2011 International Conference on Document Analysis and Recognition

Towards Improving the Accuracy of Telugu OCR Systems

P. Pavan Kumar*, Chakravarthy Bhagvati, Atul Negi, Arun Agarwal, B. L. Deekshatulu

Dept. of Computer and Information Sciences
University of Hyderabad
Hyderabad 500 046, INDIA
*Email:*pavan.ppkumar@gmail.com, {chakcs, atulcs, aruncs, bldcs}@uohyd.ernet.in*

***Abstract*—Design of a high accuracy OCR system is a challenging task as the system performance is affected by its component modules. Each module has its own impact on the overall accuracy of the OCR system. An improvement in a module reflects upon overall system performance. In the present work, we have developed an OCR system for Telugu. Our experiments on a corpus of about 1000 images has shown that the system performance is degraded due to broken characters caused by the binarization module as well as due to improper character segmentation. Therefore, we address the issues of handling broken characters and poor segmentation. A novel approach which is based on feedback from the distance measure used by the classifier is proposed to handle broken characters. For character segmentation, our proposed approach exploits the orthographic properties of Telugu script. As a result, significant improvement is obtained in the performance of the system. These algorithms are generic and may be applicable to other Indian scripts, especially to south Indian scripts. In our experiments, an end-to-end system performance is evaluated which is not reported in the literature.**

***Keywords*-OCR system; Telugu script; system performance; Indian scripts**

I. INTRODUCTION

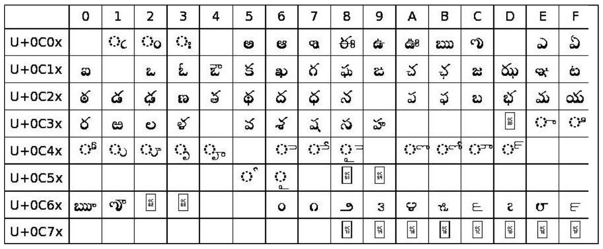
Design of a high accuracy OCR system is a challenging task as its performance is affected by various modules like pre-processing (noise removal, binarization, skew detection and correction etc), line, word and character segmentation, feature extraction, classification etc. The accuracy of the overall system depends on the accuracy of each module. Errors in any module propagate to successive modules and degrade the performance of the overall system. Therefore, there is an imperative need to handle errors in any of the modules to get a more robust OCR system. Most of these errors occur if the input document image is degraded.

In the present work, we consider an OCR system that is being developed for Telugu, as a part of consortium project funded by Govt. of India. A corpus of around 1000 scanned document pages taken from different books along with their annotated ground truth is created. Many of these documents are degraded as they are taken from old books which have bad print quality. In the literature other attempts to Telugu OCR [1], [2] have neither shown results on such large data nor reported end-to-end performance. From our experimentation on the corpus, we observe that the overall system performance is degraded by broken characters due to binarization and also by improper character segmentation. Here we describe a novel approach to handle broken characters based on feedback from the distance measure used by the classifier. Issues in character segmentation are tackled by a novel approach that exploits the orthographic properties of Telugu script. As a result, significant improvement is obtained in the overall accuracy of the end-to-end system.

The paper is organized as follows. Telugu script and its complexity is discussed in Section II. Section III gives an overview of Telugu OCR system. In Section IV, proposed algorithm for handling broken characters is discussed. In Section V, proposed algorithm for character segmentation is discussed. In Section VI, experimental results are shown and the paper is concluded in Section VII.

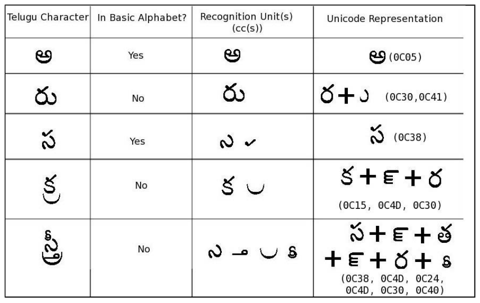
II. TELUGU SCRIPT AND ITS COMPLEXITY

As discussed in [1], [3]–[5], Telugu is a phonetic language, written from left to right with each character representing a syllable. Telugu alphabet consists of 50 letters with 14 vowels and 36 consonants. In addition, there are half vowel symbols called vowel modifiers and half consonant symbols called consonant modifiers. Telugu characters or *aksharas* are formed by the conjunction of vowel and consonant modifiers with basic consonants. These *aksharas* form an extended symbol set which is not a part of the alphabet (estimated to be around 10000 in the language) and create serious difficulty in the design of recognition process. An innovative and practical solution [3], [5], was to employ connected components (ccs) as basic recognition units, and to reduce the complexity to around 400 ccs. Now it is to be noted that each *akshara* could have more than one cc, and its ccs have to be arranged in order so that the corresponding valid UNICODE representation is produced. This ordering can be computed before or after their recognition. Violation of the ordering for an *akshara*, would result in an erroneous UNICODE representation. Further, a valid UNICODE representation of an *akshara* in Telugu is obtained by a composition of the UNICODE representation of a base consonant, followed by the UNICODE representation(s) of the consonant modifier(s) (if



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0C0x		◌ఁ	◌ం	◌ః		అ	ఆ	ఇ	ఈ	ఉ	ఊ	ఋ	ఌ		ఎ	ఏ
U+0C1x	ఐ		ఒ	ఓ	ఔ	క	ఖ	గ	ఘ	ఙ	చ	ఛ	జ	ఝ	ఞ	ట
U+0C2x	ఠ	డ	ఢ	ణ	త	థ	ద	ధ	న		ప	ఫ	బ	భ	మ	య
U+0C3x	ర	ఱ	ల	ళ		వ	శ	ష	స	హ				0C3D	◌ా	◌ి
U+0C4x	◌ీ	◌ు	◌ూ	◌ృ	◌ౄ		◌ె	◌ే	◌ై		◌ొ	◌ో	◌ౌ	◌్		
U+0C5x						◌ౕ	◌ౖ		0C58	0C59						
U+0C6x	ౠ	ౡ	0C62	0C63			౦	౧	౨	౩	౪	౫	౬	౭	౮	౯
U+0C7x									0C78	0C79	0C7A	0C7B	0C7C	0C7D	0C7E	0C7F

Figure 1. Telugu UNICODE chart



Telugu Character	In Basic Alphabet?	Recognition Unit(s) (cc(s))	Unicode Representation
అ	Yes	అ	అ (0C05)
రు	No	రు	ర + ు (0C30,0C41)
స	Yes	[illegible] ✓	స (0C38)
క్ర	No	క ◡	క + ్ + ర (0C15, 0C4D, 0C30)
స్త్రీ	No	[illegible] [illegible] ◡ [illegible]	స + ్ + త + ్ + ర + ీ (0C38, 0C4D, 0C24, 0C4D, 0C30, 0C40)

Figure 2. Some Telugu characters, their ccs and their UNICODE representation

any present) and finally followed by those of the vowel modifier(s) (if any present). This complexity is illustrated by some examples in Figure 2.

In Telugu, UNICODES [6] are defined only for the basic alphabet and vowel modifiers as shown in Figure 1. From the figure, we see that UNICODE codepoints 0C01 to 0C39 represent characters in the basic alphabet, while codepoints from 0C3E to 0C56 represent vowel modifiers, and codepoints from 0C66 to 0C6F represent Telugu numerals. Representation for consonant modifiers is arrived by composing the UNICODE representations for special symbol *halant* (UNICODE 0C4D), followed by the corresponding consonant. UNICODE representations and ccs for various Telugu aksharas are shown in Figure 2. First character is a vowel represented by single UNICODE. Third one is a consonant with two ccs, but has single UNICODE representation. These two characters are a part of basic alphabet. Second character is a combination of a base consonant and a vowel modifier and has one cc. It is represented by two UNICODES: base consonant and a vowel modifier. Fourth character has a consonant modifier with base consonant and it has two ccs, represented by three UNICODES: base consonant and the last two for the consonant modifier (*halant* followed by the corresponding consonant). Last example is one of the most complex characters commonly found in Telugu. It has a base consonant with two consonant modifiers and a vowel modifier and it has 4 ccs. As observed from the figure, its UNICODE representation is also complex: base consonant, *halant*, consonant corresponding to the first consonant modifier, *halant*, consonant corresponding to the second modifier and finally vowel modifier.

III. An Overview of Telugu OCR System

Telugu OCR system has various modules like pre-processing (binarization, skew detection etc.), Line, word and character segmentation, feature extraction, classification and finally followed by post-processing. For binarization, different methods are discussed in the survey paper [7] and on our corpus, adaptive methods show good performance. Line, word and character segmentation are performed based on projection profiles [8]. For classification, sophisticated classifiers like SVM, Neural Networks [9] etc, with different features [3], [10]–[12] are used and it is observed from the experimentation on our corpus that they do not perform well due to large number of classes/broken characters. K-NN (*K Nearest Neighbour*) classifier using distance measure based on fringe map feature [3] is used as it has shown good performance on our corpus.

Errors may occur at any of the above mentioned modules. For example, binarization may result in noise, broken or touching characters, line segmentation may combine two adjacent lines or split a line into two or more lines and word segmentation may combine two adjacent words or split a word in two or more words. Character segmentation may take ccs from or miss ccs to the surrounding characters or it may violate the order of ccs and classifier may give wrong labels to the ccs (recognition unit for Telugu). This error analysis at each module helps to improve its performance as well as the overall system performance.

As mentioned earlier, based on our experimentation on the corpus, it is observed that the major degradation in the system performance is due to the broken characters and improper character segmentation. To handle these issues, algorithms are proposed which are discussed in the following sections.

IV. Algorithm for Handling Broken Characters

Broken characters [13]–[15] are generally formed due to poor binarization or they can be introduced in the process of scanning the document itself. Our proposed algorithm handles broken characters obtained by any of the above two processes. The algorithm operates at word level in the sense, after word segmentation, for each word, ccs in a left to right order are inspected and broken ccs are combined. This combination is logical as it only modifies the minimum bounding rectangle (MBR) such that it covers both the ccs. For example, in Figure. 3(a), MBRs of two broken parts are shown and after combining the two broken parts, the expanded MBR which covers the two parts is shown in Figure. 3(b).

This algorithm is based on the idea that the broken or touching ccs exhibit higher values of the minimum distances to the templates in the database with any distance measure

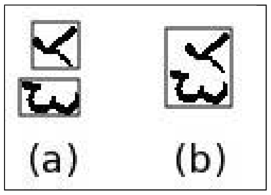


Figure 3. (a) MBRs of two broken parts shown in rectangles (b) Expanded MBR after combining the two broken parts shown in (a).

used by the classifier. The reason is that broken or touching ccs are not present in the template database and hence even the minimum distance as given by the distance measure is high. Higher value (greater than a threshold) of the minimum distance for a cc means that cc is abnormal (touching or broken) and vice-versa. Our approach uses this feedback from the distance measure to detect whether a cc is abnormal or not. For a given word, after its ccs are extracted, their minimum distances to the template database are computed. If an abnormal cc is detected, it is combined with its adjacent cc and the combination is checked for abnormality by again computing the minimum distance for the combination. If this distance is reduced, that means the combination approaches to a normal character and hence the combination is retained. Otherwise, abnormality is increased by the combination and the two ccs are not combined. Thus, once an abnormal cc is detected, it is combined with its adjacent ccs until the minimum distance is reduced. But sometimes even a touching cc may combine with the adjacent normal cc as the abnormality of the touching cc is reduced by the combination. Therefore, abnormality of the adjacent cc also has to be checked before combining a abnormal cc with it.

But to detect an abnormal cc, a threshold for the significant value of minimum distance has to be computed. In our case, as the distance measure is based on fringe maps, the threshold for the significant value of minimum fringe distance is computed by taking a huge number of ccs (around 3×10^5) from our corpus and plotting their minimum fringe distances. In that 1-D plot, a point (100) is found at which 90% of normal ccs fall below it and 90% of touching or broken ccs fall above it. That value is taken as our threshold (V) and for a given cc, if its minimum fringe distance is less than 100, it is normal. Otherwise, it is abnormal. The detailed algorithm is discussed below:

1) For each word in the document image, repeat the steps 2 to 12.
2) Arrange the ccs of the given word in a left to right order.
3) Combine ccs of the given word by performing the following steps. Initial combination has only first cc in the word.
4) Compute the fringe map of current combination of ccs and its minimum fringe distance (MFD) to the templates in the database.
5) Combine the current combination with the adjacent cc (if present) and compute the fringe maps of the resultant combination and the adjacent cc. Then find the MFDs of the resultant combination and the adjacent cc. If there is no adjacent cc, exit.
6) If MFD of the resultant combination is greater than MFD of the current combination (abnormality increased), do not combine adjacent cc with the current combination.
7) Otherwise, If MFD of current combination is greater than the threshold V (current combination of ccs is abnormal) and MFD of adjacent cc is less than V (adjacent cc is normal), (that means, a touching cc may be combined with a normal cc), then do not combine adjacent cc with the current combination.
8) Otherwise, if MFD of current combination is less than V (current combination is normal) and MFD of adjacent cc is less than V (adjacent cc is normal), then do not combine adjacent cc with the current combination.
9) If all of the above cases fail, then combine the adjacent cc with the current combination and the resultant combination now becomes current combination (this combination approaches to a normal shape) and go to step 4.

In Figure 4, our algorithm is illustrated. In Figure 4(a), it is observed that the first two ccs are broken parts of the last one and have significant (greater than 100) minimum fringe distance values 339.75 and 213.89 respectively and the last one has the value 36.69, which is reduced due to the combination and its shape approaches to a template in the database. Similarly, in Figure 4(b) first three ccs are broken and the last is the combined one and their distances are 120.9, 310.16, 1747.75 and 30.04 respectively. In Figure 4(c), two more examples of broken characters that are combined by the proposed approach are shown. In this figure, breaks are shown in rectangles. Finally, in Figure 4(d), a touching character which has the minimum fringe distance 555.84 is shown. Thus our algorithm is able to detect touching characters, but we do not handle them at this moment. The proposed method is applicable to other scripts as it can be used for any recognition unit and distance measure.

V. Algorithm for Character Segmentation

Character segmentation is a process of segmenting characters or *aksharas* from a word. Our algorithm for character segmentation is based on the idea that ccs in an *akshara* overlap horizontally with one another. This is not always be the case due to the complexity of Telugu script and hence a threshold which is based on font size is chosen for the overlap. The ccs of a word are divided into blocks where ccs in each block are horizontally overlapped as given by the threshold. Each block is considered as an *akshara*.

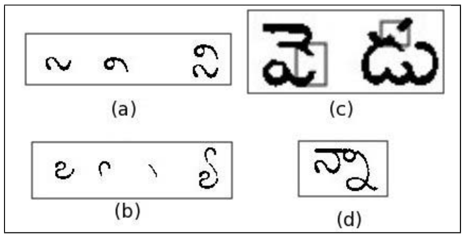


Figure 4. Illustrating proposed algorithm for handling broken characters: (a) First two ccs have minimum fringe distances 339.75 and 213.89 respectively and the last one as combined by the algorithm has the distance 36.69 (b) First three ccs are broken ccs and have distances 120.9, 310.16 and 1747.75 respectively and the last one as combined by the algorithm has the distance 30.04. (c) Two other broken characters handled by the proposed algorithm (breaks are shown in rectangles)(d) A touching character with minimum fringe distance 555.84.

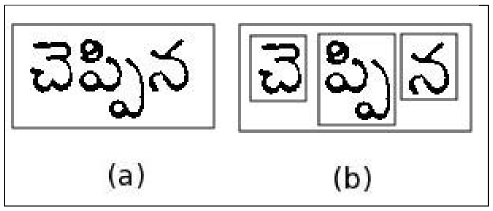


Figure 5. (a) A word image (b) *aksharas* (shown in rectangles) obtained after character segmentation on the word image shown in (a).

For example, a word image is shown in Figure. 5(a) and *aksharas* obtained after character segmentation on the word image are shown in Figure. 5(b) where each rectangle gives an *akshara*.

As discussed in Section II, ccs in each *aksharas* have to be ordered (base followed by consonant modifiers and then by vowel modifiers). This ordering can be done after their recognition as their types (base, consonant or vowel modifier) are also known after recognition. But a different approach is followed in our OCR system. For each *akshara*, the types of its ccs are found before recognition and based on the types, the ccs are ordered in the required manner. The advantage in finding the type of a cc before its recognition is that it is then matched against only the templates of that type in the template database. Thus it reduces recognition time and the recognition becomes more accurate as the number of templates to be matched are reduced. That means, our approach also serves as a first level classifier as it performs type recognition.

The type of cc in a given *akshara* is found based on the properties of Telugu script. For any *akshara*, base cc has more thickness and size than the ccs corresponding to the consonant and the vowel modifiers and this nature is exploited in our approach. A cc with maximum black pixel count (an estimate of thickness and size) in the given *akshara* is isolated and its type is assigned as base and finding types of other ccs becomes trivial once the base cc is known. If a cc is below (above) the base cc, its type is assigned as consonant (vowel) modifier. The above process is repeated for all the *aksharas* in the word. The algorithm is discussed below:

1) Extract the ccs of a given word and compute their MBRs. Let the left, top, right and bottom coordinates of a MBR be denoted by *xmin*, *ymin*, *xmax* and *ymax* respectively. Here, it is assumed that x-axis goes from left to right and y-axis from top to bottom.
2) Sort the ccs of the word in the left to right order.
3) Find the black pixel count of each cc in the word.
4) Divide the ccs of the word into blocks (each block comprises an *akshara*), such that ccs in each block are horizontally overlapped by atleast $0.25 \times f$ where f is the estimated font size (computed in the line segmentation module).
5) For each *akshara* obtained in the previous step:
    a) Find a cc with the maximum black pixel count. Assign its type as base (B).
    b) For each other cc C in the *akshara*:
        i) If *ymax* of C is less than *ymax* of B, then C is of type vowel modifier. Otherwise, C is of type consonant modifier.
6) For each *akshara*, order its ccs such that base cc should appear first, later the ccs for the consonant and vowel modifiers respectively. If there is more than one consonant modifier, arrange them in the increasing order of *ymax*.

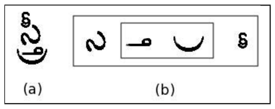


Figure 6. (a) A complex Telugu *akshara* (b) Ordered ccs for the *akshara* shown in (a).

In Figure 6(a), a complex Telugu *akshara* is shown and the ordered ccs obtained by our algorithm are shown in Figure 6(b). In Figure 6(b), first cc is base, next two are consonant modifiers (shown in the internal box) and the last one is vowel modifier (last one). The character shown in Figure 6(a) and the last character shown in Figure 2 are same. It can be observed from both the figures that the order generated by our algorithm is same as the one required for the UNICODE representation. The proposed method is applicable to other scripts, especially to south Indian scripts which are similar in nature to Telugu script.

VI. Experimental results

After incorporating the proposed algorithms into our OCR system, the previous and the improved OCR systems are compared on around 1000 images in the corpus. The error rate for a given page is computed by using a traditional string matching algorithm, *Levenshtein edit distance* [16]. Given two strings, Levenshtein edit distance gives the minimum number of substitution, insertion and deletion operations needed to convert one string to the other. For a given page, all the UNICODES of the ground truth text are taken into one string and all the UNICODES of OCR text output into the other string and Levenshtein edit distance is computed between them. The ratio of this distance to the number of UNICODES in the ground truth text gives the overall error rate for the given page. The overall error rate as reported by this distance can be considered as the accumulation of error rates of the individual modules as errors at any module ultimately reflect in the text output. As mentioned earlier, no one in the literature has reported an end-to-end system performance and also no one has used such a large corpus having a lot of degraded documents. In [2], the authors have reported only symbol recognition accuracies.

Table I
RANGES OF ERROR RATES AND THE NUMBER OF PAGES IN THAT RANGE FOR BOTH THE PREVIOUS AND THE CURRENT OCR SYSTEMS

[1]Total Number of pages = 969

Error Rates (%)	Previous system	Current system
≤ 10	232	367
> 10 and ≤ 15	515	448
> 15 and ≤ 20	134	84
> 20	88	70

In Table I, the percentage error rates on 969 pages[1] are shown. In that table, three columns show the ranges of error rates and the number of pages falling in that range for both the previous and the improved systems respectively. It is observed from the table, the number of pages which have error rates less than 10% is increased to 367 (from 232) for the current system. For the current system, the number of pages in the other ranges is reduced as most of these pages move to the first range (less than 10%).

VII. CONCLUSION

In the present work, we propose algorithms for handling broken characters and character segmentation. These algorithms show significant improvement in the performance of our Telugu OCR system. To handle broken characters, an algorithm is proposed which detects abnormal ccs based on feedback from the distance measure and combines them. Our proposed algorithm for character segmentation also determines the type of cc whether it is base, consonant or vowel modifier and hence serves as a first level classifier.

The previous and the improved Telugu OCR systems are compared on around 1000 scanned images taken from our corpus. The results on them show that the error rates are significantly reduced for the current system. The proposed algorithms are also applicable to other Indian scripts, especially to south Indian scripts which are similar in nature to Telugu script.

ACKNOWLEDGMENT

The authors thank the Ministry for Communications and Information Technology (MCIT), New Delhi, Government of India, for providing financial support under the grant No. 14(6)/2006-HCC(TDIL).

REFERENCES

[1] C. V. Lakshmi and C. Patvardhan, "An optical character recognition system for printed Telugu text," *Pattern Analysis and Applications*, vol. 7, no. 2, pp. 190–204, 2004.

[2] C. V. Lakshmi, R. Jain, and C. Patvardhan, "OCR of printed Telugu text with high recognition accuracies," *Computer Vision, Graphics and Image Processing*, pp. 786–795, 2006.

[3] A. Negi, C. Bhagvati, and B. Krishna, "An OCR system for Telugu," *ICDAR'01*, pp. 1110–1114, 2001.

[4] A. Negi, K. N. Murthy, and C. Bhagvati, "Foundational issues of document engineering in indic scripts and a case study in Telugu," *Vivek*, vol. 16, no. 2, pp. 2–7, 2006.

[5] C. Bhagvati, T. Ravi, S. M. Kumar, and A. Negi, "On developing high accuracy OCR systems for Telugu and other indic scripts," in *Proceedings of the Language Engineering Conference (LEC'02), pp 18-23*, 2002.

[6] http://UNICODE.org/charts/PDF/U0C00.pdf.

[7] O. Trier and A. K. Jain, "Goal-directed evaluation of binarization methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1191–1201, 1995.

[8] K. Y. Wong, R. G. Casey, and F. M. Wahl, "Document analysis system," *IBM Journal of Res. Develop.*, vol. 26, no. 6, pp. 647–656, 1982.

[9] V. Govindaraju and S. Srirangaraj, *Guide to OCR for Indic Scripts*. Advances in Pattern Recognition, Springer, 2010.

[10] S. Rajasekaran and B. Deekshatulu, "Recognition of printed Telugu characters," *Computer Graphics and Image Processing*, vol. 6, no. 4, pp. 335–360, 1977.

[11] A. K. Pujari, C. D. Naidu, M. S. Rao, and B. C. Jinaga, "An intelligent character recognizer for Telugu scripts using multiresolution analysis and associative memory," *Image and Vision Computing*, vol. 22, no. 14, pp. 1221–1227, 2004.

[12] G. Anuradha, "An investigation into Telugu font and character recognition," Ph.D. dissertation, University of Hyderabad, Hyderabad, April 2009.

[13] V. Bansal and R. Sinha, "A complete OCR for printed Hindi text in Devanagari script," *ICDAR'01*, pp. 800–804, 2001.

[14] B. B. Chaudhuri and U. Pal, "An OCR system to read two indian language scripts: Bangla and Devnagari (Hindi)," *ICDAR'97*, vol. 2, pp. 1011–1015, 1997.

[15] C. V. Jawahar, M. N. S. S. K. P. Kumar, and S. S. R. Kiran, "A bilingual OCR for Hindi-Telugu documents and its applications," *ICDAR'03*, vol. 1, pp. 408–412, 2003.

[16] D. O. Richard, H. E. Peter, and S. G. David, *Pattern Classification(2nd Edition)*. pp 413-421: Wiley, 2000.