OCRdroid: A Framework to Digitize Text on Smart Phones

Mi Zhang[§], Anand Joshi[†], Ritesh Kadmawala[†], Karthik Dantu[†], Sameera Poduri[†], Gaurav S. Sukhatme[†§]
[†]Computer Science Department, [§]Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089, USA
{mizhang, ananddjo, kadmawal, dantu, sameera, gaurav}@usc.edu

***Abstract*— As demand grows for mobile phone applications, research in optical character recognition, a technology well developed for scanned documents, is shifting focus to the recognition of text embedded in digital photographs. In this paper, we present OCRdroid, a generic framework for developing OCR-based applications on mobile phones. OCRdroid combines a lightweight image preprocessing suite installed inside the mobile phone and an OCR engine connected to a backend server. We demonstrate the power and functionality of this framework by implementing two applications called PocketPal and PocketReader based on OCRdroid on HTC Android G1 mobile phone. Initial evaluations of these pilot experiments demonstrate the potential of using OCRdroid framework for real-world OCR-based mobile applications.**

I. Introduction

Optical character recognition (OCR) is a powerful tool for bringing information from our analog lives into the increasingly digital world. This technology has long seen use in building digital libraries, recognizing text from natural scenes, understanding hand-written office forms, and etc. By applying OCR technologies, scanned or camera-captured documents are converted into machine editable soft copies that can be edited, searched, reproduced and transported with ease [15]. Our interest is in enabling OCR on mobile phones.

Mobile phones are one of the most commonly used electronic devices today. Commodity mobile phones with powerful microprocessors (above 500MHz), high-resolution cameras (above 2megapixels), and a variety of embedded sensors (accelerometers, compass, GPS) are widely deployed and becoming ubiquitous today. By fully exploiting these advantages, mobile phones are becoming powerful portable computing platforms, and therefore can process computing-intensive programs in real time. In particular, some modern mobile devices can use pictures of barcodes to look up detailed information about a product's ratings, price, and reviews. Some mobile phones with business card reader application installed facilitates users to transfer contact information directly from business cards. This allows business people to carry only one personalized card with no physical copies to share.

In this paper, we explore the possibility to build a generic framework for developing OCR-based applications on mobile phones. We believe this mobile solution to extract information from physical world is a good match for future trend [18]. However, camera-captured documents have some drawbacks. They suffer a lot from focus loss, uneven document lighting, and geometrical distortions (e.g., text skew, bad orientation, and text misalignment) [17]. Moreover, since the system is running on a mobile phone, real time response is also a critical challenge. We have developed a framework called OCRdroid. It utilizes embedded sensors (orientation sensor, camera) combined with image preprocessing suite built on top of the hardware to address those issues mentioned above. We have also evaluated our OCRdroid framework by implementing two applications called PocketPal and PocketReader based on this framework. Our experimental results demonstrate the OCRdroid framework is feasible for building real-world OCR-based mobile applications. The main contributions of this work are:

- An algorithm to detect text misalignment in real time, and guide users to align the text properly. To the best of our knowledge, it is the first trial on detecting text misalignment on mobile phones.
- Heuristics to identify the tilts by using orientation sensors to prevent users from taking pictures if the mobile phone is not properly oriented and positioned.
- An auto-rotation algorithm to correct skewness of text.
- An application called PocketPal, which can extract text and digits on receipts and store them in a shopping history database inside the phone.
- An application called PocketReader, which provides a text-to-speech interface to read text extracted from any text sources (magazines, newspapers, and etc).

The rest of this paper is organized as follows. Section II discusses work related to OCR and image processing on mobile phones. Section III describes the OCRdroid framework and PocketPal and PocketReader applications. Design considerations of OCRdroid are described in detail in Section IV. The system architecture and implementation are presented in Section V, with experiments and evaluations in Section VI. Section VII discusses limitations and future work, and Section VIII summarizes our contributions.

II. Related Work

There are currently several commercially available OCR systems on the market today such as ABBYY FineReader, OmniPage, and Microsoft Office Document Imaging. In addition, the research and opensource communities also offer comparable systems like GOCR [3], OCRAD [6], Tesseract[10] and OCROPUS [8]. ABBYY also provides a Mobile OCR Engine [1] for the mobile phones which is claimed to provide real time processing with a very high accuracy.

Several academic projects and commercial products have tried to use mobile phone cameras to build interesting applications. In [26], the authors presented a mobile sensing system that analyzes images of air sensors taken on cell phones and extracts indoor air pollution information by comparing the sensor to a calibrated color chart using image processing and computer vision techniques. However, all the processing in this system is performed at the backend server and not in real time. In [15], the authors designed an image preprocessing suite on top of OCR engine to improve the accuracy of text recognition in natural scene images. Again, all the processing is implemented at the back end server with no implementation on the mobile device. Nowadays some mobile phones are equipped with business card reader application which facilitates users to store contact information from business cards directly on their mobile phones [27]. Also in [14], authors have discussed about a mobile application to capture barcodes of any item and get detailed information about its ratings, price, and reviews.

III. OCRdroid Description and Applications

OCRdroid is a generic framework for developing OCR-based applications on mobile phones. This framework not only supports the baseline character recognition functionality, but also provides an interactive interface by taking advantage of high-resolution camera and embedded phone sensors. This interface enriches the interactivity between users and devices, and as a result, successfully guides users step by step to take good-quality pictures.

In this section, we describe two applications called PocketPal and PocketReader we have implemented on the OCRdroid framework. Several potential applications that may emerge on OCRdroid are also covered.

A. Pocket Pal

PocketPal is a personal receipt management tool installed in one's mobile phone. It helps users extract information from receipts and keep track of their shopping histories digitally in a mobile environment. Imagine a user, Fiona, is shopping in a local shopping mall. Unfortunately, she forgets what she bought last weekend and hasnt brought her shopping list with her. Fiona takes out her mobile phone and starts the PocketPal application. PocketPal maintains her shopping history in a chronological order for the past 2 months. She looks over what she bought last week and then decides what to buy this time. After Fiona finishes shopping, she takes a good-quality picture of the receipt under the guidance of PocketPal via both audio and visual notifications. All the information on the receipt is translated into machine editable texts and digits. Finally, this shopping record is tagged, classified and stored inside the phone for future reference. PocketPal also checks the total and reminds users in an unobtrusive way if the spending increases one's monthly budget. We have studied a lot of receipts with different formats. Some of the most common items are:

- Date
- Time
- Shop name
- Shop location
- Contact phone number
- Shop website
- Name of each item bought
- Price of each item bought
- Amount of tax
- Total amount of the purchase

PocketPal can recognize all of these items, categorize them into different categories, and display to the users.

B. Pocket Reader

PocketReader is a personal mobile screen reader that combines the OCR capability and a text-to-speech interface. PocketReader allows users to take pictures of any text source (magazines, newspapers, and etc). It identifies and interprets the text contents and then reads them out. There are many potential applications using PocketReader. For example, users can quickly capture some of the hot news from the newspaper and let PocketReader read them out if users do not have time to read them carefully. PocketReader can also act as a form of assistive technology potentially useful to people who are blind, visually impaired, and learning disabled. A visually impaired person, trying to read a newspaper or a description of a medicine, can ask PocketReader to read it loud for him.

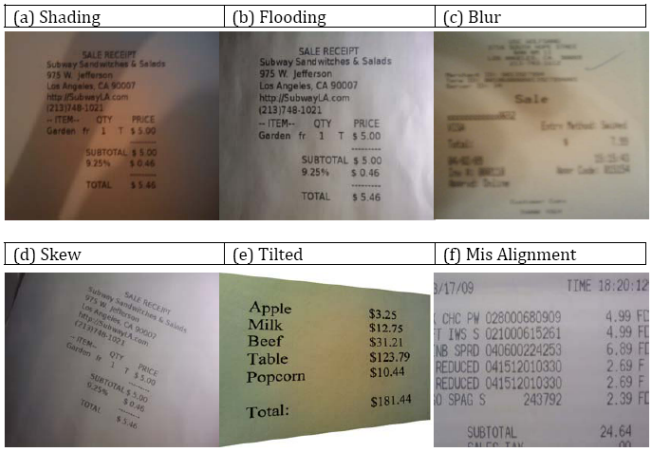
C. Other Potential Applications

It might be feasible to apply OCRdroid to digitize physical sticky notes [18] to build a mobile memo and reminder. If the message contains important timing information, such as a meeting schedule, the system can tag this event and set an alarm to remind user of this event. In addition, OCRDroid framework can be combined with a natural language translator to diminish the language barrier faced by tourists. As a result, tourists can take pictures of public signage and have the same access to information as locals [15].

IV. Design Considerations

OCR systems have been under developed in academia and industry since the 1950s. Such systems use knowledge-based and statistical pattern recognition techniques to transform scanned or photographed text images into machine-editable text files. Normally, a complete OCR process includes 5 main steps [13]: (1) noise attenuation, (2) image binarization (to black and white), (3) text segmentation, (4) character recognition, and (5) post-processing, such as spell checking. These steps are very effective when applied to document text, which when collected by a scanner, is generally aligned and has clear contrast between text and its uniform background. However, taking pictures from a portable camera, especially the one embedded inside a mobile device, may leads to various artifacts in the images and as a result, causes even the best available OCR engine to fail. Problems include uneven document lighting, perception distortion, text skew, misalignment etc. Some of these issues are illustrated in Figure 1. In addition, since the system is installed on mobile devices, real time response is another critical issue that needs

to be considered. Among the issues mentioned above, some of them exhibit inherent tradeoffs and must be addressed in a manner that suits our applications. This section presents a pertinent range of design issues and tradeoffs, and discusses proposed approaches applicable to our OCRdroid framework.



1: Different issues arising in camera-captured documents: (a) shading (b) flooding (c) blur (d) skew (e) tilted (f) misalignment

A. Lighting Condition

Issue: An embedded camera inside the mobile phone has far less control of lighting conditions than scanners. Uneven lighting is common, due to both the physical environment (shadows, reflection, fluorescents) and uneven response from the devices. Further complications occur when trying to use artificial light, i.e. flash, which results in light flooding.

Proposed Approach: Binarization has long been recognized as a standard method to solve the lightning issue. The goal of binarization process is to classify image pixels from the given input grayscale or color document into either foreground (text) or background and as a result, reduces the candidate text region to be processed by later processing steps. In general, the binarization process for grayscale documents can be grouped into two broad categories: global binarization, and local binarization [24]. Global binarization methods like Otsu's algorithm [20] try to find a single threshold value for the whole document. Each pixel is then assigned to either foreground or background based on its grey value. Global binarization methods are very fast and they give good results for typical scanned documents. However, if the illumination over the document is not uniform, for instance, in the case of camera-captured documents, global binarization methods tend to produce marginal noise along the page borders. Local binarization methods, such as Niblack's algorithm [19], and Sauvola's algorithm [22], compute thresholds individually for each pixel using information from the local neighborhood of that pixel. Local algorithms are usually able to achieve good results even on severely degraded documents with uneven lightning conditions. However, they are often slow since computation of image features from the local neighborhood is to be done for each image pixel. In this work, in order to handle uneven lightning conditions for our camera-captured documents, we adopt Sauvola's local binarization algorithm. We also have tried Background Surface Thresholding algorithm in [23]. However, based on our experiments, we found Sauvolas's algorithm worked better and faster.

B. Text Skew

Issue: When OCR input is taken from a hand-held camera or other imaging device whose perspective is not fixed like a scanner, text lines may get skewed from their original orientation [13]. Based on our experiments, feeding such a rotated image to our OCR engine produces extremely poor results.

Proposed Approach: A skew detection process is needed before calling the recognition engine. If any skew is detected, an auto-rotation procedure is performed to correct the skew before processing text further. While identifying the algorithm to be used for skew detection, we found that many approaches, such as the one mentioned in [13], are based on the assumptions that documents have s et margins. However, this assumption does not always hold in our application. In addition, traditional methods based on morphological operations and projection methods are extremely slow and tend to fail in presence of camera-captured images. In this work, we choose a more robust approach based on Branch-and-Bound text line finding algorithm (RAST algorithm) [25] for skew detection and auto-rotation. The basic idea of this algorithm is to identify each line independently and use the slope of the best scoring line as the skew angle for the entire text segment. After detecting the skew angle, rotation is performed accordingly. Based on our experiments, we found this algorithm to be highly robust and extremely efficient and fast. However, it suffered from one minor limitation in the sense that it failed to detect rotation greater than 30°. We also tried an alternate approach, which could detect any angle of skew up to 90°. However, this approach was based on presence of some sort of cross on the image. Due to the lack of extensibility, we decided to stick with RAST algorithm.

C. Perception Distortion (Tilt)

Issue: Perception distortion occurs when the text plane is not parallel to the imaging plane. It happens a lot if using a hand-held camera to take pictures. The effect is characters farther away look smaller and distorted, and parallel-line assumptions no longer hold in the image [17]. From our experience, small to mild perception distortion causes significant degradation in performance of our OCR engine.

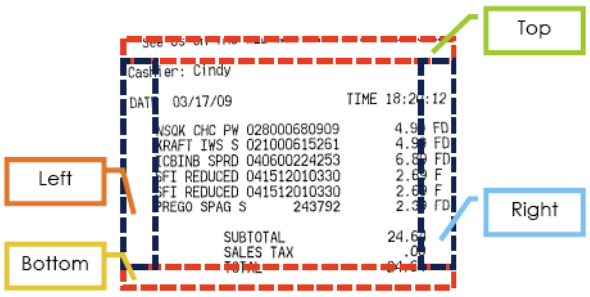
Proposed Approach: Instead of applying image processing techniques to correct the distortion, we take advantages of the embedded orientation sensors to measure the tilt of the phone. Users are prevented from taking pictures if the camera is tilted to some extent, and as a result, the chances of perception distortion are reduced considerably. We also

consider the situations where the text source itself is titled. In this case, we have provided users a facility to calibrate the phone to any orientation so that the imaging plane is parallel to the text plane. For example, if one user wants to take a picture on a poster attached on the wall, he can first calibrate the phone with the imaging plane parallel to the wall surface, and then take the picture.

D. Misalignment

Issue: Text misalignment happens when the camera screen covers a partial text region, in which irregular shapes of the text characters are captured and imported as inputs to the OCR engine. Figure 1(f) shows an example of a misaligned image. Misalignment issue occurs a lot especially when people casually take their pictures. Our experiment results indicate that the OCR result is significantly affected by misalignment. Also misaligned images may lead to loss of important data.

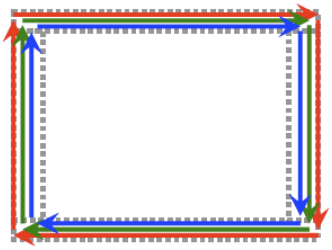
Proposed Approach: Firstly, we define the camera-captured image is misaligned if any of the four screen borders of the phone cuts through a single line of text, either horizontally, or vertically. Based on this definition, in order to detect misalignment, we define a 10-pixel wide margin along each border as depicted in Figure 2.



2: Definition of four 10-pixel wide margins

If any foreground pixel is detected within the margins, there is high probability that the text is being cut. Therefore, we conclude that the image is misaligned. Considering issues of imperfect lighting condition, a binarization pre-processing step is necessary before we can infer whether we have detected any black dot or not. We have tried Otsu's algorithm [20] and another fast global binarization method from [16]. However, they provide poor results due to shadings. Finally, we adopt Sauvola's algorithm [22]. We choose this local algorithm as the basis of our algorithm for two reasons. (1) Images captured from camera suffer from illumination variation and blur. Global binarization algorithms specifically designed for flatbed scanners fail to handle the local subtleties in the images and thus produce poor results in such situations. (2) Our alignment-checking algorithm is by nature a local algorithm since only pixels within the four margins need to be checked. However, there are two drawbacks if we directly run Sauvola's algorithm. (1) Running Sauvola's algorithm is relatively time-consuming. The reason is obvious since we need to calculate the mean and standard deviation of each pixel within the four margins. Thus, a fast version of Sauvola's algorithm is needed. (2) Sauvola's algorithm is too sensitive to noise. Even a single noise black dot would ruin our algorithm to provide wrong judgments. Therefore, a more robust algorithm is needed to reduce the noise.

Our newly designed alignment-checking algorithm is based on the fast variant of Sauvola's local binarization algorithm described in [12], and is further optimized for our application. We run our optimized Sauvola's algorithm within four margins in a Round-Robin manner as depicted in Figure 3.



3: The route to run Sauvola's binarization algorithm

Specifically, we first go around the outermost circle in red. If no foreground pixel is detected, we continue on the green route. By following this approach we can detect the misalignment faster and quit this computing-intensive process as soon as possible. Once a black pixel is detected, it is necessary to verify whether it is a true pixel belonging to a text character or just some random noise. Figure 4 demonstrates a perfectly aligned receipt with some noise dots located within the top margin, which are circled by a red ellipse. To judge whether it is a noise dot or not, whenever a black dot is detected, we then check all its neighbors within a local W x W box. The parameter W(window size) needs to be carefully chosen to match the resolution of the embedded camera. In order to meet the real time requirement

4: A perfectly aligned receipt with noise dots located within the top margin and circled by a red ellipse. A WxW local box is defined to help filter out the noise dots

and balance speed and accuracy, we do not rerun the fast

Sauvola's local binarization algorithm for every pixel within this local box to determine their thresholds individually. Instead, we assume all the neighbor pixels within the local box have the same threshold. This assumption is supported by statistical analysis, where text regions normally have concentrated distribution of black pixels while noise dots are sparsely distributed. If more than 10% of its neighbor pixels inside the local box are black dots, then we conclude that black dot being verified is a true pixel of a text character. Based on our experiments, this newly designed alignment-checking algorithm takes less than 6 seconds and boasts an accuracy of around 96% under normal lighting condition.

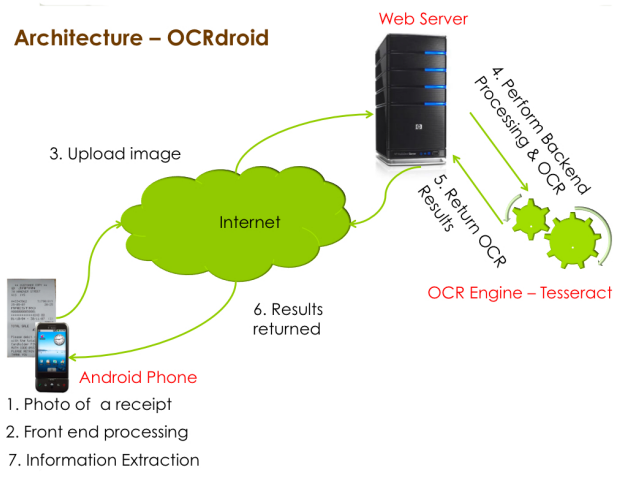
E. Blur (Out Of Focus)

Issue: Since many digital cameras are designed to operate over a variety of distances, focus becomes a significant factor. Sharp edge response is required for the best character segmentation and recognition [17]. At short distances and large apertures, even slight perspective changes can cause uneven focus.

Proposed Approach: We use the AutoFocus API provided by Android SDK to avoid any blurring seen in out of focus images. Whenever we start the application, a camera autofocus handler object is declared and initiated so that the camera itself can focus on the text sources automatically.

V. System Architecture and Implementation

Figure 5 presents the client-server architecture of OCR-droid framework. The software installed on the phone checks the camera orientation, performs alignment checking, and guides the user in an interactive way to take a good-quality picture. The mobile phone plays the role as a client sending the picture as a request to the back-end server and also doing some preprocessing to ensure good OCR results. The computing-intensive image preprocessing steps and character recognition process are performed at the server. Finally, the text results are sent back to client.



5: Overview of OCRdroid framework architecture

The OCRdroid client is currently developed on HTC G1 mobile phone supported by Google's Android OS platform. However, it can be extended with minimal effort to any other platform powered phones supporting orientation sensors and an embedded camera. The OCRdroid application server is an integration of Tesseract OCR engine from Google [10] and Apache (2.0) web server. We have implemented 2 applications called PocketPal and PocketReader. A series of screenshots and a demo video of these applications can be found at our project website [7]. We present the implementation details of both phone client and server next.

A. Phone Client

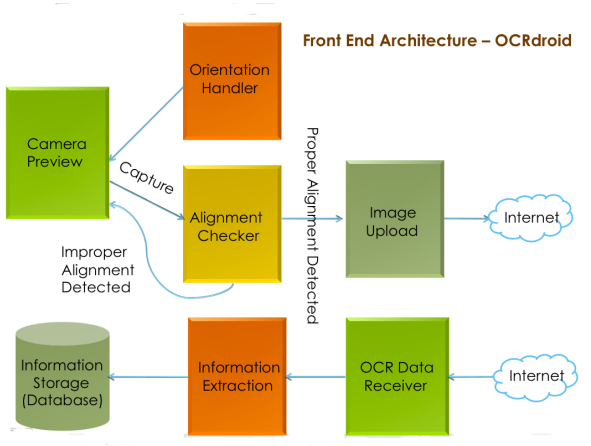
We developed the OCRdroid phone client application using Google's Android software stack (version 1.1) in the Windows environment. Android is the first truly open and comprehensive platform designed specifically for mobile devices. Compared to other mobile OS platforms like Nokia S60, and Windows Mobile, Android is more efficient and easy to program. Our client software requires access to phone camera, embedded sensors, background services, network services, relational database and file-system on the phone. The Android SDK (version 1.1) provides all the necessary APIs for this.

Compared to desktop or notebook computers, mobile devices have relatively low processing power and limited storage capacity. Meanwhile, mobile phone users always expect instantaneous response when interacting with their handsets. Our OCRdroid mobile application is designed to minimize the processing time so as to provide real time response. We enforce real time responsiveness by adopting many strategies, including:

- Designing computationally efficient algorithms to reduce processing time.
- Spawning separate threads and background services for computation-intensive workloads to keep the GUI always responsive.
- Using compressed image format to store and transmit image data.

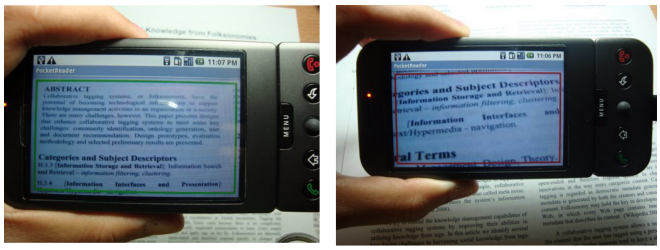
Figure 6 presents the architectural diagram for the client software.

The client software consists of 7 functional modules. By default, the client is in the idle state. When the user initiates the application to take a new picture, the client starts up a camera preview. The preview implements a callback function to retrieve the data from the underlying 3.2 mega-pixel camera, which satisfies the 300dpi resolution requirement of the OCR engine. Client also implements an orientation handler running inside a background service thread, which is responsible for preventing users from tilting the camera beyond a threshold while taking a picture. It periodically polls the orientation sensors to get the pitch and roll values. A threshold range of [-10, 10] is set for both sensors. If the phone is tilted more than the allowed threshold, the client doesn't allow the user to capture any image. The following figure shows a screenshot indicating how the tilt detection module works. The image on the right shows a case where



6: OCRdroid Client(Phone) Architecture

user has somewhat tilted the phone. As a result, a red colored bounding box is displayed, indicating to the user that the camera is not properly oriented. As soon as the orientation is perfect, the box turns to green and the user is allowed to capture an image

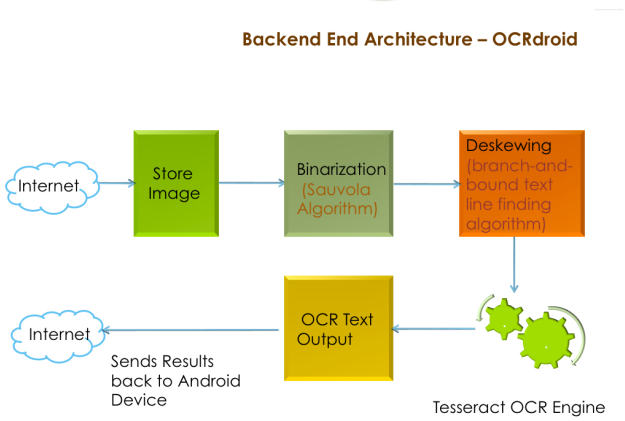


7: Screenshot demonstrating tilt detection capability of OCRDroid using embedded orientation sensor. A green (red) rectangle indicates correct (incorrect) tilt.

Once the user takes a picture, the misalignment detection module is initiated to verify if the text source is properly aligned. In case of a misaligned image, the client pops up an appropriate notification to instruct the user at which border any misalignment is detected. Once the text source is properly aligned and the user clicks the ”Perform OCR” button, the client transports the image to the server over a HTTP connection. An OCR data receiver listens on the connection and passes the OCR result sent from the remote server to information extraction module, which extracts corresponding information and parses it into different categories. Finally, the OCR result is displayed on the screen and automatically stored inside the database for future reference.

B. Back-End Server

OCRdroid application server is an integration of an OCR engine and a web server. Figure 8 presents the architectural diagram for the backend server. Once images are received at Apache (2.0) web server, shell scripts are called through PHP (5.0) in sequence to perform binarization and image rotation (if any skew is detected). Here at each step, conversion of image is done using a popular open source tool called ImageMagick [4]. Once the image preprocessing procedures are completed, the intermediate result is fed to the OCR engine to perform text segmentation and character recognition.



8: Software architecture of OCRdroid backend

There are many open source as well as commercial OCR engines that are available, each with its own unique strengths and weaknesses. A detailed list is available at [11] and [5]. We tested some of the open source OCR engines such as OCRAD [6], Tesseract [10], GOCR [3] and Simple OCR [9]. Based on our experiments, we found that Tesseract gave the best results, as compared to others. We also tried a complete document analysis tool called OCRopus. It performs document layout analysis and uses Tesseract as its OCR engine for the character recognition.However during our experiments, we did not find any significant improvements in results due to the extra preprocessing step induced by OCROPUS. In addition, it gave rise to one additional complication in our PocketPal application because of its inherent multi-column support. It actually identified receipts to be in 2 column format and as a result, it first displayed name of all items followed by their corresponding prices. This required us to carry out extra operations to identify the price of each item on the receipt. Therefore, we choose Tesseract as our OCR engine for our PocketPal application. Our framework can be easily customized to use any Document Layout Analysis tool such as OCROPUS ,instead of only the OCR engine if the desired application will be dealing with complex documents with illustrations or multiple-columns

Once the OCR process finishes, OCRdroid server stores the text content and responds to the client side with an OK message. After the client receives this message, it understands that the OCR on the server has been performed successfully. Therefore, it sends a request to the web server asking for the stored OCR result. Finally, OCRdroid server sends the text results back as soon as the request is received.

VI. Experiments and Evaluation

We evaluate the OCRdroid framework by implementing PocketPal and PocketReader applications. The OCR accuracy and timing performance are of our interest. We start by describing the text input sources and defining performance metrics. And then we present the results of a set of preprocessing steps and detailed performance analysis.

A. Test Corpus

The system was tested on 10 distinct black and white images without illustrations. Tests were performed under distinct lighting conditions. All the test images were taken with HTC G1's embedded 3.2 megapixel camera. They can be found at our project website [7].

B. Performance Metrics

In order to measure the accuracy of OCR, we use two metrics proposed by the Information Science Research Institute at UNLV for the Fifth Annual Test of OCR Accuracy [21]

1) Character Accuracy: This metric measures the effort required by a human editor to correct the OCR-generated text. Specifically, we compute the minimum number of edit operations (character insertions, deletions, and substitutions) needed to fully correct the text. We refer to this quantity as the number of errors made by the OCR system. The *character accuracy* is defined as:

$$\frac{\#characters - (\#errors)}{\#characters} \quad (1)$$

2) Word Accuracy: In a text retrieval application, the correct recognition of words is much more important than the correct recognition of numbers or punctuations. We define a word to be any sequence of one or more letters. If m out of n words are recognized correctly, the *word accuracy* is m/n. Since full-text searching is almost always performed on a case-insensitive basis, we consider a word to be correctly recognized even if one or more letters of the generated word are in the wrong case (e.g., "transPortatIon").

In addition, real time response is another very important metric for mobile applications. We evaluate timing performance in terms of total processing time taken by each of the preprocessing steps.

C. Experimental Results and Analysis

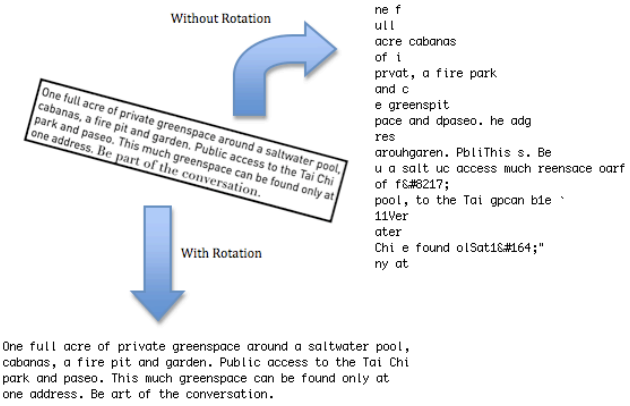
In this section, we give both qualitative and quantitative analysis of the performance improvement brought by each of our preprocessing stages.

1) Binarization: The effectiveness of binarization algorithm heavily depends upon the lighting conditions when image is captured. To measure the performance, we first define 3 lighting conditions. Then we perform experiments under these lighting conditions.

- Normal lighting condition: This refers to situations when images are captured outdoors in the presence of sunlight or indoors in an adequately lit room.
- Poor lighting condition: This describes situations when users take images outdoors during night or capture images in rooms which have very dim lighting.
- Flooding condition: can arise either outdoors during daytime, or indoors in a very dimly lit room. In such cases the source of light is very much focused on a particular portion of the image, whereas the remaining portion is dark. In such situation, sometimes binarization leads to mixing of the text with the background, which in turns lead to poor results.

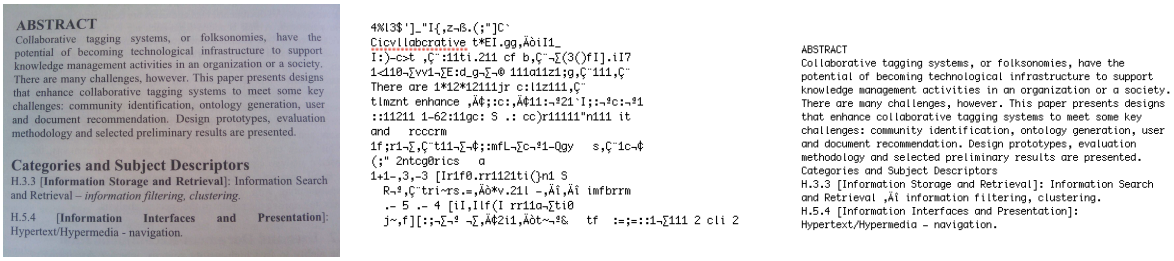
The Figure 9 shows one of the image from our test corpus captured under normal lighting conditions as well as the corresponding OCR generated text with and without our Binarization Algorithm.

2) Skew Detection and Auto-Rotation: In order to examine the performance of our skew detection and auto-rotation algorithm, we rotated all the images in our text corpus by 5°, 10°, 15°, 25°, and 35°in both clockwise and counterclockwise directions using Gimp Image Processing ToolBox [2]. These rotated images are then passed to the OCR engine. Figure 10 shows one image with a rotation angle of 15°as well as the corresponding OCR results with and without the skew detection and auto-rotation algorithm. As expected, without auto-rotation, the OCR engine gives very poor results.



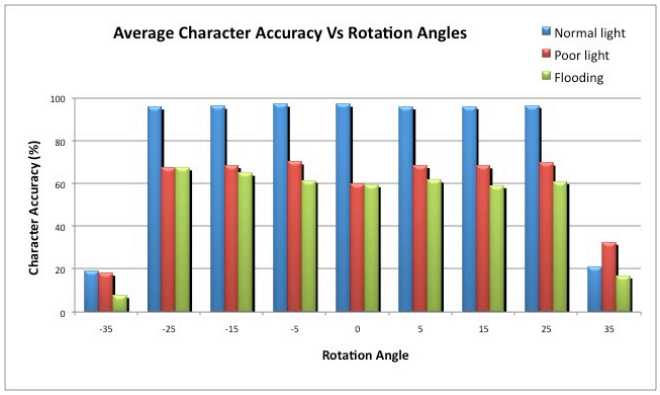
10: Comparision of OCR results with and without Skew-Detection and Auto-Rotation Module

The Figure 11 and 12 demonstrates the performance of our system in terms of the two performance metrics mentioned above- character accuracy and word accuracy respectively, for all the images in our text corpus at different rotation angles from -35 to 35 under three different lighting conditions.

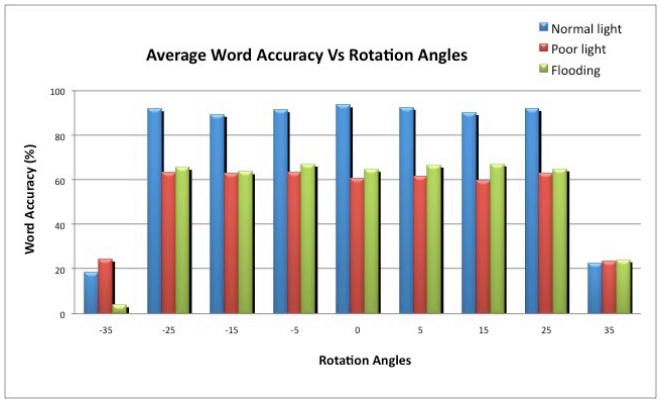


(a) Text Image (b) Without Binarization (c) With Binarization

9: Comparision of OCR results with and without Binarization Module



11: Average Character Accuracy of the OCR generated Text at Different Rotation angles.



12: Average Word Accuracy of the OCR generated Text at Different Rotation angles.

Type of lighting Conditions	Character Accuracy Without Binarization(%)	Character Accuracy With Binarization(%)
Normal	46.09	96.94
Poor	9.09	59.59
Flooding	27.74	59.33

I: Comparison of OCR Accuracy With and Without Binarization at 0° rotation for different lighting Conditions

Along with measuring the performance of our system, we also calculated the character and word accuracies for all the images in our test corpus in absence of the binarization and rotation module. At 0° rotation there is no impact of the skew detection module on the images and hence if there is any performance improvement in the OCR results, it can be directly attributed to the binarization module. The comparison between the average OCR accuracy for the Tesseract OCR engine at 0° rotation, with and without our binarization module for our text corpus is shown in the Table I. Also, during our experiments, we found that the tesseract OCR engine fails to produce any meaningful results in presence of more than 5°of skew in the input image which shows that our skew detection module is quite important if we are to allow users the freedom to capture the images without worrying much about the angle between the camera phone and the document which is to be digitized.

One interesting result we found during our experiments was that our skew detection module completely fails to correct images rotated more than 35°in either clockwise or counter-clockwise directions. However it is reasonable to believe that generally users wouldn't take images at such high degree of rotation and hence this won't be a much of a problem for real time scenarios.

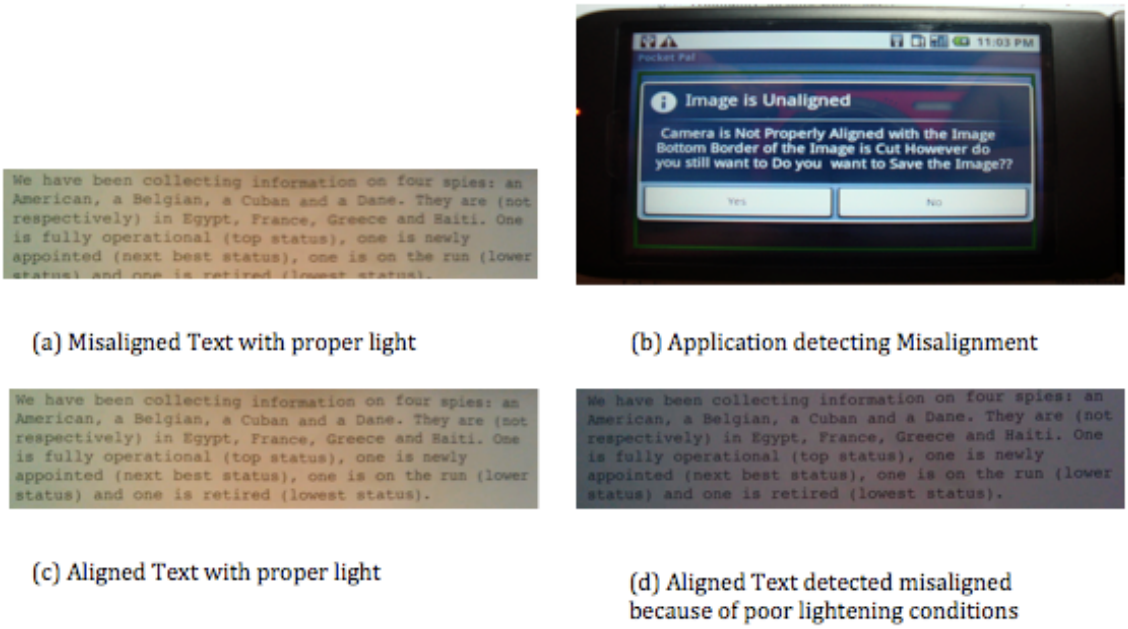
The graph also demonstrates that the lighting conditions severely affect the OCR accuracy. The images taken under poor lighting conditions tends to have many dark regions which leads to following two problems

- During the binarization module, some of the background region is recognized as foreground pixels thus leading to some garbage characters in the OCR result
- The text in the dark region of the image tends to have very dark and broad boundaries which confuses the OCR engine leading to poor results.

Similarly the images suffering from flooding faces following two problems

- Like the images captured in poor lighting conditions ,these images also tends to have some dark areas which have a severe impact on the OCR accuracy
- Also these images have some extremely bright spots as compared to remaining portion of the image. As a result, text falling in these regions are sometime detected as background pixels, thus leading to decrease in overall accuracy.

3) Misalignment Detection: Since our misalignment detection algorithm is based on the Sauvola's binarization



(a) Misaligned Text with proper light

(b) Application detecting Misalignment

(c) Aligned Text with proper light

(d) Aligned Text detected misaligned because of poor lightening conditions

13: Screenshots indicating performance of our Misalignment Detection Algorithm

algorithm, its accuracy is affected by the prevalent lightening conditions while capturing the image. In order to measure the accuracy, we carried out 30 trials for each of the three lighting conditions mentioned above . The corresponding results are shown in Table II.When the lighting conditions are not proper, there are many dark regions in the image and it becomes difficult to differentiate between foreground and the background pixels. As a result, in such cases, our algorithm tends to show higher failure rate. The Figure 13 demonstrates two example trials for our misalignment module.

lighting Conditions	Type of Images	No of Images	No of Images Detected Misaligned	No of Images Detected Properly Aligned
Normal	Misalgined	15	14	1
	Properly Aligned	15	2	13
Poor	Misalgined	15	14	1
	Properly Aligned	15	7	8
Flooding	Misalgined	15	13	2
	Properly Aligned	15	6	9

II: Experimental results indicating accuracy for Misalignment detection algorithm

4) Perception Distortion: When taking pictures, if the camera is not parallel to the text source being captured, the resulting image suffers from some perspective distortion. Based on our experiments, we found the OCR accuracy is highly susceptible to any perception distortion. During our experiments, we found that the character accuracy drops sharply if we tilt the camera over 12°. Therefore, we set the threshold for our orientation sensors to [-10°, 10°]. The Figure 14 shows the effect of the perception distortion on the OCR performance. The image on the top in the figure 14 shows a document image captured while the mobile phone is titled approx 20°and the corresponding OCR text output. The poor OCR accuracy clearly demonstrates OCR results are strongly correlated to the orientation of the camera. As a contrast, the lower image presents the image taken with perfect orientation and its OCR result, which is almost 100% accurate. This clearly demonstrates that orientation sensors indeed play an important role in ensuring good performance.

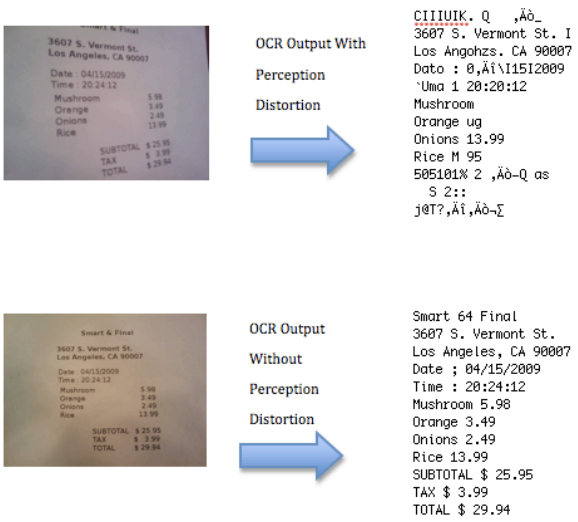
5) Running Time Performance: We measured the total running time taken for each of our preprocessing stages while testing our test corpus. The results are summarized in the Table III. The result doesnt consider network latency issues which arises in any client-server application.

Preprocessing Steps	Max Time Taken(sec)
Misalignment Detection	6
Binarization	3
Skew Detection and Auto-Rotation	2
Character Recognition	3
Total Time Taken	11

III: Experimental results indicating maximum time taken by each of the preprocessing steps

VII. Limitations and Ongoing Work

As demonstrated in the previous section, the applications built on our OCRdroid framework can produce accurate



14: Comparison of OCR results with and without perception distortion

results in real time. However there are still certain areas where we believe our prototype system could be improved. This section discusses several limitations with OCRdroid and some references to our ongoing work.

A. Text Detection from Complex Background

Our OCRdroid framework only works in the case where the background of the text source is simple and uniform. However, in some cases, the source may contain a very complex background, such as pages of magazines with illustrations, or icons of the companies printed on receipts. We are working on applying text detection algorithms to detect regions that most likely contain text and then separate text regions from the complex background.

B. Merge Multiple Images Together

In some cases, image documents or receipt are quite long and cannot be captured within one single mobile frame due to the limited screen size of the mobile phones. We are currently investigating algorithms that can merge the OCR results from images capturing different portions of the same text source and make sure that they are merged in a proper sequence and there is no data lost or repetition.

Finally, we plan to build a new application to digitize sticky notes bases on OCRdroid framework. This mobile application can be used as a mobile memo and a reminder as well.

VIII. Conclusions

In this paper, we present the design and implementation of a generic framework called OCRdroid for developing OCR-based applications on mobile phones. We focus on using orientation sensor, embedded high-resolution camera, and digital image processing techniques to solve the OCR issues related to camera-captured images. One of the key technical challenges addressed by this work is a mobile solution for real time text misalignment detection. We have developed two applications called PocketPal and PocketReader based on OCRdroid framework to evaluate its performance. Preliminary experiment results have been highly promising, which demonstrate our OCRdroid framework is feasible for building real-world OCR-based mobile applications.

References

[1] Abbyy Mobile OCR Engine. http://www.abbyy.com/mobileocr/.

[2] Gimp - the GNU Image Manipulation Program. http://www.gimp.org/.

[3] GOCR - A Free Optical Character Recognition Program. http://jocr.sourceforge.net/.

[4] ImageMagick: Convert, Edit, and Compose Images. http://www.imagemagick.org.

[5] OCR resources (OCRopus). http://sites.google.com/site/ocropus/ocr-resources.

[6] OCRAD - The GNU OCR. http://www.gnu.org/software/ocrad/.

[7] OCRdroid - website. http://www-scf.usc.edu/ anand-djo/ocrdroid/index.php.

[8] OCRopus - Open Source Document Analysis and OCR System. http://sites.google.com/site/ocropus/Home.

[9] Simple OCR - Optical Character Recognition. http://www.simpleocr.com/.

[10] Tesseract OCR Engine. http://code.google.com/p/tesseract-ocr/.

[11] Wikipedia OCR details. http://en.wikipedia.org/wiki/Optical_character_recognition.

[12] Faisal Shafait A, Daniel Keysers A, and Thomas M. Breuel B. Efficient implementation of local adaptive thresholding techniques using integral images, 2008.

[13] W. Bieniecki, S. Grabowski, and W. Rozenberg. Image preprocessing for improving ocr accuracy. *Perspective Technologies and Methods in MEMS Design, MEMSTECH 2007*, 2007.

[14] Ohbuchi Eisaku, Hanaizumi Hiroshi, and Hock Lim Ah. Barcode readers using the camera device in mobile phones. In *CW '04: Proceedings of the 2004 International Conference on Cyberworlds*. IEEE Computer Society, 2004.

[15] Megan Elmore and Margaret Martonosi. A morphological image preprocessing suite for ocr on natural scene images, 2008.

[16] E. Kavallieratou. A binarization algorithm specialized on document images and photos. In *ICDAR*, 2005.

[17] J. Liang, D. Doermann, and H. P. Li. Camera-based analysis of text and documents: a survey. *International Journal on Document Analysis and Recognition*, 7(2-3):84–104, July 2005.

[18] Pranav Mistry and P Maes. Quickies: Intelligent sticky notes. In *International Conference on Intelligent Environments*, 2008.

[19] W. Niblack. *An Introduction to Digital Image Processing*. Prentice Hall, 1986.

[20] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 1979.

[21] S. V. Rice, F. R. Jenkins, and T. A. Nartker. OCR accuracy: UNLV's fifth annual test. *INFORM*, September 1996.

[22] J. Sauvola and M. Pietikainen. Adaptive document image binarization. *Pattern Recognition*, 2000.

[23] M. Seeger and C. Dance. Binarising camera images for OCR. In *ICDAR*, 2001.

[24] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation, 2004.

[25] A. Ulges, C. H. Lampert, and T. M. Breuel. Document image dewarping using robust estimation of curled text lines. In *ICDAR*, 2005.

[26] K. Whitesell, B. Kutler, N. Ramanathan, and D. Estrin. A system determining indoor air quality from images air sensor captured cell phones, 2008.

[27] Luo Xi-Ping, Li Jun, and Zhen Li-Xin. Design and implementation of a card reader based on build-in camera. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*. IEEE Computer Society, 2004.