LAMP-TR-105
CAR-TR-987
CS-TR-4519
UMIACS-TR-2003-87

August 19, 2003

ADAPTIVE HINDI OCR USING GENERALIZED HAUSDORFF IMAGE COMPARISON

HUANFENG MA, DAVID DOERMANN

LAMP-TR-105 UMIACS-TR-2003-87
CAR-TR-987
CS-TR-4519

ADAPTIVE HINDI OCR USING GENERALIZED HAUSDORFF IMAGE COMPARISON

HUANFENG MA
DAVID DOERMANN

Language and Media Processing Laboratory
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742-3275
{*hfma,doermann*}*@umiacs.umd.edu*

Abstract

In this paper, we present an adaptive Hindi OCR using generalized Hausdorff image comparison implemented as part of a rapidly retargetable language tool effort. The system includes: script identification, character segmentation, training sample creation and character recognition. The OCR design (completed in one month) was applied to a complete Hindi-English bilingual dictionary (with 1083 pages) and a collection of ideal images extracted from Hindi documents in PDF format. Experimental results show the recognition accuracy can reach 88% for noisy images and 95% for ideal images, both at the character level. The presented method can also be extended to design OCR systems for different scripts.

Keywords: OCR, Generalized Hausdorff Image Comparison, Script Identification, Document Processing

The support of this research under DARPA cooperative agreement N660010028910 and DOD contract MDA90402C0406 is gratefully acknowledged.

1 Introduction

Digital document processing is gaining popularity for application to office and library automation, bank and postal services, publishing houses and communication technology. An important task of automatic document processing is the reading of text. The procedure of automatically processing the text components of a complex document which contains text, graphics, and/or images can be divided into three stages: (1) region extraction and text region classification using document layout analysis; (2) text line, and possibly word (glyphs separated by white space) and character segmentation; and (3) optical character recognition (OCR). Typically, the OCR classifier stage needs to be redesigned for each new script, while the other stages are easier to port. OCR technology for some scripts like Roman and Chinese is fairly mature and commercial OCR systems are available with accuracy higher than 98%. For example, *OmniPage Pro* from ScanSoft or *FineReader* from ABBYY for Roman and Cyrillic scripts, and *THOCR* from Tsinghua University for Chinese.

Although commercial systems are available for Roman, Cyrillic, far east and many middle east languages, such systems for Indian scripts, as well as many low density languages are still in the research and development stage. In some cases, this is due to technical challenges, but more often due to a lack of a commercial market. Nevertheless, there is a real need for OCR in these languages.

The DARPA TIDES program is supporting a project at the University of Maryland that is focused in part on rapidly acquiring resources from printed resources such as bilingual dictionaries. With the large number of different languages all over the world, obtaining OCR systems of all these languages is unrealistic. We need to be prepared to retarget OCR systems to be able to deal with specific tasks including new languages or new scripts.

During a recent 'Surprise Language' task for TIDES that focused on Hindi, we were faced with the challenge of rapidly acquiring Hindi OCR capabilities. Since no feasible commercial OCR system was available, we set out to develop one as rapidly as possible. In this paper, we present a Devanagari(Hindi) OCR system using Generalized Hausdorff Image Comparison (GHIC) that was developed and trained in less than a month. Trained using character samples extracted from different documents, the OCR system can be easily adapted to

perform Devanagari OCR in any font. Details of a complete system for segmenting, parsing and tagging bilingual dictionaries can be found in [12].

1.1 Background

Devanagari, an alphabetic script, is used by a number of Indian languages, including Sanskrit, Hindi and Marathi. Many other Indian languages use close variants of this script. Although Sanskrit is an ancient language and is no longer spoken, written material still exists. Hindi is a direct descendant of Sanskrit through Prakrit and Apabhramsha. It is a very expressive language, which has been influenced and enriched by Dravidian, Turkish, Farsi, Arabic, Portugese and English. Hindi is the world's third most commonly used language after Chinese and English, and there are approximately 500 million people all over the world that speak and write in Hindi. Thus, research on Devanagari script, mainly the Hindi language, attracts a lot of interest. In the rest of this paper, Hindi, the language, and Devanagari, the script used are interchangeably.

Unlike English and other Roman script languages, there are few if any commercial Hindi OCR systems on the market. Chaudhuri and Pal proposed a Devanagari OCR system that was ultimately purchased and is being marketed as a custom solution, but is not available as an off the shelf product. The basic components of the system, however are described in the literature [3] [4]. After word and character segmentation, a feature based tree classifier is used to recognize the basic characters. Error detection and correction based on a dictionary search brings the recognition accuracy of the OCR to 91.25% at word level and 97.18% at character level on clean images. In his Ph.D. thesis [1], Bansal designed a Devanagari text recognition system by integrating knowledge sources, features of characters such as horizontal zero crossings, moments, aspect ratio, pixel density in 9-zones, number and position of vertex points, with structural descriptions of characters. These are used to classify characters and perform recognition. After correction, based on dictionary search, the average accuracy is about 87% at character level for scanned document images.

It should be noted that both of the OCR systems mentioned above need vast amounts of training data with ground truth to achieve acceptable levels of performance. Collection and ground-truthing of data is time consuming and labor intensive. Even so, before feeding

a new font of a Hindi document to the OCR, the system must be retrained to obtain a reasonable accuracy. In our application, we benefit from the need for only a small number of fonts for any given dictionary. In this paper, we propose an approach to quickly build Hindi OCR. The segmentation of characters is similar to the approach proposed in [1] with minor changes, and the recognition is based on the Generalized Hausdorff Image Comparison. This OCR does not need to be trained using a large number of training samples, and is easily adapted to different types of documents.

The rest of this paper is organized as follows: Section 1.2 describes the system architecture. Section 2.1 addresses how to identify the Devangari words from bilingual or multilingual documents. Section 2.2 describes the procedure of character segmentation. Section 2.3 addresses the procedure of character recognition and some postprocessing techniques, and Section 3 provides experimental results. Summary, conclusion, and future work is discussed in Section 4.

1.2 System architecture

Our Hindi OCR, designed to work on pure Devanagari, or bilingual and multilingual document images with one script Devanagari, is shown in Figure 1. The system contains three different functional components: (1) document image preprocessing including denoising and deskewing; (2) segmentation and script identification at word level; and (3) a classifier. In the following sections, we'll briefly describe the word level script identification and focus on the design of the Hindi classifier.

1.3 Overview

The system first scans pages of Hindi text at 300 or 400 DPI. Images are first preprocessed with denoising and deskewing [5]. An implementation of DOCTRUM [14] was applied to the preprocessed images to segment them into zones, text lines and words. Components of page are segmented into entries based on the functional features of document using the approach described in [9]. Figure 2 shows the segmented dictionary entries. Script identification is applied to the segmented word images to identify Devanagari script and Roman script words (including symbols neither Roman nor Devanagari). The identified Roman script

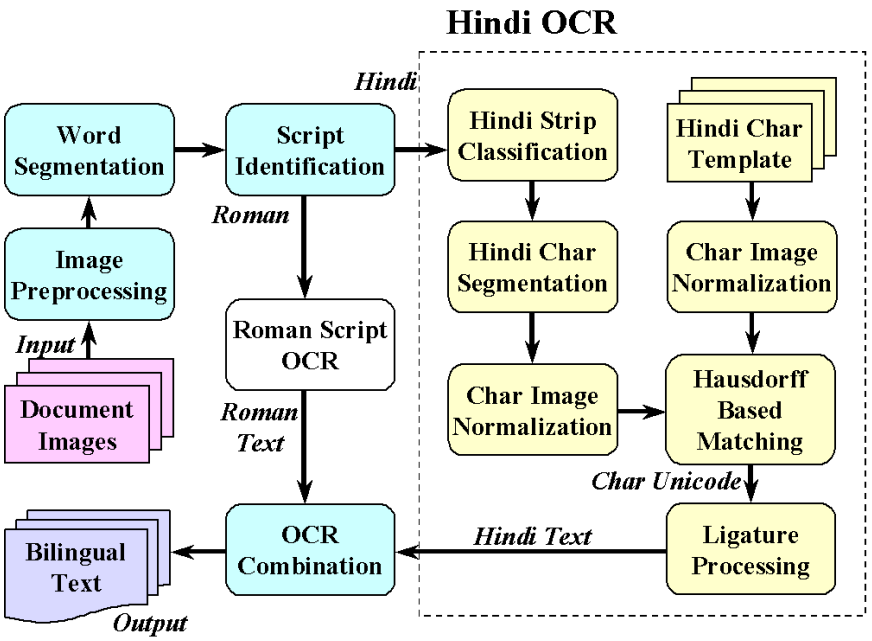


Figure 1: System architecture.

word images are fed into a commercial English OCR, while the Hindi word images are first segmented into characters, and the character images are fed into a classifier to perform classification and recognition. After postprocessing the recognized Hindi words, the output of the Hindi OCR will be combined with the OCR output of Roman script to provide a complete result. The details of approach and results are described in the following sections.

2 Technical approach

2.1 Devanagari script identification

Before describing what kinds of features can be used to identify Devanagari script words from bilingual or multilingual document images, we examine the appearance of Devanagari script. Regular Hindi words can typically be divided into three strips: top, core and bottom. For the Hindi word अकुलीन for example, the five-character-word image is show in Figure 3, where three strips are illustrated. The top strip and core strip are always separated by the header line, while there is no corresponding feature to separate the bottom strip and core

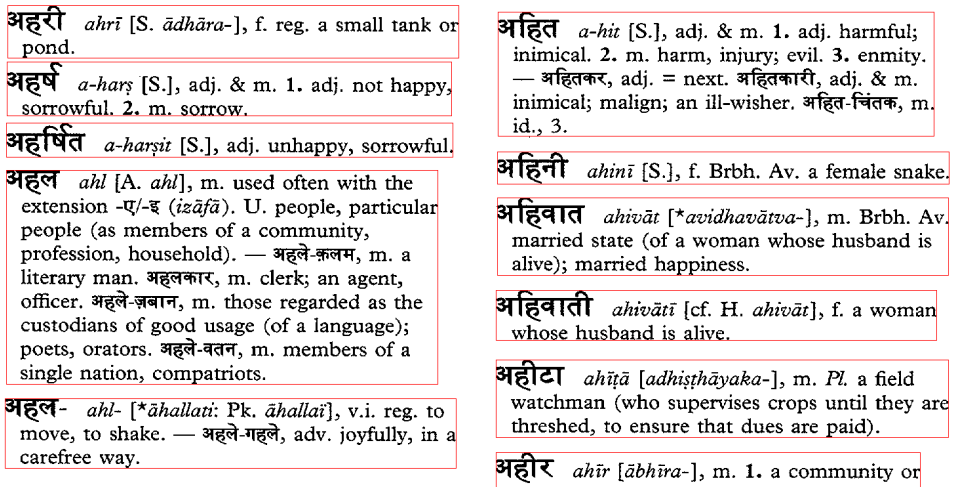


Figure 2: Segmented entries of the Hind-English dictionary.

strip. The top strip contains the top modifiers, and bottom strip contains lower modifiers. In a Hindi word, top and bottom strip are not always necessary, but depend on top and lower modifiers.

Figure 3: Three strips of a Hindi word.

In [10], we proposed an approach to identify scripts at word level based on the texture features, where the features were extracted using Gabor filters. The performance of the approach can be improved by applying Supported Vector Machines (SVM), as found in [11]. The approaches in [10] and [11], however, are designed to be used to identify scripts under the assumption that the operator knows nothing about the non-Roman script, and the only feature used to do script identification is texture features extracted in 16 Gabor channels. For a specific script, some non-texture features can be used to improve the performance, if the system knows these features. The occurrence of a header line in a Hindi word is such a

powerful feature that it can be used to identify Hindi words from bilingual or multilingual document images. For each segmented word with width W and height H, we compute the horizontal projection (denoted HP) of this word and then find the maximum value HP_{max} and the position PS_{max} of the maximum value. A word could be identified as a Hindi word if and only if:

$$HP_{max} > 0.8W$$
$$PS_{max} > 0.5H$$

In some documents, single Roman character such as ***E, e, R, T, t, I, P, D, F, l, Z, z,*** or ***B*** in a specific font face may also satisfy the above two criteria, so these misidentification cases must be handled to improve the performance. Considering the fact that in a regular document, all these characters except ***I*** seldom appear as a single character, while ***I*** is usually much narrower than a single Hindi character, these misidentification are removed by setting a word width threshold which depends on the font size and resolution of document image. Figure 4 shows the performance comparison of two script identification approaches, *SVM* and above-mentioned *Script-Oriented* approach on 20 randomly picked pages, which are sorted based on the *Script-Oriented* accuracy. The result shows both of the approaches work effectively with average accuracy higher than 93%, and the latter is much higher with average accuracy of 98.94%.

2.2 Character segmentation

2.2.1 Devanagari script overview

Devanagari has about 11 vowels (shown in the first row of Table 1) and about 33 consonants (shown in Table 2). Each vowel except अ corresponds to a modifier symbol as shown in the second row of Table 1. In Hindi, when consonants are combined with other consonants, the consonants with a vertical bar may appear as a half-form. Except for the characters क and फ , the half form of consonants are the left part of original consonants with the vertical bar and part right of the bar removed. These half-form of consonants are shown in Table 3, where the order of characters corresponds to the character order in Table 2. Table 4 gives some examples of combinations of half-consonant with other consonants. The combination

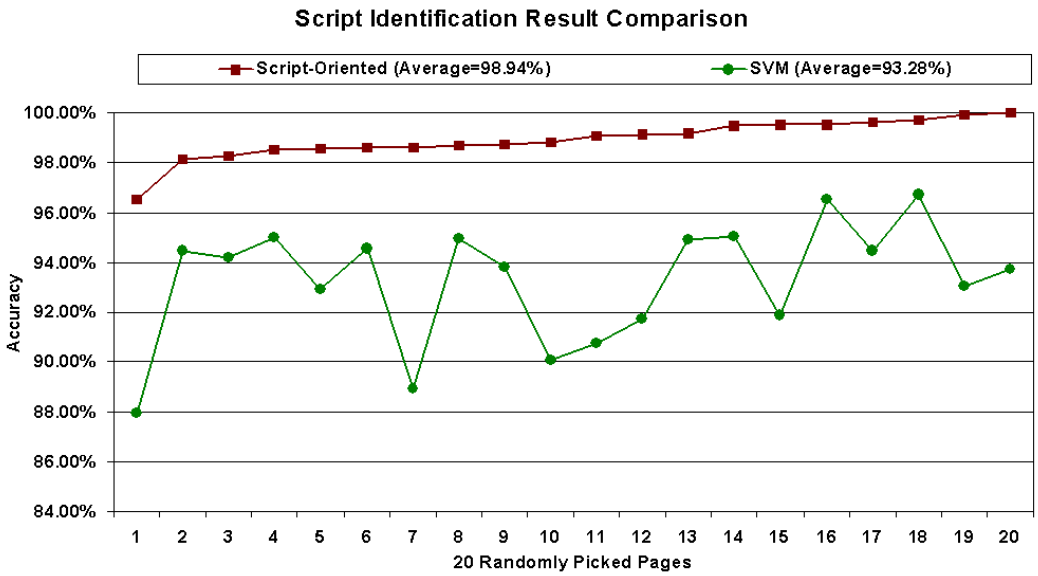


Figure 4: Accuracy comparison of two script identification approaches (Random pages, sorted by *Script-Oriented* accuracy).

of half-consonant and other consonants are not always left-right structured, sometimes the combination orientation is top-down, or even reorganized to become a new character. Some of these examples of special combination are shown in Table 5. It should be noted that the listed special combination is far from complete, so how to handle all these cases needs to be solved. In Subsection 2.3.5, we'll address how to deal with these special cases with operator feedback. In addition, Table 6 shows a list of special characters or symbols, some of them are seldom used, but they do exist in different documents, so these special symbols also need to be handled.

Vowels:	अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ
Modifiers:		ा	ि	ी	ु	ू	ृ	े	ै	ो	ौ

Table 1: Vowels and corresponding modifiers.

क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
ठ	ड	ढ	ण	त	थ	द	ध	न	प	फ
ब	भ	म	य	र	ल	व	श	ष	स	ह

Table 2: Consonants.

क्	ख्	ग्	घ्		च्		ज्	झ्	ञ्	
			ण्	त्	थ्		ध्	न्	प्	फ्
ब्	भ्	म्	य्		ल्	व्	श्	ष्	स्	

Table 3: Half form of consonants with vertical bar.

क क क्क	क ल क्ल	घ न घ्न	च ञ च्ञ	ञ च ञ्च	त न त्न	प त प्त	प ल प्ल
ब व ब्व	भ न भ्न	म ल म्ल	ल ल ल्ल	श न श्न	श ब श्ब	श ल श्ल	स न स्न

Table 4: Examples of combination of half-consonant and consonant.

क ष क्ष	ज ञ ज्ञ	ट ट ट्ट	ट ठ ट्ठ	त र त्र	द द द्द
द ध द्ध	द व द्व	द व र द्व	श र श्र	द भ द्भ	द य द्य

Table 5: Examples of special combination of half-consonant and consonant.

क़	ख़	ग़	ज़	फ़	ड़	ढ़	ं	ँ	ः	।	ऽ	ॅ

Table 6: Special symbols.

2.2.2 Hindi character segmentation

The procedure to segment a Hindi word into characters (including core characters, and top and bottom modifiers) is illustrated in Figure 5 using the segmentation of the Hindi word स्वीकृति as an example. The numbered arrow in Figure 5 represents the step of segmentation, and the characters with solid bounding boxes are the final segmentation results. The procedure to do character segmentation can be described as follows:

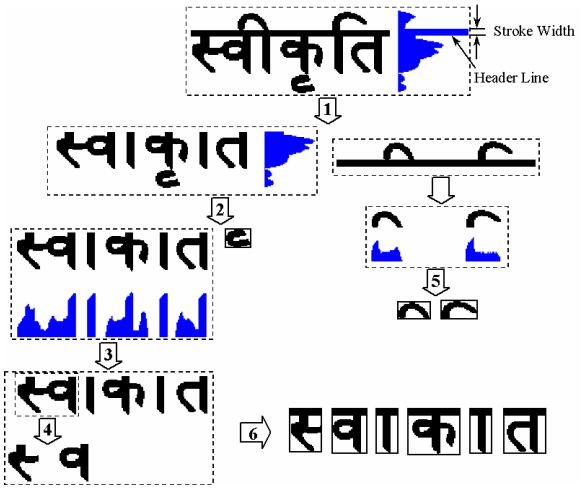


Figure 5: The procedure of Hindi character segmentation.

Step 1: Locate the header line and separate the core-bottom strip which contains the core strip and bottom strip , and a top strip which contains the header line and top modifiers.

Step 2: Identify core strip and bottom strip from the core-bottom strip, and extract low modifiers.

Step 3: Separate core strip into characters which may contain conjunct/shadow characters.

Step 4: Segment conjunct/shadow characters into single characters.

Step 5: Remove the header line from the top strip and extract top modifiers.

Step 6: Put header line back to the segmented core characters.

The details for each step are described below. We denote the width of the Hindi word bound box as W, the height as H, and the coordinates of the left-top corner are set to be (0,0).

Step 1: Separate top strip and core-bottom strip. The separation of top strip and core-bottom strip is based on the location of the header line. For each word, we compute the horizontal projection (HP) and find the row (with Y-coordinate y) having the maximum value of HP. This is the candidate of the header line position. A header line candidate can be the real header line if: $y \leq 0.4W$. If this condition is not satisfied, then set the HP value

of this row to 0 and re-search the row with maximum value until a real header line position is located. The maximum HP value is marked as HP_{max} and the position of the header line is marked as *hPosition*. Setting *hPosition* as center, traverse the adjacent 10 HP values at each side of *hPosition*, and find the continuous rows whose HP values are all greater than $0.8HP_{max}$. The number of these continuous rows is the stroke width of this word which is marked as the *StrokeWidth* and important for the postprocessing of segmentation. *hPostion* is updated as the first row's Y-coordinates of the header line. The header line separates the Hindi word into top strip (including the header line) and core-bottom strip. This procedure is shown in step 1 of Figure 5.

Step 2: Identify core strip and bottom strip from the core-bottom strip. The width and height of the core-bottom strip obtained in the last step are denoted as W_{cb} and H_{cb}. The vertical projection VP of the core-bottom strip is computed in this step. The columns with no black pixels separate the Hindi word into several character candidates which may contain conjunct/shadow characters. We find the maximum height of these characters and denote it H_{max}. The separated characters are divided into three groups. The first group contains all characters with height greater than $0.8H_{max}$, the second group contains characters whose height is between $0.8H_{max}$ and $0.64H_{max}$, and the remaining characters are put into the third group. The group with the maximum number of members is considered to contain normal characters without low modifiers, and the maximum height of members in this group is set as a threshold hTh. If $(H_{cb} - hTh) \geq 1/4H_{cb}$, the word contains low modifiers. Next step, compute the horizontal projection HP_{cb} of the core-bottom strip. Setting hTh as center, traverse the adjacent 5 HP_{cb} values at each side of hTh, the row with the minimum HP_{cb} value is the boundary which will segment the core-bottom strip into the core and bottom strips. The vertical projection of the bottom strip is computed, and low modifiers are easily located by finding the columns without black pixels or with least black pixels. This procedure is shown in step 2 of Figure 5.

Step 3: Separate the core strip into characters. In this step, the core strip is separated into characters, and the conjunct/shadow characters, which need further segmentation, will be determined in this step as well. We borrow the definition of shadow character from Bansal and Sinha [2]. A character is said to be under the shadow of another character if they do

not physically touch each other but it is impossible to separate them merely by drawing a vertical line. In their paper, Bansal and Sinha proposed an approach to separate the core strip into characters and determine the conjunct/shadow characters based on the statistical information (such as average width, minimum and maximum width) of characters on text line. The approach obviously can not model our case because in the bilingual document, Hindi words and English words are usually interlaced. It is impossible to obtain one Hindi text line which contains words with the same size. Therefore, we separate the Hindi word into characters and determine conjunct/shadow characters based on the statistical information obtained from the current Hindi word. Before extracting the information, it must be noted that the modifier र् in the Hindi word has a much smaller width than the regular characters after removing the Header line, so this character can not be applied in the computation of statistical information of character width. Fortunately, this character is easily located based on the obtained stroke width in the first step. The separation of the core strip can be describe as:

(1)Using the method in step 2, separate the core strip into characters based on the vertical projection;

(2)For each separated character, if its width is smaller than $2StrokeWidth$, consider this as र् and remove it.

(3)Find the minimum width of remaining characters and denote this as W_{min};

(4)For each remaining character, if its width is greater than $1.5W_{min}$, remove this character because it may be a conjunct/shadow character which can affect the statistical information significantly;

(5)After removing the too-narrow and too-wide characters, compute the average width of the remaining characters and denote it as W_{avg};

(6)Traverse each separated character, if it is wider than $1.2W_{min}$, this character will be considered as a conjunct/shadow character which needs further segmentation;

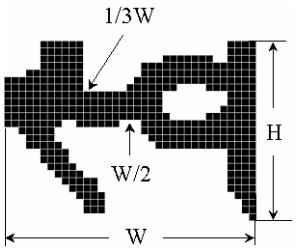
This procedure is shown in step 3 of Figure 5, where one conjunct character is located.

Step 4: Segmentation of the conjunct/shadow character. The segmentation of a conjunct/shadow character is complicated, and the segmentation of the conjunct character and shadow character is different. They are described as follows respectively:

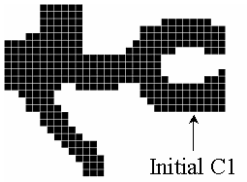
Segmentation of the conjunct character. The segmentation of the conjunct character is based on its *Collapsed Horizontal Projection CHP*, which is defined in [2] and [1]. *CHP(k)*, the collapsed horizontal projection of the k-th row of the image is 1 if at least one black pixel can be found in this row. Otherwise *CHP(k)* is 0. The location of the segmentation column contains two steps. First, segmentation column C1 is located by examining the right part of the conjunct image. Then segmentation column C2 is located by examining the left part of the conjunct image. The final segmentation column C is determined by co-relating C1 and C2.

Locate segmentation column C1: The procedure is illustrated in Figure 6. First we check if there exists a vertical bar in the right part of the image by computing the vertical projection of the right half part of the conjunct image. If there is a vertical bar, the bar and its right part will be removed from the conjunct image. The new image is shown in Figure 6(b). Suppose the right boundary of this new image is *nRight*, then the initial C1 is set at one stroke width to the left of *nRight*. Inscribing the right part of image between C1 and *nRight*, the *Collapse Horizontal Projection CHP* of the inscribed image is computed. If *CHP* has no discontinuity and the inscribed image is higher than *H/3*, C1 is the segmentation column. Otherwise, shift C1 one column left and repeat the above computation until the above criteria is satisfied. This procedure is shown in Figure 6(c), where the first two inscribed images have discontinuity, and the final C1 is shown in the last figure.

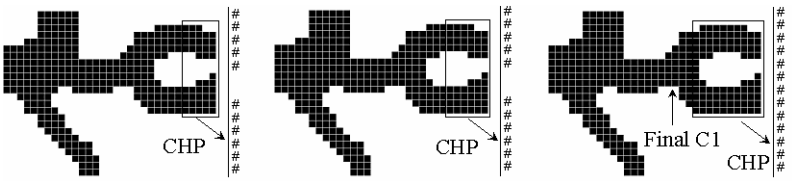
Locate segmentation column C2: Before setting the initial segmentation column C2, we check if there exists a vertical bar in the left half part of this conjunct image based on the vertical projection. If there is no vertical bar in the left part of the image, the initial C2 is set at *W/3* where *W* is the width of this conjunct image. Suppose the left boundary of this conjunct image is *nLeft*, we compute the height of the inscribed image between *nLeft* and *W/3*. If the height is less than *H/3*, then C2 is shifted one column right. If the inscribed image is higher than *H/3*, then C2 is shift one column right only if the pixel strength of the new column is not more than the present column. The pixel strength of the column is defined as the number of black pixels in this column. This procedure is shown in Figure 7, where the inscribed image is always higher than *H/3*, and the strength for the next column is shown in all three subfigures.



(a) Conjunct character



(b) Vertical bar removed



(c) Procedure to find C1

Figure 6: Segmentation of the conjunct character (to find C1).

If there is a vertical bar in the left part of the image, suppose the right column of the bar has location *bRight*, then the initial segmentation column C2 is set as *bRight+1*. Since the height of the inscribed image between *nLeft* and *C2* is always higher than *H/3* in this case, C2 is shifted one column right only if the pixel strength of the new column is not more than the present column. One example is shown in Figure 8.

Considering the fact that the left part of a conjunct character can not be wider than the right part, C2 must be smaller than *W/2*, which puts another stop condition for the determination of C2.

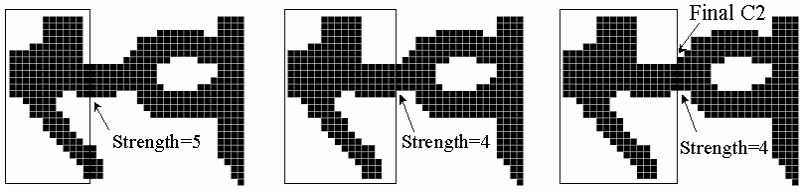


Figure 7: Segmentation of conjunct character (to find C2).

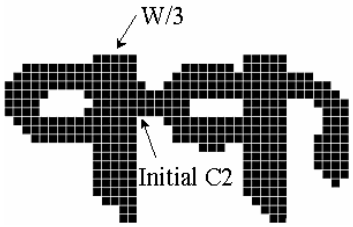


Figure 8: Example of conjunct character with two vertical bars.

Determine segmentation C by co-relating C1 and C2: Comparing C1 and C2, we could encounter the following three possibilities:

(a)C1 is less than C2: No further segmentation is needed, C1 and C2 are ignored.

(b)C1 and C2 are almost the same: If the difference between C1 and C2 is less than the stroke width, then segmentation column C is set as the average of C1 and C2.

(c)C1 is one stroke width larger than C2: The segmentation column C is set as C1, only the right part will be extracted. The remaining left part will be considered as a new conjunct character image.

Segmentation of the shadow character. The determination of a shadow character is straightforward. First we find the left most pixel of the character image, then we try to find the connected component starting from this pixel and obtain the bounding box of this connected component. If the right value of this bounding box is less than the right value of the original character image, then this character is considered a shadow character. The segmentation of a shadow character is shown in Figure 9. There are not many shadow cases in the Hindi words. Usually in the shadow character image, the right character is a character which can be represented as one single connected component. So the segmentation of a shadow character is from the right side. First we find the right most black pixel of the image, then find the connected component starting from this pixel using 8-neighbor tracing. The connected component is considered as the right character and separated from the original image. The left character is the remaining part with the detected connected component removed.

If should be noted that the above mentioned segmentation can also be extended to the segmentation of shadow top modifiers (low modifiers usually don't have a shadow part).

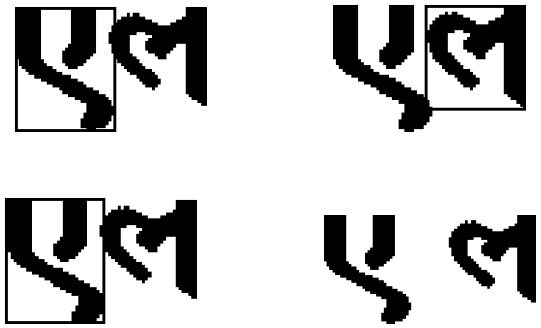


Figure 9: Segmentation of shadow character. (a) Determination of shadow character; (b) Bounding box of connected component (right character); (c) Bouding box of left character; (d) Segmented characters.

Three examples of shadow top modifiers are shown in Figure 10.

Figure 10: Examples of shadow top modifiers.

Step 5: Extract top modifiers. The extraction of top modifiers from the top strip (show in step 5 in Figure 5) is very simple. The header line is removed from the top strip first, then the vertical projection of the remaining strip is computed. The boundaries of top modifiers are located based on the column without any black pixels. There are some special cases that two top modifiers may touch each other, and they are separated as one single top modifier. Further segmentation of these top modifiers will be handled as a special case, which is described in subsection 2.3.5. As mentioned above, there exist cases that one top modifier is under the shadow of another top modifier, these cases can be handled using the same technique of segmenting shadow characters.

Step 6: Put header line back to the segmented characters. This step is straighforward, where header line is put back to each segmented core character for the recognition in the next step.

In the above description of operations in all steps, there are several constants defined. Some of the constants depend on the natural characteristics of Hindi character, such as the factors 0.4, 0.8, 0.64, 1.2, 1/4, 1/3, and 1/2, which are usually fixed even for different fonts

or different size of Hindi words. There are also some constants (such as the 5 and 10 when traverse the projection profile) which depend on the size of Hindi words, we choose these constants based on the experimental results, they can be fixed as long as the font has the standard font size used in regular documents, or can be changed based on the new size of font.

2.3 Recognition

The recognition of Hindi characters is based on the Hausdorff image comparison. Huttenlocher et. al. [6] proposed efficient algorithms of computing Hausdorff distance to compare the resemblance between two binary images with the assumption that there is only translation between two images. In this paper, the Generalized Hausdorff Image Comparison (GHIC) is applied to determine the resemblance of one point set to another, by examining the fraction of points in one set that lie near points in the other set and vice versa. There are two parameters used to determine the degree of resemblance of two point sets: (i) the maximum distance that points can be separated and still be considered close together; and (ii) what fraction of the points in one set are at most this distance away from points of the other set. Hausdorff-based distance measures differ from correspondence-based matching techniques, such as point matching methods and binary correlation, because there is no pairing of points in the two sets being compared. Often in matching and recognition problems, the two images are allowed to undergo some kind of geometric transformation in the matching process. In this case we are concerned with finding the transformations of one image that produces good matches to the other image. In the following section, we give a brief introduction of Generalized Hausdorff Image Comparison.

2.3.1 Generalized Hausdorff Image Comparison (GHIC)

Given two sets of points $A = \{a_1, ..., a_m\}$ and $B = \{b_1, ..., b_n\}$, the Hausdorff distance is defined as:

$$H(A, B) = max(h(A, B), h(B, A))$$

where $h(A, B) = max_{a \in A} min_{b \in B} ||a - b||$. The function $h(A, B)$ is called the *directed Hausdorff distance* from A to B (this function is not symmetric and thus is not a true distance).

It identifies the point $a \in A$ that is farthest from any point of B, and measures the distance from a to its nearest neighbor in B. Thus the Hausdorff distance, $H(A, B)$, measures the degree of mismatch between two sets, as it reflects the distance of the point of A that is farthest from any point of B and vice versa. It is well known that the Hausdorff distance is a metric over the set of all closed and bounded sets because it obeys the properties of identity, symmetry and triangle inequality.

The Hausdorff distance is very sensitive to even a single *outlying* point of A or B. For example, consider $A = B \cup x$, where the point x is some large distance D from any point of A. In this case $H(A, B) = D$, which is determined solely by the point x. Thus rather than using $H(A, B)$, a generalization of the Hausdorff distance (which does not obey the metric properties on A and B, but does obey them on specific subsets of A and B) is used. This generalized Hausdorff measure is given by taking the k-th ranked distance rather than the maximum, or largest ranked one:

$$h_k(A, B) = k\text{-}th_{a \in A} min_{b \in B} ||a - b||$$

where $k - th$ denotes the $k - th$ ranked value (or equivalently the quantile of m values). For example, when k=m, then $k - th$ is max. When k=$m/2$ then the median of the m individual point distances determines the overall distance. Therefore this measure generalizes the directed Hausdorff measure, by replacing the maximum with a quantile.

In the character recognition, we are interested in using the Hausdorff distance to identify instance of some *model* bitmap M (character template or sample) in one image bitmap I (the character image). The instance can only be translated. One way to phrase this problem is that we seek all translations $t \in R^2$ such that $h_k(M + t, I) \leq \delta$. The parameter k tells us how many of the model points should be near image points in order to classify a given translation as an instance of the model (i.e., we allow m-k of the m model points to be outliers). The parameter δ tells us how close each non-outlying model point must be to some image point.

In order to find each translation such that $h_k(M + t, I) \leq \delta$, we form $I' = I + C_\delta$ and then compute the correlation of M with I'. For each translation t of M with respect to I', the correlation determines p, how many points of M + t are superimposed with I' (the logical ***and*** of M+t and I'). Any translation where $p \geq k$ is a translation where an instance was

found, i.e. for which $h_k(M+t, I) \leq \delta$. We refer to p/m as the Hausdorff fraction for a given translation t (at some fixed δ). The Hausdorff fraction measures the percentage of M + t that lies near (within δ of) points of I.

The computation of $h_k(M+t, I) \leq \delta$ alone does not necessarily find very good instances of M in I, rather it finds portions of I that could contain M plus some other points. For instance, a totally filled in image will match any model at any translation. Thus we often use the other direction of the generalized Hausdorff measure, $h_k(M+t, I)$, to verify those translations where it was found that $h_k(M+t, I) \leq \delta$. This reverse Hausdorff fraction, ensures that a given portion of the points in the image (covered by the model array) are actually near points of M+ t. It thereby rules out situations such as a totally filled in image matching any model.

The details of generalized Hausdorff image measure can be found in [6]. In our recognition, forward and reverse distance thresholds are specified to compare the resemblance between character and template. Under the constraint of the two thresholds, there may still be more than one characters which satisfy the conditions, thus forward and reverse Hausdorff fractions are used to prune the character candidates. Furthermore, in our work, we also set two thresholds of these two fractions. The sum of the forward and reverse Hausdorff fraction are used compute the confidence (half of the sum which has value between 0.0 and 1.0) of character recognition, and the confidence value is useful for the following on processing, which will be discussed in Section 4.

2.3.2 Normalization of template and character images

Since we only consider the translation when computing the generalized Hausdorff image measure to perform recognition, the template and character images fed into the system must be normalized. In our work, we performed a very simple normalization based on the long edge of images, which normalize all core characters into images with long edge 32, and normalize all top and low modifiers into images with long edge 16 while preserving the aspect ratio. Denote the image's dimensions as W (width) and H (height). The normalization procedure is:

$$D_{max} = max(W, H)$$

if $low\ or\ top\ modifiers$, **then**

$Factor = 16/D_{max}$

else

$Factor = 32/D_{max}$

$NW = W \cdot Factor,\ NH = H \cdot Factor$

Resize the image into new image with size $NW \cdot NH$

2.3.3 Classification of characters

A vertical bar does not appear at the left end of a Hindi character. If a vertical bar is present, it either appears at the right end (End Bar) or in the middle (Middle Bar) of a Hindi character. Based on the presence and position of vertical bar and the conjunction number of character with the header line, all the core Hindi characters can be divided into the following six groups shown in Table 7:

Open Header	अ थ ध भ
One Conjunction End Bar	च ज ञ त न व ज़
More Conjunctions End Bar	ख घ झ प म य ष स ख़
Middle Bar	ऋ क फ क़ फ़
No Bar	इ उ ऊ ए ङ छ ट ठ ड ढ द र ह ड़ ढ़
Special Case	ग ण श ग़

Table 7: Classes of core Hindi characters.

The four characters ग ण श ग़ in the *Special Case* class are characters with an end bar, but after removing the header line and computing the vertical projection, each of these four characters will be split into two parts. The over-segmentation is handled in the next subsection. The character ए in the *No Bar* class is also special because it is the only *No Bar* character which has more than one conjunction with the header line.

2.3.4 Over-segmentation processing

In the above character segmentation procedure, there could exist over segmentation of characters, i.e. one single character could be segmented into two or more parts. Based on

the direction of over-segmentation, we divide this into two types, horizontal and vertical direction, and handle them separately.

The over-segmentation in vertical direction only happens to long characters such as ए ह and other strongly combined forms of consonants such as द्द द्ध द्व द्र and ङ्ग . After being segmented into two parts, the bottom part is usually rejected or incorrectly recognized during the procedure of recognition. This type of over-segmentation is handled by adding the over-segmented top part into the template and assigning them special codes. Once recognized, we know this is an over-segmented character, its following part will be put back. For example, Figure 11(a) shows some of the over-segmented characters we added to our templates, and characters in Figure 11(b) are the original complete forms.

Figure 11: Examples of over-segmented characters added in the template. (a) Over-segmented characters; (b) Original characters.

The handling of over-segmentation in horizontal direction is a little more complex because of the half-consonant. Taking the four characters in the Special Case class, their half form happens to be the left part of the over-segmented parts. To determine whether this is an over-segmented character depends on the following character. In these four cases, if they are over-segmented, then the next character can only be character ा, but this can not be determined until the next character has been recognized. We leave the handling of this type of over-segmentation to the 'Ligature processing' section.

2.3.5 Dealing with special characters

As mentioned above, it is not possible to add all special characters to the template because many of them are rarely used. In addition, adding more templates will significantly slow down the recognition. In our work, we provide a very simple scheme which allows the operator to add new special characters easily, which means the recognizer can easily be tuned to adapt to new cases. If the operator find one new character, he first classifies this character (end

bar, middle bar, no bar and so on) and puts this into the corresponding file which includes the template name and unicode of this character. Then he simply cuts this character from the image, save it into a TIFF file and put the filename into the training template directory. The recognizer will automatically read templates from the file and perform recognition. All the special characters listed in Table 5 and Table 6 were handled in our system.

Due to the quality of scanned document images, two or more top modifiers may touch each other, which can not be segmented using the approach described above. Some of these touched top modifiers are shown in Figure 12. These touched top modifiers are considered as special top modifiers, which are added to the class of top modifier. Fortunately there are not many such cases, so this case can be easily handled.

Figure 12: Examples of touching modifiers.

2.4 Ligature processing

Devanagari characters, like characters from many other scripts, can be combined or change shape depending on their context. A character's appearance is affected by its relation to other characters, the font used to render the character, and the application or system environment. These special characters are handled in subsection 2.3.5.

Additionally, a few Devanagari characters may result in a change in the order of the displayed characters. This re-ordering is not commonly seen in non-Indic scripts. One such character is ि, which is always displayed one consonant left of its real position. When exporting the codes of one Hindi word with such a character, the codes of characters must be reordered.

The Devanagari script is noted for a large number of consonant conjunct forms that serve as orthographical abbreviations (ligatures) of two or more adjacent letter forms. This abbreviation takes place only in the context of a consonant cluster.

Some independent characters such as ई ,ऐ ,ओ ,औ ,अं ,ो and ौ have a top strip which is

segmented and classified as top modifiers, when exporting the encoding of these characters, the top modifier should be put back to the corresponding core character to generate a correct code.

Independent characters such as आ , ओ , औ will be segmented into अ plus ा, ो, ौ. When exporting the encoding of these characters, the separated parts should also be put back to generate one correct single code.

All the above schemes are handled in our system. Since most of them are caused by the ligature of Devanagari script, we discuss this in the 'Ligature processing' part.

3 Experimental results

3.1 OCR Evaluation

The proposed system was applied to the 1083 pages of the Oxford Hindi-English dictionary [13] and to a collection of PDF-converted Hindi document images. The dictionary binding was burst and scanned at 400 DPI. The PDF-converted Hindi document images are obtained directly from the PDF file without the introduction of scanner noise, so they are considered ideal images.

An example of the scanned dictionary image is shown in Figure 13, where (a) is the original image, and (b) shows the identified Hindi words (with errors) and the segmented characters. The OCR result which combines the Hindi OCR and the Latin OCR is shown in Figure 14.

To evaluate the accuracy of this OCR, we randomly chose 7 pages and counted the number of Hindi words and characters recognized. The result evaluation is shown in Table 8.

From Figure 13, it can be seen that the identified Hindi words can be correctly segmented into isolated characters using the proposed approach. The evaluation result shown in Table 8 shows the recognition accuracy at character level reaches 87.75%, while accuracy at word level reaches about 67%. The experiment was done on scanned images which obviously contain noise, and the result is the pure recognition result without any spell checking and word correction based on dictionary search. We strongly believe, with the availability of Hindi language constraints and electronic text, correction techniques can be applied to the

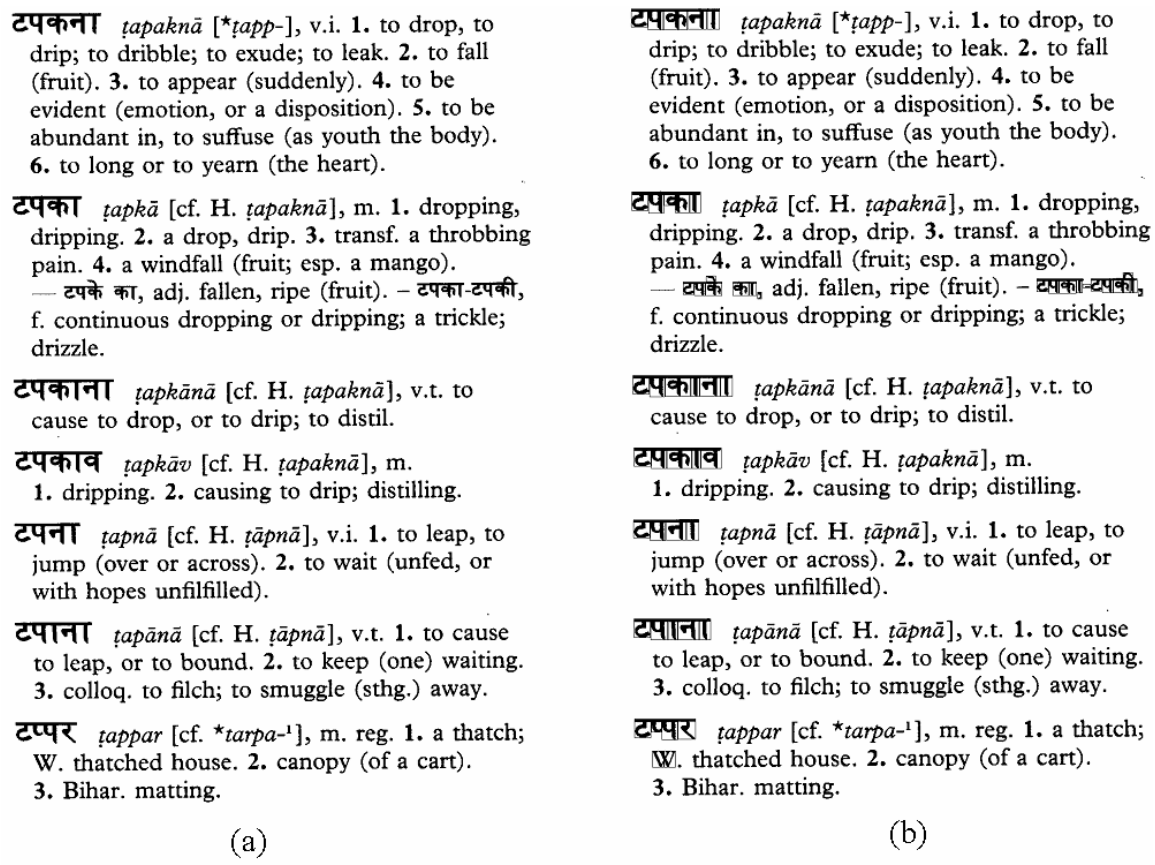


Figure 13: One example of the bilingual dictionary. (a) Original image; (b) Identified Hindi words and character segmentation.

Pages	Total Characters	Recognized	Correct	Accuracy1	Accuracy2	Total Words	Correct	Accuracy
p0098	451	443	407	90.24%	91.87%	110	79	71.82%
p0160	317	311	272	85.80%	87.46%	73	54	73.97%
p0179	480	477	409	85.21%	85.74%	113	67	59.29%
p0401	294	290	264	89.8%	91.03%	71	53	74.65%
p0799	437	451	379	86.73%	84.04%	80	50	62.5%
p0987	405	402	359	88.64%	89.3%	67	39	58.2%
p1023	343	338	303	88.34%	89.64%	64	44	68.75%
Total	2727	2712	2393	**87.75%**	**88.24%**	578	386	**66.78%**

Table 8: Result evaluation of the Hindi-English dictionary, where Accuracy1 is with respect to Total Characters and Accuracy2 is with respect to Recognized.

postprocessing stage of this system to improve the performance.

To test the effectiveness of the proposed approach working on ideal clean images, we processed PDF converted ideal images and evaluated them. The accuracy is about 95% (with 2584 characters and 2450 correctly recognized for one page). Figure 15 shows part of

टपकना *tapaknd [*tapp-],* v.i. 1. to drop, to drip; to dribble; to exude; to leak. 2. to fall (fruit). 3. to appear (suddenly). 4. to be evident (emotion, or a disposition). 5. to be abundant in, to suffuse (as youth the body). 6. to long or to yearn (the heart).

टपब्ष्ा *tapka* [cf. H. *tapakna], m.* 1. dropping, dripping. 2. a drop, drip. 3. transf. a throbbing pain. 4. a windfall (fruit; esp. a mango).

- टपके का, adj. fallen, ripe (fruit). - टपकाटपकी, f. continuous dropping or dripping; a trickle; drizzle.

टपकाव *tapkav* [cf. H. *tapaknd], m.* 1. dripping. 2. causing to drip; distilling.

टपना *tapnd* [cf. H. *tápnd],* v.i. 1. to leap, to jump (over or across). 2. to wait (unfed, or with hopes unfilfilled).

टपाना *tapand* [c£ H. *tapna],* v.t. 1. to cause to leap, or to bound. 2. to keep (one) waiting. 3. colloq. to filch; to smuggle (sthg.) away.

टप्पर *tappar* [cf. **tarpa-'], m.* reg. 1. a thatch; श्श thatched house. 2. canopy (of a cart). 3. Bihar. matting.

Figure 14: OCR results of images shown in Figure 13 (Reconstructed by combing Hindi and Latin results).

one converted ideal image and its OCR result.

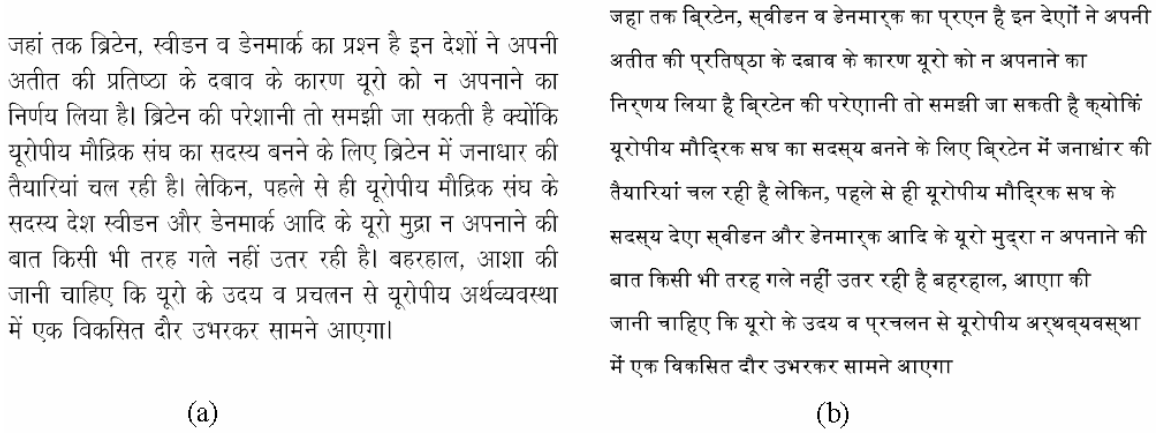


Figure 15: OCR result of the PDF converted ideal image. (a) Original image; (b) OCR result.

3.2 Discussion

In examining the data, we found that a number of factors contributed to incorrect recognition including:

(1) *Incorrect word segmentation.* The word segmentation performance is dependent on the quality of document images. Noise may cause the incorrect under- or over-segmentation of words on merging of words and other symbols. Incorrect word segmentation can further affect the script identification result, which leads to the degradation of OCR performance.

(2) *Incorrect character segmentation.* Segmentation is a challenging task, especially for Hindi scanned images. Due to the appearance of noise, average width and height of characters may not be obtained accurately. During segmentation, many decisions such as identifying conjunct/shadow characters, determining lower modifiers, and determining stroke width, are made based on these statistics. Incorrect statistics can significantly degrade the performance of character segmentation, and furthermore degrade the final performance of recognition.

(3) *Missing punctuation such as commas, periods and parenthesises.* Often the space between the words and the puctuation that follows is small. Punctuation can therefore be merged with Hindi words during word segmentation. The direct result is these symbols can negatively influence the distribution of the features used to perform segmentation. In our work, although we tried to handle this by detecting these symbols first, there are still some cases which can not be detected correctly.

(4) *Character mis-classification due to noise.* This typically happens with 'Open Header' and 'Middle Bar' characters. Noise may cause a 'Open Header' character to become a closed header character, or cause the detected vertical bar in a 'Middle Bar' character shifted. The classes typically do not overlapp, so if one character is mis-classified, it most likely will not be recognized correctly.

(5) *Character similarity.* There exist Hindi characters with similar appearance. Sometimes the scanning noise makes them almost indistinguishable. During recognition, they may have the same confidence value, which can then make the final selection of OCR output of this character ambiguous. Higher level context or language model may help fix this problem finally.

(6) *Special symbols which are noise-like.* There are some special symbols (including notations) which are visually noise-like, although the position of these symbols relative the rest of the text provides strong context. Sometimes these symbols are removed as noise, othertimes noise is left resulting in incorrect modifiers. This causes incorrect classification

and degrades the performance of OCR.

To improve the performance of the OCR, we need to consider all the factors discussed above and try to remove their affects. Since a majority of the incorrect recognition was caused by noise, applying new denoising techniques or make the parameters more flexible with the changing of image quality, the performance can be improved, a possible approach can be found in [15]. For the incorrect recognition caused by mis-classification, more rules, or even new classes could be added to make the classification more accurate.

4 Conclusion and future work

We have presented an adaptive Hindi OCR system which use a Generalized Hausdorff Image Comparison implemented as part of a rapidly retargetable language tool effort. The system includes three stages: (i)script identification; (ii)character segmentation; and (iii)training sample creation and character recognition. Based on the Generalized Hausdorff Image Comparison, the system is easy to retarget to different Hindi fonts or even a different script provided the segmentation can be handled using the same or similar approach. The OCR is also designed to handle unknown special characters, and provides a simple interactive interface to allow the user to add them. The OCR (designed from scratch in one month) was applied to a complete Hindi-English bilingual dictionary and a set of ideal images extracted from Hindi documents in PDF format. Experimental results show the average recognition accuracy can reach 87.82%, while for ideal image, the accuracy can reach 95%, both at character level and without any spelling checking and word correction.

A major thrust of future work will be to perform OCR correction or to resolve ambiguity among the candidates. One advantage of our approach is that we are giving a confidence as a side effect of recognition. By setting two thresholds on the Hausdorff distance, the forward and reverse Hausdorff fractions under the constraint of these two thresholds, can be used to compute the confidence of character and word recognition. This confidence is a real value between 0.0 and 1.0, which is much more intuitive and usable than current commercial OCR software such as *ScanSoft Developer's Kit 2000* from ScanSoft and *FineReader Engine* from ABBYY. For these packages, the confidence of each character is a boolean value, which gives the same weight to all characters when computing the confidence of a word.

Given the confidence of characters and words, we can further consider the word correction based on dictionary search. The word correction engine would determine whether a word need to be replaced with another correct word coming from the dictionary, which can significantly improve the recognition performance at both the character and word level. Since there is possibility that the recognized result may be over-corrected, the word correction can also provide a probability for the replacement, which makes the correction easily tuned. Details of this correction can be found in [7] and [8].

Another advantage of our approach is that we can consider adapting to different image qualities. For example, if the scanned document image quality is poor, the Hausdorff thresholds can be set to lower values, which makes the classifier and recognizer more tolerant to allow more choices. While if the image quality is high, the thresholds can be set to higher values, which can speed up the recognition procedure.

A next step for the recognition phase is to apply new, possibly multiclassifier techniques, and combine them with the current Hausdorff classifier to provide improved performance. The approach assumes we can trian the system using a small number of samples, so the new classification techniques must also have this property.

Our OCR system was designed under the assumption that no vast amount of training samples are available, so it can be easily extended for the recognition of other languages or scripts under the following two conditions: (i) the language uses symbols of the same basic class; (ii) the words of this new language can be segmented into characters. Under these two conditions, the segmentation can be changed based on the characteristics of new script, while the recognition will be adapted through exemplars if necessary. The user can create new rules to classify characters, obtain samples from document images, set output codes for each character, set thresholds for the Hausdorff distance, and perform recognition. If it is difficult to extract features which can be used to classify characters into different classes, the operator can put the whole set of characters into one single class, compute the Hausdorff distance between the segmented characters and each character in the single class, and then perform recognition.

Dealing with special characters can also be extended for the recognition of some symbols in cases which are difficult to segment correctly (such as complex ligatures). The user can

just add these parts into the samples and perform recognition. The only thing the user needs to do before performing recognition is to make sure the encoding of these special characters is correct.

Finally, a key to making our system generally adaptive is to consider how to allow system parameters to dynamically adjust for changes in image quality.

5 Acknowledgement

The support of this research under DARPA cooperative agreement N660010028910 and DOD contract MDA90402C0406 is gratefully acknowledged. Thanks to Vishal Khandelwal and Armrita Misra for doing the evaluation and explaining the encoding of special Hindi symbols.

References

[1] Veena Bansal. *Integrating Knowledge Sources in Devanagari Text Recognition.* PhD thesis, Indian Institute of Technology, Kanpur, India, 1999.

[2] Veena Bansal and R.M.K. Sinha. Segmentation of touching and fused devanagari characters. *Pattern Recognition*, 35:875–893, 2002.

[3] B.B. Chaudhuri and U. Pal. An ocr system to read two indian language scripts: Bangla and devanagari (hindi). In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 1011–1016, Germany, 1997.

[4] B.B. Chaudhuri and U. Pal. Skew angle detection of digitized indian script documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):182–186, 1997.

[5] J. J. Hull. Document image skew detection: Survey and annotated bibliography. In J. J. Hull and S. L. Taylor, editors, *Document Analysis Systems II.* Word Scientific, 1998.

[6] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[7] Okan Kolak and Philip Resnik. Ocr error correction using a noisy channel model. In *Human Language Technology Conference (HLT 2002)*, 2002.

[8] Okan Kolak, Philip Resnik, and William Byrne. A generative probabilistic ocr model for nlp applications. In *HLT-NAACL 2003*, 2003. to appear.

[9] Huanfeng Ma and David Doermann. Bootstrapping structured page segmentation. In *SPIE Conference Document Recognition and Retrieval*, pages 179–188, Santa Clara, CA, 2003.

[10] Huanfeng Ma and David Doermann. Gabor filter based multi-class classifier for scanned document images. In *7th International Conference on Document Analysis and Recognition*, pages 968–972, Edinburgh, Scotland, 2003.

[11] Huanfeng Ma and David Doermann. Application of three classifiers to word level script identification on scanned document images. In *SPIE Conference Document Recognition and Retrieval*, San Jose, CA, 2004. submitted.

[12] Huanfeng Ma, Burcu Karagol-Ayan, David Doermann, Jianqiang Wang, and Doug Oard. Parsing and tagging of bilingual dictionaries. *Traitement Automatique Des Languages*, 2003.

[13] R.S. McGregor. *The OXFORD Hindi-English Dictionary*. OXFORD DELHI, OXFORD UNIVERSITY PRESS, 1993.

[14] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(11):1162–1173, 1993.

[15] Yefeng Zheng, Huiping Li, and David Doermann. Text identification in noisy document images using markov random field. In *7th International Conference on Document Analysis and Recognition (ICDAR)*, pages 599–605, Edinburgh, Scotland, 2003.