OCRdroid: A Framework to Digitize Text Using Mobile Phones

Anand Joshi[†], Mi Zhang[§], Ritesh Kadmawala[†], Karthik Dantu[†], Sameera Poduri[†] and Gaurav S. Sukhatme[†§]

[†]Computer Science Department, [§]Electrical Engineering Department
University of Southern California, Los Angeles, CA 90089, USA
{ananddjo,mizhang,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 light-weight 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.

1 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. By fully exploiting these advantages, mobile phones are becoming powerful portable computing platforms, and therefore can process computing-intensive programs in real time.

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 [17]. However, camera-captured documents have some drawbacks. They suffer a lot from focus loss, uneven document lighting, and geometrical distortions, such as text skew, bad orientation, and text misalignment [16]. 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 to address those issues mentioned above. In addition, we have 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:

– A real time algorithm to detect text misalignment 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.
– utilizing 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.
– A mobile application called PocketPal, which extracts text and digits on receipts and keeps track of one's shopping history digitally.
– A mobile application called PocketReader, which provides a text-to-speech interface to read text contents extracted from any text sources (magazines, newspapers, and etc).

The rest of this paper is organized as follows. Section 2 discusses work related to OCR and image processing on mobile phones. Section 3 describes the OCRdroid framework and PocketPal and PocketReader applications. Design considerations of OCRdroid are described in detail in Section 4. The system architecture and implementation are presented in Section 5, with experiments and evaluations in Section 6. Section 7 discusses limitations and future work, and Section 8 summarizes our work.

2 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 [25], 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 [26].

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.

3 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 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 OCRdroid framework. Several potential applications that can be built on this framework are also covered.

3.1 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 hasn't 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.

3.2 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. PocketReader can be applied in many situations. For example, users can quickly capture some hot news from the newspaper and let PocketReader read them out if users do not have time to read. What's more, PocketReader can act as a form of assistive technology potentially useful to people who are blind, visually impaired, or 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.

3.3 Other Potential Applications

It is feasible to apply OCRdroid to digitize physical sticky notes [17] 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].

4 Design Considerations

OCR systems have been under developed in both 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, and misalignment. 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.

4.1 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

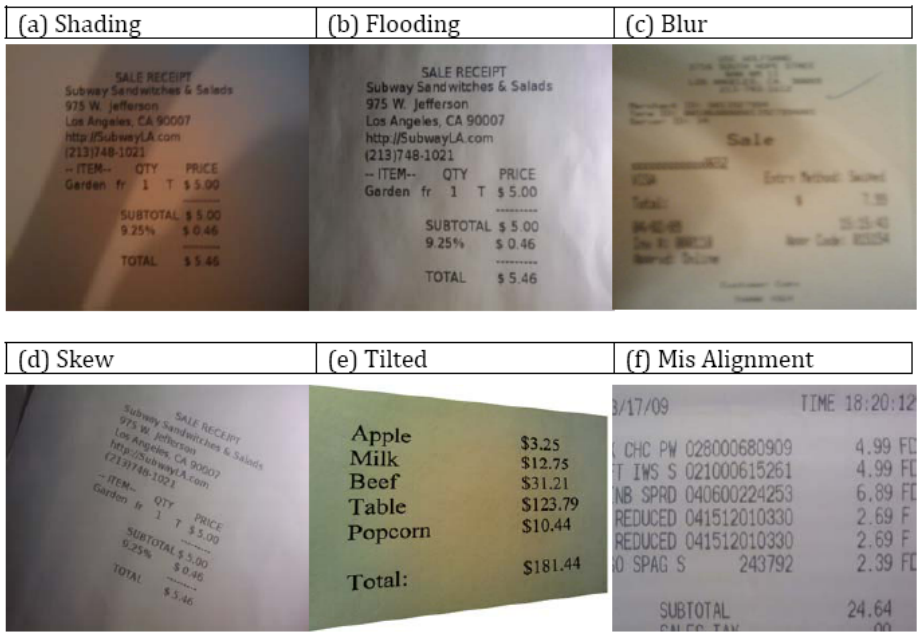


Fig. 1: Different issues arising in camera-captured documents: (a) shading (b) flooding (c) blur (d) skew (e) tilted (f) misalignment

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 [23]. Global binarization methods like Otsu's algorithm [19] 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 [18], and Sauvola's algorithm [21], 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 have also tried Background Surface Thresholding algorithm in [22]. However, based on our experiments, we found Sauvolas's algorithm worked better and faster.

4.2 Text Skew

Issue: When OCR input is taken from a hand-held camera or other imaging devices 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 set margins. However, this assumption does not always holds true in real world scenarios. In addition, traditional methods based on morphological operations and projection methods are extremely slow and tends to fail for camera-captured images. In this work, we choose a more robust approach based on Branch-and-Bound text line finding algorithm (RAST algorithm) [24] 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°.

4.3 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 [16]. 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 advantage 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 situation 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 of 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.

4.4 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 generally arises when people casually take their pictures. Our experiment results indicate that the OCR result is significantly affected by misalignment. Moreover, misaligned images may lead to loss of important data.

Proposed Approach: We define that a 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, we set a 10-pixel wide margin along each border as depicted in Figure 2.

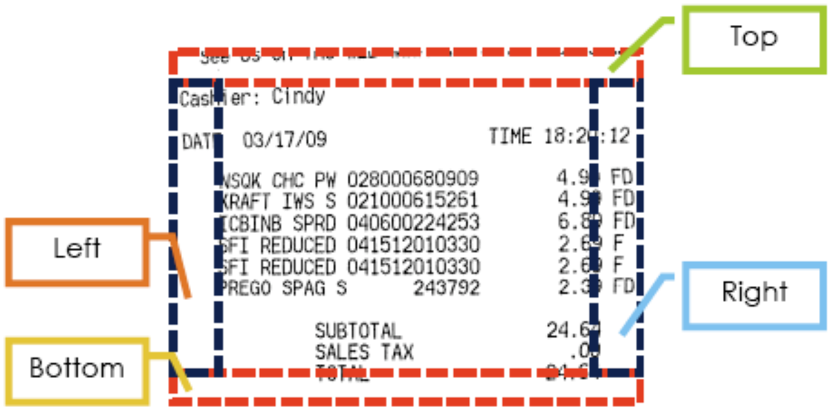

Top
Left
Right
Bottom
Cashier: Cindy
DATE 03/17/09 TIME 18:24:12
NSQK CHC PW 028000680909 4.99 FD
KRAFT IWS S 021000615261 4.99 FD
ICBINB SPRD 040600224253 6.89 FD
SFI REDUCED 041512010330 2.69 F
SFI REDUCED 041512010330 2.69 F
PREGO SPAG S 243792 2.39 FD
SUBTOTAL 24.64
SALES TAX .00
TOTAL

Fig. 2: Four 10-pixel wide margins

If any foreground pixel is detected within any of those 4 margins, there is high probability that the text is being cut and hence we can conclude that the image is misaligned. We faced two major challenges while designing an algorithm for this task

1. Our algorithm should provide real time response.
2. Our algorithm must be able to deal with random noise dots and should not classify them as part of any text.

Considering issues of imperfect lighting condition, we decided to perform a binarization preprocessing step to infer whether we have detected a foreground pixel or not. Despite the fact that global binarization algorithms tends to run faster, we preferred to adopt local binarization algorithm as the basis for our alignment detection algorithm because of 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 a local algorithm by nature, since only pixels within the four margins needs to be checked.

We based our alignment-checking algorithm on the fast variant of Sauvola's local binarization algorithm described in [12] so as to provide real time resposne. What is more, we run the Sauvola's algorithm within four margins in a Round-Robin manner as depicted in Figure 3.

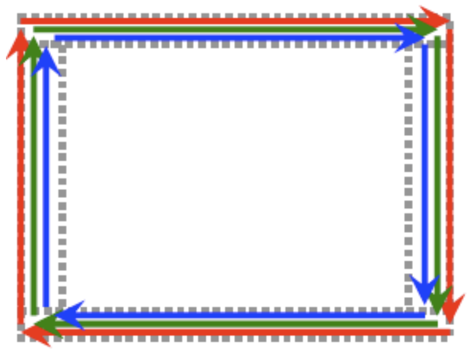


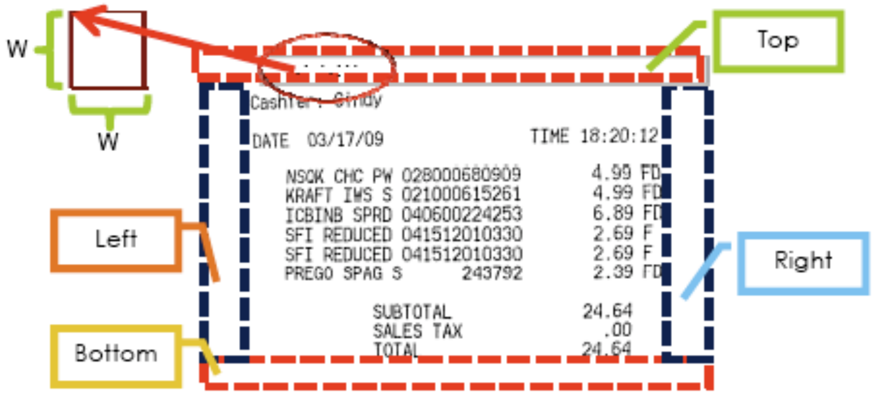
Fig. 3: The route to run Sauvola's binarization algorithm

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. If more than 10% of its neighboring pixels inside the local box are also black dots, then we conclude that current pixel belongs to a text character. This inference is based on the observation that text characters always have many black pixels besides each other, whereas the noise is generally randomly distributed. Based on our experiments, this newly designed alignment-checking algorithm takes no more than 6 seconds and boasts an accuracy of around 96% under normal lighting condition.

4.5 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 [16]. At short distances and large apertures, even slight perspective changes can cause uneven focus.

Proposed Approach: We adopted 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 initiated so that the camera itself can focus on the text sources automatically.



```
Cashier: Cindy

DATE  03/17/09                    TIME 18:20:12

  NSQK CHC PW 028000680909         4.99 FD
  KRAFT IWS S 021000615261         4.99 FD
  ICBINB SPRD 040600224253         6.89 FD
  SFI REDUCED 041512010330         2.69 F
  SFI REDUCED 041512010330         2.69 F
  PREGO SPAG S      243792         2.39 FD

           SUBTOTAL               24.64
           SALES TAX                .00
           TOTAL                  24.64
```

Fig. 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

5 System Architecture and Implementation

Figure 5 presents the client-server architecture of OCRdroid framework. The software installed inside 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 to the back-end server. The computing-intensive image preprocessing steps and character recognition process are performed at the server. Finally, the text results are sent back to the client.

The OCRdroid client program is currently developed on HTC G1 mobile phone powered by Google's Android platform. However, it can be extended with minimal effort to any other platform powered phones with orientation sensors and an embedded camera integrated. The OCRdroid 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 based on this framework. 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.

5.1 Phone Client

We developed the OCRdroid phone client program 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 platforms such as 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 inside the phone. All these necessary APIs are provided by Android SDK (version 1.1).

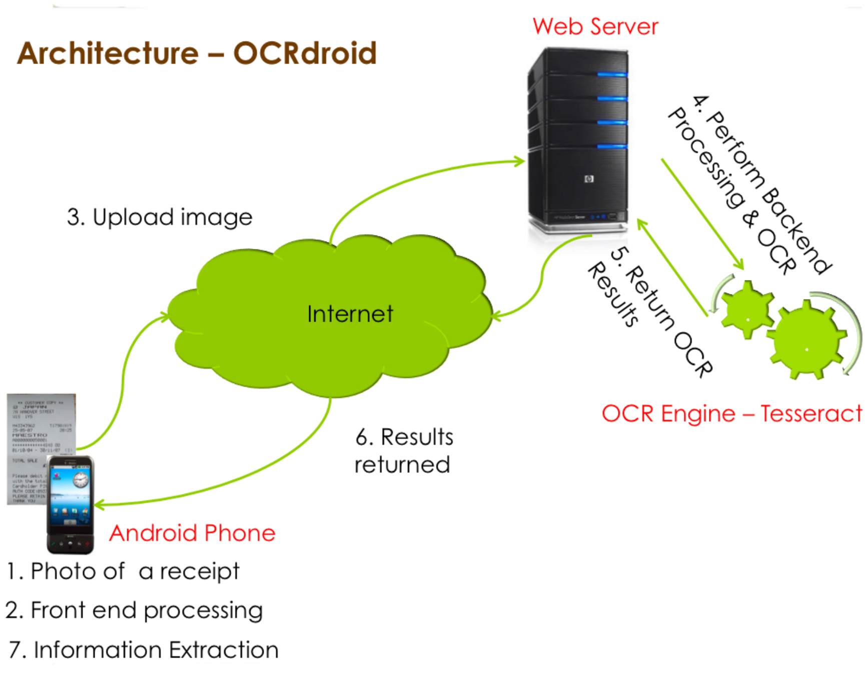


Fig. 5: Overview of OCRdroid framework architecture

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. OCRdroid framework 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 architectural diagram for the client software. The client software consists of 7 functional modules. By default, the client is in idle state. When the user initiates the application, the client starts up a camera preview to take a picture. The preview object implements a callback function to retrieve the data from the embedded 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

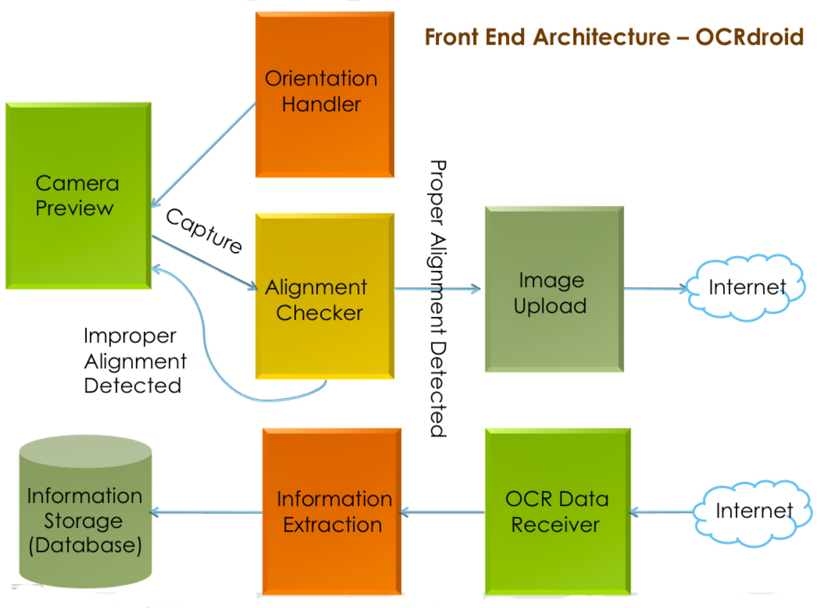


Fig. 6: OCRdroid Client(Phone) Architecture

phone is tilted more than the allowed threshold, users are prevented from capturing any images. Figure 7 presents a screenshot showing how the tilt detection module works. The image on the right shows a case where user has somewhat tilted the phone. As a result, a red colored bounding box is displayed, indicating 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.

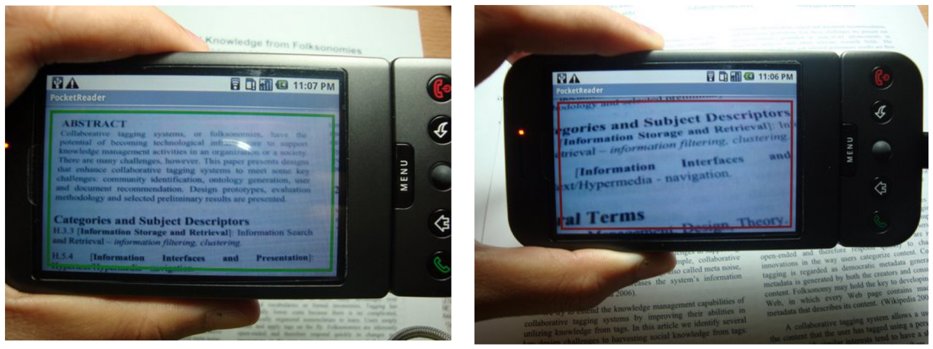


Fig. 7: Screenshot demonstrating tilt detection capability of OCRDroid using embedded orientation sensor. A green (red) rectangle indicates correct (incorrect) tilt.

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 where misalignment is detected. As soon as the text source is properly aligned, the new image is transported to the server over a HTTP connection. An OCR data receiver module keeps listening on the connection and passes the OCR result sent from the server to information extraction module, where corresponding information is extracted and parsed into different categories. Finally, the OCR result is displayed on the screen and automatically stored inside the local database for future references.

5.2 Back-End Server

OCRdroid backend 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 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.

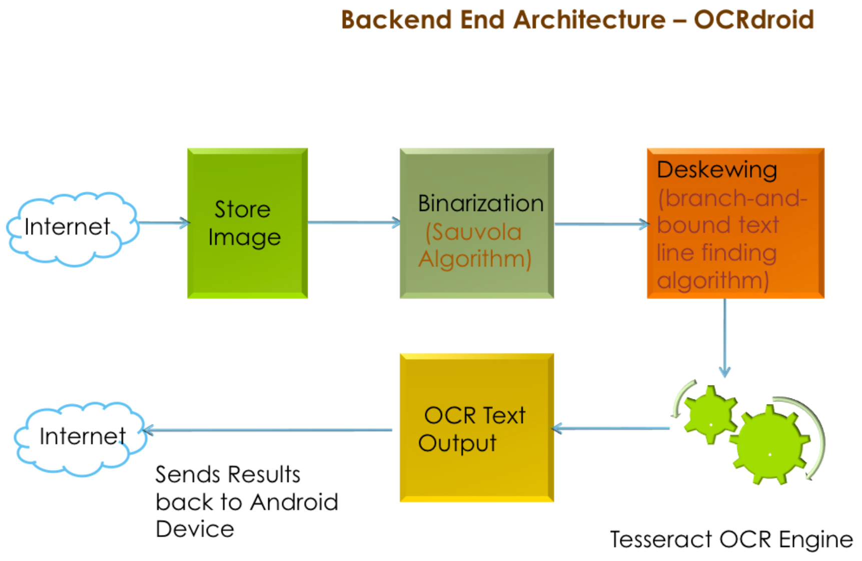

Backend End Architecture – OCRdroid
Internet
Store Image
Binarization (Sauvola Algorithm)
Deskewing (branch-and-bound text line finding algorithm)
Tesseract OCR Engine
OCR Text Output
Internet
Sends Results back to Android Device

Fig. 8: Software architecture For OCRdroid Backend

There are many open source as well as commercial OCR engines available, each with its own unique strengths and weaknesses. A detailed list of OCR engines is available at [11]. 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. We also tested a complete document analysis tool called OCRopus [5]. It first performed document layout analysis and then used Tesseract as its OCR engine for character recognition. However, OCRopus gave rise to one additional complication in our PocketPal application because of its inherent multi-column support. The document layout analyser identified receipts to be in a 2-column format. As a result, it displayed the names of all shopping items followed by their corresponding prices. This required us to carry out extra operations to match the name and the price of each item. Therefore, we choose Tesseract as our OCR engine.

Once the whole process finishes successfully, OCRdroid server responds to the client side with an OK message. On receieving this message, the client sends back a HTTP request to ask for the OCR result. Finally, OCRdroid server sends the text result back as soon as the request is received.

6 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. Then we present the results of a set of preprocessing steps and detailed performance analysis.

6.1 Test Corpus

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

6.2 Performance Metrics

In order to measure the accuracy of OCR, we adopt two metrics proposed by the Information Science Research Institute at UNLV for the Fifth Annual Test of OCR Accuracy [20]

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 defiend as:

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

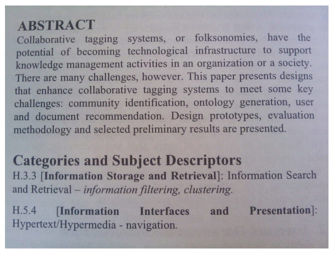
Word Accuracy In a text retrieval application, the correct recognition of words is much more important than the correct recognition of numbers or punctuations. We define a word to be any sequence of one or more letters. If *m* out of *n* words are recognized correctly, the *word accuracy* is *m/n*. Since 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 processing time taken by each of the preprocessing steps.

6.3 Experimental Results and Analysis

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

Binarization Binarization plays a very important role in OCR preprocessing procedure. Figure 9 presents one test case to demonstrate the importance of binarization process. The effectiveness of binarization algorithm heavily depends upon the lighting

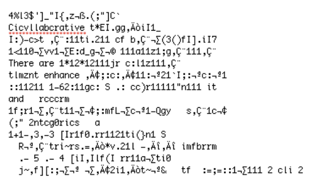


ABSTRACT
Collaborative tagging systems, or folksonomies, have the potential of becoming technological infrastructure to support knowledge management activities in an organization or a society. There are many challenges, however. This paper presents designs that enhance collaborative tagging systems to meet some key challenges: community identification, ontology generation, user and document recommendation. Design prototypes, evaluation methodology and selected preliminary results are presented.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *information filtering, clustering.*

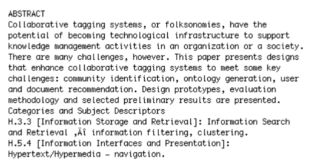
H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia - navigation.

(a) An image of text source taken under normal lighting condition



```
4%l3$']_"I{,z¬ß.(;"]C`
Cicvllabcrative t*EI.gg,ÄòiI1_
I:)-c>t ,Ç¨:11ti.211 cf b,Ç¨¬∑(3()fI].iI7
1<110¬∑vv1¬∑E:d_g¬∑¬® 111a11z1;g,Ç¨111,Ç¨
There are 1*12*12111jr c:l1z111,Ç¨
tlmznt enhance ,Ä¢;:c:,Ä¢11:¬º21`I;:¬ºc:¬º1
::11211 1-62:11gc: S .: cc)r11111"n111 it
and   rcccrm
1f;r1¬∑,Ç¨t11¬∑¬¢;:mfL¬∑c¬º1-Qgy   s,Ç¨1c¬¢
(;" 2ntcg0rics   a
1+1-,3,-3 [Ir1f0.rr1121ti()n1 S
  R¬º,Ç¨tri~rs.=,Äò*v.21l -,Äî,Äî imfbrrm
  .- 5 .- 4 [iI,Ilf(I rr11a¬∑ti0
  j~,f][:;¬∑¬º ¬∑,Ä¢2i1,Äòt~¬º&   tf  :=;=::1¬∑111 2 cli 2
```

(b) OCR output without binarization



```
ABSTRACT
Collaborative tagging systems, or folksonomies, have the
potential of becoming technological infrastructure to support
knowledge management activities in an organization or a society.
There are many challenges, however. This paper presents designs
that enhance collaborative tagging systems to meet some key
challenges: community identification, ontology generation, user
and document recommendation. Design prototypes, evaluation
methodology and selected preliminary results are presented.
Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search
and Retrieval ,Äî information filtering, clustering.
H.5.4 [Information Interfaces and Presentation]:
Hypertext/Hypermedia - navigation.
```

(c) OCR output with binarization

Fig. 9: Comparision of OCR results with and without Binarization Module

conditions when image is captured. To measure the performance, we carried out our experiments under three distinct 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 lightening condition: This describes situations when users take images outdoors during night or capture images in rooms which have very dim lighting.
– Flooding condition: This describes situations when the source of light is very much focused on a particular portion of the image, whereas the remaining portion is dark.

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

Table 1: Comparision of OCR character accuracy with and without binarization under three lighting conditions

Table 1 lists the character accuracy of OCR output with and without binarization under three lighting conditions. As expected, OCR output with binarization achieves much higher peformance than their counterparts without binarization. Under normal lighting condition, the OCR output with binarization achieves 96.94% character accuracy. However, the performance degrades significantly if lighting condition is poor or flooded. The main reason to cause this problem is the text in the shaded regions tends to have very dark and broad boundaries. This confuses the OCR engine, and as a result, leads to relatively low character accuracy.

Skew Detection and Auto-Rotation Our Tesseract OCR engine failed to produce any meaningful results if the input image is skewed more than 5°. 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 counter-clockwise directions using Gimp Image Processing ToolBox [2]. These rotated images are pictured under three different lighting conditions and then passed to the OCR engine. Figure 10 and Figure 11 demonstrate the performance in terms of average character accuracy and average word accuracy.

As presented, our skew detection and auto rotation algorithm works quite well for image rotation up to 30°in both clockwise and counter-clockwise directions. On the other hand, even under normal lighting condition, the performance drops sharply with image rotation at 35°. However, in real-world applications, it is reasonable to believe that general users would not take images at such high degree of rotation.

Misalignment Detection Since our misalignment detection algorithm is based on the Sauvola's binarization algorithm, the accuracy depends on lighting conditions as well. Figure 12 presents some test cases under both normal and poor lighting conditions. To measure the accuracy, we followed the definitions of three different lighting conditions, and carried out 30 trials under each lighting condition. Finally, the total number of false positives and false negatives are counted and summarized below.

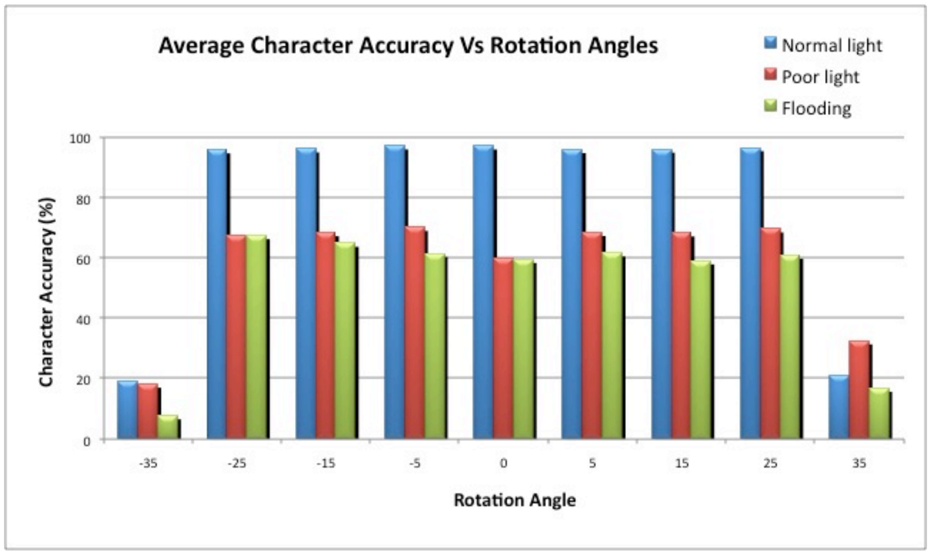


Fig. 10: Average Character Accuracy of OCR output at different rotation angles under three different lighting conditions

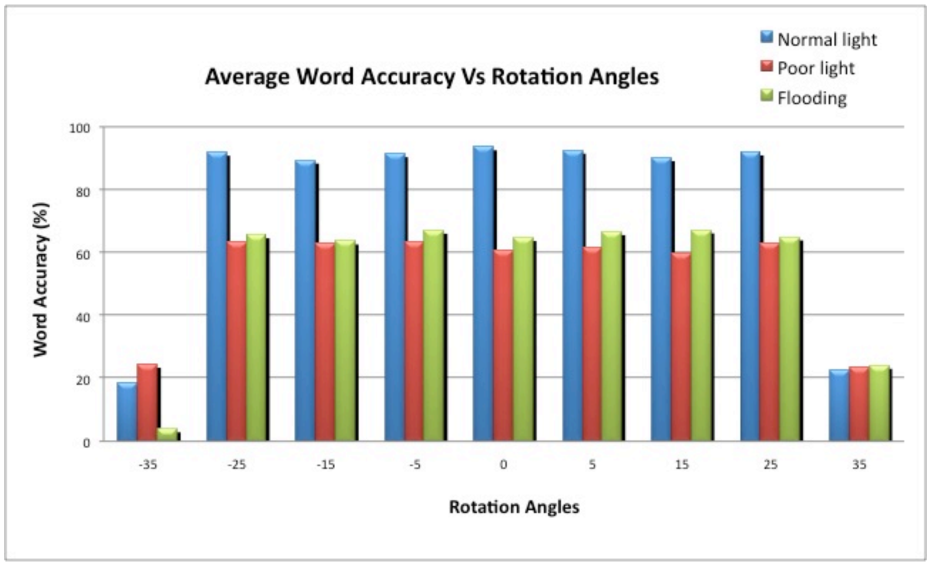
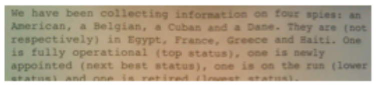


Fig. 11: Average Word Accuracy of OCR output at different rotation angles under three different lighting conditions



We have been collecting information on four spies: an American, a Belgian, a Cuban and a Dane. They are (not respectively) in Egypt, France, Greece and Haiti. One is fully operational (top status), one is newly appointed (next best status), one is on the run (lower status) and one is retired (lowest status).

(a) Misaligned Text with proper light

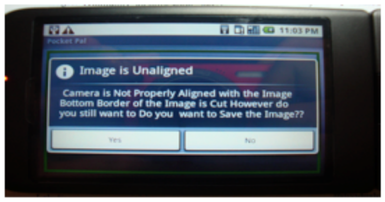
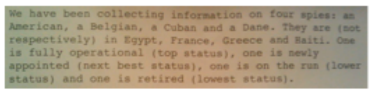


Image is Unaligned

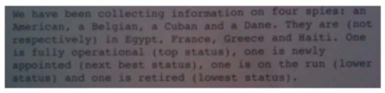
Camera is Not Properly Aligned with the Image Bottom Border of the Image is Cut However do you still want to Do you want to Save the Image??

(b) Application detecting Misalignment



We have been collecting information on four spies: an American, a Belgian, a Cuban and a Dane. They are (not respectively) in Egypt, France, Greece and Haiti. One is fully operational (top status), one is newly appointed (next best status), one is on the run (lower status) and one is retired (lowest status).

(c) Aligned Text with proper light



We have been collecting information on four spies: an American, a Belgian, a Cuban and a Dane. They are (not respectively) in Egypt, France, Greece and Haiti. One is fully operational (top status), one is newly appointed (next best status), one is on the run (lower status) and one is retired (lowest status).

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

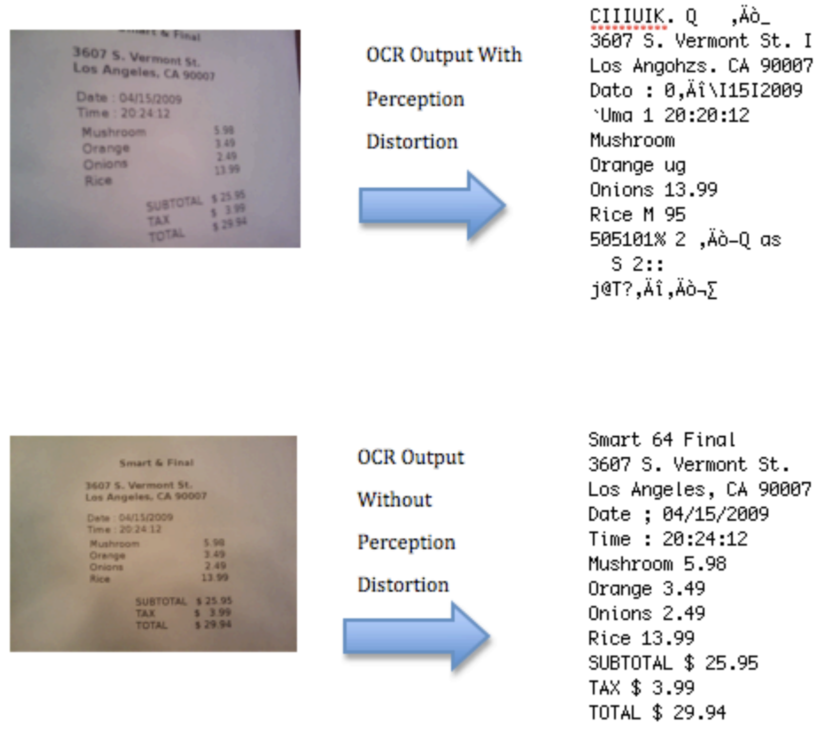
Fig. 12: Screenshots of test cases for misalignment detection

Lighting Conditions	Type of Images	No of Images	No of Images Detected Misalgined	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

Table 2: Experimental results indicating accuracy for Misalignment detection algorithm

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. However, it is very hard for us to measure the tilting angles directly. Therefore, we tuned the thresholds in the program and then checked the performance. Based on our experiments, we found the OCR accuracy is highly susceptible to camera orientation. The character accuracy drops sharply if we tilt the camera over 12°. Therefore, we set the threshold for our orientation sensor to [-10°, 10°]. The upper graph in Figure 13 shows a document image captured when the mobile phone is titled 20°and the corresponding OCR text output. The poor OCR accuracy indicates OCR results are strongly correlated to the orientation of the camera. As a contrast, the lower graph presents the image taken with 5°orientation and its OCR result with 100% accuracy. This contrast clearly demonstrates that orientation sensor plays an important role in ensuring good performance.

Timing Performance We evaluate timing performance in terms of processing time taken by each of the preprocessing steps. Since binarization, skew detection and character recognition are performed at the server, we close all other processes in the server to reduce the measurement error. For misalignment detection, the measurement error


OCR Output With Perception Distortion
OCR Output Without Perception Distortion

```
CIIIUIK. Q   ,Äò_
3607 S. Vermont St. I
Los Angohzs. CA 90007
Dato : 0,Äî\I15I2009
`Uma 1 20:20:12
Mushroom
Orange ug
Onions 13.99
Rice M 95
505101% 2 ,Äò-Q as
  S 2::
j@T?,Äî,Äò¬∑
```

```
Smart 64 Final
3607 S. Vermont St.
Los Angeles, CA 90007
Date ; 04/15/2009
Time : 20:24:12
Mushroom 5.98
Orange 3.49
Onions 2.49
Rice 13.99
SUBTOTAL $ 25.95
TAX $ 3.99
TOTAL $ 29.94
```

Fig. 13: Comparison of OCR results with and without perception distortion

is relatively big since we can not shut down some native applications running in the phone. In addition, since we used Wi-Fi as the link between the mobile device and the backend server, the network latency is negligible compared to other processing steps. The result is summarized in Table 3.

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

Table 3: Experimental results indicating maximum time taken by each of the preprocessing steps

It takes maximum 11 seconds to complete the whole process. As expected, due to limited processing power of mobile phone, misalignment detection is the most time-consuming step among the entire process. However, the entire process quite meets the real time processing requirement.

7 Limitations and Ongoing Work

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

7.1 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 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.

7.2 Merge Multiple Images Together

In some cases, image documents or receipts are quite long and can not be captured within one single frame due to the limited screen size of 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 they are merged in a proper sequence without data lost or repetition.

7.3 New Applications on top of OCRDroid Framework

We are working on a new application on the top of the OCRdroid framework - PocketScan. This application allows users to quickly scan ingredients of any medicines as well as food and check if they contain some particular chemical to which the user is allergic.

8 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 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. In addition, we have developed two applications called PocketPal and

PocketReader based on OCRdroid framework to evaluate its performance. Preliminary experiment results are highly promising, which demonstrates 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/ ananddjo/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. 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.
17. Pranav Mistry and P Maes. Quickies: Intelligent sticky notes. In *International Conference on Intelligent Environments*, 2008.
18. W. Niblack. *An Introduction to Digital Image Processing*. Prentice Hall, 1986.
19. N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 1979.
20. S. V. Rice, F. R. Jenkins, and T. A. Nartker. OCR accuracy: UNLV's fifth annual test. *INFORM*, September 1996.
21. J. Sauvola and M. Pietikainen. Adaptive document image binarization. *Pattern Recognition*, 2000.
22. M. Seeger and C. Dance. Binarising camera images for OCR. In *ICDAR*, 2001.
23. M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation, 2004.
24. A. Ulges, C. H. Lampert, and T. M. Breuel. Document image dewarping using robust estimation of curled text lines. In *ICDAR*, 2005.
25. K. Whitesell, B. Kutler, N. Ramanathan, and D. Estrin. A system determining indoor air quality from images air sensor captured cell phones, 2008.
26. 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.