Interactive Text Recognition and Translation on a Mobile Device

Michael Hsueh

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-57
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-57.html

May 13, 2011

Interactive Text Recognition and Translation on a Mobile Device

Michael Hsueh
University of California, Berkeley
Computer Science Division

***Abstract*—The convergence of powerful processors and high resolution cameras on mobile devices has made them an attractive platform for optical character recognition (OCR) software. Most traditional OCR applications have been designed to be highly automated and used on desktop machines. These recognition engines perform well but usually require high quality input images that are reliably obtained. In adapting recognition systems for use on mobile devices, conventional assumptions about the need for automation, available processing power, and input image quality should be re-evaluated. This paper presents a mobile text recognition and translation system that is designed with consideration for these factors. The application presented runs on the Nokia N900 smartphone and introduces human-assisted elements into the OCR pipeline to enhance accuracy. These elements include manual cropping, classification, segmentation, and thresholding. The system also employs vignetting correction and a web-based recognition service in order to address the camera and performance limitations of the N900. The completed application was deployed publicly for testing by the Maemo community under the name *MIR Translator*. Feedback for the system was positive overall and confirms the utility of text recognition software on mobile devices.**

I. Introduction

The overall capabilities of mobile devices have rapidly increased in recent years in terms of processing power, connectivity, and available sensors. These advancements, together with the growing prevalence of smartphones, have made it feasible and in some cases preferable to run OCR software on mobile platforms. Naturally, image processing applications that are ported to mobile devices frequently must adapt to very different technical and usage assumptions than those under which they were originally developed. For example, traditional OCR users often use workstations to set up long running, automated jobs to process large amounts of scanned documents. In contrast, mobile users might only need to process short segments of text on demand and with the expectation of low response times.

One challenge in adapting OCR for mobile devices is in handling lower quality input images. Conventional recognition systems assume input data to be high resolution images obtained from flatbed or sheet fed scanners. As a result, many existing engines are optimized for processing clean and sharp images of text. Unfortunately, hardware limitations and volatile operating environments usually cause cell phone cameras to produce lower fidelity images that suffer from defects such as noise or uneven lighting.

A second drawback in running OCR on mobile devices is that they have much lower processing capabilities than workstations. This presents an obstacle in particular for the recognition of complex scripts and characters such as those of certain East Asian languages. Recognition of these languages is resource intensive, as the engine must distinguish among significant numbers of characters. The problem is magnified by the fact that use of mobile applications is commonly characterized by short bursts of "on-the-go" activity [1]. This usage behavior means short response times are a necessity for any mobile application.

Fortunately, mobile devices also offer advantages to recognition applications that extend beyond simply portability and ubiquity. Smartphones can be used to capture images of text from arbitrary objects at distances that are impractical for scanners. In addition, mobile OCR is particularly useful when the user needs to input a foreign language, especially those based on a foreign script. Take for example a traveler in a foreign country who has neither the input method nor knowledge to enter the local language into his or her mobile device. The user may need to translate a physically encountered piece of text but has no convenient way of doing this, even given access to a multilingual electronic dictionary. Objects requiring quick translation can include anything from signs or product labels to restaurant menu items.

Various mobile recognition–translation applications for consumers currently exist, but none have yet to completely resolve the issues outlined above. A number of these offerings are very promising but have been primarily focused only on languages based on the Latin alphabet. Recognition of a wide range of languages including those with complex, logographic writing systems is necessary for a solution to be truly adequate.

A practical solution addressing these issues can be developed by leveraging (1) the camera and Internet connectivity built into most modern mobile platforms and (2) an OCR pipeline adapted to address the platform's unique limitations and advantages. This paper presents MIRATS, a system designed around these points.

The presented system uses web services to increase the mobile platform's effective processing power for character recognition. It also addresses some limitations specific to the N900 camera that interfere with OCR accuracy. MIRATS modifies the typical recognition pipeline from being entirely automated to more user interactive. In this way, it takes advantage of the high user engagement afforded by smartphone usage patterns and aids the accuracy of the system in face of the aforementioned challenges.

The rest of this paper is organized as follows. Section II discusses related work on text recognition and translation on mobile devices. Section III gives an overview of the design and implementation of the presented system. Section IV describes the N900 application used to interface with the system. In Section V, experiments and an evaluation of the system are explained. Section VI proposes future enhancements to the system, and a summary of this work follows in Section VII.

II. Related work

Since the popularization of digital cameras, efforts have been made to adapt text recognition algorithms to work on digital camera platforms. In shifting from the pseudo-binary nature of scanned text images to real world pictures, much focus has been on the problem of text localization and extraction. Work in areas such as text line detection, perspective removal, and environmental background removal from natural pictures have yielded accuracy rates ranging from 50 to 70 percent [2]–[4], depending on the images tested. For surveys on general issues surrounding camera-based text digitization, see [5], [6].

Researchers have built early examples of standalone OCR systems that run on cell phone hardware [7]–[9]. Senda et al. [10] demonstrate a "camera-typing" interface that relies on the concatenation of multiple photos taken by a smartphone camera for text input. The system built by Laine and Nevalainen [9] was found to consume most of its computational time performing tasks that are specific to adapting text recognition for mobile environments (e.g., skew adjustment). Experiments from the mentioned systems suggested that performing OCR on commodity mobile hardware was of borderline practicality due to long processing times. This problem is likely to have been mitigated to an extent by recent advances in mobile technology. Work in mobile OCR by Joshi et al. [11] takes advantage of new hardware features such as the tilt sensor to aid proper user input.

A number of OCR engines are widely available and in current use. Open-source projects include GOCR[1], OCRAD[2], OCROpus [12], OpenOCR[3], and Tesseract [13], [14]. Commercial products supporting mobile platforms include ABBYY FineReader[4], Nuance OmniPage[5], and ExperVision OpenRTK[6], among others.

Image recognition systems that incorporate human-assistance have been developed. Wilkinson, Garris, and Geist [15] used an OCR engine augmented with human verification to generate large test sets for other OCR systems. Blostein [16] approached assistance from the user interface perspective, employing specialized input devices and N-best lists to correct errors in diagram recognition. User-driven segmentation and classification were shown to improve recognition of UML diagrams in a system built by Lank, Thorley, and Chen [17]. Researchers in the area of mathematics recognition have demonstrated style-preserving morphs for inputting and manipulating hand-written formulas [18], [19]. Mankoff, Abowd, and Hudson [20] created a user interface toolkit consisting of reusable error correction elements and architectural extensions to provide structural support for handling ambiguity in recognizers.

Solutions combining recognition and translation services on mobile devices have come largely from the industry. By far the most popular offering is Google Goggles[7], an image analysis and visual search tool available for Android and iPhone devices. It supports recognition and search of a growing variety of subjects including landmarks, barcodes, and everyday objects. The application is also capable of text recognition and translation; at the time of this writing, the supported languages are English, French, German, and Spanish. Google Goggles works by capturing images of interest on the device and then sending them over the Internet for recognition in the company's databases.

Augmented reality is a relatively new area where several text recognition and translation applications have emerged. Nokia Point & Find[8] was an experimental application that recognized movie posters in real time and displayed relevant information to the user. World Lens[9] is an application for iPhone that translates text visible in the viewfinder and overlays the results on the screen in real time. The system works without Internet connectivity and currently supports translation from English to Spanish and vice versa. In the academic domain, Fragoso et al. [21] built a similar application that relies on a web-based service for translation.

III. System Overview

The MIRATS system was developed in Qt. The N900 application also uses the open-source library FCam API[10] for interfacing with the platform's camera. A recognition and translation cycle in the system follows the process outlined in Fig. 1 and the steps below.

1) The user begins by capturing an image containing text of interest using the N900 camera.
2) The user manually specifies the region of the image containing the text to be recognized.
3) The specified area of the image is partially preprocessed on the device in order to optimize it for transfer and input to the OCR server.
4) The preprocessed data is sent to a web service where recognition takes place. The results are delivered back to the device.
5) The device sends the results of the recognition to a translation web service and obtains a translated string.

[1] http://jocr.sourceforge.net/
[2] http://www.gnu.org/software/ocrad/
[3] http://en.openocr.org/
[4] http://finereader.abbyy.com/
[5] http://www.nuance.com/for-individuals/by-product/omnipage/
[6] http://www.expervision.com/ocr-sdk-toolkit/openrtk-ocr-toolkit-sdk
[7] http://www.google.com/mobile/goggles/
[8] http://pointandfind.nokia.com/
[9] http://questvisual.com/
[10] http://fcam.garage.maemo.org/

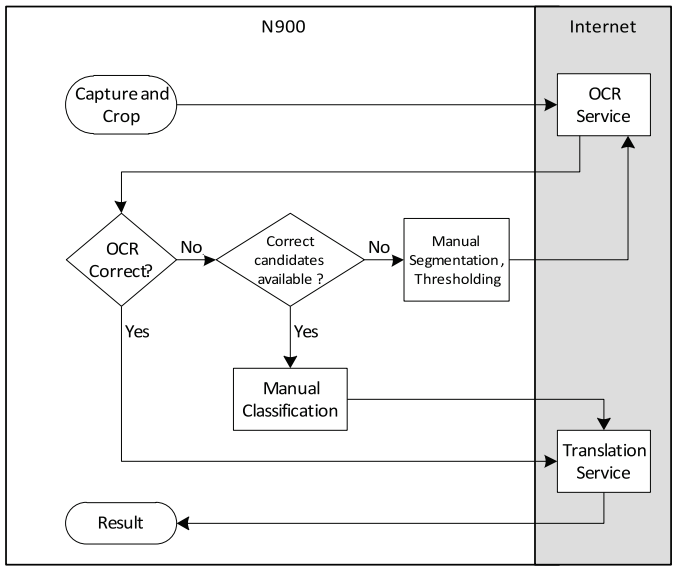

Fig. 1. MIRATS system organization. The application captures an input image and sends it to an OCR web service. The results are returned to the device and inspected by the user. The user performs manual error correction tasks as needed. When the recognized text is correct, it is sent to a translation web service as the final step.

6) The user examines the results and makes manual adjustments to the recognized text if necessary to produce a better translation.

In designing the system, it had to be decided how the steps for recognition would be divided between the device and the server. It is possible to perform the OCR process partially on the mobile device. Specifically, it was considered to have the N900 complete feature extraction locally and then send the resulting feature vectors to the server for classification. Because the client would be responsible for inspecting the input image, image preprocessing steps would be forced to take place on the mobile device. This organization avoids network transmission of raster image data, which if uncompressed is almost certain to be much larger than feature vectors (the N900 has a 5MP camera). Ultimately, it was determined that input images can usually be sufficiently reduced in size by downsampling, compressing, and converting them (to grayscale) such that they become smaller than their associated feature vectors. These steps can be parameterized to minimize their impact on recognition accuracy in most cases. This is the design route taken for MIRATS.

A. Server

It was important for the MIRATS server to support recognition of as many different languages as possible. Especially difficult are languages such as Chinese or Japanese, which contain large sets of symbols that are problematic for OCR engines designed for small alphabets. Several language-specific engines address this issue, such as the one by Huo, Ge, and Feng [22], which uses Gabor features and discriminative feature extraction. On the opposite end, language-independent systems such as the one built by Lu et al. [23] using hidden Markov models have shown good results. The solution chosen for MIRATS was Tesseract due to its open-source availability and prepackaged support for many languages [14]. It is UTF-8 capable, fully trainable, and available with trained data for over 30 languages.

The server listens for connections from client devices requesting recognition service. A request includes the incoming image as well as metadata specifying the desired language for recognition. The server runs a version of the Tesseract engine that has been modified to provide additional information about the confidence, classification choices, and segmentation boundaries for each recognized character. Specifically, the engine outputs the confidence level as percentage ranging from 0-100 for each character that is recognized. It also produces a configurable number of the best alternative characters that were considered during classification but not chosen. Finally, the engine attaches a bounding box to each recognized character that indicates the region of the input image in which the character is located. It should be noted that while this system uses Tesseract to power its OCR, any other engine capable of producing these additional pieces of information can be easily integrated into the MIRATS system design.

The server is intended to be deployed as a web service but can also run locally on the mobile device. This may be necessary in areas of limited connectivity or to avoid expensive bandwidth charges while roaming. The server module has been successfully deployed and tested on instances of Amazon EC2, a desktop workstation, and the N900 itself. The main disadvantage of running OCR on the N900 is limited performance and a need to install large trained language files for Tesseract. Additionally, if translation is desired, Internet access to a translation web service is still necessary. Local translation software can be used, but the limitations described for local recognition apply similarly.

B. Client

The application begins with the user capturing an image and then manually cropping out a region to process. The application translates the cropping coordinates from viewport space to image space and begins preprocessing the region. Preprocessing involves vignetting correction (discussed in the next section), conversion to grayscale, and compression. For efficiency, only the selected region of the image is processed. Subsequent recrops of the same source image will use previously computed data. If a new crop is made but includes previously selected areas, only the new, unprocessed regions need to be computed.

Once the cropped area is preprocessed, it is sent to the recognition server. The application parses the results returned by the server and sends them through a POST request to the Google Translate API [11], specifying the user selected source and destination languages.

[11] http://code.google.com/apis/language/translate/overview.html

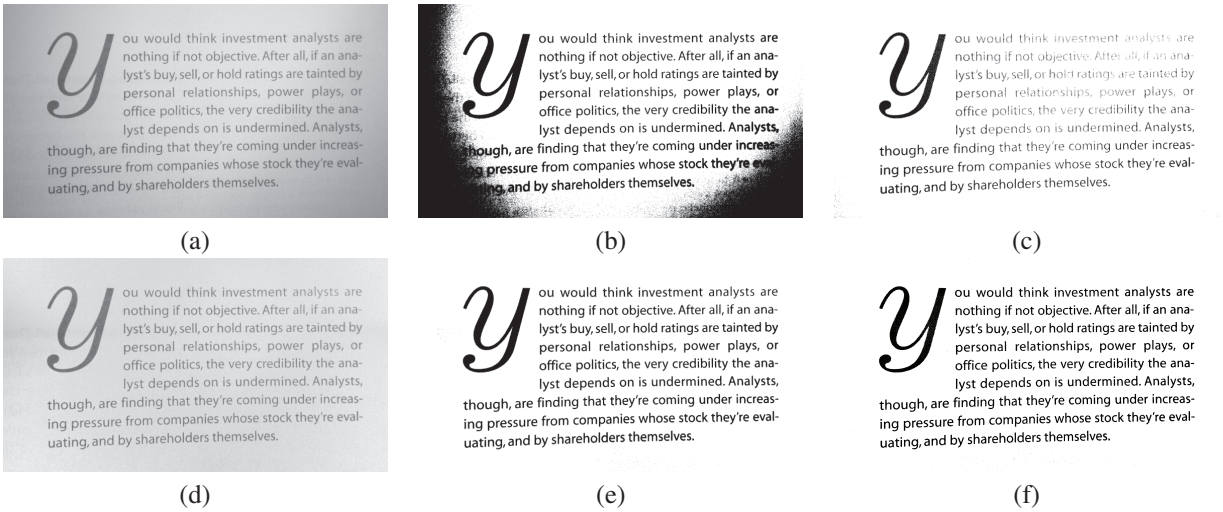

Fig. 2. **(a)** - An example of a non-corrected image that the application sends to the OCR server. **(b)** - A manually thresholded version of (a) that attempts to keep all text pixels dark. **(c)** - A manually thresholded version of (a) that attempts to keep all background pixels white. In this image, the text contained in lighter areas begin to disappear. **(d)** - The image from (a) with the vignetting correction function applied. **(e)** - The image from (d) thresholded by the OCR engine. **(f)** - The image from (a) after local adaptive thresholding with window size 100×100 pixels and a pixel value bias of -10%.

C. Vignetting Correction

The N900 camera produces a severe amount of vignetting (i.e., darkening) around the outer edges of captured images. This issue appears in Nokia's default camera application when in the viewfinder mode but is resolved in captured images by the software's internal processing.

FCam outputs raw images captured by the CCD and therefore does not correct the vignetting effect. Although it does not cause visual problems in casual pictures, it presents a serious difficulty for OCR. Tesseract binarizes images into text and background areas according to Otsu's method [24], which performs histogram shape-based thresholding over the image. Because of the significant vignetting on the N900, this technique alone yields poor results in many cases. When the input image spans non-vignetted areas to severely vignetted areas (see Fig. 2(a)), there is no appropriate global thresholding value that can be chosen (see Fig. 2(b)(c)), even manually. In these cases, the non-text pixels of vignetted areas appear just as dark as the text pixels of the non-vignetted areas.

One way to resolve this issue is to employ local adaptive thresholding over the image. This would correct, in addition to the vignetting specific to any particular camera sensor, the more general problem of uneven lighting . A drawback of this approach is that it incurs considerable performance penalties over the use of global thresholding. A more significant issue is that local adaptive thresholding is most effective if its parameters are chosen correctly. The two parameters that need to be specified are (1) the size of the local area to be sampled per pixel and (2) the amount of bias given to a pixel once a threshold for its local area has been calculated. In Fig. 2(f), a vignetted source image undergoes local adaptive thresholding with a 100×100 pixel window size and a -10% bias for thresholded pixels. As can be seen from the picture, the results are good.

It was decided not to automatically enable local adaptive thresholding but rather to leave this method available as a manual task for the user. This feature will be explained in the latter part of this paper. The decision to forgo automatic adaptive thresholding was motivated by the fact that poorly chosen parameters can degrade instead of improve results. Window sizes that are too small will end up hollowing out text with thick lines. Large window sizes reduce the adaptive advantages of local thresholding and result in poor performance.

To correct vignetting, a per-pixel correction function was used. The function is based on comparing any given pixel's change in brightness to its distance from the center of the image. To estimate the correction, several pictures of uniformly colored surfaces were taken with the N900 camera. For every image, the HSV value for each individual pixel was compared with the average value V_c of the pixels in the central 10×10 pixel region of the image. For any given pixel p_x with Value $V(p_x)$, the point $K(p_x)$ plotted is simply the relative change going from $V(p_x)$ to V_c:

$$K(p_x) = \frac{V_c - V(p_x)}{V(p_x)} \tag{1}$$

The results are shown in the Fig. 3. A curve fitting produced a second-degree polynomial with $R^2 = 0.92$ that approximates how much brightness to apply to each pixel to counteract vignetting. Experiments indicate that the function brings pixels sufficiently close to appropriate values for Ostu's global thresholding algorithm to work effectively.

Because the function needs to be evaluated per pixel, it is important for performance reasons that only the cropped out region of the entire source image undergoes vignetting correction. For a pixel p_x with Euclidean distance $D(p_x)$ from the center of the image, the correction factor $C(p_x)$ and corrected pixel value $V'(p_x)$ at the same location in the

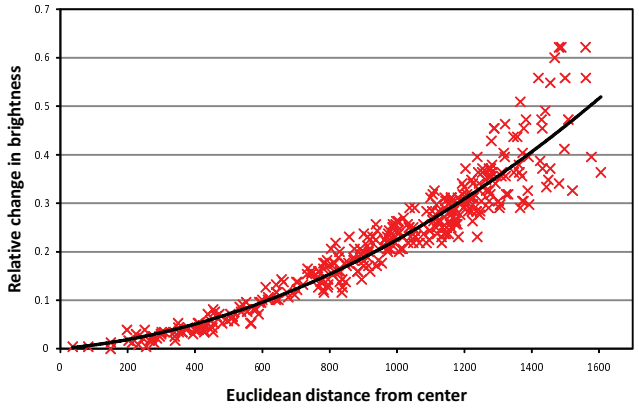

Fig. 3. Relative change in pixel brightness versus Euclidean distance from the image center.

corrected image is given by the following:

$$C(p_x) = 2 \times 10^{-7} \times D(p_x)^2 + 7 \times 10^{-5} \times D(p_x) \quad (2)$$

$$V'(p_x) = V(p_x) \times (1 + C(p_x)) \quad (3)$$

Since the $C(p_x)$ correction factors depend only on the distances of individual pixels, their values can be precomputed during the first run of the application and stored for all possible pixel-to-center offsets. This allows the correction function to subsequently look up the factors at run time instead of calculating them, avoiding costly multiplication operations. It is only necessary to store correction factors for one quadrant of the entire image area, since the vignetting effect is approximately symmetric across all quadrants. The N900 camera produces images at a resolution of 2560×1944, and thus $C(p_x)$ values are stored only for offsets landing in the upper left 1280×972 area.

It is important to note that the stored factors are indexed by offsets (from the center pixel) rather than actual Euclidean distance. This is to avoid pixel-wise calculations of distance at run time. A further optimization can be realized by omitting redundant storage of factors. Specifically, it is only necessary to store values for the set of all unordered pairs of offsets in each direction. For example, let x_c and y_c be the horizontal and vertical coordinates of the center, respectively. Then the pixels at positions $(k_1 + x_c, k_2 + y_c)$ and $(k_2 + x_c, k_1 + y_c)$ have the same Euclidean distance and thus only one $C(p_x)$ value is stored for the both of them. With the factors stored as 32-bit floating-point numbers, the precomputed file is 3MB in size.

The vignetting profile of the sensors on different N900s may not be exactly the same. If this is the case, a calibration process that automates the above steps can be used to determine the appropriate correction function for each individual device. The results are expected to work well so as long as the vignetting profiles are not radically different across devices.

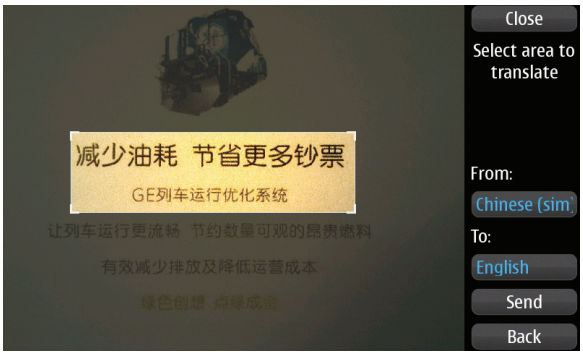

Fig. 4. Interface for cropping the area to translate.

IV. User Interface

Typical usage of the application is summarized in Fig. 1 and below.

1) The viewfinder is used to capture an image containing text of interest.
2) The user manually selects the area containing the text to be recognized from the image (see Fig. 4). The application is designed for cropping to take place after the image is captured. The reason for this is that it is significantly easier to crop from a stationary image than directly on a live viewfinder. The latter approach is taken by Google Goggles.
3) The source and destination languages are specified using drop down menus. The application currently supports over 30 different languages for both input and output.
4) The user presses the *Send* button and awaits the results.
5) The results screen displays the recognized text as well as its translation (see Fig. 5). After inspecting the results, the user has the option of performing manual corrective tasks to fix any errors in the recognized text. These tasks are described in the following sections.
6) The user can return to the original captured photo and select another area to translate, or capture a new photo to process.

MIRATS is designed with the belief that human assistance is both useful and viable for mobile text recognition. There are two reasons for this assumption. The first is that users are not expected to use smartphones to digitize large corpuses. Mobile OCR is more likely to serve as a handy tool for capturing short segments of text that may be difficult or inconvenient to enter manually. Consequently, full automation of the OCR pipeline is not a strict requirement. The second reason is that usage sessions on mobile devices tend to be short, occur in physically stable situations, and have a high level of user engagement [1]. It is expected, therefore, that mobile users will be willing to engage in the interactive use of OCR if response times are kept low. MIRATS provides several interactive mechanisms with which the user can improve recognition accuracy given a reasonable amount of effort.

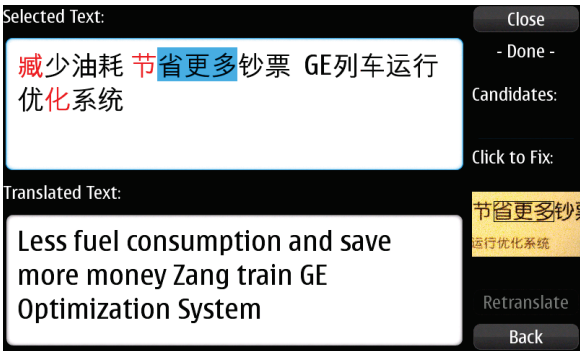

Fig. 5. Results screen showing color coding on the recognized string. The window shown on the right provides visual reference of the three selected characters.

Fig. 6. Manual classification of a selected character using the menu shown on the right.

A. Visual Reference of OCR Results

Recognized characters are displayed in either black or red font. Red characters indicate a recognition confidence level below a configurable threshold. The system uses 75% as the default threshold. Color coding confidence levels allows rapid verification of results. Individual or clusters of red characters immediately draw attention to areas that may require user attention.

In order to aid manual correction, the application displays visual references of the recognized characters. The bounding rectangles returned by the recognition service record the regions in the input image from which each character was extracted. When the user highlights an individual character, the local region of the image corresponding to that character is displayed in a window on the interface. The image is appropriately scaled and the character's precise bounding box is drawn over the image. This gives the user feedback about the precise segmentation that took place during recognition. If instead a group of characters is selected, the visual reference window presents the maximum bounding rectangle encompassing all the selected characters (see Fig. 5).

B. Manual Classification

Because translation is highly sensitive to even minor errors in recognized text, the application allows the user to assist in classifying segmented characters. Manual classification is most helpful in the recognition of languages containing large numbers of characters that share strong resemblances. In some cases involving small or blurry print, it is difficult for even humans to make the distinction between similar characters without semantic, contextual clues.

Upon selection of a recognized character, the application displays a list of the best alternative characters that were considered but not chosen during classification (see Fig. 6). These choices are displayed in order of decreasing confidence. The user can select one of these alternatives to replace the original character. Once changes to all desired characters are made, a new translation request can be sent with the revised text.

C. Manual Segmentation

Segmentation errors are more serious and cannot usually be corrected by manual classification. When manual classification is insufficient or when a large region of the image is incorrectly recognized, the user can specify an area of the image to reprocess. To do this, the user first selects the erroneous characters from the recognized text and then clicks on the visual reference window. The resulting dialog box displays an area in which the user manually adjusts a cropping rectangle to contain the region of the input image to reprocess (see Fig. 7). The cropping rectangle is initially configured as the bounding box over the selected text. In some cases, the original segmentation was correct and no adjustments to the cropping rectangle are necessary. When the user has appropriately adjusted the cropping area, a new recognition request of the selected region is sent to the OCR server. The response to the request is inserted into the existing result body, replacing the initially selected contents.

Manual segmentation provides a "second chance" opportunity to perform recognition on a particular area of the image. This is helpful for cases in which there are extraneous marks within the translation region that cannot be removed except through a localized, intermediate segmentation step. Manual segmentation also amounts roughly to a user-driven form of adaptive thresholding. One can imagine the extreme case of an input picture that is shaded on one side and brightly lit on the opposite. Manual segmentation allows thresholding and recognition of the two distinct parts of the image independently of the parent image.

Other examples of failed segmentation include multiple characters being mistakenly grouped together or a single character being split apart. Either of these scenarios can be remedied by manual segmentation. The same techniques can be used to recover omitted characters from the recognized string. Recoverable areas include those that were wiped out for falling under the thresholding value or simply ignored by the recognition engine. Entire areas of input images are occasionally omitted by the engine because the text they contain is dismissed as noise. This can happen for example if the text is of a significantly smaller font than in other parts of

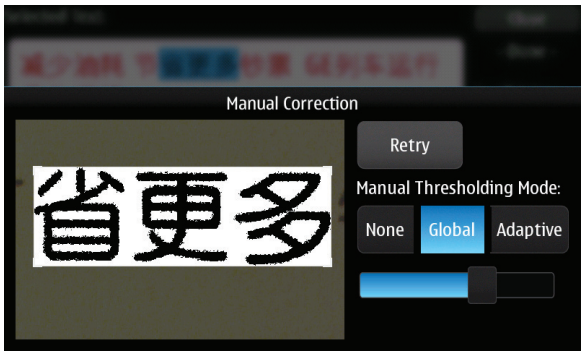

Fig. 7. Interface for manual segmentation and thresholding. Manual global thresholding is active.

the image. The technique of using localized follow-up requests usually yields more accurate results than single requests over the entire image.

D. Manual Thresholding

The correction dialog described above also allows the user to forgo automatic thresholding by the OCR engine in favor of manual thresholding. Upon selecting either Global or Adaptive modes from the thresholding options, the user is presented with a slider bar for adjusting thresholding parameters. In Global mode, the slider sets the [0-255] grayscale value used for global thresholding. As the slider is adjusted, thresholding is applied dynamically and the binarized result is updated in real time for the user to verify (see Fig. 7).

In Adaptive mode, the application applies Sauvola's adaptive thresholding [25] according to the method described by Shafait, Keysers, and Breuel [27], allowing a run time independent of the size of the thresholding window. Efficient implementation is required for the application to be able to provide continuous visual feedback on the thresholded image. When Adaptive mode is first selected, the integral image of the currently cropped region is computed. Subsequent adjustment of the slider changes the window size used for thresholding, which ranges from 0 to the maximum of either the length or width of the cropped region.

All manually thresholded images are binarized on the client, significantly reducing the amount of data that needs to be sent to the server for reprocessing. It should also be noted that while manual thresholding is most likely to be used to fix localized areas of the input image, it can be just as readily applied to the entire image.

V. Evaluation

The accuracy of recognition in the system was measured on a variety of images of small segments of text taken by the N900. All images used were of printed materials or objects. 50 images were captured each for both English and Simplified Chinese text. To measure performance, the average amount of time required to perform recognition and vignetting correction on these images was recorded. In the tests that follow, four manual classification alternatives were sent per recognized character. The workstation used for testing was running 64-bit Windows 7 with a 3.0GHz processor and 4GB RAM.

TABLE I
Phrase Match Accuracy

Language	Corrected	Auto Classification	Manual Classification
English	No	32.0%	40.0%
English	Yes	44.0%	48.0%
Chinese	No	24.0%	38.0%
Chinese	Yes	34.0%	40.0%

A. Accuracy

The results of the accuracy tests are summarized in Tables I and II. The results are divided by language (English and Chinese), whether the input image underwent vignetting correction (Yes/No), and whether the results allowed for manually classified characters (Yes/No). Two methods for scoring accuracy are used. The first is the rate of exact phrase matches. This measures the number of requests in which the recognized characters matched those in the source image exactly. Phrase matching is difficult but important because translations are sensitive to minor errors in the recognized string. The second score is character-wise matches, which reflects approximately how well the recognition performs in terms of single characters. Given two strings A and B, the character-wise match score between A and B is given by:

$$score(A, B) = \frac{|A \cap B|}{\max(|A|, |B|)} \tag{4}$$

Based on these tests, accuracy for individual character recognition is reasonable at around 80% while phrase matching leaves much room for improvement. The difficulty with phrase matching is that it is likely for a series of recognized characters, each having a roughly 20% chance of being incorrect, to contain errors. A single incorrectly recognized or omitted character results in a phrase match failure. The tests revealed that recognition accuracy tends to be low when dealing with stylized fonts or text with non-uniformly colored backgrounds.

A large number of failed tests were due to incorrect segmentation. For example, a "W" character can be misinterpreted as the visually similar string "\/\/". Resulting translations are unlikely to be helpful in these cases. Still, this type of error is not overly problematic for the system, since user-assisted segmentation can be used when manual classification is insufficient. A more serious type of error is global segmentation mistakes. This occurs when the overall granularity of text is misinterpreted and usually causes catastrophic recognition results. Grainy images in which the OCR engine mistakes noise for text often cause global segmentation errors. These issues reinforce that the system's manual assistance features are appropriate.

Overall, the tests show that both vignetting correction and manual classification measurably improve recognition accuracy. Also, while phrase matching is important, its deficiency is not viewed as fundamentally problematic for the system.

TABLE II
CHARACTER-WISE MATCH ACCURACY

Language	Corrected	Auto Classification	Manual Classification
English	No	56.0%	57.1%
English	Yes	78.5%	80.0%
Chinese	No	73.7%	77.8%
Chinese	Yes	80.6%	83.9%

This is because manual segmentation makes phrase matching more feasible.

B. Performance

The measured tasks run approximately seven to nine times faster on the server than on the N900 (see Table III). Especially costly is the recognition of Chinese on the N900, which takes over four times the duration required for English on the same platform. It is expected that other languages with large character sets such as Japanese should incur similar performance penalties. The increased complexity of these languages is reflected in the significantly larger sizes of their trained data files. For Tesseract, Chinese trained data occupies 40MB compared to just 2MB for English. Given these differences, it may be practical to employ different approaches to the recognition of logographic versus alphabetical scripts. For example, the application can choose to perform English recognition on the N900 but rely on the server for Chinese recognition.

The application might also dynamically adjust how to divide the work of image processing between the device and the server. Currently, all image processing steps are performed on the N900 so that only a grayscale, compressed image needs to be transferred over the network. By far the most costly of these steps is vignetting correction, which works best on the color version of an image. Therefore, moving this step to the server would require transferring a full-color image and consuming more bandwidth. Whether the application should attempt local image processing depends on how much bandwidth is currently available. If the amount of time saved by performing vignetting correction on the server outweighs the extra time needed to send a full-color image, then it is worthwhile to process the image on the server.

To see this more clearly, call T_s the time to transfer the color image, T_n the time to transfer the grayscale image, V_s the time for the server to perform image processing, and V_n the time for the N900 to perform image processing. Server-side image processing should be used if $T_s - T_n < V_n - V_s$. T_s and T_n can be calculated by dividing the estimated file sizes of the color and grayscale images by the available bandwidth. Since vignetting correction accounts for the majority of image processing times, V_s and V_n can be roughly approximated by the vignetting correction times measured for the server and N900. The exact values of V_s and V_n depend on the size of the image to be processed, so a linear model estimating processing time based on image dimensions can be devised if greater precision is required.

TABLE III
RUN TIME PERFORMANCE

Task	Server time	N900 time
English Recognition	376 ms	3374 ms
Chinese Recognition	1760 ms	14182 ms
Vignetting Correction	307 ms	2239 ms

C. Public Deployment

The system was deployed publicly for testing by the Maemo community under the name *MIR Translator*[12]. The server was set up on an Amazon EC2 micro instance, with support for translation to and from 35 different languages. Public usage of the service was monitored via server logs over the first week of deployment, during which over one thousand recognition requests were serviced per day on average. The average size of input images for requests was approximately 100KB. Manual inspection of the requests confirmed that the recognition module is dependent on input images being of good quality. Results are best on images that are reasonably well and evenly lit and have good contrast. Stylized fonts that are not well aligned or are shown on complex backgrounds tend to strongly degrade the accuracy of recognition. In addition, Tesseract OCR uses rectangular regions for segmenting images and does not employ any un-skewing step. As a result, italicized or connected fonts can often present difficulties for the engine. It may be possible to incorporate skew correction as a manual step in future work.

Several examples of successfully processed images by the server are presented in the top row of Fig. 8. The bottom row contains input images from requests that failed. Images in the top row fit the criteria of containing typographic-like fonts that are well lit and approximately axis-aligned. They have well defined and distinct areas corresponding to the text and background. The extraneous handwritten markings in Fig. 8(c) did not prevent the OCR module from producing a satisfactory recognition.

Tesseract does not perform page layout analysis and thus Fig. 8(d) was incorrectly recognized due to the frequent extraneous border markings present in the image. Manual segmentation was sufficient to separate the two distinct regions of text in the image and produce a satisfactory result. Recognition of Fig. 8(e) was affected by faulty segmentation because the first and last letters in the word “Smile” appear in a stylized font. Manual segmentation to crop out the tail-like features of the stylized ‘S’ and ‘e’ characters yielded a satisfactory result. The subtle pattern in the background of this image was successfully eliminated through automatic thresholding and did not present a problem. Fig. 8(f) suffers from a significant amount of uneven lighting and non-axis aligned text. These factors, along with the extraneous grey region in the lower portion of the image, make it difficult to process. Recognition on this image is not achievable even when manual correction methods are used.

A considerable number of users expressed interest in an

[12]http://mhsueh.com/mir/

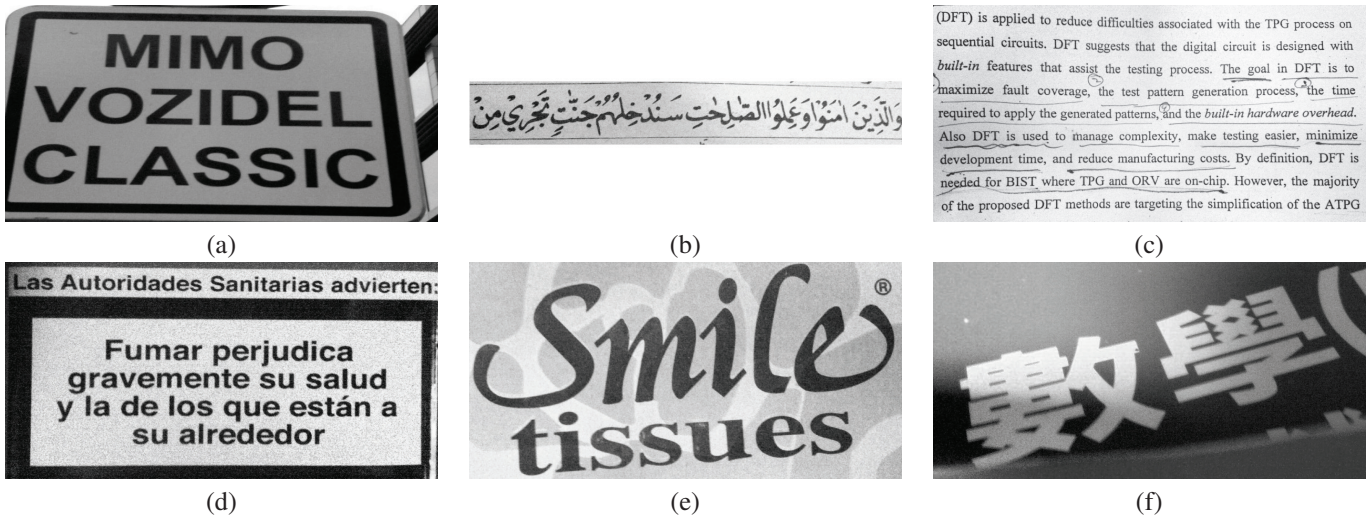

Fig. 8. A sample of images processed by the public MIR Translator OCR server. The top row [**(a)**,**(b)**,**(c)**] shows images that were recognized correctly. The bottom row [**(d)**,**(e)**,**(f)**] contains examples of failed cases. **(d)** - This image failed segmentation due to extraneous border markings throughout the image. It was correctable with manual segmentation. **(e)** - This image failed segmentation due to the stylized font used in the 'S' and 'e' characters of "Smile." It was correctable with manual segmentation. **(f)** - This image failed recognition due to uneven lighting, non-axis aligned characters, and extraneous marks. It was not manually correctable.

offline version of the application. This is driven by the fact that Internet roaming charges in foreign countries, in which a photo translation tool might be handy, are often expensive. Fortunately, offline operation of the application is feasible. As described earlier, one solution is to deploy the server module on the N900 for local processing. The MIR Translator application was not initially released as local-only because (1) it is desirable to minimize OCR processing times for complex languages and (2) an online recognition service significantly reduces the local complexity and install footprint of the application. The current trained language data for the approximately 35 supported languages amount to roughly 250MB. In contrast, the total package size for the client application is only 1.5MB. An offline solution might handle this by only supporting languages that the user preinstalls. The languages to be supported can be configured at install time or downloaded later.

VI. Future work

A number of potentially helpful future enhancements to the system are described below.

Interactive mode. It may be helpful for the application to support an entirely interactive mode in which the user does not make corrections but rather controls all steps of the recognition pipeline. For example, a translation request might consist of a series of prompts for the user to adjust segmentation boundaries or un-skew lines. This process significantly increases the required level of user engagement, but can nevertheless be useful if absolute accuracy is needed.

Automatic language detection for recognition. Users may not be sure about the language of the text they are attempting to translate. A system-level approach to language detection is to attempt recognition in multiple languages in parallel. For each attempted language, the confidences for the recognized characters are multiplied together. The product of the confidences corresponds roughly to the likelihood for a particular language to match the input. The detected language is defined as the one yielding the highest aggregate confidence level. Language detection might also be handled internally by the OCR engine, for example by detecting certain characters (with high confidence) that are unique to specific languages. An experimental language detection feature is currently under development for Tesseract 3.01.

Exploit parallelism. There are many opportunities for parallel processing in the system. One of the most expensive tasks in a request is preprocessing the image on the phone before sending it to the server. Since the application is centered around interactive sessions, it may be useful to optimistically begin preprocessing in the background while the user is making a cropping selection. This way, there is a chance that some of the user's cropped area will have already been processed upon confirmation of the selection.

Opportunities for parallelism also exist between the web service and mobile device. The client can split the source image into blocks that are dynamically provided to the server for processing in parallel. The application might also begin other preprocessing steps, such as computing the integral image, while waiting for results from the server or transferring data.

Make use of user-generated data. Manual classification provides human feedback on results from the recognition module. Specifically, user-selected alternatives are expected to correctly identify the characters being recognized in the source image. Therefore, they can be prepared and iteratively fed back into the OCR system as training data. This can work by having the application send manually classified characters and their associated segmentation boundaries to the recognition server when making a re-translation requests. The server would then

extract the portions of the source images indicated by the bounding boxes and annotate them using the selected alternatives. The annotated images can then optionally be filtered by humans for quality before being used as supplemental training data. This amounts to a semi-supervised process for adapting the recognizer. Plauché and Nallasamy [28] proposed a similar method for speech recognition adaptation.

Optimization of bandwidth use. Bandwidth use can be reduced by experimentally determining the optimal number of alternative characters to send for a given confidence level. Additionally, bandwidth used for re-recognition requests can be minimized by relying on already-transferred data. The application currently makes a new, standalone request per manual segmentation or manual thresholding task. Since the server already has the original input image, it is sufficient to send only the parameters defining these manual corrections. Follow up requests would be reduced to only consist of the manually specified segmentation coordinates and thresholding value.

Offline mode. Feedback gathered from the public suggests that an offline operating mode would be helpful, especially in avoiding heavy roaming costs while using the application abroad. An offline application will have to address the issue of large trained data files. This can be addressed by supporting individual language packages that are installed by the user. A similar strategy can be applied for local translation modules.

VII. Conclusion

This paper presented MIRATS, a system for performing text recognition and translation on images captured by the N900 smartphone. The system uses vignetting correction to overcome the platform's camera limitations and a server-based OCR service to lower response times.

Mobile OCR systems must adapt to the unique conditions under which they operate. Specifically, mobile image analysis software is subject to limited computing performance and lower-quality input images. Fortunately, mobile applications benefit from short usage sessions characterized by high user engagement. MIRATS minimizes the platform's drawbacks and takes advantage of its strengths by introducing human assistance into OCR, a process that has traditionally been automated.

Experiments show that MIRATS performs well for individual character recognition. Phrase recognition remains short of a satisfactory level, and addressing this issue will likely require additional work on the OCR engine itself. Engine improvements aside, overall accuracy can be significantly improved by using manual classification, segmentation, and thresholding throughout the recognition process.

Users are unlikely to attempt large-scale text recognition on portable devices and thus full automation is not a strict requirement for a mobile OCR solution. Introducing human assistance into the OCR process removes the all-or-nothing nature of fully automated pipelines. This approach can be extended and applied to other types of recognition software. For example, an object recognition application might present a series of guided questions about the target object being identified in order to tend the recognition towards the right answer. Doing so reduces the risk of an inflexible error in which the correct object was almost determined but discarded in favor of a similar, but incorrect object. The opportunity to interactively and successively improve results is helpful for tasks that are sensitive to even small errors in recognition. The translation component of MIRATS is an example of such a task.

References

[1] Y. Cui and V. Roto, "How people use the web on mobile devices," in *WWW '08 Proc. 17th Int. Conf. World Wide Web*. Beijing, China, 2008, pp. 905-914.

[2] P. Clark and M. Mirmehdi, "Recognising text in real scenes," *International Journal on Document Analysis and Recognition*, vol. 4, no. 4, pp. 243-257, 2002.

[3] R. Lienhart and A. Wernicke, "Localizing and Segmenting Text in Images and Videos," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 256-268, April, 2002.

[4] T. de Campos, B. Babu, and M. Varma, "Character recognition in natural images," in *VISAPP 2009*, Lisboa, Portugal, 2009, pp. 92-103.

[5] D. Doermann, J. Liang, and H. Li, "Progress in camera-based document image analysis," in *ICDAR '03 Proc. 7th Int. Conf. Document Analysis and Recognition*, Edinburgh, UK, 2003, pp. 606-616.

[6] J. Liang, D. Doermann, and H. Li, "Camera-based analysis of text and documents: a survey," *International Journal on Document Analysis and Recognition*, vol. 7, no. 2, pp. 84-104, July, 2005.

[7] M. Koga et al., "Camera-based kanji OCR for mobile-phones: practical issues," in *Proc. 8th Int. Conf. Document Analysis and Recognition*, Seoul, Korea, 2005, pp. 635-639.

[8] K. S. Bae, K. K. Kim, Y.G. Chung, and W. P. Yu, "Character recognition system for cellular phone with camera," in *COMPSAC '05 29th Annu. Int. Computer Software and Applications Conf.*, Edinburgh, UK, 2005, pp. 539-544.

[9] M. Laine and O. Nevalainen, "A standalone OCR system for mobile cameraphones," in *PIMRC '06 17th Annu. IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications*, Helsinki, Finland, 2006, pp. 1-5.

[10] S. Senda, K. Nishiyama, T. Asahi, and K. Yamada, "Camera-typing interface for ubiquitous information services," in *PERCOM '04 Proc. 2nd IEEE Annu. Conf. Pervasive Computing and Communications*, Orlando, FL, 2004, pp. 366-370.

[11] A. Joshi et al., "OCRdroid: A Framework to Digitize Text Using Mobile Phones," unpublished.

[12] T. M. Breuel, "The OCRopus open source OCR system," in *Proc. IS&T/SPIE 20th Annu. Symp.*, 2008.

[13] R. Smith, "An overview of the Tesseract OCR engine." in *ICDAR '07 Proc. 9th Int. Conf. Document Analysis and Recognition*, Curitiba, Brazil, 2007, pp. 629-633.

[14] R. Smith, D. Antonova, and D. Lee, "MOCR '09 Adapting the Tesseract open source OCR engine for multilingual OCR," in *Proc. Int. Workshop Multilingual OCR*, Barcelona, 2009.

[15] R. A. Wilkinson, M. D. Garris, and J. Geist, "Machine-assisted human classification of segmented characters for OCR testing and training," in *IS&T/SPIE Symp. Electronic Imaging: Science and Technology,*, San Jose, CA, 1993, pp. 44-54.

[16] D. Blostein, E. Lank, A. Rose, R. Zanibbi, and W. P. Yu, "User interfaces for on-line diagram recognition," in *GREC '01 Selected Papers from 4th Int. Workshop Graphics Recognition Algorithms and Applications*, Kingston, Canada, 2001, pp. 92-103.

[17] E. Lank, J. Thorley, and S. J. Chen, "An interactive system for recognizing hand drawn UML diagrams," in *CASCON '00 Proc. 2000 Conf. Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, 2000, pp. 7.

[18] S. Smithies, K. Novins, and J. Arvo, "Equation entry and editing via handwriting and gesture recognition," *Behaviour and Information Technology*, vol. 20, no. 1, pp. 53-67, January, 2001.

[19] R. Zanibbi, K. Novins, J. Arvo, and K. Zanibbi, "Aiding manipulation of handwritten mathematical expressions through style-preserving morphs," in *GRIN '01 Proc. Graphics Interface 2001*, Ottawa, Canada, 2001, pp. 127-134.

[20] J. Mankoff, G. D. Abowd, and S. E. Hudson, "OOPS: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces," *Computers & Graphics*, vol. 24, no. 6, pp. 819-834, December 2000.

[21] Fragoso et al., "TranslatAR: a mobile augmented reality translator on the Nokia N900," unpublished.

[22] Q. Huo, Y. Ge, and Z. Feng, "High performance Chinese OCR based on Gabor features, discriminative feature extraction and model training," in *Proc. 2001 IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Salt Lake City, UT, 2001, pp. 1517-1520.

[23] Z. Lu et al., "A robust, language-independent OCR system," in *Proc. 27th AIPR Workshop: Advances in Computer-Assisted Recognition*, Washington, DC, 1998, pp. 96-104.

[24] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, pp. 62-66, January, 1979.

[25] J. Sauvola and M. Pietikainen, "Adaptive document image binarization," *Pattern Recognition*, vol. 33, no. 2, pp. 225-236, 2000.

[26] F. Shafait, D. Keysers, and T. M. Breuel, "Efficient implementation of local adaptive thresholding techniques using integral images," in *DPR '08 Proc. 15th Document Recognition and Retrieval Conf.*, San Jose, CA, 2008.

[27] F. Shafait, D. Keysers, and T. M. Breuel, "Efficient implementation of local adaptive thresholding techniques using integral images," in *DPR '08 Proc. 15th Document Recognition and Retrieval Conf.*, San Jose, CA, 2008.

[28] M. Plauché and U. Nallasamy, "Speech interfaces for equitable access to information technology," *Information Technologies and Int. Development*, vol. 4, no. 1, pp. 69-86, 2007.