

EMBEDDED SPELL CHECKER FOR OPTICAL CHARACTER RECOGNITION SYSTEM FOR INDIAN LANGUAGES (HINDI)

A DISSERTATION

*Submitted in partial fulfilment of the
requirements for the award of the degree*

of

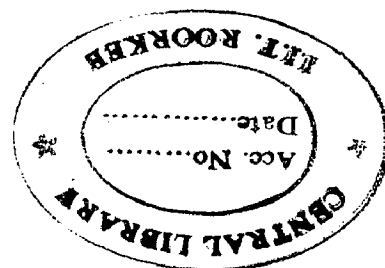
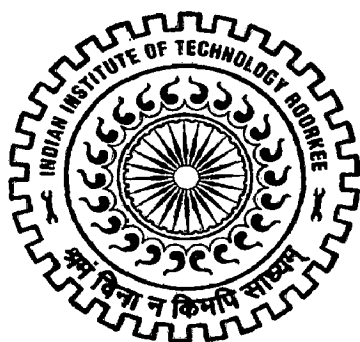
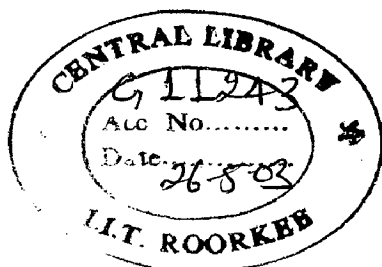
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

NAVEEN KUMAR JAYANT



**ER & DCI
NOIDA**

**IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307**

FEBRUARY, 2003

621.380285

JAY


CANDIDATE'S DECLARATION

This is to certify that the work, which is being presented in this dissertation, entitled **“EMBEDDED SPELL CHECKER FOR OPTICAL CHARACTER RECOGNITION SYSTEM FOR INDIAN LANGUAGES”**, in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology** submitted in **IIT, Roorkee – ER&DCI Campus, Noida**, is an authentic record of my own work carried out from August 2002 to February 2003, under the supervision of **Mr. V.N. Shukla**, Director (Special Applications), Electronics Research and Development Centre of India, Noida.

I have not submitted the matter embodied in this dissertation for the award of any other degree.

Date: 24/02/03

Place: Noida


(Naveen Kumar Jayant)


CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 24/02/03

Place: Noida

Guide:



(Mr. V.N. Shukla)

Director, Special Applications,
ER&DCI, Noida

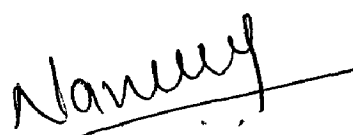
ACKNOWLEDGEMENT

I hereby take the privilege to express my deepest sense of gratitude for **Prof. Prem Vrath**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K. Verma**, Executive Director, ER&DCI, Noida for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Prof. A.K. Awasthi**, Dean PGS&R IIT-Roorkee, our program coordinator and **Prof. R.P. Agrawal** course coordinator M.Tech (IT) IIT-Roorkee for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I would like to thank my guide, **Mr. V.N. Shukla**, Director Special Applications, ER&DCI, Noida, for his guidance and invaluable suggestions during the entire course of this work. He provided me continuous inspiration and support throughout the course of this dissertation.

I am highly indebted to **Mr. B.K. Sahu**, Senior Project Engineer, and **Mr. Shailendra Jayant**, Project Engineer, Natural Language Processing Laboratory, ER&DCI, Noida, for without their constant support this work could not have been completed. My sincere thanks are due to **Dr. P.R. Gupta**, Reader, ER&DCI, and Noida for the continuous inspiration and support, she provided me with throughout the course of this dissertation. I am also grateful to **Mr. Munish Kumar**, Project Engineer, and ER&DCI, Noida for the cooperation extended by him in the successful completion of this work.

It is impossible to mention the names of all those persons who have been involved, directly or indirectly, with this work and I extend my gratitude to all of them. However, I feel, I owe special thanks to all my friends who have helped me formulate my ideas and have been a constant support. I find myself short of words to thank my father, mother and brother who have always been by my side throughout my life.


(Naveen Kumar Jayant)
Enrollment No: 019027

CONTENTS

CANDIDATE'S DECLARATION	(i)
ACKNOWLEDGEMENT	(ii)
ABSTRACT	1
1. INTRODUCTION	3
1.1 Objective	3
1.2 Scope	3
1.3 Statement of work done	3
1.4 Organization of Dissertation	4
2. LITERATURE SURVEY	5
2.1 Optical Character Recognition	5
2.2 Commercial Applications	6
2.3 Devanagari Script	7
2.3.1 Notable Features	8
2.4 Indian Standard Code for Information Interchange (ISCII)	10
2.4.1 Nature Of Indian Alphabet	12
2.4.2 Structure of the ISCII code	15
3. THEORETICAL ASPECTS OF THE PROBLEM	19
3.1 OCR Error Correction	19
3.2 OCR Error Pattern of HINDI Text	21
3.3 Categorization of Errors	21
3.4 Converter	25
3.5 Dictionary Design	26
3.5.1 Minimum Distance Algorithm	27
3.5.2 Searched Procedure	28
4. IMPLEMENTATION ASPECTS	29
4.1 Working With Dictionary	29
4.2 Working with Converter	29
4.3 Working with Error Module	30

5. RESULTS AND DISCUSSION	37
5.1 Results	37
5.2 Suggestions For Future work	43
6. CONCLUSION	45
REFERENCES	47
Appendix A: Layout of the Hindi OCR Software	49
Appendix B: PC-ISCII Code Table	50
Appendix C: MFC Classes and method used in software	51

ABSTRACT

This work is an attempt to develop “Embedded Spell Checker Module for Indian Language (Hindi)”. The spell checker module developed to identify various types of errors and to correct them in Hindi text. The OCR recognizes various types of errors, which commonly occur in Devanagari. A special property of the present technique is that the multiple errors in a single word can also be corrected. The errors are detected and corrected in two steps. In first step, detection of invalid Hindi patterns is performed and in second step heuristically correction of the word followed by the dictionary look-up for generation of probable correct options is done.

This Spell Checker Module checks the various errors on scanned text document. An Optical Character Recognition System takes scanned image of a text document as input and produces the document in Text format. When the OCR provides the document, document has some recognition errors. When the document is provided to this Spell Checker module, this module scans whole the document and recognizes the wrong word in the given text and corrects each and every wrong word.

The software has been developed as one of the modules of ‘Chitraksharika’, an Optical Character Recognition system for Devanagari script being developed at ER&DCI, Noida.

INTRODUCTION

1.1 Objective

To design and develop an Embedded Spell Checker for Optical Character Recognition System for Indian Languages (Hindi).

1.2 Scope

The embedded Spell Checker for the OCR System must be able to achieve the following objectives:

(i) It should be able to recognize the error and try to correct them. These errors may be categories in different ways

a) Recognition Error

b) Grammatical Error

(ii) In the Spell Checker if user does not want any change for the selected word than he can ignore the word. Because the selected word could be correct but in the document it shows incorrect. This feature should be in the module.

(iii) It should be able to display best four options for the selected wrong word and user can choose any one option for the wrong word and that wrong word can be easily replaced by the correct word. If user want more options than this feature should be in the module.

(iv) If user want to insert the word in the dictionary than user can add the selected word into user directory, so that in future selected text can not identify incorrect word for the same word.

1.3 Statement of work done

The software is being developed as one of the modules of the Optical Character Recognition (OCR) System for Indian Languages. When the OCR run over the scanned page the characters do not recognize correctly and there are some recognition errors in the word. The error module corrects those errors, which are incorrect and provide the

correct document to the user. The software is an attempt to provide the above features to an OCR System for Indian Languages.

In this dissertation, we design a module, called Spell Checker Module, which detect the errors on the document and make correction of the word followed by the dictionary look-up for generation of probable correct options.

Some specific tasks which is carried out in this dissertation are as follows:

1. PC-ISCII to KrutiDev 010 Converter.
2. KrutiDev 010 to PC-ISCII Converter.
3. Detection of errors and generation of probable correct options for Hindi text.

1.4 Organization of Dissertation

The first chapter gave an overview of Spell Checker module and discussed the problem to be solved. The second chapter describes the literature Survey for OCR for Hindi Language, ISCII code. The third chapter carries out a detailed analysis of the problem and discusses relevant theoretical issues, and discusses the various type of error and features of spell checker. Chapter fourth presents the detailed design of the proposed solution. In chapter fifth, results obtained from the software developed are presented and discussed. Concludes the thesis in sixth chapter.

LITERATURE SURVEY

2.1 Optical Character Recognition

OCR stands for Optical Character Recognition. It is the process of converting scanned images of machine printed or handwritten text (numerals, letters, and symbols), into a computer process able format (such as ASCII). Optical character recognition (OCR) is the translation of optically scanned bitmaps of printed or written text characters into character codes, such as ASCII. This is an efficient way to turn hard-copy materials into data files that can be edited and otherwise manipulated on a computer.

The OCR systems are useful for direct transfer of large data into the computer with the help of scanner. The hard copy is fed into a scanner, the image file is then analyzed and image processing is applied for transforming into ASCII file. Efforts are underway to develop OCR for Hindi. The software system that recognizes characters from a registered image can be divided into three operational steps: document analysis, character recognition, and contextual processing

Document Analysis

Text is extracted from the document image is a process known as document analysis. Reliable character segmentation and recognition depend upon both original document quality and registered image quality. Prior to character recognition it is necessary to isolate individual characters from the text image. Many OCR systems use connected components for this process. For those connected components that represents multiple or partial characters, more sophisticated algorithms are used.

Character Recognition

Two essential components in a character recognition algorithm are the feature extractor and the classifier. Feature analysis determines the descriptors, or feature set, used to describe all characters. Given a character image, the feature extractor derives the

features that the character possesses. The derived features are then used as input to the character classifier.

Template matching, or matrix matching, is one of the most common classification methods. In template matching, individual image pixels are used as features. Classification is performed by comparing an input character image with a set of templates (or prototypes) from each character class. Each comparison results in a similarity measure between the input character and the template. One measure increases the amount of similarity when a pixel in the observed character is identical to the same pixel in the template image. If the pixels differ the measure of similarity may be decreased. After all templates have been compared with the observed character image, the character's identity is assigned as the identity of the most similar template.

Contextual Processing

Contextual information can be used in recognition. The number of word choices for a given field can be limited by knowing the content of another field. The result of recognition can be post-processed to correct the recognition errors. One method used to post process character recognition results is to apply a spelling checker to verify word spelling.

2.2 Commercial Applications

Commercial OCR systems can largely be grouped into two categories: task-specific readers and general purpose page readers. A task-specific reader handles only specific document types. Some of the most common task-specific readers read bank cheque, letter mail, forms or credit card slips. These readers usually utilize custom-made image lift hardware that captures only a few predefined document regions. For example, a bank check reader may just scan the courtesy amount field and a postal OCR system may just scan the address block on a mail piece. Such systems emphasize high throughput rates and low error rates. Applications such as letter mail reading have throughput rates of 12 letters per second with error rates less than 2%. The character recognizer in many of task-specific readers is able to recognize both handwritten and machine-printed text.

Address Readers: The address reader in a postal mail sorter locates the destination address block on a mail piece and reads the ZIP Code in this address block.

Form Readers: A form reading system needs to discriminate between pre-printed form instructions and filled-in data. The system is first trained with a blank form. The system registers those areas on the form where the data should be printed. During the form recognition phase, the system uses the spatial information obtained from training to scan the regions that should be filled with data.

Cheque Readers: A cheque reader captures check images and recognizes courtesy amounts and account information on the cheque.

Bill Processing Systems: A bill processing system is used to read payment slips, utility bills and inventory documents. The system focuses on certain regions on a document where the expected information is located, e.g. account number and payment value.

Passport Readers: An automated passport reader reads a traveler's name, date of birth, and passport number on the passport and checks these against the database records that contain information on fugitive felons and smugglers.

General-purpose page readers are designed to handle a broader range of documents such as business letters, technical writings and newspapers. These systems capture an image of a document page and separate the page into text regions and non-text regions. Non-text regions such as graphics and line drawings are often saved separately from the text and associated recognition results. Text regions are segmented into lines, words, and characters and the characters are passed to the recognizer. Recognition results are output in a format that can be post processed by application software. Most of these page readers can read machine written text but only a few can read hand-printed alphanumeric.

2.3 Devanagari Script

Origin: The Devanagari alphabet descended from the Brahmi script sometime around the 11th century AD. Its name come from the Sanskrit deva 'heavenly' + nagari 'script of the city'.

Devanagari is a compound word with two roots: deva means "immortal", and nagari means "city". Literally, together they mēan "city of immortals."

2.3.1 Notable Features

This is a syllabic alphabet in which consonants all have an inherent vowel. This vowel can be muted with a special diacritic called a virama.

Vowels can be written as independent letters, or by using a variety of diacritical marks, which are written above, below, before or after the consonant they belong to. This feature is common to most of the alphabets of South and South East Asia.

When consonants occur together in clusters, special conjunct letters are used. The symbols for the consonants other than the final one in the group are reduced and the inherent vowel only applies to the final consonant.

Devanagari - A script that is used by over 300 million people all over the world. It is the base script of many languages in India, such as Hindi and Sanskrit, and many more languages throughout India and the world use variants of this script. From an OCR point of view, Devanagari owes its complexity to its rich set of conjuncts. The richness of this script deserves a more detailed explanation.

1. अस्पताल में भरती रोगी की मेडिकल रिपोर्ट, एक्स-रे अथवा अल्ट्रा साउंड आदि की रिपोर्ट अपने लैप-टॉप कंप्यूटर पर देखकर उचित सलाह दी जा सकती है।
2. किसी उद्योग संस्थान की कार्यशाला में कार्यरत कर्मी द्वारा किए जा रहे कार्यों को मुख्यालय में 'लाइव' देखा जा सकता है।
3. युद्ध के क्षेत्र में युद्ध के दृश्यों को ऑपरेशनल मुख्यालय में 'लाइव' देखकर कमांडर द्वारा उचित निर्देश दिए जा सकते हैं। इसी प्रकार के अन्य अनेक कार्यों को मोबाइल इंटरनेट की सहायता से देखा व सुना जा सकता है।

Figure 2.1: Sample of Hindi text

A sample of text in Devanagari is shown in Figure 2.1 as is obvious from the character structure, Devanagari has a wide range of glyphs and is a challenging subject vis a vis the OCR of Devanagari documents. Add to that issues like fused characters, the

क़ ख़ क़ ग़ च़ छ़ ज़ ज़
 ट़ ड़ ङ़ ड़ त्क़ त् स्र ख़ ड़ क़ क़ द्र व

Figure 2.4 compound characters

Devanagari text can be represented in 2 popular ways - Transliteration and Unicode formats. Both formats are widely used, though each makes its' own claim towards having covered the whole Devanagari character set. A transliteration map is shown below. Transliteration is used to convert English alphabets into Devanagari characters. It is based upon phonetic translation.

अ आ इ ई उ ऊ ऋ ॠ ए ऐ ओ औ
 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

Figure 2.5 Consonants and vowels

2.4 Indian Standard Code for Information Interchange (ISCII)

In 1991, the Bureau of Indian Standards adopted the Indian Standard Code for Information Interchange (ISCII), the ISCII standard that was evolved by a standardization committee, of which C-DAC was a member, under Department of Electronics during 1986-88. The ISCII document is available as IS13194: 1991 from the BIS offices.

All GIST products are based on ISCII. Other than this, IBM for PC-DOS, Apple for ILK has used it, and several companies are developing products and solutions based on this representation. This has been made mandatory for the data being collected by

organizations like The Election Commission and for projects as Land Records Project etc.

The ISCII code standard specifies a 7-bit code table, which can be used in 7 or 8-bit ISO compatible environment. It allows English and Indian script alphabets to be used simultaneously. It shall not be used in incompatible environments like that of IBM-PC, and with computers which do not allow 8-bit characters, or which do not follow ISO code extension techniques. It cannot be used in the 5-bit Baudot code used for telecommunications.

The Bureau of Indian Standards adopted this Indian Standard after the Electronics and Telecommunication Division Council have approved the draft finalized by the Computer Media Sectional Committee.

This standard conforms to IS 10401:1982, "8-bit coded character set for information interchange"(equivalent to ISO 4873). It is intended for use in all computer and communication media which allow usage of 7 or 8 bit-character set - code extension techniques".

In an 8-bit environment, the lower 128 characters are the same as defined in IS 10315:1982 (ISO 646 IRV) "7-bit coded character set for information interchange" also known as ASCII character set. The top 128 characters cater to all the 10 Indian scripts based on the ancient Brahmi script. In a 7-bit environment the control code SI can be used for invocation of the ISCII code set, and control code SO can be used for reselection of the ASCII code set.

An attribute mechanism has been provided for selection of different Indian script font and display attributes. An Extension mechanism allows use of more characters along with the ISCII code. These are only meant for the environment where no other alternative selection mechanism is available. The ISCII code table is a super-set of all the characters required in the ten Brahmi-based Indian scripts. For convenience, the alphabet of the official script Devanagari has been used in the standard.

In a 7-bit environment the control code SI can be used for invocation of the ISCII code set, and control code SO can be used for reselection of the ASCII code set. The official language of India, Hindi is written in the Devanagari script. Devanagari is also used for writing Marathi and Sanskrit.

2.4.1 Nature of Indian Alphabet

All the Indian scripts have originated from the ancient Brahmi script, which is phonetic in nature. The alphabet in each may vary, they all share a common phonetic structure. The differences between scripts primarily are in their written forms, where different combination rules get used.

The Consonants

Indian script consonants have an implicit + (a) vowel included in them. They have been categorized according to their phonetic properties. There are 5 Vargs (Groups) and non-Varg consonants. Each Varg contains 5 consonants, the last of which is a nasal one. The first four consonants of each Varg, constitute the Primary and Secondary pair. The second consonant of each pair is the aspirated counterpart

	Primary		Secondary		Nasal			
Varg 1	क	ख	ग	घ	ङ			
	ka	kha	ga	gha	ṅa			
Varg 2	च	छ	ज	झ	ञ			
	ca	cha	ja	jha	ña			
Varg 3	ट	ठ	ड	ढ	ण			
	ṭa	ṭha	ḍa	ḍha	ṇa			
Varg 4	त	थ	द	ध	न			
	ṭa	ṭha	ḍa	ḍha	ṇa			
Varg 5	प	फ	ब	भ	म			
	pa	pha	ba	bha	ma			
non-Varg								
	य	र	ल	व	श	ष	स	ह
	ya	ra	la	va	śa	ṣa	sa	ha

Note that the consonants श (śa) ष (ṣa) are pronounced identically today.

Anuswar

Anuswar indicates a nasal consonant sound. When an Anuswar comes before a consonant belonging to any of the 5 Vargs, then it represents the nasal consonant belonging to the Varg. Before a non-Varg consonant however the anuswar represents a

Varg 1 अङ्क=अंक aṅka	पङ्ख=पंख paṅkha	गङ्गा=गंगा gaṅgā	सङ्घ=संघ saṅgha
Varg 2 मञ्च=मंच mañca	पञ्ची=पंछी pañchī	पञ्जा=पंजा pañjā	साञ्ज=सांझ sāñjha
Varg 3 घण्टा=घंटा ghaṅṭā	कण्ठ=कंठ kaṅṭha	झण्डा=झंडा jhaṅḍā	दूण्ड=दूंद dhūṅḍha
Varg 4 सन्त=संत santa	पन्थ=पंथ pantha	बन्द=बंद banda	गन्ध=गंध ganḍha
Varg 5 चम्पा=चंपा santa	गुम्फ=गुंफ pantha	खम्बा=खंबा banda	स्तम्भ=स्तंभ ganḍha

different nasal sound. Some Hindi examples:

Nasalization Sign: Chandrabindu

The ̣ denotes nasalization of the preceding vowel (can be implicit अ vowel within a consonant).

Example: औंख, पौंच, हुमायूं, हेँ, मेँ.

In Devanagari script it often gets substituted with Anuswar, as the latter is more convenient for writing. In some words, however, Anuswar and Chandrabindu can give different meanings. Hindi example: हैंस (Laugh), हंस (Swan).

Visarg

Comes after a vowel sound, and represents a sound similar to "h".

Vowels and Vowel signs (Matras)

There are separate symbols for all the vowels in Indian scripts which are pronounced independently (either at the beginning of a word, or after a vowel sound). The consonants in the Indian script themselves have an implicit vowel + (a). To indicate a

vowel sound other than the implicit one, a vowel-sign (Matra) is attached to the consonant. Thus there are equivalent Matras for all the vowels, excepting the + vowel.

Roman	ā	i	ī	u	ū	ṛ	ṅ
Vowel	आ	इ	ई	उ	ऊ	ऋ	ॠ
Matra	ा	ि	ी	ु	ू	ृ	ं
Matra on क	का	कि	की	कु	कू	कृ	कं
Roman	ē	ai	ē	o	ō	au	ō
Vowel	ए	ऐ	ऎ	ओ	औ	औ	औ
Matra	े	ै	ँ	ो	ौ	ौ	ौ
Matra on क	कै	कं	कँ	को	को	को	को

Vowel Omission Sign: Halant

In Indian scripts consonants are assumed to have an implicit vowel + "a" within them unless an explicit Matra (vowel-sign) is attached. Thus a special sign Halant is needed for indicating that the consonant does not have the implicit + vowel in it. In

Ashok = अशोक => अशोक

Northern languages, the Halant at the end of a word generally gets dropped, though the ending still gets pronounced without a vowel. Example:

Conjuncts

Indian scripts contain numerous conjuncts, which essentially are clusters of up to four consonants without the intervening implicit vowels. The shape of these conjuncts can differ from those of the constituting consonants. These conjuncts are formed in the ISCII code by putting the Halant character, between the constituent consonants.

Example: क्षत्रिय = क्श्त्रिय
 कर्म = कर्म्
 क्रम = क्रम्

Diacritic Mark: Nukta

The Nukta is used for deriving 5 other consonants in the Devanagari and Punjabi scripts, required for Urdu.

क ख ग ज ड ढ फ
क ख ग ज ड ढ फ

Punctuation:

All punctuation marks used in Indian scripts are borrowed from English, except for the full-stop, instead of which a Viram (|) is used in the Northern scripts. The Viram is, however, being increasingly substituted by a gull-stop. A double Viram (||) is also used in Sanskrit texts for indicating a verse ending.

Numerals

Many Indian scripts today use only the international numerals. Even in others, the usage of international numerals instead of the original forms is increasing. Although the Devanagari script has its own numerals, the official numeral system is the international one.

2.4.2 Structure of the ISCII code

A common alphabet for all the Indian scripts is made possible by their common origin from the same ancient Brahmi script. The ISCII code contains only the basic alphabet required by the Indian scripts. All the composite characters are formed through combinations of these basic characters.

Vowels and Matras

The ISCII code contains separate vowels and Matras (Vowel signs). While a vowel sign can be used independently, the Matra sign is valid only after a consonant. Thus:

कई = क ई, की = क ी

Vowel Modifiers

After a consonant, vowel or Matra character, a character can be used which modifies the vowel sound and is called a "Vowel Modifier". This can be a Chandrabindu,

Anuswar or Visarg

Example:

हंस = ह ं स, अंत ् अ ं त, अता ् अ त ा

Halant

The implicit vowel in a consonant can be removed by addition of a Halant sign. In the ISCII code conjuncts are formed by typing a Halant character between consonants. A conjunct may consist of upto 4 consonants joined by Halants. Example:

क ् त = क्त	श ् र = श्र
श ् य = श्य	ष ् ट ् र = ष्ट्र
क ् ष = क्ष	त ् र = त्र
ज ् य = ज्य	र ् द ् य = रद्य

Explicit Halant

A Halant is used between consonants to form conjuncts. But many times in Sanskrit and Vedic texts, one may wish to show an Explicit Halant which would be shown on the previous consonant, and which would prevent the consonant from joining with the next one. Two consecutive Halants form an Explicit Halant. Example:

क ् त	= क्त	क ् ् त	= क्त
क ् त ि	= क्ति	क ् ् त ि	= क्ति
ड ् क ि	= ड्कि	ड ् ् क ि	= ड्कि
ट ् र ि	= ट्रि	ट ् ् र ि	= ट्रि

Soft Halant

A Soft Halant is formed by typing a Nukta character after a Halant. In Devanagari the Soft Halant allows retention of the "half form" for the preceding consonant, and prevents it from combining with the following consonant. Example:

श ् य	= श्य	श ् ् य	= श्य
क ् त ि	= क्ति	क ् ् त ि	= क्ति

Invisible Consonant INV

The INV (Invisible) code is used for formation of composite characters which require consonantal base, but where the consonant itself ought to be invisible. These may be required only for some special display purposes. Example:

क् INV=क्	INV ्र = ्र
र् INV=र्	INV ृ = ृ
INV ि = ि	INV ी = ी

The Nukta Character

The Nukta consonants get formed by adding a Nukta character immediately after the appropriate consonant. In the ISCII code the same Nukta character is thought of as an operator to derive some of the lesser used Sanskrit characters which are not directly available on the Inscript keyboard. A Nukta can be typed after a Halant to form a Soft Halant.

Table 4: ISCII characters derived by appending a Nukta

Char	Nukta Char	Name
क	क्	Consonant QA (Urdu)
ख	ख्	Consonant KHHA (Urdu)
ग	ग़	Consonant GHHA (Urdu)
ज	ज़	Consonant ZA (Urdu)
ड	ड़	Consonant Flapped DA
ढ	ढ़	Consonant Flapped DHA
फ	फ़	Consonant FA (Urdu)
ऋ	ऋ़	Vowel RII (Sanskrit)
ॠ	ॠ़	Vowel Sign RII (Sanskrit)
इ	इ़	Vowel LI (Sanskrit)
ि	ि़	Vowel Sign LI (Sanskrit)
ई	ई़	Vowel LII (Sanskrit)
ी	ी़	Vowel Sign LII (Sanskrit)
ं	ं़	Sign OM
।	।़	Vowel Stress Sign AVAGRAH (Sanskrit)

Attribute Code (ATR)

The Attribute code, followed by a displayable ASCII character, defines a font attribute applicable for the following characters. The mechanism is meant for use in that medium where alternative font selection mechanism is not available.

Extension Code (EXT)

The Extension code, followed by an ISCII character, defines a new character which can combine with the previous ISCII character. This provision has been primarily made for supplementing Vedic signs along with the Devanagari text.

Numerals

In all the Indian scripts the international numerals are being used increasingly.

From the software viewpoint, usage of the same numerals as given in the ASCII set allows proper handling of numerals by existing software. For display rendition purposes however, it may be sometimes desirable to have separate Indian script numerals which are given in the ISCII table. The ATR mechanism also allows display rendition of the ASCII numerals in an Indian script form. The ISCII numerals should be used only when it is not possible to use the ATR mechanism for selecting numerals in an Indian script.

THEORETICAL ASPECTS OF THE PROBLEM

3.1 OCR Error Correction

OCR (Optical Character Recognition) error detection and Correction technique for a highly inflectional Indian language. The technique is based on morphological parsing where using two separate lexicons of root words and suffixes, the candidate root-suffix pairs of each input string, are detected, their grammatical agreement is tested and the root/suffix part in which the error has occurred is noted. The correction is made to the corresponding error part of the input string by means of a fast dictionary access technique. To do so, the information about the error patterns generated by the OCR system is examined, and some alternative strings are generated for an erroneous word. Among the alternative strings, those satisfying grammatical agreement in root and suffix are finally chosen as suggested words.

The problems of automatic error detection in words and automatic correction are great research challenges. Finding solutions will be very important for text and code editing, computer aided authoring, OCR, machine translation and natural language processing (NLP), database retrieval and information retrieval interfaces,

Word errors belong to either of two distinct categories, namely, non-word errors and Real word errors. Invalid words are non-words. By real word error, we mean a valid but not intended word in a sentence, thus making the sentence syntactically or semantically ill formed or incorrect. In both cases, the problem are to detect the word error and either suggests correct alternatives or automatically replace it with the appropriate valid word. The detection of real word errors needs higher-level knowledge compared to the detection of non-word errors. For real word errors, it is often not possible to separate the problem of error detection from that of correction.

Here we are concerned with non-word errors only. We assume that the document subject to OCR system contains only valid words and that the sentences are syntactically and semantically well formed. Of course, the OCR still can create real word errors, but as will be shown later, such errors are rare.

We are concerned here error correction of Hindi, the most popular language in India. The complex character grapheme structure of Hindi script creates difficulty in error detection and correction. There are 12 vowels and 34 consonants in the Hindi alphabet. These are known as basic characters.

But the vowels, depending on their position in a word, take different shapes, called modifiers. You may observe that some of the characters have upper and lower modifiers. Vowel modifiers and their shapes when attached to a consonant. Moreover, two or more basic character combine to form a new complex shaped character called a compound character. Making recognition as well as error detection and correction more difficult than, say that of Roman script.

Vowel modifiers

प	पा	पि	पी	पु	पू	पृ	पृ	पे	पै	पो	पौ
pa	pā	pi	pī	pu	pū	pr	pṛ	pe	pai	po	pau

Figure 3.1 Vowel modifiers

Compound character

क्क	क्ख	क्त	ग्ध	च्च	च्छ	ञ्ज	ज्ज					
ट्ट	ट्ठ	ड्ड	डु	त्क	त्त	स्त्र	क्ष	ड्ड	कृ	क	द	व

Figure 3.2 Compound characters

Using a huge lexicon for error detection and correction is not a very practical proposition. For speed and storage efficiency, it is necessary that the lexicon size be reasonably small while tackling the large number of surface words using a clever suffix handling approach. For this purpose, two lexicons are used in our system, one for the root words and the other for the suffixes. Given a character string, first we look for matches in the root word lexicon, and then we search in the suffix lexicon for suffix matches. When properly done, this approach has an additional advantage in that it performs morphological parsing of words, which is the first stage in any natural language processing (NLP) task. We shall very briefly describe in section 5 how morphological parsing is conducted. Moreover, the basic structure of the approach presented here is very suitable for the design of spellcheckers in word processing applications for Hindi and other Indian languages.

3.2 OCR Error Pattern of HINDI Text

A document image captured by flatbed scanner is subjected to initial processing, such as skew correction and line, word and character segmentation. For convenience of recognition, the basic, modified and compound characters are classified into separate groups. A feature based tree classifier does basic and modified character recognition. A decision is made at each node of the tree based on certain feature(s) from the feature set. The compound characters are initially grouped into small subsets by a feature based tree classifier. A run length based template-matching approach then recognizes characters in each group. Characters, which are not recognized with the system, are marked by the marker '?'. There are few errors, which are generally recognized in the document. We have categorized them below.

3.3 Categorization of Errors

We have identified the different types of errors and categorized them as under.

- Matra error
- Exchange of matra with vowel error
- Nukta error
- Halant error
- Consonant Exchange error
- Splitting of word error
- Phonetic error

3.3.1 Matra error

In Hindi consonants do not indicate the consonant sounds only. They stand for a particular consonant + vowel (अ). Thus (क) is not simply क but क, ँ. But if the simple consonant without inherent (अ) is to be expressed then (Halant (ँ)) is to be put below the letter. When some vowel other than this inherent (अ) comes after a consonant then Matra is tagged on to a consonant letter.

In Hindi-Writing putting the correct Matra on the consonant is a very common error due to the phonetic similarity in the Matras. To correct Matra errors we generate the

consonant pattern of the incorrect word , which is the Key for searching in the Dictionary. Dictionary structure is designed in such a way that the word list having same consonant pattern get selected in the single search step. On this word list, We apply the “Minimal Distance Algorithm” (MDA) and the most suitable option is displayed for the user. The “MDA” uses the following pairs of Matras, which has been identified based on the phonetic similarity.

(ि ि) (ु ू) (ॅ ै) (ि ि) (ॅ ै)

And a special class (Matra No-Matra)

For Example, If incorrect word is पूरूष

and generated correction option words are पौरूष पुरूष

Penalties for these words will be 2 1 respectively

Hence the option words display sequence will be पुरूष पौरूष

3.3.2 Exchange of matra with vowel error

This is a special kind of Matra error where the abbreviated form of vowel i.e. the Matra is written in the actual form of the vowel due to the phonetic similarity. This sort of error generally occurs due to the pronunciation difference caused by regional effect.

For Example:

Incorrect	Correct
भइया	भैया

For such cases Matra & Vowels tuples are identified. Using these tuples the words are generated and these generated words are searched in the dictionary to prove their validity.

3.3.3 Nukta error

There are some loan words from Arabic, Persian, and Turkish which have become very common in Hindi. These words can have Nukta consonants which are

formed by adding a special symbol Nukta (.) after the equivalent Hindi consonants. This is a small set of consonants, which may carry a Nukta character immediately after them.

This Nukta error generally occurs either due to missing the Nukta on the required consonant or unnecessarily putting the Nukta on the consonant or putting Nukta after the consonant where Nukta can never occur.

These three types of Nukta errors are dealt by the following way.

- a) Checking the Nukta consonant in the word.
- b) Putting/Removing the Nukta from the consonant
- c) Searching the generated words in the dictionary for the validity.

3.3.4 Halant Error

To represent the consonant without the inherent (अ) is expressed by putting the Halant after the consonant. And two or more consonants with no vowel including the inherent (अ) between them can be combined together and thus form a conjunct. A conjunct may consist of up to 4 consonants joined by Halants.

for example क + ण + या = क्णया

It is however, not usual to write a conjunct with the help of a Halant. This mark is rarely used except the final consonant. Most of the consonant, when forming a conjunct, omit a part of their original form. This time Halant is called soft Halant. But some consonant are written explicitly putting the Halant, this time it is called Hard Halant.

It has been analysed that generally two types of a mistake occurs for this character.

- Putting a Soft Halant instead of Hard Halant.
- Or missing the Halant at last as with some words of Sanskrit at the last consonant.

To correct this error conjuncts have been identified for ISCII codes because Dictionary data uses ISCII storage which is the basis of UNICODE.

3.3.5 Consonant Exchange error

This error occurs due to the exchange of adjacent characters in a word. This may happen due to

a) Regional way of speaking

Example

Incorrect	correct
मतबल	मतलब

b) Wrong sequence of key depressions

Incorrect	correct
हय	यह

These are handled by mutually exchanging the characters of the word. It has been observed that this type of exchange does not occur with first and the last characters of the word.

3.3.6 Splitting of word error

For virtually all spelling error detection and correction techniques, word boundaries are defined by white space characters. This fact turns out to be problematic since a significant portion of text error involve running together one or more words or splitting of the word. In this paper only splitting of the word error has been dealt with, provided the splitted words in itself are not the valid words.

3.3.7 Phonetic error

This type of error occurs due to the placement of the phonetically similar characters. To dealt with this error, a list of pairs for phonetically similar character have been identified.

क ग	ग घ	च ज	ब व	ब भ
र ङ	स श	श ष	द त	ध थ

ठ ढ	छ झ	ख घ	फ भ	न ण
ड ड़	छ क्ष	ड ढ	ड़ ढ़	ट ड

Error correction is done just substituting the other character from the pairs.

Examples

Incorrect	Correct
वर्श	वर्ष
खरगोस	खरगोश
छत्रिय	क्षत्रिय
पड़ाई	पढ़ाई
घटग	घटक

3.4 Converter

Although PC-ISCII has been declared as a standard, still developers of Hindi Word-processors have not adopted it fully. The different vendors are using their own proprietary storage formats (Font). This is done knowingly too, to avoid a User from using any Word-processor of other vendor. Though it is a negative Approach of trade, it is prevailing widely in the market. To make the spell-check Module able to work along with different vendors, the font converters are also developed as a part of our project work. These converters first convert a vendor's Proprietary font format to PC-ISCII and after doing the spell check, the data is Converted back to the vendor's own font format. As explained earlier, the dictionary is in the standardized PC-ISCII format. So, Before applying the search module, the given text is converted into PC-ISCII Format. And after the completion of search procedure & spell-checking, the options that are generated, are reformatted back into the working context of the text, So, for each of the text format (font), we are required two converters.

The converters, we have designed so far are-

- - Kruti Dev 010 to PC-ISCII

- PC-ISCII to Kruti Dev 010

The concept behind the development of converters is the mapping that is applied for the transformation of the character in one font to another font. The mapping shows how the different characters are stored in the memory and what are their relative sequences.

For example

In kruti Dev 010 the word is written as

विभाग

In PC-ISCII it is stored as

व ि भ ा ग

3.5 Dictionary Design

Data representation is crucial to the design of a dictionary. Some of the important consideration in the design of a dictionary is storage requirement, search efficiency for the words, and the complexity of the retrieval of related words. Here dictionary data is organized on the basis of mode of writing the vowel with the consonant. The dictionary stores all the inflectional variants of the words. The Devanagri script text is represented in the form of ISCII. The ISCII code contains only the basic alphabets required by the Indian scripts. All the composite characters are formed through combinations of these characters.

In the said technique, the patterns have been generated from the valid words containing actual forms of the vowels & consonants for which basic ISCII code is available. All These patterns are organized as key to the Hash Table and the words satisfying these patterns are arranged horizontally as the value to the key of Hash Table. To implement this structure Hash Function is used.

This technique of words organization has unique feature and very useful for handling multiple character error in a single word. It also makes the searching process fast.

Dictionary is designed in such a way that it can take care of multiple Matra error in a single word. This is limited to the Matra errors only.

Incorrect	Correct
सोहार्दपुर्ण	सौहार्दपूर्ण
चीकीत्सा	चिकित्सा
अनूसुचीत	अनुसूचित

3.5.1 Options Display sequence by Minimum Distance Algorithm (MDA)

MDA used for ranking of generated possible correct options for a given incorrect word. The distance between two words is the number of editing operations required to transform one of the words into the other multiplied by respective penalties. The allowed editing operations are removal, insertion, and replacement of Matra or a Character. Different editing operations are classified into five different classes a, b, c, d and e and they have penalties p_1 , p_2 , p_3 , p_4 and p_5 respectively. Hence the following formula is applied for calculation of total penalty on the generated option word with respect to the incorrect word.

$$P = n_1 * p_1 + n_2 * p_2 + n_3 * p_3 + n_4 * p_4 + n_5 * p_5$$

Where

- P is the total penalty for generated word -option, with respect to incorrect word.
- n_1, n_2, n_3, n_4, n_5 are the editing operation counts for classes a, b, c, d, e respectively
- p_1, p_2, p_3, p_4, p_5 are the penalties for classes a, b, c, d, e respectively
- class 'a' contains phonetically similar Matra pairs and editing operation is assumed changing of Matra with other Matra of the same pair
- class 'b' contains Matra and No-Matra pair, special class and editing operations can be insertion and removal of Matra
- class 'c' can have any phonetically dis-similar Matra pair and editing operations can be insertion, deletion and exchange of Matra
- class 'd' can have any (character, character) and (character, No-character) pair and editing operations can be insertion, deletion and exchange of character

- class 'e' can have (Halant/Ref, blank) pair and editing operation can be insertion and deletion
- Values for p1, p2, p3, p4, and p5 are assigned 1, 1, 2, 2 and 3 respectively.

Penalties on each generated option words are calculated with respect to the incorrect word. Minimum of these Penalty counts will be Minimum Distance. Now generated option words are displayed in the increasing order of penalties.

This technique takes care of not only for correction of errors resulting from keyboard input but also for correction of phonetic spelling errors.

For example :

if incorreced word is चत्र

and generated correction option words are

चित्र चतुर चरित्र चित्रों

Penalties for these words will be

1 3 2 3 respectively

Hence the option words display sequence will be चित्र चरित्र चित्रों चतुर

3.5.2 Searched Procedure

In the first step as the input word arrives, it is first processed by the remove_matra () function to remove all the matra, halant & nukta to get the corresponding without matra word. On the basis of the without matra word thus obtained, the search procedure is fired that can result into the following possibilities-

Here two cases can arise:

a) The Without – matra word does not match to any word (Without – matra word) in the dictionary. This condition also shows that the desired word is not present in the dictionary.

b) The Without – matra word matches to the word (Without – matra word) in the dictionary. Here this line of the dictionary is picked up & is stored in array.

IMPLEMENTATION ASPECTS

4.1 Working With Dictionary

Dictionary is the important part of this module. When the spell checker module is run, initially this dictionary is loaded and the word is searched in the dictionary. The words are in sorted form in the dictionary.

The root word, without matra word and its variants are stored in line. In the dictionary information is stored in the ISCII form and when the word is searched it return the variants corresponding to the root word.

The dictionary is maintained in such a way that it supports index sequential access to the data stored therein. To fast retrieval of data, two- depth indexing is used...The first level index tells the dictionary file & the initial position for the word. To be searched. The second level index tells the no of lines (count) to check for the word to be searched. The whole of the dictionary is implemented in the split form. The dictionary (information) is split upon the basis of the first character that may be there in the ISCII word.

4.2 Working with Converter

The font converters are also developed as a part of our project work. These converters first convert a vendor's Proprietary font format to PC-ISCII and after doing the spell check, the data is Converted back to the vendor's own font format. The dictionary is in the standardized PC-ISCII format. So, before applying the search module, the given text is converted into PC-ISCII Format. And after the completion of search procedure & spell checking, the options that are generated are reformatted back into the working context of the text.

The concept behind the development of converters is the mapping that is applied for the transformation of the character in one font to another font. The mapping shows how the different characters are stored in the memory and what are their relative sequences.

Kruti Dev 010 to PC-ISCII
PC-ISCII to Kruti Dev 010

4.3 Working with Error Module

The functions, which are using in the spell checker module, are shown here. Each function is described in details.

4.3.1 Consonant Errors

This type of error occurs by the misplacement of the letters in the words.

Example

मतबलमतलब

Search Criteria:

Generally in this type of error the first letter remains the same. Means, keeping the first letter untouched, we calculate the permutations of the next three letters (max $3! = 6$ words). Then we carry out the search procedure (in the file opened corresponding to the first letter) to check out the validity, i.e. to see if the words are there in the dictionary or not. The valid words then can be placed in the option list.

Function : consonantErrors()

4.3.2 Nukta Error

This type of error occurs due to misplacement or non-placement of the nukta (.) characters in the word.

Example

जमाना ज़माना

Search Criteria

There are only seven characters that can have nukta mark.

1). First find out the positions of the letters that can have nukta error.

2). Depending upon the letters found, say n, we fire n! Nukta correction permutations and search for their validity. The valid words then can be represented in the option list.

Function: nuktaError()

4.3.3 Upper_r_kar Error

This error occurs due to the misplacement of upper_r_kar (८).

Example

दुदर्शा दुर्दशा

SearchCriteria

- 1) First find out the position of upper_r_kar
- 2) Remove the upper_r_kar from the word and make the following two cases-
 - a). Put upper_r_kar on the character succeeding the original position and check out for the validity.
 - b). Put uoooep_r_kar on the character preceding the original position and check out for the validity.

Function: upperRKarError()

4.3.4 Nmn_r_kar_error Error

This type of error occurs due to the placement of र.....or...रि. In place of (८) and vice-versa.

Example

ब ज

बृज ब्रज

Search Criteria

Find out the position of , *(\ + र) * in the word and replace it with (८) and vice versa.

Function: nmnrKarError()

4.3.5 Mnv_kar_error Error

This error occurs due to placement of \ before or after र .

Examples

प्रमाणुपरमाणु

Search Criteria

- 1). Find out the positions of before or after र .
- 2). Remove them in various permutations (max.3) and search for their validity.

Function: mnrKarError()

4.3.6 Soft_halant_error Error

This error occurs due to placement of non-placement of soft halant- (\ + .)

Example

गन्तवय गन्तवय

Search Criteria

A letter having a matra can t have a soft halant

- 1). First check the letter with no matra.
- 2). Now make the permutations of the selected letters (except the last one) and apply the soft halant to each of the permutations and check for validity.

Function: softHalantError()

4.3.7 Hard_halant_error Error

This error occurs due to the presence of special character combination which should be replaced by specific character.

Example

क्षत्रीय.....क्षत्रिय

Search Criteria

1). Find out the letters without matra (except on the last one). Chose the first letter from this list

2). Now put a halant () on this letter

3). After putting the halant, if the combination of this letter and the next consecutive letter makes a new letter (as shown above) then it is replaced by the specific character and is left intact and both of there letter are removed from the list if then are present. Chose the next letter form the list and go to the step (2).

4). The new word is checked for validity.

Function: hardHalantError()

4.3.8 Halant_at_last_error Error

This error occurs due to the presence or absence of halant (्) at the last character.

Example

नवम.....नवम्

Search Criteria

1). If halant is present at the last character then remove it.

2). If halant is not is present at the last character then put it.

3). Check for the validity in each of the cases.

Function: halantAtLastError()

4.3.9 Explicit_halant_error Error

This error occurs due to placement or non-placement of soft halant(ँ)

Example

कतिकृति

Search Criteria

A letter having a matra can't have a explicit halant.

1). First check the letter with no matra.

2). Now make the permutations if the selected letter (except the last one and apply the soft halant to each of the permutations and check for validity.

Function: Explicit_halant_error ()

4.3.10 Ex_soft_error Error

This error occurs due to the interchange of the soft halant with explicit halant.

Example

क्त्ति.....क्ति

Search Criteria

Interchange the two halants and check for their validity.

Function: exSoftHalantError()

4.3.11 Matra Error

This error occurs due to placement of the wrong matras

Example

रचीयतारचियता

गोतमगौतम

Search Criterion

Interchange the different matras according to the combinations and check for the validity after each of the replacement.

Function: matraError()

4.3.12 Anuswar Anunasik Error

This type of error occurs due to the interchange and the presence or absence of the anuswar(अं) and anunasik(अँ).

अँक.....अंक

Search Criterion

Anunasik comes only upon the character, which have अँ and आँ Matra.

1) Interchange the anunasik to anusware.

2) Interchange the anuswar to anunasik only if anuswar come upon the character which have अँ and आँ Matra.

3) Remove entirely.

4) Check the validity in each case.

Function: anuswarAnunasikError()

4.3.13 Double Matra Error

This type of error occurs due to the presence of double matra on a single letter

Example

मुकेशमुकेश

Search Criterion

1) Find the character with double matra.

2) Remove the matra one by one and check for the validity.

Function: sameDoubleMatraError()

4.3.14 Phonetic Error

This type of error occurs due to placement of the phonetically wrong word.

Example

कल्यानकल्याण

Search Criterion

Interchange the different character according to the combinations and check for the validity after each of the replacement.

Function: phoneticError()

4.3.14 Visarg Error

This error occurs due to the wrongly placement of the visarg in the end.

Example

अन्यथः.....अन्यथा

Search Criterion

If there is a visarg at the last character then remove it and put (ऩ) matra.

Function: visargError()

4.3.15 NMN1_R_KAR Error

This type of error occurs due to placement of ि - in place of ँ

Examples

कृतृम कृत्रिम

Search Criterion

Find out the position of ि - (र + ि)* in the word and replace it

with (ँ) and vice versa.

Function: nmn1RKarError()

RESULTS AND DISCUSSION

5.1 Results

The results are shown in the following windows. Each window has a different functionality. Here the document, which have different errors is opened and than spell checker is run over the document and finally the document is produced which have no error .In this figure 5.1, the starting windows is shows.

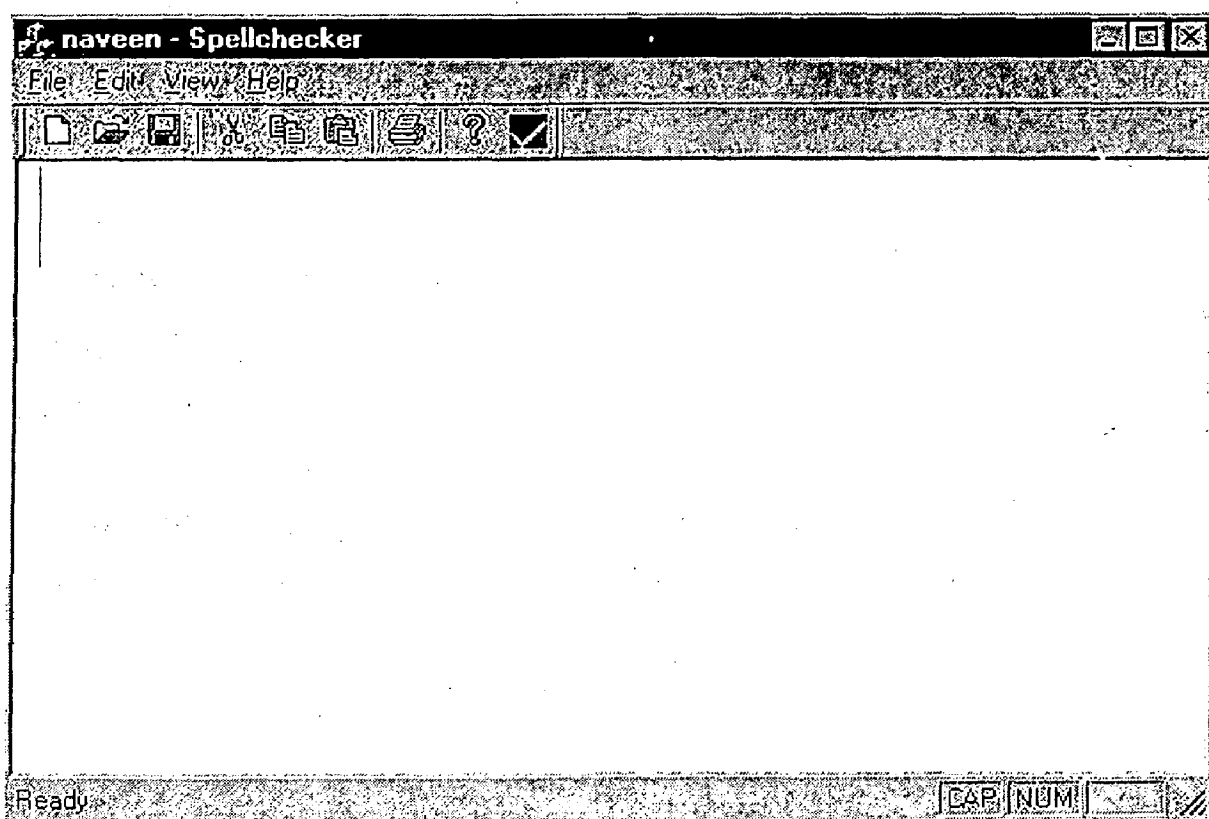


Figure 5.1: Starting Window

The document is selected by the user and open in the window. The document has may errors, which have to be correct by using the spell checker

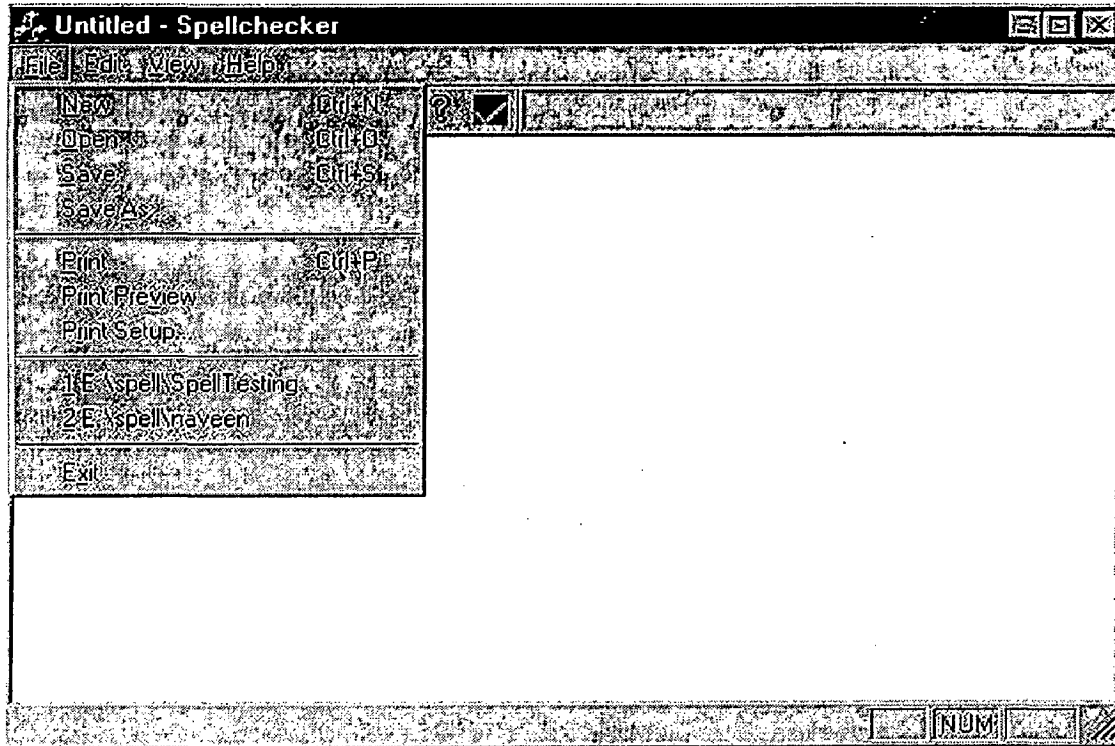


Figure 5.2: Document is to be opened in the window

In the figure 5.3, this window has following options. Each button have different functions through user can make modification on the documentation.

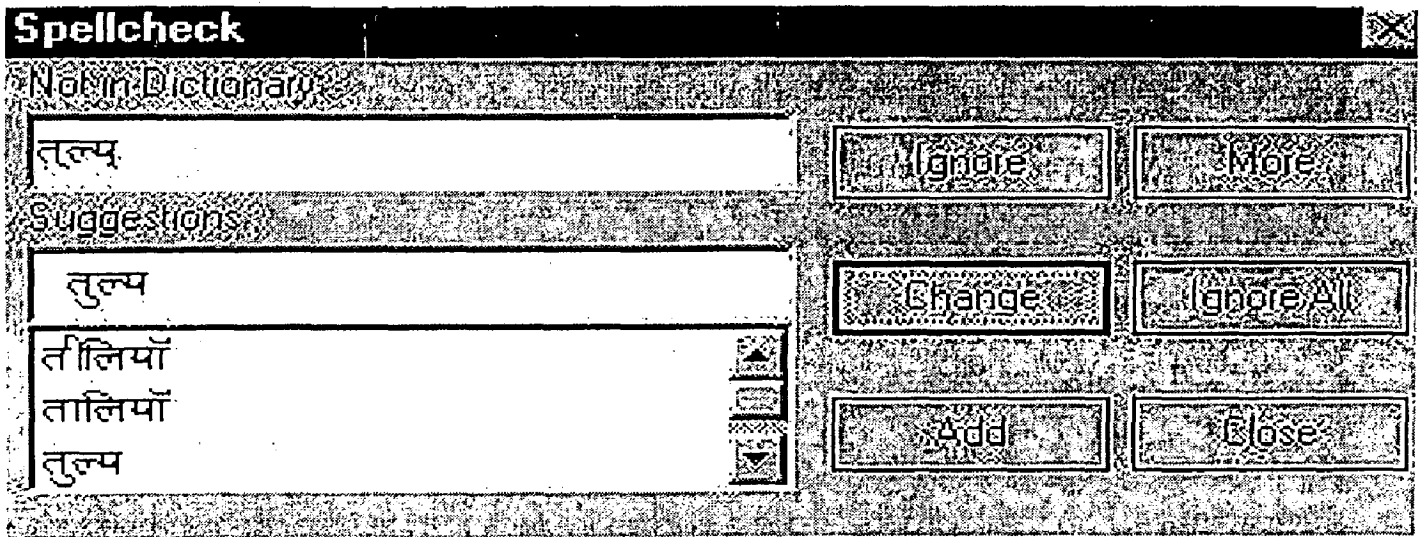


Figure 5.3: Spell Checker Dialog Box

1. **Ignore Button:** If user does not want any change for the selected word than this button is used for ignorance.
2. **More Button:** If user want more options for a given word. This button is used and user gets more options in the list.
3. **Change Button:** If user want to change the wrong word. By using this button he can replace the wrong word to correct word.
4. **Ignore all Button:** if user want to ignore the selected word in the document. This button is used. In the document this word considered as a correct word.
5. **Add Button:** if user want to insert the word in the dictionary. By using this button he can insert the word in new dictionary called user dictionary.
6. **Close Button:** For closing the spell checker dialog box.
7. **Options list:** Here different options of the wrong word are shows in the list and user can choose any one correct word from the list.
8. **Wrong Word:** Here wrong word of the document is shows in this box.

The Document is opened in the window. Here there are many errors in the document, which have to be corrected by using the spell checker.

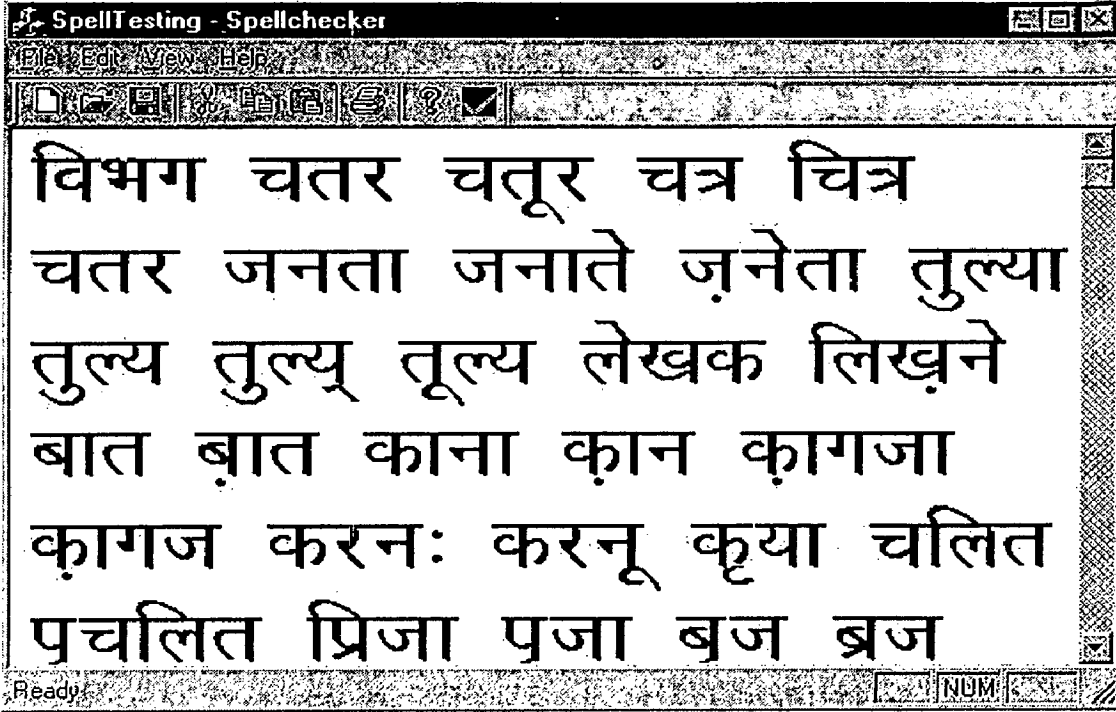


Figure 5.4: Document is opened in the window

When spell checker is run over the document than wrong word is recognized. Spell checker shows the options of the wrong word and user choose nay one correct word and replace wrong word to the correct word.

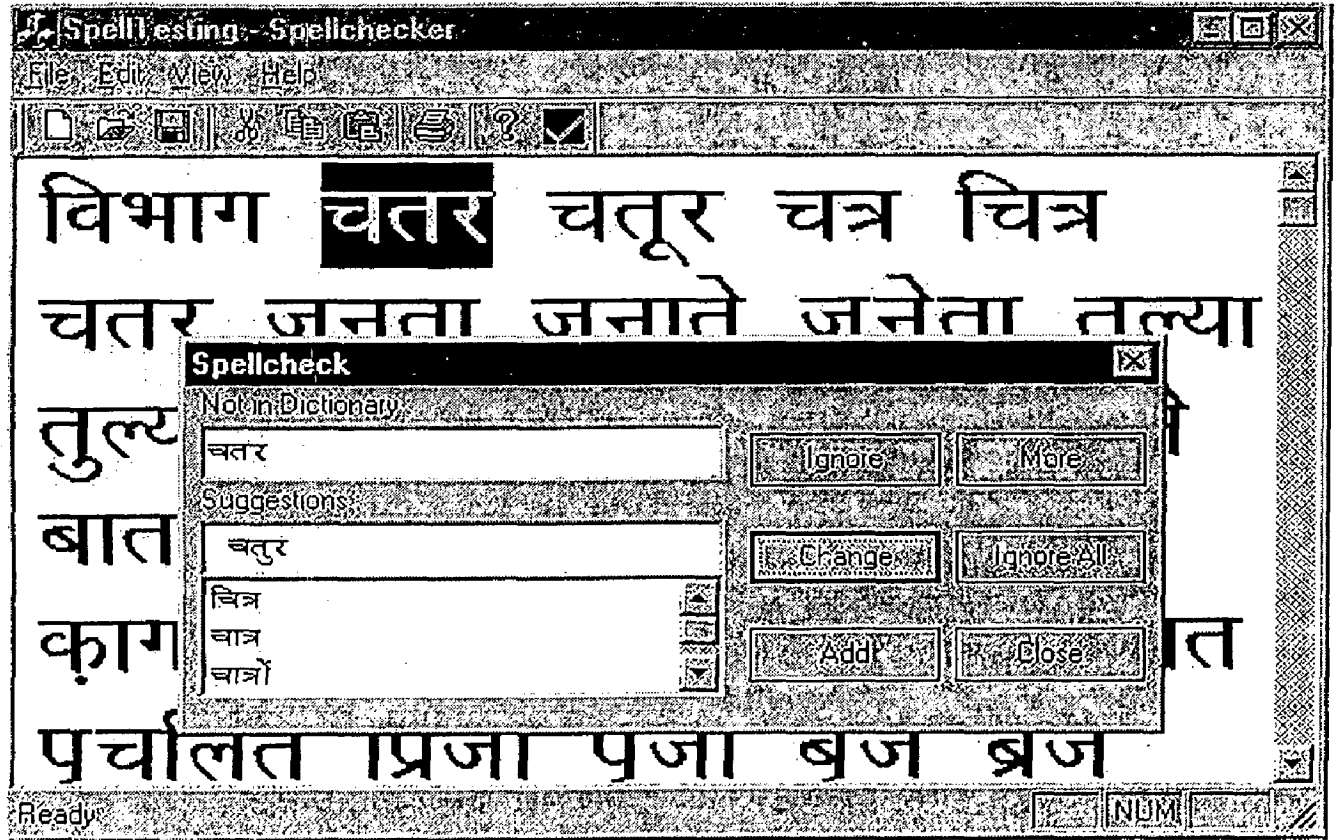


Figure 5.5: Spell checker has options of the wrong word

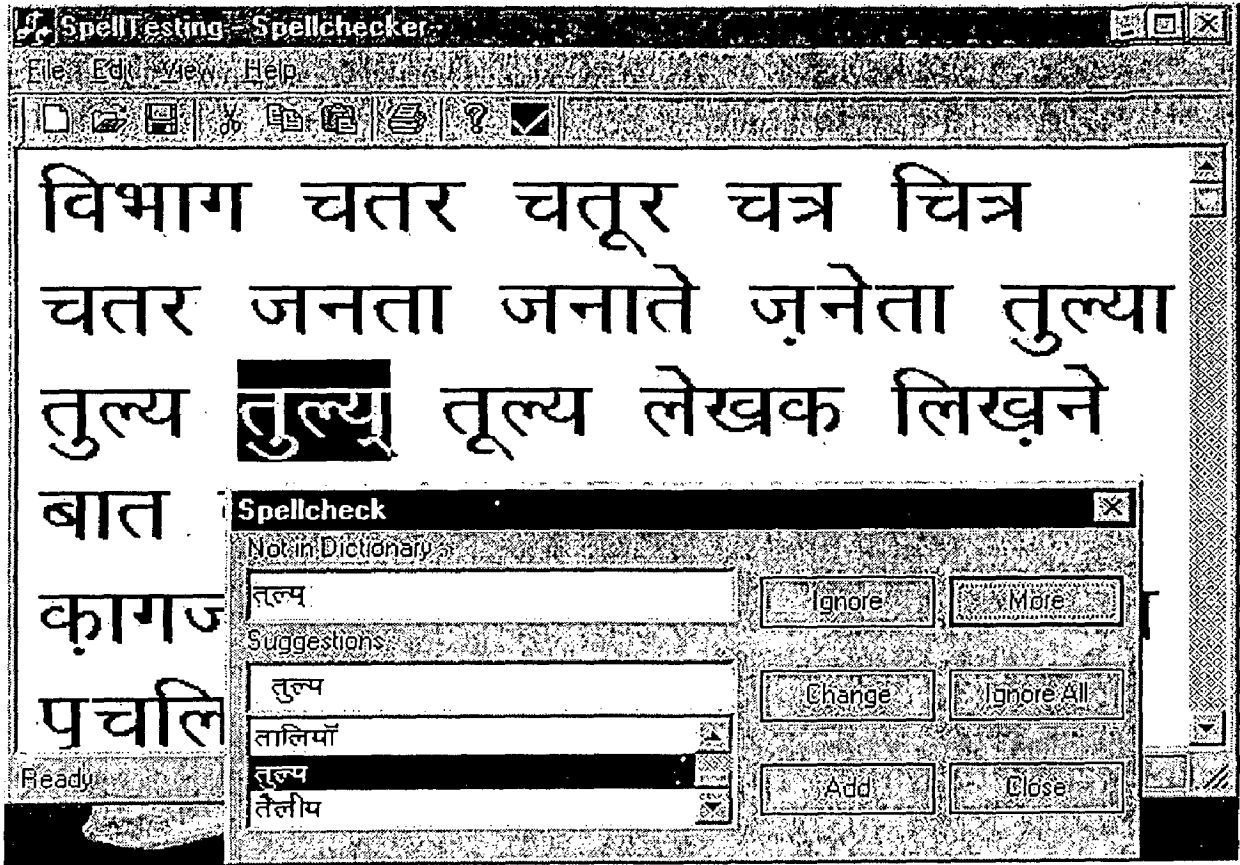


Figure 5.6: Spell checkers has all the options after clicking the more button

Here an object of the dictionary is created by using this window. The dictionary has 50,000 words, which have to Access in spell checker module. So that words is easily assessable.

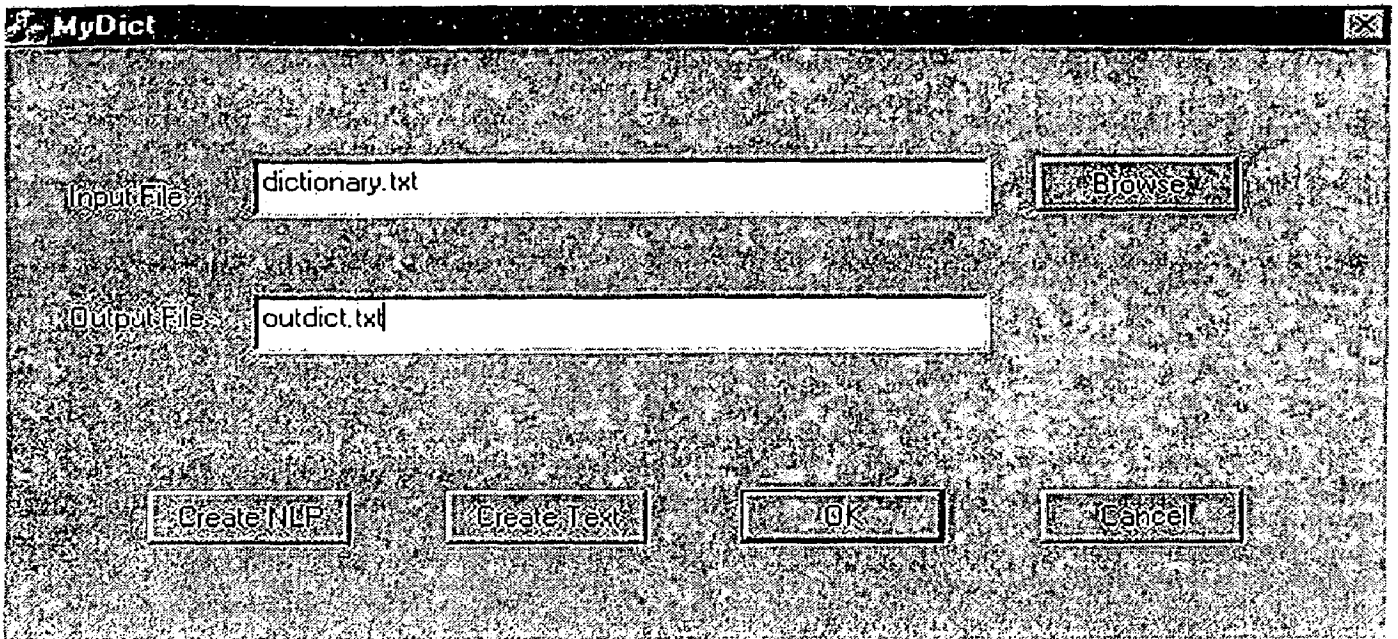


Figure 5.7:Dictionary is created in this window

5.2 Suggestions for future work

In this spell checker module grammatical errors have not been considered. In the next phase a HINDI parser will be developed and incorporated to detect and correct grammatical errors also.

CONCLUSION

The results of Spell Checker show that mostly error is caused due to wrong recognition of the word and Matras. We have shown in the Spell Checker Module that Spell Checker is powerful enough to detect errors and to rank the correct word options. Ranking of the generated-options will lead to more desired sequence if the Word Frequency is also used.

In Spell Checker the word is considered to be incorrect,

- If it contains any invalid pattern which is not possible in Hindi words OR
- The word is not in the dictionary.

And the validity of correctness of the constructed options is checked with the help of dictionary.

For example: दीन

Which is a correct word but not in the dictionary and correction algorithm leads to दिन which is another correct word but present in the dictionary.

The module has been tested on incorrect words and contains 50,000 words in the dictionary and in 80% of the cases the correct options are generated. The results also show the expected words appear in first three options in most of the cases.

Sample Result table applied on 150 incorrect words.

Error Type	Total incorrect words	% of correct option generated	% of correct option in first 03 positions
Matra error	63	100%	85%
Exchange of matra with vowel error	03	70%	100%
Nukta error	04	100%	100%
Halant error	10	65%	100%

Consonant Exchange error	10	70%	80%
One character missing error	15	100%	60%
Splitting of word error	13	40%	80%
Phonetic error	09	95%	100%

References

1. B.B. Chaudhuri and U. Pal, "An OCR system to read two Indian Language scripts: Bangla and Devanagari (Hindi)", In Proc. 4th Int. Conf. On Document Analysis and Recognition, Ulm, Vol. 2, pp. 1011-1015, 1997.
2. Veena Bansal and R.M.K. Sinha, On how to Describe Shapes of Devanagari Characters and Use Them for Recognition, 5th International Conference on Document Analysis and Recognition(ICDAR '99), 1999, Bangalore, India.
3. R.M.K. Sinha. and K.S. Singh A Programme for correction of single spelling errors in Hindi words., Journal of Institution of Electronics and Telecommunication Engineers, Vol. 30, No. 6, 1984, p. 249-251.
4. J.L. Peterson 1980. Computer programs for detecting and correcting spelling errors. Commun. ACM 23, 12, (Dec.) 676-684.
5. R.M.K. Sinha, 'Computer processing of Indian languages and scripts - potentialities and problems', Jour. of Inst. Electron. & Telecom. Engrs., vol.30,no.6, 1984,pp. 133-49.
6. Veena Bansal and R.M.K. Sinha, Partitioning and Searching Dictionary for Correction of Optically Read Devanagari Character Strings, Int. Jour. On Document Analysis and Recognition, Vol. 4, 2002 pp 269-280 (Presented at 5th International Conference on Document Analysis and Recognition 1999, Bangalore, India.)
7. Indian Standard: Indian Script Code for Information Interchange – ISCII. Electronics Information & Planning, Feb., 1992.
8. Microsoft Press," Microsoft Visual C++ MFC Library Reference(Visual C++ 5.0 Documentation Library", Vol 1., Part 1, [ISBN 1572315180].
9. Microsoft Press," Microsoft Visual C++ MFC Library Reference(Visual C++ 5.0 Documentation Library", Vol 2., Part 2, [ISBN 1572315199].

Layout of the Hindi OCR Software

The system flow of HINDI OCR is shown in figure a1.

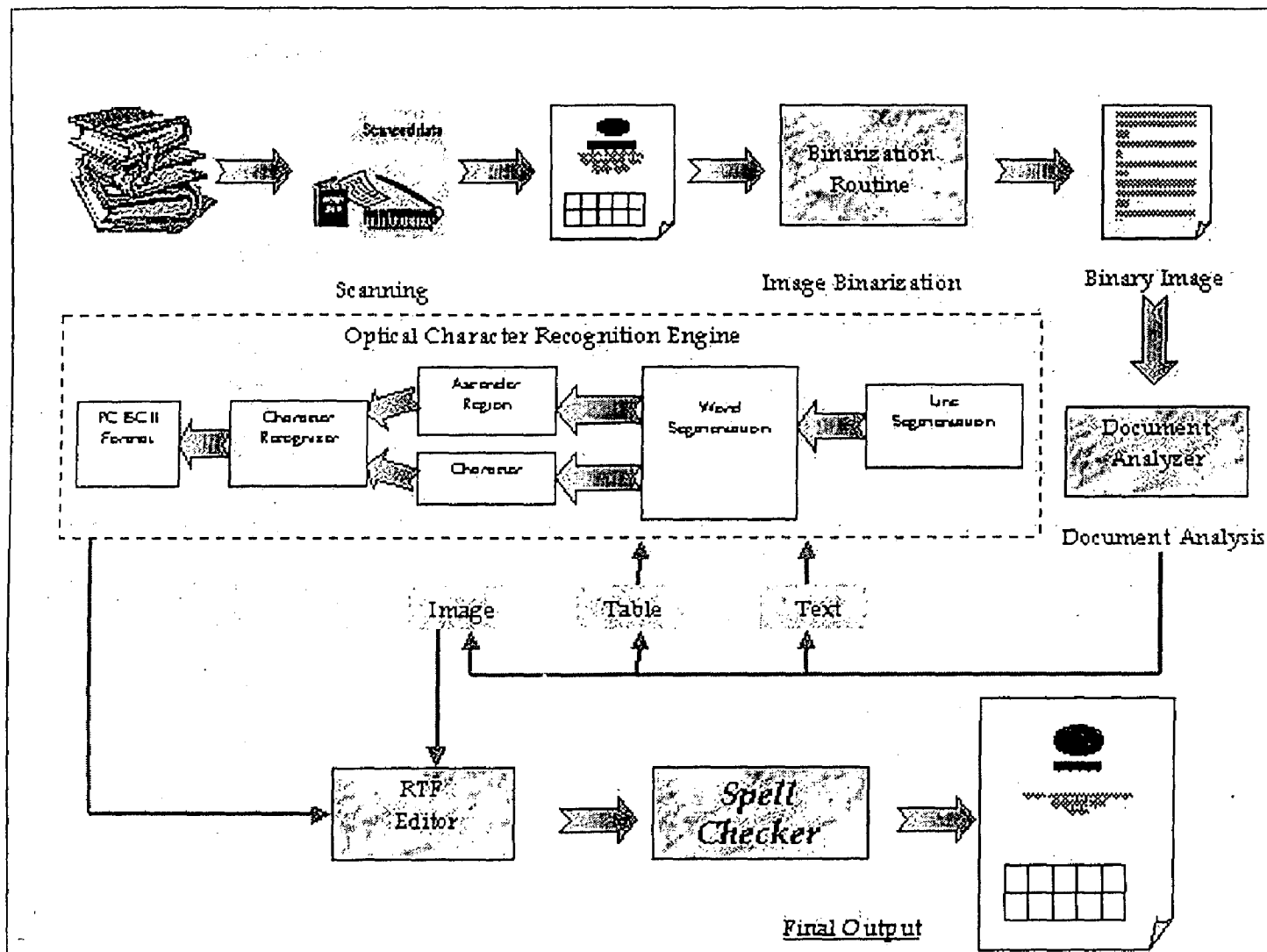


Figure a1: System flow of HINDI OCR

PC-ISCII Code Table

The PC-ISCII code is the version of ISCII code defined by Center for Development of Advanced Computing (CDAC), Pune, for compatibility with IBM-PC 8-bit character set. IBM-PC does not follow the ISO 8-bit code recommendation.

PC-ISCII CODE

Hex	Dec.	8	9	A	B	C	D	E	F
		128	144	160	176	192	208	224	240
0	0	ँ	औ	ण				ATR	EXT
1	1	ं	ॉ	त्				ल	ँ
2	2	ः	क	थ				ळ	े
3	3	अ	ख	द				क	ै
4	4	आ	ग	घ				ष	ँ
5	5	इ	घ	न				श	ी
6	6	ई	ङ	त्				ष	ो
7	7	उ	च	प				स	ी
8	8	ऊ	छ	फ				ह	ी
9	9	ऋ	ज	झ				INV	२
A	10	ऐ	झ	भ				ा	ं
B	11	ए	अ	म				ि	।
C	12	ऐ	ट	य				ी	।
D	13	एँ	ठ	स				२	
E	14	औ	ड	र				२	
F	15	ओ	ड	र				२	

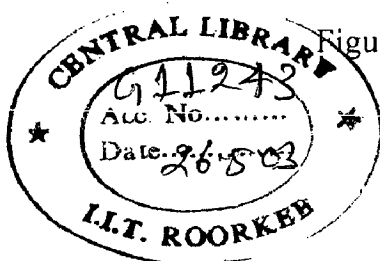


Figure b2:PC-ISCII CODE

MFC CLASSES AND METHOD USED IN SOFTWARE

The MFC classes and method used in this software is:

C.1 CStringList

This class supports lists of CString objects. All comparisons are done by value, meaning that the characters in the string are compared instead of the addresses of the strings. When a CStringList object is deleted, or when its elements are removed, the CString objects are deleted as appropriate.

C.1.1 Class Members Used in Software

CObList::CObList

This constructor constructs an empty CObject pointer list. As the list grows, memory is allocated in units of nBlockSize entries. If a memory allocation fails, a MemoryException is thrown.

```
CObList( int nBlockSize = 10 );
```

Parameters

nBlockSize

Specifies the memory-allocation granularity for extending the list.

CObList::GetHead

This method retrieves the CObject pointer that represents the head element of this list. You must ensure that the list is not empty before calling GetHead. If the list is empty, then the Debug version of the Microsoft Foundation Class Library asserts.

```
CObject*& GetHead();
```

Return Value

If the list is accessed directly or through a pointer to a CObList, then GetHead returns a reference to a CObject pointer. This allows the function to be used on either side of an assignment statement and thus allows the list entries to be modified.

CObList::AddHead

This method adds a new element to the head of this list. The list can be empty before the operation.

```
POSITION AddHead( COBJECT* newElement );
```

Parameters

newElement

Specifies the COBJECT pointer to be added to this list.

Return Value

Returns the POSITION value of the newly inserted element.

COBList::AddTail

This method adds a new element to the tail of this list. The list can be empty before the operation.

```
POSITION AddTail( COBJECT* newElement );
```

Parameters

newElement

Specifies the COBJECT pointer to be added to this list.

Return Value

Returns the POSITION value of the newly inserted element.

COBList::RemoveAll

This method removes all of the elements from this list and frees the associated COBList memory. No error is generated if the list is already empty. When you remove elements from a COBList, you remove the object pointers from the list. It is your responsibility to delete the objects themselves.

```
void RemoveAll();
```

COBList::GetHeadPosition

This method retrieves the position of the head element of this list.

```
POSITION GetHeadPosition()const;
```

Return Value

A POSITION value that can be used for iteration or object pointer retrieval; it is NULL if the list is empty.

COBList::GetNext

This method retrieves the list element identified by rPosition, and then sets rPosition to the POSITION value of the next entry in the list. You can use GetNext in a forward iteration loop if you establish the initial position with a call to GetHeadPosition or Find.

```
COBJect* GetNext( POSITION& rPosition ) const;
```

Parameters

rPosition

Specifies a reference to a POSITION value returned by a previous GetNext, GetHeadPosition, or other method call.

Return Value

If the list is accessed through a pointer to a const COBJList, then GetHead returns a COBJect pointer. This allows the function to be used only on the right side of an assignment statement and thus protects the list from modification.

COBJList::GetAt

This method retrieves the COBJect pointer associated with a specified position. A variable of type POSITION is a key for the list. It is not the same as an index, and you cannot operate on a POSITION value yourself. You must ensure that your POSITION value represents a valid position in the list. If it is invalid, then the Debug version of MFC for Windows CE asserts.

```
COBJect* GetAt( POSITION position ) const;
```

Parameters

position

Specifies a POSITION value returned by a previous GetHeadPosition or Find method call.

Return Value

If the list is accessed through a pointer to a const COBJList, then GetHead returns a COBJect pointer. This allows the function to be used only on the right side of an assignment statement and thus protects the list from modification.

COBJList::Find

This method searches the list sequentially for the first COBJect pointer that matches the specified COBJect pointer. Note that the pointer values are compared, not the contents of the objects.

```
POSITION Find( COBJect* searchValue, POSITION startAfter = NULL ) const;
```

Parameters

searchValue

Specifies the object pointer to be found in this list.

startAfter

Specifies the start position for the search.

Return Value

A POSITION value that can be used for iteration or object pointer retrieval; it is NULL if the object is not found.

CObList::GetCount

This method retrieves the number of elements in this list.

```
int GetCount( )const;
```

Return Value

An integer value containing the element count.

CObList::IsEmpty

This method determines whether this list contains no elements.

```
BOOL IsEmpty( ) const;
```

Return Value

Nonzero if this list is empty; otherwise, it is zero.

C.2 CString

CString does not have a base class. A CString object consists of a variable-length sequence of characters. CString provides functions and operators using a syntax similar to that of Basic. Concatenation and comparison operators, together with simplified memory management, make CString objects easier to use than ordinary character arrays.

C.2.1 CString Class Member Used in Software.

CString::GetLength

```
int GetLength( ) const;
```

Return Value

A count of the bytes in the string.

Remarks

Call this member function to get a count of the bytes in this CString object. The count does not include a null terminator.

CString::Empty

```
void Empty();
```

Remarks

Makes this CString object an empty string and frees memory as appropriate.

CString::GetAt

```
TCHAR GetAt( int nIndex ) const;
```

Return Value

A TCHAR containing the character at the specified position in the string.

Parameters

nIndex

Zero-based index of the character in the CString object. The nIndex parameter must be greater than or equal to 0 and less than the value returned by GetLength.

Remarks

The GetAt member function returns a single character specified by an index number.

CString::Left

```
CString Left( int nCount ) const;
```

Return Value

A CString object containing a copy of the specified range of characters.

Parameters

nCount

The number of characters to extract from this CString object.

Remarks

Extracts the first (that is, leftmost) nCount characters from this CString object and returns a copy of the extracted substring. If nCount exceeds the string length, then the entire string is extracted.

CString::Right

CString Right(int nCount) const;

Return Value

A CString object that contains a copy of the specified range of characters.

Parameters

nCount

The number of characters to extract from this CString object.

Remarks

Extracts the last (that is, rightmost) nCount characters from this CString object and returns a copy of the extracted substring. If nCount exceeds the string length, then the entire string is extracted.

CString::TrimLeft

void TrimLeft();

Remarks

Call the version of this member function trim leading whitespace characters from the string. TrimLeft removes newline, space, and tab characters.

CString::TrimRight

void TrimRight();

Remarks

Call the version of this member function trim leading whitespace characters from the string. TrimRight removes newline, space, and tab characters.

CString::FindOneOf

int FindOneOf(LPCTSTR lpszCharSet) const;

Return Value

The zero-based index of the first character in this string that is also in lpszCharSet; -1 if there is no match.

Parameters

lpszCharSet

String containing characters for matching.

Remarks

Searches this string for the first character that matches any character contained in lpszCharSet.

C.3 CComboBox

A combo box consists of a list box combined with either a static control or edit control. The list-box portion of the control may be displayed at all times or may only drop down when the user selects the drop-down arrow next to the control.

The currently selected item (if any) in the list box is displayed in the static or edit control. In addition, if the combo box has the drop-down list style, the user can type the initial character of one of the items in the list, and the list box, if visible, will highlight the next item with that initial character.

C.3.1 CComboBox Class Member Used in Software

CComboBox::GetCount

```
int GetCount( ) const;
```

Return Value

The number of items. The returned count is one greater than the index value of the last item (the index is zero-based).

Remarks

Call this member function to retrieve the number of items in the list-box portion of a combo box.

CComboBox::DeleteString

```
int DeleteString( UINT nIndex );
```

Return Value

If the return value is greater than or equal to 0, then it is a count of the strings remaining in the list.

Parameters

nIndex

Specifies the index to the string that is to be deleted.

Remarks

Deletes a string in the list box of a combo box.

CComboBox::InsertString

```
int InsertString( int nIndex, LPCTSTR lpszString );
```

Return Value

The zero-based index of the position at which the string was inserted. The return value is CB_ERR if an error occurs.

Parameters

nIndex

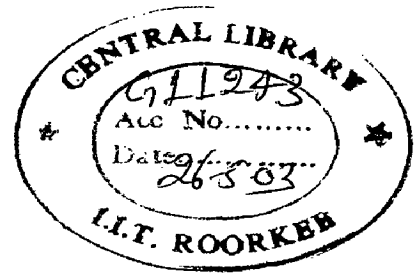
Contains the zero-based index to the position in the list box that will receive the string. If this parameter is -1, the string is added to the end of the list.

lpszString

Points to the null-terminated string that is to be inserted.

Remarks

Inserts a string into the list box of a combo box



CComboBox::ResetContent

```
void ResetContent();
```

Remarks

Removes all items from the list box and edit control of a combo box.

CComboBox::SelectString

```
int SelectString( int nStartAfter, LPCTSTR lpszString );
```

Return Value

The zero-based index of the selected item if the string was found.

Parameters

nStartAfter

Contains the zero-based index of the item before the first item to be searched. When the search reaches the bottom of the list box, it continues from the top of the list box back to the item specified by nStartAfter. If -1, the entire list box is searched from the beginning.

lpszString

Points to the null-terminated string that contains the prefix to search for. The search is case independent, so this string can contain any combination of uppercase and lowercase letters.

Remarks

Searches for a string in the list box of a combo box, and if the string is found, selects the string in the list box and copies it to the edit control.

A string is selected only if its initial characters (from the starting point) match the characters in the prefix string.