Free-Flow: Unintrusive Reading Device for a Printed Text

Hae Jin Song (hjsong@mit.edu)
Suvrit Sra (suvrit@mit.edu)

***Abstract*—This project develops a software for a handheld, pen-style device that allows a reader to click on words in printed texts and look up the definitions. Unlike most of the current stylus pens for digital devices, our device targets printed texts, which makes it a more attractive and useful reading aid for a larger group of readers. The software implements four major actions, "Capture", "Preprocess", "Extract (OCR engine)" and "Search". It is a completely offline software which uses the camera hardware installed in most modern mobile devices. Users use our Android application to first capture the word to search on the printed text. The captured image is scaled and binarized in the Preprocess module. Then, the OCR engine, which is trained on the provided training data, recognizes the word. Lastly, the word is searched on a dictionary stored in the burst trie data structure. The technology used in each of Free-Flow's main modules existed over a decade, but Free-Flow is the first end-to-end system that combines them together to create a full pipeline. It reduces the latency and energy consumption by removing any unnecessary networking connections, improves the accuracy of the Optical Character Recognition (OCR) via preprocessing, and supports a fast closest match search. It achieves a high recognition accuracy of 95.25% on average, is robust against different ambient lightings and is responsive with a small latency of 1.11 second. Its achievements open up the new possibilities to take advantage of mobile devices to make reading on the printed texts more interactive and intuitive.**

I. Introduction

INCREASING computational power and memory of mobile devices have enhanced the reading experiences on smartphones, tablets, and Ebooks. Users of such digital devices have access to software which can quickly look up definitions by selecting the words directly on the screen. The results are often displayed on top of the digital text. As a result, the users don't need to leave their current screen during the interaction, and their reading flow is not disrupted by the search. Unfortunately, there is no such tool for traditional texts such as paperbacks, printed newspapers, and magazines. If they want to look up a word while reading, they have to stop and look up the dictionaries themselves on online or paperback dictionaries. This process forces them to jump back and forth between their text and dictionaries (or computer screens), thus interrupting the reading flow. This intrusive and cumbersome transition cause an inevitable tradeoff between the quality of reading experience and the level of comprehension.

We feel a strong need to bridge this technical gap for the printed texts and aim to build a new tool that can minimize such intrusive transitions and help maintain a continuous reading flow. We chose the readers of the printed texts in their non-native languages as the main target of our project. We used hardware already available in most of the modern mobile devices and developed the software, Free-Flow, that performs image-processing, the Optical Character Recognition (OCR) and a fast closest match search on a dictionary. It runs on the Android platform, and it is now available for download from Google App Store. On average, Free-Flow achieves a short response time of 1.117 seconds up on the user's request and 95.25% recognition accuracy of the selected word in various ambient settings. The application is simple yet robust and does not require any additional hardware or network connectivity. Its accuracy is consistent in three different lighting settings (cloudy outdoor, sunlit indoor, normal indoor) and robust against camera noise and user's handshakes.

Free-Flow consists of four major modules, **"Capture"**, **"Pre-process"**, **"Extract"** and **"Search"**. The first module, *Capture*, takes a picture of the scene around the word that a user wants to search. The second module, *Pre-process* rescales and binarizes the image using a median filter. Binarization refers to the conversion of pixel values from color to black and white so that the resulting image is a collection of 0 (for black pixels) and 1 (for white pixels). Then, the binary image is run through the OCR engine based on Tesseract in order to extract a string of characters during the *Extract* module. The final module, *Search*, performs a closest match search on the recognized word and displays back the definition to the user's screen on the same mobile device.

Fig. 1. A user holds the mobile device to capture the word to search

II. Related Work

A. OCR and Tesseract

Optical Character Recognition OCR refers to the conversion of images of text $printed, typedorhandwritten$ to a machine-encoded string of characters. It has a wide application in systems like robot navigation, document analysis, and object categorization and has been an active field of research in Computer Vision. However, it still remains a difficult problem especially in a noisy environment such as damaged documents or low resolution of the scanning device. The challenge in our project is that the mobile device a user will use to capture the image is susceptible to the user's hand shakes and different lighting settings. In addition, mobile devices have limited computational resources and memory space and require a quick response time to be usable in practice.

The core of our OCR engine is based on the state-of-the-art OCR system *Tesseract*. Its four major features are the adaptive thresholding, the line finding, feature/classification methods, and the adaptive classifier [2]. Figure 2 shows the workflow of Tesseract. When it receives an input image, it first binarizes the image via adaptive thresholding. Then, it runs a connected component analysis during which the outlines of the components are gathered together into *Blobs*. The third step is to organize these blobs into text lines, which are then broken into words. Once the Tesseract has a collection of words, it initiates the recognition phase through a two-pass process [2]. In the first pass, Tesseract attempts to recognize the word as in the conventional recognition stage.

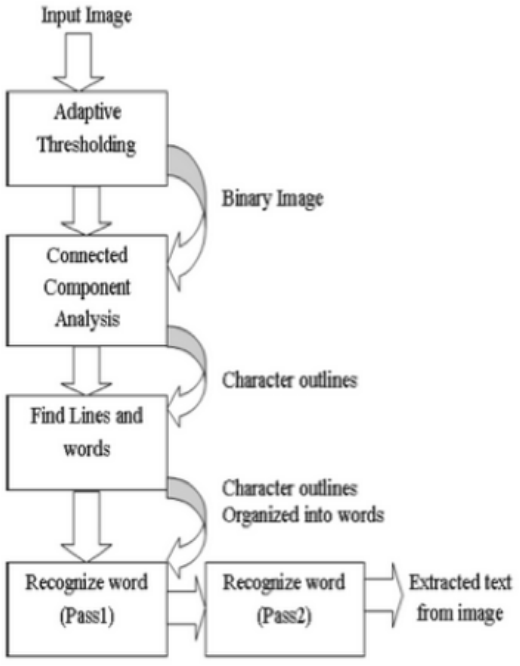

Fig. 2. Workflow of Tesseract engine [1]

Only the words that are reasonably well recognized in the first stage are passed into an adaptive filter as training data. The result of this first stage is a set of training data and the adaptive classifier trained on the training data. In the second stage, the adaptive classifier performs a second pass of the entire page and attempts to recognize words that are not recognized correctly in the first pass. Previous tests on vehicle number plates confirm Tesseract achieves higher percent accuracy and faster processing speed than another popular OCR engine, Transym [3]. This performance is also supported by the evaluation by a group of researchers in Pozna Supercomputing and Networking Center, Poland. Their evaluation shows that Tesseract's performance achieved higher accuracy of 75% to 88% [1]. Based on this evaluation, we chose to build our OCR engine using Tesseract's Android library, *tess-two*.

Previous work has been done in both academia and industry to improve the Tesseract's accuracy. We looked at the incorporate document-specific modeling [4,5,6], which adjusts the system to the input document's fonts, lexicons and noise models. Another approach worth mentioning is to train the recognizer on a pre-selected training data that only include high-precision characters which were correctly recognized from different OCR systems [7].

B. Image processing

The captured image must be pre-processed before the OCR engine to achieve high accuracy. Gross and Brrajovic introduce an image preprocessing algorithm for illumination invariant face recognition in **??**. The algorithm is simple but successfully mimics the human visual system by computing the estimate of the illumination field and then compensates for it. Based on their experiments on multiple face databases including the Yale database, the algorithm improves the accuracies of standard face recognition algorithms and maintains improved accuracies under different lightings. Another approach is to apply Kalman filter to compute a face class average that better represents unique facial features **??**. However, this illuminance-invariant image processing is not suitable for our application as it requires a large computational power and takes longer than the desired response time of no longer than 2 seconds. As discussed in Section VI, we found that mean and median filters are simple to implement while still performing well enough for a high recognition accuracy. These filters better satisfy our constraints on computational power and response time.

C. Closest match search

An intuitive way to implement search is to construct a hashmap with the key of a string of characters and the value of the corresponding definition. Hashing provides a fast search in amortized constant time, yet it does not preserve any order. Consequently, it is not suitable for a closest match search. An alternative data structure that provides a fast search as well as an access to the closest neighbors at each node is Trie. Consider the case of searching for a word whose maximum length can be M in the text containing N words. The time complexity of a trie is $\mathcal{O}(M)$, which is linear in the length of the word to be searched. Since the search word is usually

[1] https://blog.cedric.ws/how-to-train-tesseract-301

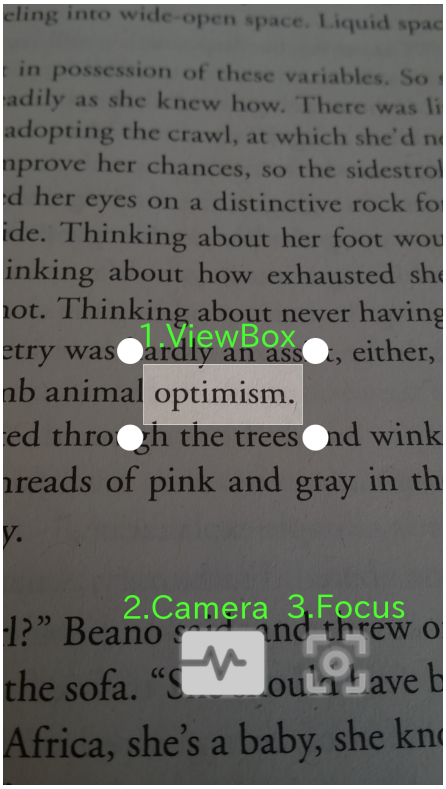

Fig. 3. Main Screen: 1.ViewBox defines the word boundary; 2.Camera button triggers the image capture; 3. Focus button refocuses the camera on the ViewBox.

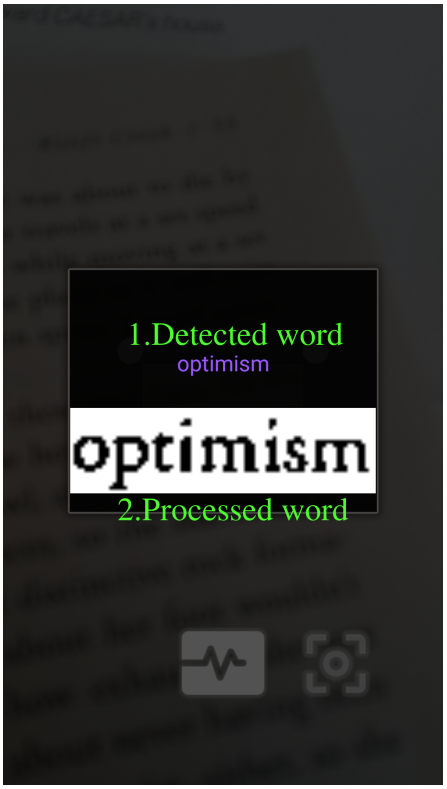

Fig. 4. Preprocessed image of the capture word and the OCR detected characters are displayed.

Fig. 5. The detected word is searched in the dictionary stored in the Trie data structure to facilitate the closest match search

very short in our case, this is nearly a constant time search. The advantage of a trie over hashing is that it supports the prefix search, which allows users to find all words with the same prefix. An example of the prefix search is Google's search box which provides a list of suggestions as a user types. However, a trie has a large space complexity as it requires $\mathcal{O}(Alphabet * M * N)$ space. For example, if an English dictionary is to be stored in a trie, each node will correspond to a character of a word with 26 (i.e. the size of the alphabet) pointers for its children. Due to the memory constraint of mobile devices, a trie's large space complexity is not suitable for our purpose. A better option is a Ternary Search Tree (TST). TST is a special trie whose child nodes are ordered as a binary search tree. It supports trie's operations such as prefix search and closest neighbor search. The biggest advantage of TST over a trie is that it uses less space. For instance, if we use a TST to store a dictionary, each node contains only 3 pointers. The first pointer points to the node whose value is less than the value in the current node, the second pointer to the node whose value is equal to the current value and the third pointer to the node whose value is greater than the current value.

D. Similar end-to-end system

Currently, there is no end-to-end system that performs a quick search of a word in the printed text using a hand-held device. The closest system is the state-of-art Google Translate with the technology first developed for *Word Lens*. Word Lens is a mobile application that translates the road signs in foreign languages and overlays the translated information on the smartphone screen. Other pen-like devices such as *IRISPen* and *Wizcom InfoScane Pen Scanner* focus on digitizing a bulk of texts (lines, paragraphs or pages). We were not able to find an end-to-end product which focuses on detecting a single word from the middle of a text. *Free-Flow* is distinguishable from currently available tools in that it is a first full end-to-end system that brings together the image processing, OCR and Search algorithms to support a robust recognition of a word in the printed text.

III. Overview

Free-Flow is a mobile application that consists of four major modules: **Capture**, **Pre-process**, **Extract** and **Search**. It uses a built-in camera to capture the word and does not require any other additional hardware or network connectivity. It performs a highly accurate recognition and a fast closest match search. In order to use it, a user selects a word to search by holding his mobile device on top of the reading material (Figure 1). The application uses the built-ins camera and allows the user to capture the word to search and control the camera's focus if needed. He can easily interact with the main UI (Figure 3) via screen touch to set the word boundary and refocus the camera. The main UI consists of a ViewBox(1, Figure 3), a Camera button(2, Figure 3) and a Focus button(3, Figure 3).

The ViewBox is draggable and is used to set the boundary of the word to capture. The Focus button refocuses the camera, and the Camera button takes a picture of the scene in the ViewBox. The application is initiated by the user's click on the Camera button. It then preprocesses the captured image (**Pre-process**) and extracts a string of characters via the OCR engine based on Tesseract (**Extract**). The recognized word is then searched on the dictionary via the closest match search (**Search**), and the final result (i.e. the definition) is displayed back to the user on the screen.

IV. System Architecture

Free-Flow's system has four main modules: **Capture**, **Pre-process**, **Extract**, and **Search**. Upon the user's click of the Camera button, the scene within the ViewBox 1, Figure 3) is captured and processed in preparation for the OCR engine. The preprocessing involves scaling and binarization via a median filter. More details will be discussed in Section V. The processed binary image is passed on to the OCR engine which is trained on the provided training data. Our application allows users to provide new user-specific training data and can be easily extended to recognize different languages. The results of image processing and the OCR engine are displayed back on the screen as shown in Figure 4). The dictionary is stored in memory in the burst trie data structure to facilitate the closest match search with the minimum space required. Finally, the result is displayed on the screen (Figure 5).

V. Technical Approach

Free-Flow consists of four major modules, *"Capture", "Pre-process", "Extract"* and *"Search"* as shown in Figure 6). In this section, we explain each module in details.

The first module, ***Capture***, is initiated by a user's click on the Camera button in the main UI (1, Figure 3). It takes an image of what the camera views ("scene") with the focus adjusted as the user clicks on the Focus button (2, Figure 3). Restricting the region of interest to the ViewBox reduces the amount of computation and reduces the response time. It also achieves more locally optimized pre-processing and improves the quality of the binarization as discussed in the following.

The second module, ***Pre-process***, scales and binarizes the captured image. Binarization is the conversion of pixel values from color to grayscale. First, the image is scaled by taking into account the camera resolution, screen resolution and ViewBox size. We assume the mobile device is placed at the minimal focal distance away from the text and has a fixed zoom level. These assumptions are reasonable and serve well in practice since users tend to position their mobile devices as close to the text as possible (that is, at the minimal focal distance), and the zoom level is fixed as our design choice. Since the OCR Engine is trained on the font size of 12pt (0.167 in), this scaling is important to enhance the performances of the subsequent OCR operations.

After the image is scaled, we apply a median filter to find the threshold for binarization. The median filter chooses its threshold to be the median of the captured image's pixel values. It then binarizes the image by assigning the pixels with values higher than the threshold to 1 and those with lower values to 0. We tested both mean and median filters and found out that the median filter achieves a more robust binarization. The qualitative comparison of the two filters are discussed in Figure 8 in Section VI. A median filter is simple to implement and computationally efficient as our image size is small (1120 pixels on average). Its benefits are discussed in Section VI.

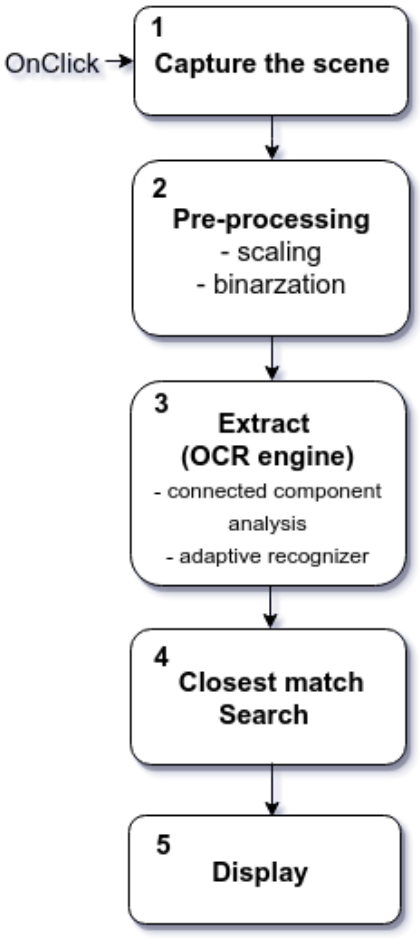

Fig. 6. Users' click on the Camera button initiates the camera to capture the scene. The image is then preprocessed, and the OCR engine extracts a string of characters, which is searched on the dictionary to find the closest match. The matched word's definition is displayed on the user's sceen

The third module, ***Extract***, is the OCR engine that performs the character recognition using Tesseract for the Android platform, *Tess-Two* [2]. This module is key to our application as it decides which word to look up in the dictionary. The recognizer must be trained in advance with specific language data. The Tesseract project provides large amounts of train data in various languages [3], which are available to the public. We used the English train data to train our recognizer, yet the scope of recognition can be easily extended to other languages by training the recognizer on new train data. If a user-specific recognizer is needed (for instance to recognize a handwriting or rare fonts), users can collect the data and construct the training data using Tesseract's libraries. The training data are

[2]https://github.com/rmtheis/tess-two
[3]https://github.com/tesseract-ocr/langdata

used to construct “Bounding Boxes” around each character and extract the structure during the component analysis process (Figure 2). We already provide 2,3-gram language data during the training. For example, the 2-gram language data is a list of 2-grams (such as “QU”, “TH”, “BE”) with their frequencies in the sorted order. It informs the OCR engine which character is more likely to appear based on the surrounding characters. After the OCR engine finishes the recognition task, it passes the recognized string of characters to the Search module.

The fourth module, ***Search***, performs a closest match search of the recognized word in the dictionary. We first scrapped the word-definition pairs from the Oxford English Dictionary and stored them in the burst trie structure. As explained in Section II, a burst trie is a variant of a trie data structure that is highly efficient for managing strings in memory. Its time complexity of search is linear in the length of the searched word, which is usually very short, and it uses no more space than a standard tree or a hash table [11]. More importantly, unlike a hash table, it preserves the sort order and can be implemented to support prefix search and closest match search. The fast search and improved space complexity well satisfy the constraints of our application. Using the insertion and bursting algorithms as described in [11], we constructed the burst trie with the contents of the Oxford English Dictionary. When the Extract module finishes, the recognized word is searched in the dictionary in two distinct stages as described in the Search algorithm in [11]. First, the leading characters of the recognized word are used to access the correct container of the burst trie. Then, the rest of the characters are used to search the record, or the closest neighbor if no exact match is found, within the container. The result (i.e. definition) is then displayed back on the user’s screen as shown in Figure 5.

VI. Evaluations

The design goal of our project is to build a simple yet accurate and robust software for a fast search on the printed texts. *Free-Flow* achieves these goals in four stages, Capture, Pre-process, Extract, and Search. In this section, we evaluate its performance measured by the recognition accuracy and runtime for each module. For evaluation, we generated two random test texts with 1000 words from our dictionary, one in Ubuntu font and the other in Times New Roman font. The test texts are available in the Github repository for the project, [4]. We tested Free-Flow on each test text under three different lighting settings: cloudy outdoor, sunlit indoor and normal indoor. Detailed analyses of the results are as follows.

A. Pre-processing

The Pre-process module of Free-Flow performs scaling and binarization. Our preliminary test without scaling achieved less than 50% of recognition accuracy. This low accuracy is caused by the difference in the font size between the captured word and the training data. The public training data available at Tesseract project’s GitHub [5] is constructed from a single font size (12pt). We tested mean and median filters during the pre-processing of the captured images on each experiment. The median filter achieved more robust binarization while their runtime difference was less than 0.001 second and was considered negligible. Figure 8 shows the qualitative comparisons of the results. As shown in the figure, the median filter preserves more foregrounds (i.e. black letters) and is less susceptible to the thinning effects of the edges. This effect is due to the lower threshold detected by the mean filter as printed texts often consist of more white backgrounds than black foregrounds. Consequently, the median filter preserves the original shape of the characters better and helps the OCR engine to recognize the word with higher accuracy. We also tested Otsu thresholding [10] for binarization. The processed image and the detection accuracy were similar to the median filter, but the runtime was nearly 7 times slower.

[4]https://github.com/cocoaaa/free-flow.git
[5]https://github.com/tesseract-ocr/tessdata

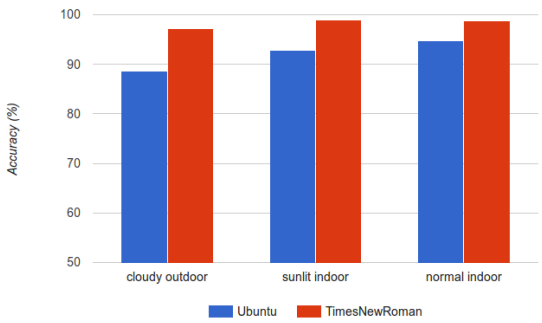

Fig. 7. Free-Flow’s accuracy is tested on two fonts (Ubuntu and Times New Roman), each under three different lightings: cloudy outdoor, sunlit window side of a library, and normal lecture hall. The average accuracy is at minimum 88% and on average 95.25%. It is higher on the test text in Times New Roman

B. Recognition accuracy and robustness

The recognition accuracy was measured by comparing the distance between the detected word and the ground truth (i.e. the word printed on the test text). We used the Levenshtein distance(LD) as the distance metric. *Levenshtein distance* is defined as the minimum number of single-character transformations (insertions, deletions or substitutions) required to change one word to another **??**. For example, LD(“hop”,“happy”) is 3, LD(“hole”, “hoe”) is 1 and LD(“sky”,“ski”) is 1. We tested Free-Flow on 35 words from the two test texts (Ubuntu and Times New Roman) under three different lighting settings. Table **??** and Figure 7 show the recognition accuracies.

Font	cloudy outdoor	sunlit indoor	normal indoor	Average
Ubuntu	88.7	92.89	94.82	92.14
TNR	97.2	98.99	98.8	98.36

TABLE I
Free-Flow’s recognition accuracy on two distinct fonts under three different lightings

The results from the three different lighting settings 7 show that Free-Flow’s recognition is robust under different ambient lightings. Since the median filter’s performance is barely affected by the ambient lightings, it allows the OCR engine to extract correct words in different environments. The cloudy

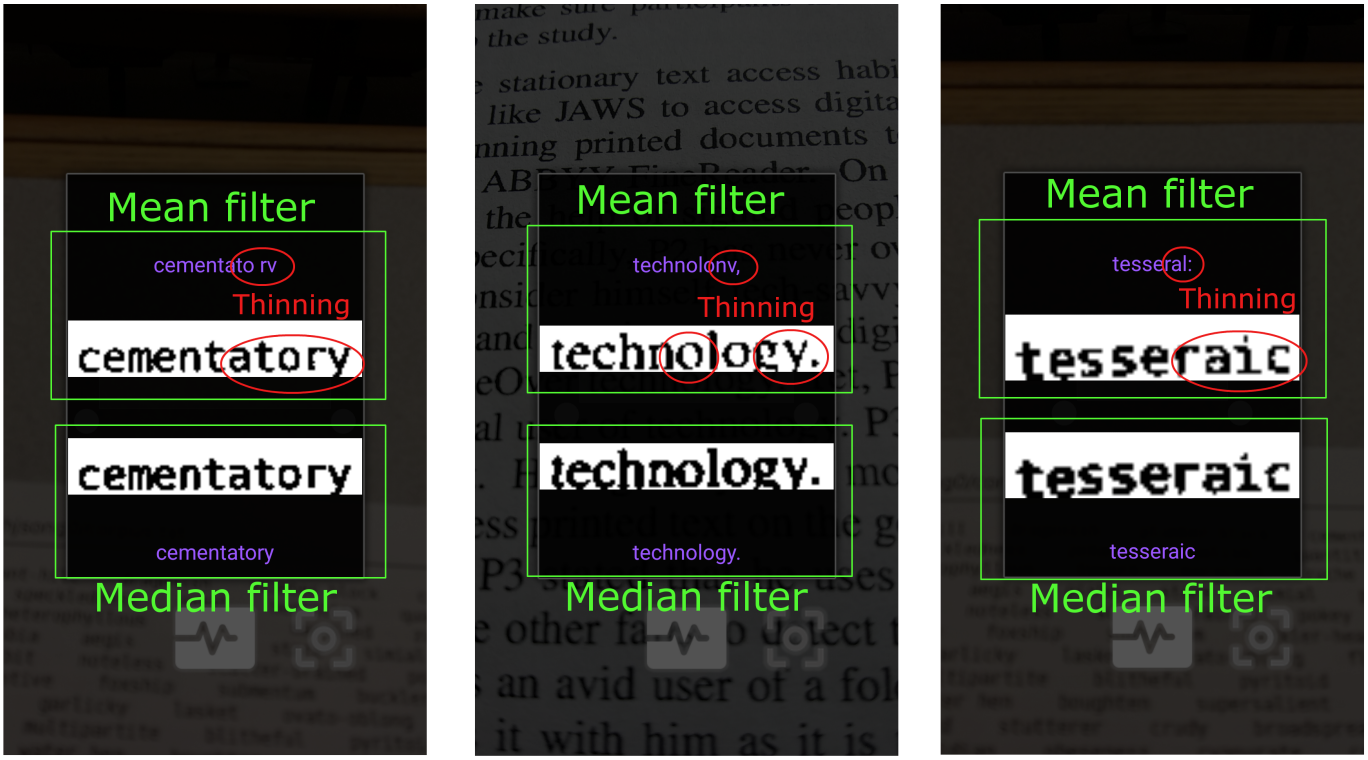

Fig. 8. Images preprocessed by mean and median filters. Mean filter thins out some of the characters and leads to recognition errors.

outdoor setting results in the lowest accuracy as it causes the threshold of the median filter to decrease and affects more character pixels to be binarized as white. This makes the edges of the characters thinner and as shown in Figure 8 and confuses the OCR engine. Figure 7 and Table **??** show the results of the median filter. Overall, it achieves a high recognition accuracy of 95.25% on average and performs consistently in different lighting settings.

C. Runtime analysis

The runtime of each module and the size of the captured images are recorded during our experiments. Each measurement is first calculated per pixel and adjusted to the fixed image size of 1000 pixels. We chose 1000 pixels as it is a close approximation of the average image size, 1120 pixels. Table II shows each module's average runtime.

Module	Pre-process	Extract(OCR)	Search	Total
Runtime(s)	0.001866	1.102	0.0129	1.117

TABLE II
RUNTIME ANALYSIS OF EACH MODULE PER 1000 PIXELS

As Table II shows, the Extract module is the bottleneck of the runtime. The pre-processing and the closest match search run in three and two magnitudes faster, respectively and do not affect the response time in a noticeable way. This result is as expected since the Extract module performs a series of more complicated computations such as connected component analysis and two passes of word recognition 2.

VII. FUTURE RESEARCH QUESTIONS

A. Extension of language coverage

Free-Flow can be easily extended to cover different languages. The only modification required to support a new language is appropriate training data. Tesseract provides training data for over 27 languages [6] as well as tools to help construct a user's own training data. With extended training data, Free-Flow can serve as a multilingual translator . More interesting work will involve hand-written letters. Currently, recognizing handwritings is considered a much harder problem than recognizing typed characters. User-specific handwritings can be collected and processed to construct training data to further customize Free-Flow.

B. Speedup of the Extract Module

As discussed in the Evaluation section, the current runtime of Free-Flow is limited by the Extract module. More work can be done to simplify some functionalities of the OCR engine to speed up the process. The tradeoff between the runtime and accuracy/robustness for each submodule within the Extract Module will be informative to future researchers applying OCR to their systems.

C. Automated Word Detection

Currently, users must drag and resize the ViewBox in order to select a word, and they often need to refocus the camera multiple times. In the future, the size of the ViewBox and

[6]https://github.com/tesseract-ocr/tessdata

the camera focus can be automatically adjusted to make the pipeline more fully automated. It is a challenging research question how to infer which word in the middle of texts lies under the user's focus.

D. More robust search

The current search algorithm finds the closest match only in the scope of the dictionary. As a result, if the recognized word is farther away from the actual word than the closest match in the dictionary, the result of the search is incorrect. A more robust search can be implemented by performing spell-check before a search or runs two passes of search where the result of the first search can be used to return a list of possible words and the second search can be informed by the first pass. Statistical linguistic models can be augmented as a heuristic for a faster search.

E. Faster search

A different data structure such as a cache-conscious HAT-trie [13] can bring more speed up to the search.

VIII. Conclusion

We have described *Free-Flow*, a full end-to-end mobile application that brings together image-processing, OCR and Search algorithms to quickly search a word in the printed texts. Free-Flow uses a built-in camera of an Android smartphone and fully functions without any network connectivity. Its system consists of four main modules, **Capture**, **Pre-process**, **Extract**, and **Search**. Users hold their mobile phones to take a picture of the word to search (Capture). The captured image is appropriately scaled and binarized through a median filter (Pre-process). The OCR engine based on the Tesseract engine is trained on the default (or user-provided) training data. Upon the binarization of the captured image, it performs connected component analysis and two rounds of adaptive recognition to extract a string of words (Extract). Free-Flow can be easily extended to support other languages by adding training data for the target languages including user-specific fonts like handwritings. Once a word is recognized by the Extract module, it is searched on the dictionary via closest match search algorithm. The dictionary is stored in a burst trie data structure in memory. The burst trie achieves a fast search with time complexity linear in the length of the search word and preserves the alphabetical sort order to support the closest match search. It is much more space-efficient than a standard trie as it achieves the same space complexity as a standard tree or a hash table. Our experiments on test texts in Ubuntu and Times New Roman under three different lightings show Free-Flow achieves 95.25% recognition accuracy on average and maintains is robust against ambient lightings. Its response time is on average 1.117 sec on 1000 pixels, which is an approximate average size of the captured images. Free-Flow thus proves the viability of the idea to take advantage of mobile devices to make a search on the printed texts more unintrusive and interactive.

References

[1] M.Heliski, M.Kmieciak and T.Parkoa, "Report on the comparison of Tesseract and ABBYY FineReader OCR engines," in *PCSS*, 2012.

[2] Ray Smith, "An Overview of the Tesseract OCR Engine," in *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society*, 2007, pp. 629-633.

[3] Chirag Patel, Atul Patel, and Dharmendra Patel. "Optical character recognition by open source OCR tool tesseract: A case study," in *International Journal of Computer Applications 55.10*, 2012, pp 50-56.

[4] J. Edwards and D. Forsyth. "Searching for character models," in *Neural Information Processing Systems*, 2005.

[5] T. K. Ho. "Bootstrapping text recognition from stop words," in *International Conference on Pattern Recognition*, 1998.

[6] T. K. Ho and G. Nagy, "OCR with no shape training," in *International Conference on Pattern Recognition*, 2000.

[7] Andrew Kae *et al*, "Improving state-of-the-art OCR through high-precision document-specific modeling," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE*, 2010.

[8] Gross, Ralph, and Vladimir Brajovic, "An image preprocessing algorithm for illumination invariant face recognition," in *Audio-and Video-Based Biometric Person Authentication. Springer Berlin Heidelberg,* 2003.

[9] Eidenberger, Horst, "Illumination-invariant face recognition by Kalman filtering," in *Multimedia Signal Processing and Communications, 48th International Symposium ELMAR-2006 focused on. IEEE*, 2006.

[10] Jian, Gong, Li Liyuan, and Chen Weinan, "A fast recursive algorithm for two-dimensional thresholding," *Signal Processing, 1996., 3rd International Conference on. Vol. 2. IEEE*, 1996.

[11] Heinz, Steffen, Justin Zobel, and Hugh E. Williams, "Burst tries: A fast, efficient data structure for string keys", *ACM Transactions on Information Systems (TOIS) 20.2 (2002): 192-223*.

[12] Levenshtein, Vladimir I, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics doklady. Vol. 10. No. 8*, 1966.

[13] Askitis, Nikolas, and Ranjan Sinha, "HAT-trie: a cache-conscious trie-based data structure for strings," *Proceedings of the thirtieth Australasian conference on Computer science-Volume 62. Australian Computer Society, Inc.*, 2007.