We also want to improve the performance of our system by allowing users to take photos from different angles and illuimnation conditions.

Figure 1: Put events in calendar by snapping a picture

II. SYSTEM OVERVIEW

Since Tesseract OCR would require high complexity cost, running this engine on a mobile client with responsive interactions is infeasible. So we decide to offload the work of preprocessing and OCR part to a high-performance server over the network. Thus this Android app uses a client-server communication system framework[2]. A flowchart of the application pipeline is showed in Figure 2.

There are major stages in the pipeline:

1) Capture and upload Image: Using Android device to capture the image. After downsizing the image, upload it to the server.

2) Preprocessing: The stage several image processing methods are used to prepare the image for text detection including: resizing of image, edge detection, histogram equalization, homograhy rectifying, and etc. Section III discusses it in details.

3) Text Detection: Two heuristic methods based on text variance and edge density are firstly used to locate the text area. And then filters are used to eliminate the useless regions by area, height, orientation, solidity and etc. It would be discussed in Section III.

4) OCR: Detected bounding box for text is sent to the Tesseract OCR engine seperately and the result would be concatnated into one string text and be pushed back to the Android device via internet. Secion IV covers it.

5) NLP Information Extraction: The downloaded text file would be parsed into dates and venue by some natural language processing methods. It is introduced in Section V.

6) Importing Google Calendar: The dates and venue information would be add to google calendar as discussed in Section V.

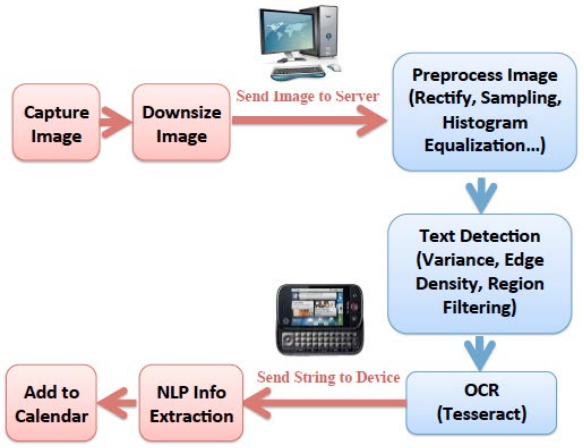


Figure 2: System pipeline. The blocks in red are implemented in Android devide and the blocks in blue are in the server

III. IMAGE PREPROCESSING

Preprocessing is an essential step to obtain accurate text recognition results with the Tesseract OCR tool, because an poster image captured by a hand held camera is far from ideal input for an OCR engine. 1) The image would have perspective distortions, while OCR engine assumes the image containing text is taken from a perpendicular upright view. 2) The illumination of the image is not uniform everywhere. 3) There are usually multiple text blocks in a flyer, and the blocks are not homogeneous (different font size, color, background). Our preprocessing workflow is designed to tackle these issues. And they are described in the following subsections.

A. Correcting Perspective distortion

We assume that the surface of the poster/flyer forms a flat plane in 3-D space, and is presented on a rectangular shape of paper. Therefore, as long as we can find the boundary quadrilateral of the poster/flyer in the image, we can form a homographic transform that transform the irregular quadrilateral to a horizontal rectangular domain. To find the quadrilateral, we run the following steps:

1. Use Canny edge detector to generate an edge map of the original image. The parameters for Canny edge detectors can be refined to better suit our purpose. Because the edge we want to detect should be relatively strong and spatially long, therefore the gradient threshold should be set to rule out the weaker edges, and the pre-smoothing filter needs to be applied in order to get rid of edges that come from local image textures.
2. Computing Hough transform on the edge map we obtained, in order to obtain boundary line candidates.
3. Locate the four sides of the bounding quadrilateral by detecting two near-horizontal edge candidates and two near-vertical edge candidates. This is done by using *houghpeaks* and *houghlines* functions. Because there can be more candidates than we need, we select and rank the candidates based on several criterions:
    a. We only accept the edge candidates to have ±15 deg deviation from nominal axis directions.
    b. We only accept edges that are longer than $1/4^{th}$ of the image's dimension along the corresponding axis. And the longer ones get higher priority.
    c. We assign higher priorities to the edges that are further away from the image center.

    Also some safeguarding is added to handle the case where some of the edges cannot be found (because of occlusion or the actual edges are outside of the captured image domain). **Figure 3** shows the detection of candidate edges that could form bounding quadrilateral.
4. Once we determine the encompassing quadrilateral, we form a homographic transform to map the image content inside the distorted quadrilateral to a standard rectangular area, thus achieving geometric correction. Figure 4 shows the detected final encompassing quadrilateral, and Figure 5 shows the result after undoing the perspective distortion.

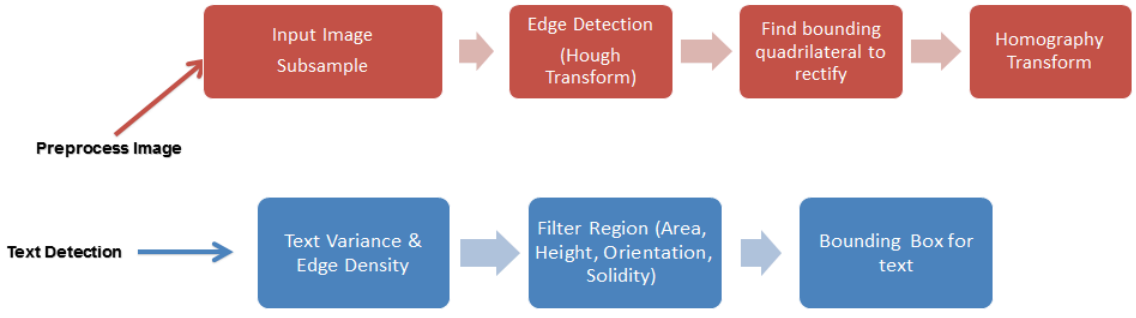
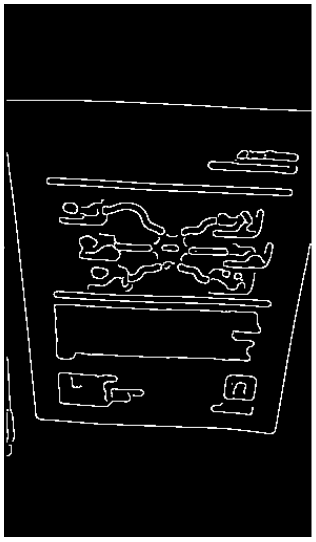
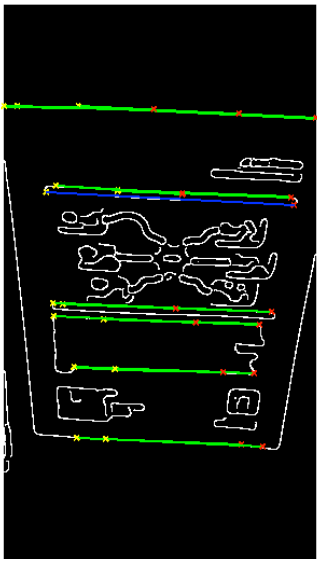


Fig. Image Processing Pipeline

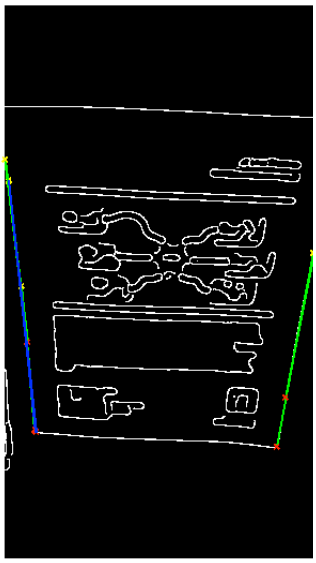
(a)



(b)



(c)



(d)

Figure 3: Using the Canny's edge map enables the detection of candidate edges that could from bounding quadrilaterals. (a): Input subsampled camera image; (b) Canny edge detection result; (c) near-horizontal edges from Hough transform; (d): near-vertical edges from Hough transform.

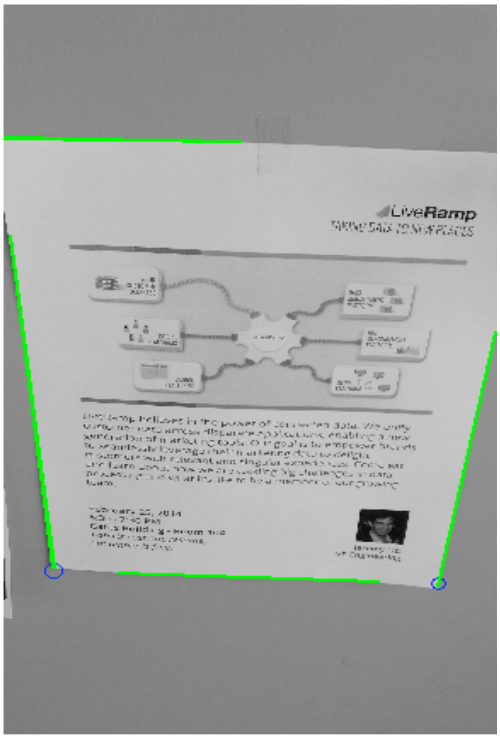


Figure 4: Final selection of the bounding quadrilaterals, drawn in green lines. Blue dots show the vertices.

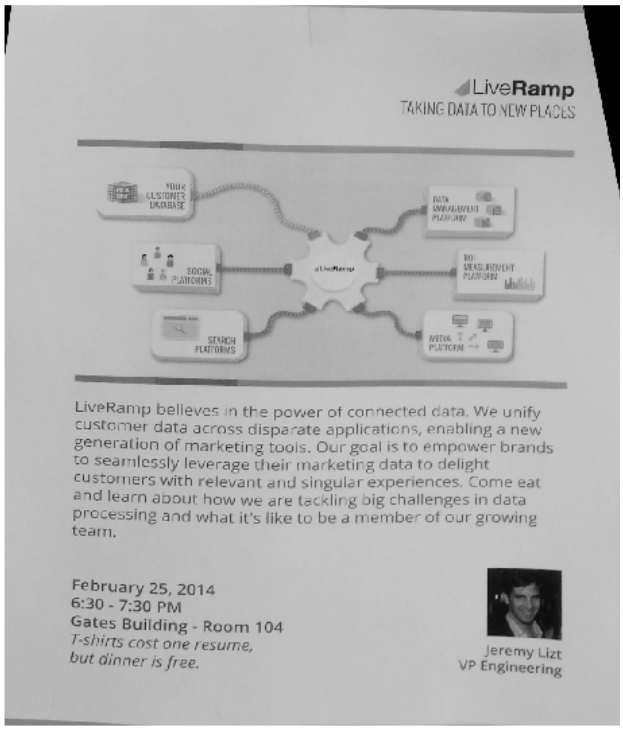


Figure 5: The geometrically corrected (rectified) image after performing the inferred homographic mapping.

B. Text Detection and Individual Textbox Extraction

We use the geometrically corrected image to detect text regions. We have two major steps in this module: first is to detect where the text locates; second is to group the areas containing text into individual text blocks. By making it homogeneous inside each text block, we will achieve much better recognition result from Tesseract. To find text regions, we use two heuristic criterions:

1. Local image variance. For every image location, we compute the image's brightness variance within a neighborhood window of 31 pixels. Locations with brightness variance lower than a certain threshold (10% of the maximum variance found in the image) would be identified as non-text (i.e. plain background).
2. Local edge density. We first compute an edge map using Canny's method. Then for every image location, we compute the proportion of edge pixels among all pixels in a neighborhood window of 31 pixels, we refer this quantity as "edge density". Locations having their edge density within a certain range [0.05 0.5] will be considered text. This measure will differentiate the text region from regions containing image content that has rich local details and thus a higher edge density.

Figure 6 shows the two heuristics computed for the example image.

After the candidate text regions are identified as a binary indicator map, we apply image dilation to close small holes, and group the text regions based on special connectivity. The grouped candidate regions are further filtered by several heuristics:

1. Regions that with an orientation that deviates from horizontal directions for more than 10 degrees are rejected.
2. Regions with too big or too small areas are rejected.
3. Regions with aspect ratio (width/height) less than 2.0 are rejected. (Assume texts are horizontal)
4. Regions with solidity less than 60% are rejected. (These regions are too far from being rectangular)

Figure 7 shows the identified text regions and the final result after property filtering (Yellow labels).

Finally, we take bounding boxes of the accepted text regions, and window them out as individual text blocks. The resulting text blocks are shown in Figure 8.

We could apply more sophisticated schemes to improve the result of text detection, nonetheless under the scope of our project, we found that the results are sufficient for our purpose: to correctly identify the key event information from these camera images.

C. Binarization

We binarize each text blocks separately using Otsu's method, which essentially achieves locally adaptive binarization. We notice that Tesseract is able to detect both black text on white background and vice versa, therefore our program does not need to make such distinction.

Figure 6: The computed local brightness variance (left) and edge density map (right). The two quantities are filtered with different threshold values.

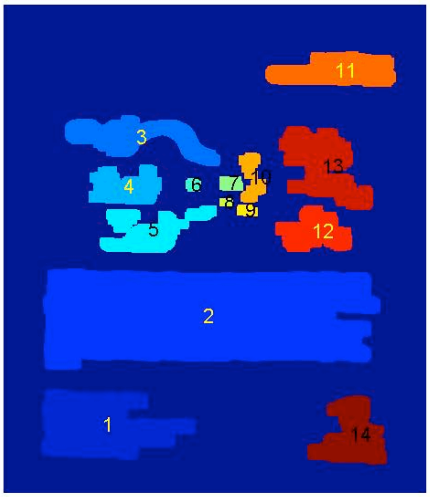


Figure 7: The identified potential text regions from analyzing the two heuristics shown on Figure 6. The regions with YELLOW No. labels are the final ones that passed filtering on region properties.

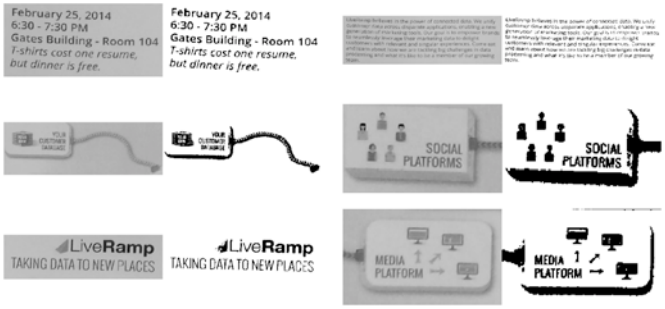


Figure 8: The six individual text block images extracted using the region information shown in Figure 7, and their corresponding binarized images.

IV. OPTICAL CHARACTER RECOGNITION

A. *Tesseract*

Tesseract is an open-source OCR engine developed at HP labs and now maitained by Google. It is one of the most accurate open source OCR engine with the ability to read a wide variety of image formats and convert them to text in over 60 languages[3]. The Tesseract library would be used on our server.

Tesseract algorithm would assume its input is a binary image and would do its own preprocessing first followed by a recognition stage. An adaptive classifier is used to identify the words.

B. *Server side OCR*

On our server, the input for the Tesseract engine are several identified text locations on the image indicated by the bounding box detected by our algorithm instead of the entire image. The resulting text of each region are then concatenated together as the final output of OCR. Though there may be some false positive regions, we consider the output of OCR is correct as long as the dates and venue information are included in the final output text.

V. POSTPROCESSING

A. *Text Information Extraction*

Once we get the text from posters by Tesseract OCR, we want to extract information of time and location from the text. For time extraction, I use a time parser library in Java [4]. The parser will extract all times from the given string, then I will filter those times based on the matching text and text position and choose a start time and end time. Standard times that we got would be like "Tue Feb 04 18:30:00 PST 2014".

For location extraction, at first, we do string matching of common places at Stanford campus such as "Gates", "Packard", "Huang" etc. Then we will match keywords that indicate locations such as "building", "room" etc. If there are numbers following those keywords, we will add those numbers to the keyword. Otherwise, we will extract words preceding those keywords and insert them to the front of the keyword. Standard locations that we got would be like "Gates Room 104", "Huang Mackenzie Room" etc.

B. *Adding Event Information to Calendar*

After the event information is extracted from the string, we would add them to the Google calendar. In the user part, once he/she finished taking a photo, a calendar window would appear on the phone with captured location and time information filled in the calendar part. Figure 10 (e) shows the app appearance.

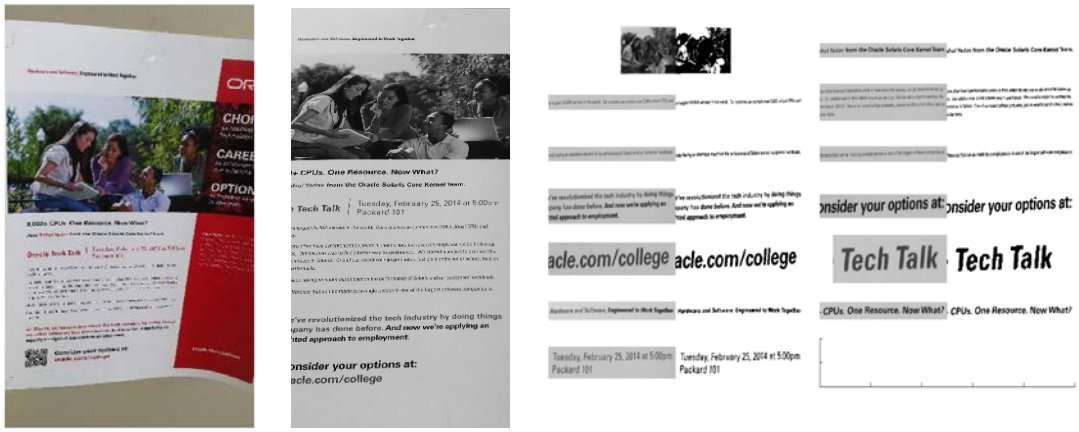
VI. RESULTS

In this section, we present the output of each step for several sample input images. Figure 10 shows a case where the image is taken with perspective distortion and papers are uneven.

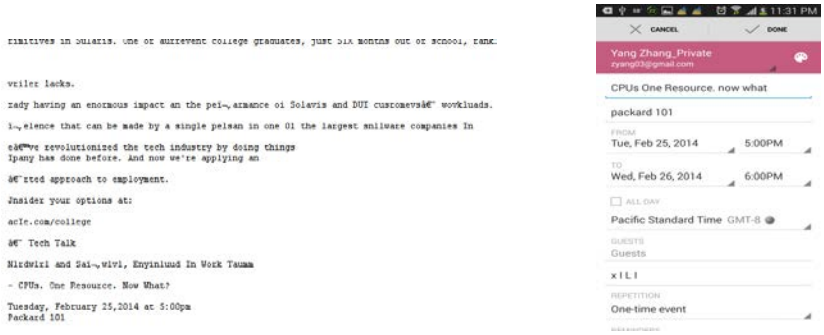
For those test cases, we can see that for the first step our algorithm is able to find the bounding quadrilateral and rectify the image. For the next step, we can correctly find regions for the text. OCR would return a quite good result if we are able to feed in proper regions of text. Finally our application is able to get the correct time and location from the text and add to the calendar.

Our algorithm will sometimes fail when the poster background is too complex and we are not able to identify regions of text from the poster, such as the input image in Figure 9. We are able to correctly rectify the image. But since the background image is too cluttered, we are not able to find regions of the text based on variance, edge density or geometries of the region. Therefore, we got incorrect results in this case.

Figure 9: Sample input image with incorrect result



(a) Input Image (b) Rectified Image (c) Text Crop



(d) OCR Result (e) App Screenshot

Figure 10: Result images of each step for a sample input image. (a) is the input image, (b) is the rectified image, (c) is text crops that we identify, (d) is the result text of OCR (e) is a screenshot of our app.

VII. FUTURE WORK

In our system, we only extract time and location from posters. It would be great if we can also extract the main topic of a poster. We can detect topic by considering font of the text in the image, regarding text with bigger font as the topic. In addition, we can improve the robustness of our system by applying techniques such as Stroke Width Transform to improve text detection and performing better text segmentation from cluttered background rather than simply Otsu's Method.

VIII. CONCLUSION

In this project, we have implement an Android app which is capable of capturing a poster image, uploading the image to a server, downloading the processed text from the server, extracting event information and finally adding them to Goolge canlendar. Lots of image processing methods are used to preprocess the image including canny filter edge detection, histogram equalization, rectifying homography, and etc. Two heuristic methods and variuos fitlers are used to detect the text area. Tesseract is used to recognize the text. And natural language processing methods are used to extract the event information.

Though our algorithm is a heuristic one, it is observed that it is robust to difficult conditions like partial occlusions, complex backgrounds, differnet camera perspective, uneven illuminations and etc. And compared to the raw OCR result, our result has an obvious improvement on the accuracy.

REFERENCES

[1] https://stacks.stanford.edu/file/druid:np318ty6250/Sharma_Fujii_Automatic_Contact_Importer.pdf

[2] http://www.stanford.edu/class/ee368/Android/Tutorial-3-Server-Client-Communication-for-Android.pdf

[3] https://code.google.com/p/tesseract-ocr/

[4] http://natty.joestelmach.com

APPENDIX

The work presented in this report and the accompanying source codes are the results of joint work of the following students.

- Yang Zhang proposed the initial project idea, and was in charge of the image preprocessing and text block detection-extraction components implementation. He also produced the demo video with the help of the other team members.
- Hao Zhang developed a prototype of the Android app to run Tesseract OCR engine and the Google calendar part combination with the app. He also built the server part of the project.
- Haoran Li implemented the post-processing part and made the poster.
- Joint efforts: Most of the brainstorming and debugging parts are done among all members. And all members create the final report together.