DOCUMENT ANNOTATION AND RETRIEVAL SYSTEMS

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (by Research)
in
Computer Science

by

A. Balasubramanian
200307012
balu@research.iiit.ac.in

International Institute of Information Technology
Hyderabad, INDIA
June 2006

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "DOCUMENT ANNOTATION AND RETRIEVAL SYSTEMS" by A.Balasubramanian, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Dr. C. V. Jawahar

Acknowledgments

Dr. C.V. Jawahar, my adviser has been the key instrumental person in motivating and getting the best out of my thesis. He has been the biggest force, who has been my philosopher and guide and helped come over during my lean and difficult patches. He has also played a role in defining and directing my career.

Dr. P.J. Narayanan, who had been constantly providing me feedback and his advise. Dr. Anoop Namboodri has helped me immensely with his expertise in online handwriting and his constructive suggestions in research paper writings.

I also wish to thank Sri Ganesh Madhavanath (HP Labs, Bangalore) for his technical help during the online handwriting annotation toolkit project, and HP Labs Bangalore for its financial support.

During the course of my Masters in IIIT Hyderabad, I've been associated with close-knitted friends who have been of immense of help to me. M.N.S.S.K Pavan from whom I've learned how to design and implement systems, especially in the online handwriting annotation project. Vandana Roy, who has been more than a friend, helping me debug my code, giving leads and inputs at the appropriate time. K.S.Sesh Kumar for being with me all throughout from my starting days at the CVIT research center. P. Suman Karthik, with whom I had technical and philosophical brainstorming. Also Satyanarayana (CVIT), who helped all throughout bridging my infrastructure gap, and I'll be failing in my duties if I don't mention Million Meshesha, PhD student here and Anand Kumar, an MS student for their technical advise and I thoroughly enjoyed writing papers and conducting experiments during my stay here.

Last, but not the least, the almighty, my parents, my relatives and all those from CVIT who had at some or the other point in time helped me with their invaluable suggestions and feedback, and my research center, Center for Visual Information Technology (CVIT), for funding my MS by research in IIIT Hyderabad.

Abstract

Digital documents are now omnipresent. Techniques and algorithms to process and understand these documents are still evolving. This thesis focuses on the non-textual documents of textual content. Example of this category are online handwritten documents and scanned printed books. Algorithms for accessing such documents at the content-level are still missing, specially for Indian Languages. This thesis addresses two fundamental problems in this area – Annotation and Retrieval. Annotated datasets of handwriting are a prerequisite for the design and training of handwriting recognition algorithms. Retrieval from annotated data sets is relatively straightforward. However retrieval from unannotated datasets is still an open problem. We explore algorithms which make these two tasks possible.

Annotation of large datasets is a tedious and expensive process. The problem becomes compounded for handwritten documents, where the characters correspond to one or more strokes. We have developed a versatile, robust annotation tool for online handwriting data. This tool is aimed at supporting the emerging UPX/hwDataset schema, a promising successor of the UNIPEN. We provide easy-to-use interface for the annotation tool. However, still the annotation is highly manual. We then propose a novel, automated method for annotation of online handwriting data at the character level, given a parallel corpus of online handwritten data and typed text. The method employs a model-based handwriting synthesis unit to map the two corpora to the same space. Annotation is then propagated to the word level and finally to the individual characters using elastic matching. The initial results of annotation are used to improve the handwriting synthesis model for the user under consideration, which in turn refine the annotation. The method takes care of errors in the handwriting such as spurious and missing strokes and characters. The output is stored in the UPX format.

Search and retrieval of online handwriting is gaining importance due to the increase in availability of such data. However, the problem is challenging due to variations in handwriting between various writers, digitizers and writing conditions. We propose a retrieval mechanism for online handwriting, which can handle different writing styles, specifically for Indian languages. The proposed approach provides a keyboard-based search interface, enabling the search of handwritten data from any computer, in addition to pen-based and example-based queries. Textual queries are supported for handwritten data sets with the help of a handwriting synthesis module. Synthesis of handwriting has a variety of applications including generation of personalized documents, study of writing styles, automatic generation of data for training recognizers, and matching of handwritten data for retrieval etc. Most of the existing algorithms for handwriting synthesis deal with English, where the spatial layout of the components are relatively simple, while the cursiveness of the script introduces many challenges. We present a synthesis model for generating handwritten data for Indian languages, where the layout of characters is complex while the script is fundamentally non-cursive. The retrieval framework, which employs handwriting synthesis and holistic matching of online words, also allows for cross-lingual document retrieval across Indian languages.

We also demonstrate the retrieval scheme on a set of offline printed documents. The system for retrieval of relevant documents from large document image collections is developed by adapting existing search engines. We achieve effective search and retrieval from a large collection of printed document images by matching image features at word-level. For representations of the words, profile-based and shape-based features are employed. Our scheme groups together similar words

during the indexing process. The system supports cross-lingual search using OM-Trans transliteration and a dictionary-based approach. System-level issues for retrieval (eg. scalability, effective delivery etc.) are the focus.

Digitized books and manuscripts in digital libraries are often stored as images or graphics. They are not searchable at the content level due to the lack of OCRs or poor quality of the scanned images. Portable Document Format (PDF) has emerged as the most popular document representation schema for wider access across platforms. When there is no textual (eg. UNICODE, ASCII) representation available, scanned images are stored in the graphics stream of a PDF file. We propose a novel solution to search the textual data in the graphics stream of the PDF files at content level. The proposed solution is demonstrated by enhancing an opensource PDF viewer (Xpdf). Indian language support is also provided. Users can type a word in Roman (ITRANS), view it in a font, and search in textual and graphics stream of PDF documents simultaneously.

In short, the contributions of this thesis are:

1. Development of a versatile annotation tool for online handwriting data and to store the annotation in UPX/hwDataset format, which is considered as the successor to the popular UNIPEN standard.

2. Development of an algorithm to annotate the online handwriting data with the help of an unaligned parallel text.

3. A synthesis scheme is proposed for online handwriting in Indian languages by addressing specialties of the Indian scripts. Given a text, corresponding realistic handwriting is synthesized.

4. A document retrieval system is presented for online handwriting, which does not require a recognizer. It accepts textual queries, synthesizes the handwriting and then does a word level elastic matching for finding the most similar words.

5. Scalability of recognition-free retrieval systems is verified by adapting an existing opensource search engine for large collection of document images.

6. Portable document format (PDF) representation of document images are not content-level accessible if the data is stored in the graphics stream of the PDF. We find the application of the recognition-free retrieval scheme in the graphics stream of PDF and verify the performance by integrating it into an opensource PDF reader.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

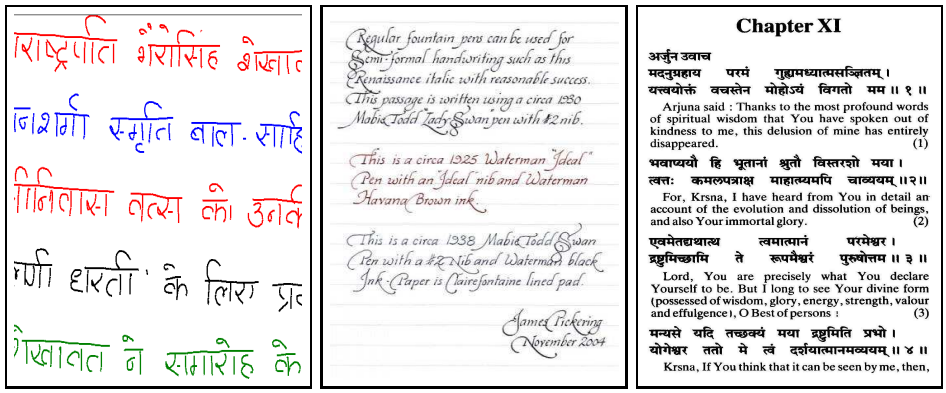
The "Digital Divide" describes the gap between individuals and communities with greater and lesser access to technology resources and training. This gap needs to narrow down at a very fast rate. For this it is increasingly becoming important to provide people with regular and effective access to the information. Computer systems share a major part in the digital technology systems. Developing document understanding system has become the major challenge ahead. For textual documents, summarization and translation issues are still open. For non-textual documents, many more fundamental problems are unsolved even now. In this thesis, we limit our interest to non-textual documents of textual content. Throughout the thesis, document is primarily used in this context.

1.1.1 Document categories

When we refer to a paper document or a papyrus document it is distinguished by the fact that it is on paper. However the notion of "digital document" is one which digital systems can understand and present to the user in an articulated manner. There are several types of documents which present information to a person that can be conceived and comprehended. Documents can be primarily divided into three different categories [5], and they are:

- **Online**: Documents that fall into the online paradigm consist of *online handwriting* (digital ink), that consist handwriting data captured by a digitizer that captures handwriting of a writer. These digitizers are specialized devices that capture a writer's ink information, his speed, the pressure applied etc. which can be later used for further processing (refer Figure 1.1 (a)). Digitizers typically include the TabletPC, Palmtops, and many other handled devices.
- **Offline**: The least common denominator for handwriting is the paper and pen. Offline documents consist of scanned copies of handwriting information that were *a priori* written on a sheet of paper. Scanner assumes the role of the digitizer here and since we scan the document after the writer has written his content, we do not access to information such the speed or pressure with which the writer must have written. Handwriting data in both *Online* and *Offline* could be either cursive, discrete or mixed (refer Figure 1.1 (b)). Handwritten letters, personal notes and handwritten diaries, historical manuscripts are some examples of *Offline* documents.

- **Printed**: Printed documents contain textual information which are scanned copies from a book. The textual content in books are in the *Printed* form following a specific font style, font size and maintaining a standard uniformity across all pages. These are typically referred to as *document images* and could also contain images or pictures in addition to textual content (refer Figure 1.1 (c)). OCRs are used to extract the textual content from these documents. Books, journals, articles, newspapers, magazines are some of the examples of a *Printed* document. Some of the popular digitizers for both *Offline* and *Printed* documents include the digital cameras, hand-held and flat-bed scanners etc.



(a) (b) (c)

Figure 1.1: Sample Documents for (a) Online (b) Offline (c) Printed

Most of these documents that are available are typically in very large numbers and to manually group and file these documents is a very taxing procedure. At the same time it is of paramount importance that these documents are made accessible to the users who would in fact like to search them with relative ease. All of *Online*, *Offline*, *Printed* documents do not contain searchable text as is, but contain their image or ink information which cannot be searched by existing text search engines. Conventional text search is based on matching or comparison of textual description (say in ASCII/UNICODE) These techniques can not be used to access content at the image/ink level, where text is represented as pixels but not as ASCII/UNICODE. We extend the conventional textual search to image/ink representation of these documents.

1.2 Background

This thesis builds on two ongoing activities for annotation and retrieval of documents.

1.2.1 UPX Format as a Successor to UNIPEN [1, 2, 3]

In order to create and evaluate recognition engines for online handwriting, significant resources in the form of annotated datasets are required. Many of the existing representation schemes cater only to a subset of requirements. UNIPEN [6] is one of the popular standard for annotated datasets. Over time, XML has evolved as an effective standard for representation of information. UPX [2], which is a new evolving standard represents the digital ink as a separate representation and the annotation as another representation. This is because, independent of annotation, digit ink by itself can be a standard representation for efficient storage and transfer of ink for a wide variety of applications. InkML [7] is an XML-based standard for representation of digital ink which the

UPX has adopted as its underlying representation of handwritten data. The UPX representation includes several elements for detailed annotation of handwriting.

There are several shortcomings in the UNIPEN format which the UPX tries to fill. Some of these are: (a) UNIPEN is unstructured, as there is no standard way of organizing information into well-defined classes, (b) UNIPEN does not enforce any strict constraints, and (c) There is no standard order in which UNIPEN keywords can appear. On the other hand the UPX, an XML-based successor to UNIPEN address the above mentioned shortcomings of the UNIPEN and provides for a path for migrating existing UNIPEN data into the new representation. In addition, UPX effort is to create a standard representation for handwriting datasets so that it supports all scripts and allows semantic interpretation of the writing at user-defined logical levels, captures information about writers and the data capture environment and supports automatic generation of annotation using recognizers, and subsequent manual validation processes. One more finer aspect being that UPX keeps handwriting data separate from its semantic interpretations.

UNIPEN has been the result of a large effort in which numerous institutes and commercial organizations have provided data. In fact, because of the heterogeneity of the data (different writers, different recording conditions, different languages and writing setup), UNIPEN provides an excellent test case for the assessment for UPX. Actually, it was one of the goals of the UNIPEN collection efforts to provide such variety. Since UNIPEN and hwDataset have very similar goals, they are functionally quite similar though they might accomplish the same ends differently.

Handwriting Annotation Tool In order to support and build UPX datasets, we developed a handwriting annotation tool in collaboration with HP labs, Bangalore, India. The handwriting annotation tool is a graphical tool that supports InkML and UPX representations. While the tool is designed to read and write UPX documents, it is also capable of importing digital ink in input formats such as InkML, UNIPEN, and simple ASCII encoding of stroke data. The tool supports input and output viewing, editing and annotation of UPX files. The tool is supplemented by a library of basic functions that can be used to access and extract handwriting data from UPX documents based on user-specific criteria.

1.2.2 Retrieval without Recognition [4]

Large digital libraries, such as Digital Library of India (DLI) [8] are emerging for archiving large collection of printed documents. Much of the books scanned in DLI are in Indian languages, where robust OCRs are not yet available for converting these scanned images into textual form. Therefore, these books can only be searched based on the metadata of the books, and not by the content within it. Indexing and retrieval from document image collections were studied by different researchers. Success of these procedures depends on the performance of the OCRs, which convert the document images into text. For Indian, African and many other oriental languages, we need alternate methods to retrieve relevant documents from the digital libraries containing scanned images.

Matching and Retrieval

Generic content-based image retrieval systems use colour, shape or/and texture features for characterizing the content. In the case of document images, features can be more specific to the domain as they contain image description of the textual content in it. Word images, particularly from newspapers and books, are of extremely poor quality. Common problems in such document database will have to be analyzed before identifying the relevant features. Popular artifacts in printed document images include: (a) Excessive dusty noise, (b) Large ink-blobs joining disjoint characters or

components, (c) Vertical cuts due to folding of the paper, (d) Cuts in arbitrary direction due to paper quality or foreign material, (d) Degradation of printed text due to the poor quality of paper and ink, and (e) Floating ink from facing pages.

An effective representation of the word images will have to take care of these artifacts for successful indexing and retrieval. Million *et al* [4, 9] found out that there are broadly three categories of features that are needed to address these artifacts: word profiles, structural features and transform domain representations. More information about these features can be found in Appendix E.3. Million [4] also observed that these representations work well for popular fonts, but only with limited success for fancy fonts. Document images are preprocessed in an offline mode to threshold, skew-correct, remove the noise and thereafter to segment into words. Then the features are extracted for individual words. They are also normalized such that the word representations become insensitive to variations in size, font and various degradations popularly present in the text documents. For proper search, we need to identify the similar words. Distance or dissimilarity between words is computed using the features discussed above. Similarity of words are computed based on two components: (a) A sequence alignment score computed using a Dynamic Time Warping (DTW) procedure (for details refer Appendix E.4). (b) Structural similarity of word images by comparing the shapes.

Word Spotting The word spotting technique involves the segmentation of each document into its corresponding lines and then into words. Each document is indexed by the visual image features of its words. Word level matching has been attempted for printed [10] , offline [11] and online [12] documents. They are useful for locating similar occurrences of the query word. There have been successful attempts on locating a specific word in a handwritten text by matching image features for historical documents [11]. Word spotting approach [11, 13] has been extended to searching queried words from printed document images of newspapers and books. Dynamic time warping based word-spotting algorithm for indexing and retrieval of online documents is also reported in [12]. Features for matching words are computed from the constituent strokes. None of these matching schemes are designed to address partial matches, which is very important for addressing word-form variations for effective search.

Information Retrieval Measures On the other hand, Million *et al* [9], have addressed the issue for partial matching for *printed* documents along with the issues of retrieval. This work extends it to cross-lingual retrieval by transliteration among Indian languages and a table-lookup translation for other languages. Million [4] has also proposed information retrieval measures such as *term frequency* (TF) and *inverse document frequency* (IDF) in the context of document images. More information can be found in Appendix E.5.

1.3 Annotation and Retrieval

The need for annotating documents is increasingly getting important. Annotated datasets of handwriting are a prerequisite to attempt a variety of problems such as building recognizers, segmentation algorithms, writer identification algorithms etc. Annotated datasets of handwriting covering a variety of writing styles are essential for development and evaluation of handwriting recognition engines. Lack of linguistic resources for many scripts, in the form of annotated handwriting datasets has been one of major hurdles to research in these scripts. Manual annotation of handwriting datasets is laborious and error-prone, especially if one were to annotate at the character or stroke

level. This brings about the need for automatic annotation. Annotation by alignment is studied, provided a parallel corpus of text is given.

1.4 Problem Statement

This thesis addresses the problem of annotation and retrieval of documents with special emphasis on Indian Language documents. We explore the specific algorithms, and verify the schemes by building prototype systems. The specific problems of interest are:

- Development of a tool for annotating online handwriting data in UPX/hwdataset format, which is considered as a successor to the popular UNIPEN format. Special issues related to the Indian Languages are kept in mind during design and implementation. Extensive evaluations are done keeping in mind the popular use.
- Development of a model-driven annotation scheme for highly automated annotation of online handwritten data. This addresses the need of developing huge annotated corpora for building recognizers. Recognition engines need annotation at the character/akshara level.
- Model driven annotation starting from the parallel unaligned corpora needs the synthesis of handwriting. We propose a synthesis scheme for online handwriting in this Thesis.
- Retrieval is a relatively straightforward process when the data is annotated. Often the document collections are unannotated. We investigate a retrieval scheme for online handwriting, without depending on a recognition algorithms. User prefers to query in text, while retrieval of documents are digital ink.
- Similar techniques could also be applied for offline printed documents. However for making large collections of documents accessible, we need scalable systems. We propose a scheme which helps in adapting the existing search engines suitable for efficient indexing and retrieval of document images.
- In the final problem, we explore the access to the graphics stream of PDF files, where the textual documents are represented as graphics in the PDFs due to the unavailability of text during the PDF generation. We propose a retrieval scheme without explicit recognition in the graphics stream of the PDF files with support for Indian languages.

The thesis address the above problems in a general recognition free framework. However, the results of this thesis could be highly useful for developing recognition engines.

1.5 Related Work

Related articles to this thesis are referred and discussed in detail in the relevant parts of the next three chapters. However, a brief sketch of the related work is provided to give the background of the thesis.

1.5.1 Related Work in Online Handwriting Annotation

- First TabletPC, *GRiD pad* was introduced in September 1989, by GRiD systems. It had MS-DOS as the underlying operating system.
- Jot [14] was introduced as a specification for an ink storage and interchange format in 1993.
- UNIPEN [6] format was introduced by Guyon *et al* [15] in 1994 and is a standard for representation of online handwriting ink.
- INKML [7] slowly evolved as the standard for ink representation and transfer. It is stored in XML format.
- *hwDataSet* [3] was proposed as a new annotation storage format for online handwriting data in 2004.
- Simultaneously [3], an annotation tool for online handwriting that stores the ink information in UPX format was built. There have been similar tools [16, 17] for offline cursive documents that can be annotated at word-level prior to our tool.
- UPX [1, 2] was proposed as a successor to UNIPEN in 2006. UPX evolved from *hwDataSet*.
- As regards to annotation by alignment is concerned, Zimmer *et al* [18] have studied the problem of alignment of parallel handwritten and text corpora for offline documents in 2000.
- Korn *et al* [19] matched word images to text based on global properties of the word extracted from both the handwritten word images as well as text words that are rendered using a specific font.
- Anand *et al* [20] use annotation by alignment for online handwriting data in 2006.

1.5.2 Related Work in Online Handwriting Synthesis and Retrieval

Synthesis of Handwriting

- Isabelle Guyon [21] started the synthesis work for online handwriting as back as 1996.
- In 1997, Guyon*et al* [22] then gave an overview and synthesis of online cursive handwriting recognition techniques.
- Later, Wang *et al* [23] proposed synthesis for online cursive handwriting.
- In 2002, Wang *et al* [24] also proposed learning-based cursive handwriting synthesis for offline handwriting datasets.
- Recently, in 2005, Zheng and Doermann [25] worked on handwriting matching and its application to handwriting synthesis.
- Jawahar and Balasubramaian [26] proposed handwriting synthesis for Indian languages for the first time.

Handwriting Matching and Retrieval

- Manmatha*et al* [27] pioneered the work of word spotting approach in as early as 1996.
- Plamondon *et al* [28] published a comprehensive survey on online and offline handwriting recognition in 2000.
- Rath *et al* [29] have explored the problem of retrieval in the context of offline historic word images for a single writer document collections in 2002.
- Russell *et al* [30] have worked with the help of a recognizer for indexing and retrieval for multi-user and single script collections in 2002.
- Jain and Anoop [31] have worked on inline ink matching and retrieval for a single writer document collections in 2003.
- Srihari *et al* [32] have studied context of "Forensic document retrieval" in 2004.
- In 2006, Balasubramanian *et al* [33] attempted for printed words for digital library collections. The Indexing process is little slow as its an offline activity.

1.5.3 Related Work in Document Retrieval Systems

- In 1925, AT&T produced *wire photo*, the first commercial image scanning system.
- First color scanner was patented by Alexander Murray and Richard Morse in 1937,
- One million book scanning activity [34, 8] gets started.
- Now even web search engine giants like Google, Yahoo and MSN are involved in archiving books more than a million.
- In 1998, Doermann [35] published a survey on "The Indexing and Retrieval of Document Images".
- Rath *et al* [11, 36] worked on feature selection and matching for scanned handwritten documents.
- In 2003, Chaudhury *et al* [37] worked on devising interactive access techniques for Indian language document images.
- Greenstone slowly [38] evolved as a popular opensource search engine.
- PDF [39] becomes the default standard for document printing and document transfer.
- Hitherto, PostScript [40] was the *de facto* standard, upon which PDF was built.
- On the PDF viewer front, Xpdf [41] became the most popular opensource PDF viewer.
- OCR-based search for latin scripts in PDF files is made available in Acrobat Professional 7.0.
- Avinash Chopde proposed ITRANS [42] for transliteration of Indian languages.
- Later OMTRANS [43] from Carnegie Mellon was proposed, which not case-sensitive unlike ITRANS.

1.6 Organization of the Thesis

Annotation of online handwriting data is described in Chapter 2, and annotation procedures such manual annotation, automatic and semi-automatic annotation procedures are also extensively covered. Annotation for the purpose archival, retrieval, and recognition are also well covered in Chapter 2. Annotation representation formats like UNIPEN and UPX are presented in lieu with the requirements for storage of online handwriting. Chapter 2 also covers our tool that supports annotation of online handwriting and compares it with existing annotation tools that are available. Finally Chapter 2 presents the Model-based Annotation framework for automatic annotation of online handwriting data.

Chapter 3 describes in detail, the retrieval of online handwriting by synthesis and matching procedures. The primary solution to handling online handwritten data is to employ a HWR (handwriting recognizer) to convert the ink into text, and use the results to search and retrieve documents. However, this approach is suited only where the handwritten data is purely text and where robust HWR is available for the language contained in the document. Chapter 3 describes an alternative approach wherein matching and retrieval are done in ink domain itself. Chapter 3 also discusses the challenges involved due to the large amount of variations that is present in online handwriting. This chapter also covers the handwriting synthesis that is used to generate handwritten data for a given input text, which is used as a template for matching. The synthesis has been achieved for Indian language scripts and is also used for cross-language retrieval.

Building an effective access to the document images requires designing a mechanism for effective search and retrieval of textual data from these collections. Chapter 4 describes the issues associated with the implementation of a scalable system for Indian language document images. We demonstrate the development of document retrieval systems for *printed* documents by modifying an existing opensource search engine and demonstrating the document word image search. For the first time, the objects in the graphics stream of PDFs are shown to be content-level accessible with the help of word spotting technique. Chapter 4 elaborates the implementation of the proposed solution in an opensource PDF reader (Xpdf) to demonstrate that textual search is possible in the graphics stream.

Finally in Chapter 5, we conclude and summarize our work by giving a direction to the future work that can be pursued.

Chapter 2

Annotation of Online Handwritten Data

2.1 Introduction to Annotation of Online Handwritten Data

Annotation is the process of identifying an object with a textual description that precisely describes the object or entities that make up the object. The need for annotating documents is increasingly getting important. Annotated datasets of handwriting are a prerequisite to attempt a variety of problems related to recognition algorithms, segmentation algorithms, writer identification algorithms etc.

Annotated datasets of handwriting covering a variety of writing styles are essential for the development and evaluation of handwriting recognition engines, especially those which utilize the data-driven learning approaches [3]. Lack of linguistic resources for many scripts, in the form of annotated handwriting datasets has been one of the major hurdles to research in these scripts. Manual annotation of handwriting datasets is laborious and error-prone process, especially if one were to annotate handwriting at the character or stroke level. There have been approaches that do automatic and semi-automatic annotation of handwriting data. Annotation can also be used for offline content, primarily the document images. Document image annotation has been widely studied, especially in the context of giving textual captions to image patches in a document image. These captions are then subjected to indexing so that these document images can be retrieved when a textual query is given. Annotation has also been studied in the context of CBIR, where a correlation is established between textual words and image features. Prior to annotation, typically there is a data collection process that is carried out in a variety of settings and then the collected data is correspondingly annotated depending on the data collection was carried out.

2.1.1 Annotation of Online Documents

Online handwriting devices are finding wide spread applications in everyday life. This necessitates effective representation of handwritten data meeting the relevant objectives. For example, the development of online handwriting recognition technology requires significant resources in the form of annotated datasets, in order to support the creation and evaluation of recognition engines. Though representation schemes have been already proposed for storage and manipulation of handwritten data [44, 45, 7, 46], many of them cater only to a subset of the requirements of the current handwriting applications. Early standards for digital ink such as ITU-T 150 (1988) and Jot(1992) [14] focused on the representation of digital ink, and did not address the issue of annotation of hand-

written data for research and development of recognition engines. Recently, XPEN(2003) [44] was proposed for representing the handwritten data which primarily needs to be exchanged over the network with distributed recognition systems. The lack of a common annotation standard often resulted in duplication of data collection efforts for individual research. This made systematic evaluation and comparison of different recognition algorithms difficult. UNIPEN [6, 46] was proposed in early 1990s as one such common standard for annotated datasets of online handwritten data.

Over time, XML has evolved as an effective standard for representation of information with extensibility, flexibility and simplicity. The creation of annotated datasets of handwriting requires two constituent representations, one for digital ink and the other for its annotation. Independent of annotation, digital ink by itself required a standard representation that can support efficient storage and transmission in a wide variety of integrated and distributed ink applications.

InkML is a recent XML-based standard which was proposed for the interchange of handwritten data within various software applications. Digital Ink Markup Language or InkML [7], a W3C draft standard, is being drafted to meet the requirements of representation of Digital Ink. The markup is designed to support input, storage and processing of handwriting, gestures, sketches, music and other notational languages in Web-based applications. The purpose of InkML is to define a common format for the exchange of ink data between various software modules, which is different from the UNIPEN's objective. InkML is a representation standard for the ink data and needs extension to store annotations.

The *hwDataset* representation proposed in this chapter uses the emerging InkML as the underlying representation of handwritten data. This representation incorporates the annotation functionalities needed for the handwriting recognition research. The specific requirements for such an annotation scheme are discussed in Section 2.4. The proposed method for storage of ink data along with its corresponding annotation using an XML based standard, is presented in Section 2.5. The advantages of the proposed model is discussed in Section 2.7 along with comparisons with similar standards. The standard is built on InkML using the core InkML for representation of the *trace* data. It incorporates the annotation storage functionalities from the UNIPEN standard, and also enhances it by adding more flexibility. This standard is proposed as an improved scheme for storage of online data for online handwriting applications. Creation of significant annotated corpora for building or evaluating recognition engines, needs effective tools. A tool for easy annotation is proposed in Section 2.6. The annotation can be done at various levels, catering to the futuristic applications. User interface is designed in such a way that annotation can be done close to the typing speed of the user.

Annotated datasets of handwriting are a prerequisite for the design and training of handwriting recognition algorithms. However, the annotation of large datasets is a tedious and expensive process, especially at the character or stroke level. Here we propose a novel, automated method for annotation at the character level, provided a parallel corpus of online handwritten and typed text. The method employs a model-based handwriting synthesis unit to map the two corpora to the same space and the annotation is propagated to the word level and then to the individual characters using elastic matching. The initial results of annotation are used to improve the handwriting synthesis model for the user under consideration, which in turn can refine the annotation. The method can take care of errors in the handwriting such as spurious and missing strokes and characters. The output is stored in the UPX format which is a new representation for storing annotation of handwriting.

2.1.2 Annotation Procedures

Annotated datasets of handwriting covering a variety of writing styles are essential for the development and evaluation of handwriting recognition engines, especially those which utilizes the data-driven learning approaches [3].

Manual Annotation Scheme

During manual annotation, users spend time annotating huge corpus of documents manually. Every region of interest that needs to be annotated, that could be typically, an image, a text region. Similar objects and other elements need to be manually identified one after another using graphic selection tools and are annotated appropriately. Handwriting data can also be annotated in a similar manner where the region of interest, that needs to be annotated could a paragraph, line or word of a handwritten text. However, annotation of large corpus of handwritten data is a time-consuming and error-prone process, especially at the character level.

Semi-Automatic and Automatic schemes

On the other hand, to bypass manual annotation, there are Semi-Automatic and Automatic schemes of annotation that reduces the manual effort considerably. Plain transcripts of handwritten data can be made available due to two factors: i) one could hire an experienced typist in a language to generate transcripts of available handwritten data, and ii) many handwritten datasets are collected based on text that is already available in the electronic form.

However, such a text need not exactly align with the handwriting due to errors in the handwriting or the transcription process. We propose a model-based annotation framework, where the handwriting style of the writer is learned and used to propagate the transcription to words and characters. The method automatically aligns the handwritten data with the text data and thus generate annotation of handwritten data at character level.

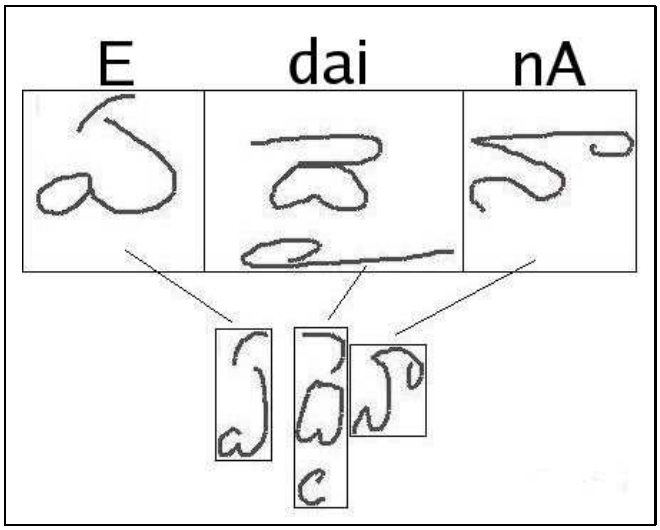


Figure 2.1: Alignment of Handwritten Data to Synthesized Handwriting.

The problem of propagation might seem relatively easy to solve since we have parallel corpora in the handwritten and text formats. However, this is true only if: i) we have an error free segmentation of the handwritten data at the word and character levels, and ii) the transcription

strictly matches the handwriting, without any errors. Both these assumptions are often violated in real world handwritten datasets and their transcriptions. Hence the process of alignment of the handwritten data with a corresponding corpus includes the identification of the word and character boundaries of the handwritten data as well as the mapping of the handwritten strokes to the characters in the text (see Figure 2.1).

2.2 Related Work

2.2.1 Document Annotation Tools

There have been many studies on automatic document image annotation. There have been document annotation tools that have been designed to handle online and offline content. A truthing tool for generating a database of cursive words was studied in [16].Their data set consisted of around nine hundred sheets of cursive writing. The segmentation was done based on the word spacing and annotation was carried out at word level. PerfectDoc [17], is a suite of tools for manual groundtruthing. The suite consists of tools to create highly accurate ground truth, and also mechanisms to deliver output suitable for web based viewing (XML/HTML). This was suite was meant for annotating printed documents. Manmtha *et al* [47] annotated most of the historical manuscript images manually. As these historical documents are handwritten, conventional recognizers seem to produce accuracies as low as 50% and thus they have to be annotated manually. Manual annotation is labour intensive, and there have been efforts for automatic and semi-automatic annotation of documents. In [47] automatic annotation involving probabilistic annotation models have been studied. Chaudhury *et al* [10] have built tools to annotate word images in a document so that they can be indexed and retrieved using text based search engine methodologies. Hua *et al* [48] have used the OCR to annotate text content in a video, which can be used for video retrieval. Firstly text detection modules identify the presence of text, and then after finding the location of the text, the text is sent to OCR after initial pre-processing techniques. GroundsKeeper [49] is an X-windows based tool that allows a user to display a document image, draw zones of various types around the different page features, and all these drawing zones are fully editable. Each of these zones could be a text (or word) region or an image region within a document image that can be annotated manually. This tool was further used to benchmark segmentation algorithms. Vind(x) [50] is another system that annotates the handwriting information using its recognizer dScript that labels the handwritten data. Vind(x) system was fundamentally meant for pen based annotation and implements concepts like query-by-drawing, query-by-example and text-based querying.

2.2.2 UNIPEN and Annotation Schema

The aim of UNIPEN was to provide the online handwriting community with a common format for facilitating the data exchange [15]. In other pattern recognition fields, such as speech and OCR, significant progress have been made since large corpora of training and test data are publicly available and public competitions are organized to compare recognition techniques on a fair basis. There haven't been similar efforts on the online handwriting recognition front though many companies and universities started collecting data on their own for their internal use. To remedy this problem, the emergence of UNIPEN came into being. UNIPEN format was developed and tested in collaboration with a work group that consisted of members from the online handwriting recognition community. It is an ASCII format designed specifically for data collected with any touch sensitive, resistive or electromagnetic device providing discretized pen trajectory information. Coordinate information such as X and Y are to be provided bare minimum, and there is a provision to extract

additional information such as pen angle and pressure information. The UNIPEN format was not meant for optimized data storage or real time data transmission unlike Jot [14]. It also does not handle color, image rotation, scaling and others which typical ink manipulating applications handle. UNIPEN can handle lots of meta information pertaining to the ink, like the information about writers, segmentation, layout, data quality, labeling and recognition results.

The format is a series of instructions consisting of a keyword followed by arguments. Keywords are reserved words that starts with a dot at the first coloumn in a line. Arguments on the other hand are strings or numbers that are separated by whitespaces. All variables are global, that is, declared variable hold until the next similar declaration. Most of the elements in UNIPEN are optional providing a way to create simple data sets. The pen trajectory is encoded as a sequence of components within the keywords .PEN_DOWN and .PEN_UP containing pen coordinates X and Y (eg. XY or XYT as described by the .COORD keyword). Segmentation and labeling are provided by .SEGMENT instruction, while .HIERARCHY specifies a segmentation hierarchy (such as SENTENCE WORD CHARACTER).

Unipen Tools To facilitate viewing and generation of UNIPEN data, the UNIPEN foundation have released some software tools. *Upview* is a UNIPEN viewer that can display UNIPEN data in various hierarchies as described in UNIPEN file. The user can navigate across multiple files and through hierarchies and can view the segment information including annotation information for that particular stroke or group of strokes in that hierarchy. Most of the UNIPEN files typically have a *.dat* extension. With *Upworks* the output looks much nicer and more colorful. More specifically, the colors allow for visualizing the subsegments within a current segment by the use of color. *Upworks* was built using Tcl/Tk which uses extensive system resources when the large files are presented to them that have bigger segment sizes. There are several limitations in the UNIPEN format. There is no standard way of organizing information in UNIPEN and it does not enforce any strict constraints and it has a scope proble. In order to safeguard the public academic distribution of the UNIPEN data, the International Unipen Foundation (iUF) was raised.

Our Tool We deveoped a which allows for both annotation and visualization of online handwriting data. Several limitations that *Upview* had, have beem well covered. Our tool is meant to read and write UPX file format for online handwriting data. We also additionally support UNIPEN file formats and map the meta information present in the UNIPEN file into the UPX format. The tool supports input and output viewing, editing and annotation of UPX files. The tool is supplemented by a library of basic functions that can be used to access and extract handwriting data from UPX documents based on user-specific criteria.

All these tools help is in building annotated datasets for online handwriting. Manual annotation requires immense efforts and paramount concentration in annotating handwritten data. This leads us to need the need for automatic annotation which is described in the next section in detail.

2.2.3 Annotation by Alignment

The problem of alignment of parallel handwritten and text corpora has been addressed before in the context of segmentation of text lines to words by Zimmerman and Bunke [18]. However, this method assumes the presence of a reliable recognition engine, which is the goal of our annotation process. Tomai *et al.* [51] proposes a similar approach to annotate words of historic handwritten data, where the recognizer is constrained to output the words in the transcript.

The work that is closest to our approach is by Kornfield *et al* [19], where word images are matched to text based on global properties of the word extracted from both the handwritten word images as

well as text words that are rendered using a specific font. The matching is done using dynamic time warping (DTW). Rothfeder *et al.* Rath [52] extends this approach to use an HMM for matching the words, that handles a limited amount of errors in the segmentation as well as transcription process.

The problem of representation of the annotation or ground truth has been studied before. Bhaskarbhatla *et al* [3] presented an XML-based representation scheme for annotation of online handwritten data called *hwDataset* and also created a tool based on the proposed representation. Although the tool attempts to address the requirements of creation of annotated data sets of handwritten data in different scripts, the process still requires selection and annotation of individual characters.

In this chapter, we propose a model-based synthesis and annotation framework, that automatically segments and aligns transcripts for online handwritten documents. In this sense, our approach combines the advantages of the segmentation algorithm in [18] and the alignment capabilities of [19]. Moreover, we do not assume the availability of a recognizer and the algorithm is robust towards noise in the handwriting, as well as errors in the transcription process. The frameworks allows us to learn and update the handwriting model for a writer from the initial annotated data, which in turn can improve the annotation. Our algorithm is tested on a dataset containing multiple writers with varied writing styles as opposed a the single writer collection used by previously reported works. Moreover, the approach can produce annotations at the character level for Indian languages. We use the InkML representation to store the annotation information generated by our algorithm.

2.3 Applications of Annotation

2.3.1 Annotation for Archival

Annotation for archival is one of the major application areas for annotation. Digital libraries, tend to maintain a huge resource archive of digital documents, as the number of such documents typically run into a very huge number. The documents thus are not only to be archived based on their content but in fact for quick indexing must be archived at the meta content level. Meta information could be the, the genre of the document, its author, publisher, ISBN number if its a digitized books, year of publication, edition. To summarize one needs to annotate information at the meta level the ontology information pertaining to that document or sets of documents that share a common meta annotation.

This way of indexing and archiving documents is much more efficient as people tend to search documents based on such meta information mentioned above and is ideal for a digital library application. The Digital Library of India [8] uses a similar mechanism to produce meta information during its archival activity where librarians manually enter/annotate meta information.

Similarly people have created digital archive infrastructure for video and other multimedia data. The challenge of proper retrieval become more difficult for archives containing documents and data that are semantically mixed and search terms are ambiguous. This can be controlled by adding more fields in the meta section that can reduce the ambiguity. Meta information can also be produced using an OCR. Typically document images are converted to text using an OCR, and this in turn can be used as a meta information or an automatic form of annotation for archival.

Annotation has also extensively applied in the field of forensic applications, especially for archiving fingerprint data. Annotation information is extremely helpful in such situations when the amount of fingerprint information runs typically into a large number, and retrieving them becomes a tedious task. Annotation for archival is also well explored in the field of Biometrics, Medical Images.

2.3.2 Annotation for Retrieval

Libraries have traditionally used manual image annotation for indexing and then later retrieving their image collections. There have been automatic image annotation and retrieval methods [53] that have been successfully used. Annotation for retrieval is used not only for images, but also for handwriting documents, video and other multimedia data. Manmatha *et al* [47] built a search engine for historical manuscript images, wherein the retrieval system was trained using an annotated set of 100 pages of George Washington's manuscripts and is used to query a dataset containing images from the same collection. The current approach is to use meta data or indices, which are manually created. This makes automatic approaches to searching and accessing the content very attractive. Another approach to such a problem is to use handwriting recognizers followed by a text search engine. However, in real time the documents, especially the historical documents, are poor of quality which makes the handwriting recognizers vulnerable to poor results. [47] used an alternative approach bypassing explicit recognition.

We applied a similar methodology [9] to retrieve printed document images. In this case, the documents were chosen from the Digital Library of India [8], and were subjected to an offline activity, that pre-processed these images, segmented them into word images, and their features were extracted. Then all these segmented word images were clustered based on their similarity, and the clustered word images were annotated manually giving it a text representation and thereby indexing the whole document image collection for the printed texts. Laverko *et al* [54] used annotation for retrieval video data. The video data primarily consisted of news videos, which needed to be annotated. After the annotation process, similar approaches of using text based retrieval methods were applied as described earlier. In general, annotation based retrieval techniques are widely applied for many multimedia data.

2.3.3 Annotation for Recognition

As seen earlier, annotation has been used widely for retrieval purposes. Annotation plays a vital role in recognition based applications too. Annotation information, which acts as ground truth will be vital in OCR applications that learn and improve classification based on the annotated text and the recognized text. Large amounts of ground truth data that would be suitable as training data for building an OCR with degradations and can be used for testing an OCR and performing analysis of its errors in the form of confusion matrices that is useful for OCR fine tuning.

There are many ways of generating the ground truth, and as seen in the previous section ground truth data for recognition purposes, like the OCR generally are accompanied along with their original documents. Ground truth information, in other words, the annotation can be associated with the document images at various hierarchies. That is, we can have corresponding text associated at the page level, the paragraph level, the sentence level, the word level, and up until the character or stroke level. Typically these annotation information is also very useful for segmentation based routines that can also build upon their segmentation results so that they can further accuracies.

Handwriting recognizers are not known to have high accuracies, with the main constraint being that every individual has his unique style of writing which is the biggest impediment during recognition. Once these styles of handwriting are annotated, then one can group together all handwriting samples at a particular hierarchy. This initial grouping is very important during the initial training phase. Since we already know the text label of the handwriting cluster, we can further extract features that are common across varied samples within that cluster. In this way one can extract a common style information across all writers for a particular chunk of ink that belong to a cluster.

Annotation has widely been used in the area of speech recognition [55] as well. During the

training phase various phonetic sounds are tagged using various tagging techniques that enhance the recognition of a speech recognition system. Sphinx [56] is a popular speech recognition engine from the Carnegie Mellon University, while Festival [57] from University Edinburgh is a popular speech synthesis system.

2.4 Requirements for Annotation of Online HW Data

2.4.1 Desirabilities of the Representation Schema

UNIPEN is a versatile standard for online data storage. However the annotation information is stored as plain text which makes it difficult for any application without a specific parser to understand the contents in it. InkML solves this problem by using an XML based representation which can be understood currently by a large number of applications. InkML is built to represent the trace information for diverse online handwriting applications. Annotation information is not a part of InkML. This leads to the requirement of a standard which can store additional information about the ink, (like the writer, content source, etc) along with the trace data in an InkML framework. Annotations may be needed at various levels of the document structure hierarchy. Some information may pertain to the whole document, where as some information may pertain only to the traces in the document. Thus, a standard which can support the annotations at various levels in the document structure hierarchy is necessary.

Groundtruthing of documents is a key step which allows the developers to evaluate the performance of many document understanding algorithms [58]. Facility should be thus provided in the standard for storing annotations at various levels of the hierarchy. Specific script related issues will have to be kept in mind while working with such a standard. Often recognition engines for non-Roman scripts suffer due to lack of sufficient data and effective representation [58]. The extensibility in XML provides a very convenient way to add more elements if need arises with certain languages.

The creation of annotated datasets on a global scale requires a standard representation for annotation that supports the following requirements (i) script-independence (ii) semantic interpretation of the writing at various user-defined logical levels (e.g. Word, Character) (iii) capture of information about script, writing style, quality (iv) capture of information about writers and data capture environment (v) support for automatic generation of annotation using recognizers, and subsequent manual validation (vi) separation of handwritten data from its semantic interpretations (vii) support for planned as well as casual data collection, among others.

2.4.2 Desirabilities of the Annotation Tool

A tool that confirms to the standard is necessary to create documents in the current UPX format. Tools are required to annotate, manipulate, store and access the online handwritten data. Provision to add the meta information as specified in the standard has to be included. User needs to have a convenient interface for annotating or correcting the annotation at various levels of the document hierarchy. The following lists some basic desirabilities of the annotation tool.

1. Ease of browsing: the annotation tools must be designed such that it provides easy use while navigating through documents and flexibility to browse across files either sequentially one after another or in a batch mode.

2. File formats: The annotation tool must be able to handle various kinds of ink input formats. Popular formats such as the UNIPEN format, the InkML format, formats from popular

digitizers and pure raw ink information must be handled. This would greatly help in using already available ink data. The annotation tool has to save the ink information with the annotation information in a custom format, so that they partially annotated files can be later opened and can be further annotated accordingly.

3. Hierarchy views: The tool must provide ways and easier mechanisms fore viewing multiple hierarchies of the handwritten documents, such as the Paragraph, Sentence, Words and Characters such that are easily discernible to the end user.

4. Manual segmentation and editing: As most of the handwriting segmentation routines are not perfect, the tool must provide ways in which errors in segmentation can be corrected and at the same time group chunk of inks according to the desired hierarchy. Special keys from the keyboard are reserved for editing ink information.

5. Ease of annotation: As the main objective is annotation, the tool must provide appropriate keyboard and mouse interfaces such that the ink information can be properly labeled. It has to be designed so that people proficient in typing can annotate chunk of ink fast and effectively.

6. Display of annotation label: As the user keeps annotating chunk of ink, the label identifying the ink has to be displayed appropriately. This being manual annotation, the user keeps checking of what the ink information convey and the label he has assigned. This is of paramount importance especially when it comes to Indian scripts which are typed in Roman (ITRANS [42]) and are displayed as Unicode characters (fonts) on the screen. Typing in ITRANS is more intuitive rather than using pre-allocated character positions for Indian scripts. INSCRIPT [59] keyboard is one such example. But the annotation tool must also provide for such a facility, if the annotator is familiar with such keyboard scheme.

7. Meta data: There must also be a form filling mechanism to enter meta information related to the ink, such as the writer, his skill sets, about the annotator, the script in which it was written, segmentation and hierarchy information and others pertaining to ink.

8. Ink display: The tool must support zoom facility so as to view ink strokes according to a users wish. Additional image processing related to stroke thickness, smoothness, location, flipping should be configurable according to user choice.

2.5 UPX Representation [1, 2, 3]

Our representation for annotated datasets of handwriting is called hwDataset, and it includes several elements for detailed annotation of handwriting, which extend the basic traceRefGroup element of the core InkML. The hwDataset element is the root of the XML document and captures meta-data about the dataset under datasetInfo, various definitions as part of datasetDefs, and hierarchical annotation of handwritten data under hwData These elements are described briefly in the following paragraphs.

DatasetInfo: The datasetInfo element captures metadata related to the dataset as a whole. It contains the following elements: (a) *name* - name for referring to the dataset (b) *category* - type of dataset (c) *version* - version number and/or datestamp of publication (d) *contact* - contact info for dataset-related queries (e) *source* - source of collected data (f) *setup* - physical conditions in which data was collected (g) *dataInfo* - information about the data

The dataInfo element in turn contains the following sub-elements: (a) *script* - language/script captured in dataset (b) *quality* - quality of handwritten data captured in dataset (c) *truth* - truth of what is captured (d) *methodology* - design of data and collection procedure (e) *annotationScheme* - description of annotation scheme

datasetDefs The datasetDefs element captures information about different writers and sources of labels (annotation) represented in the dataset, and provides the means for referring to them later in the document. It contains the following elements:

- writerDefs - declarations of writers as a sequence of writer elements
- labelSrcDefs - declarations of sources of annotation as a sequence of labelSrc elements

The writer element in turn contains the following elements: (a)*date* - date when writing occurred (meant as a coarse description as opposed to the trace timestamps in the core InkML) (b) *personal*- personal information including (c) *hand* - left/right handedness (d) *gender* - gender (e)*age* - age at the time of capture (f) *skill* - level of skill with script (g) *style* - predominant writing style (h) *region* - native region

The labelSrcDefs element contains the following elements: (a) *name* - name of the human or automated source of labels (b) *source* - organization that this label source represents (c) *time* - date and time of annotation (d) *contact* - contact details of label source (e) *labelTypes* is an attribute and describes the categories of labels generated by the source (e.g. truth, quality, script, style, etc) and their character encoding (e.g. Unicode).

The above elements provide a mechanism for representing the writing of different writers in the same dataset, as well as multiple sources and categories of annotation for the same handwritten data. An algorithm for script identification might be used as a source of script labels, while a human annotator may provide labels for truth as well as script, style and quality of writing. Of course, the representation can also accommodate multiple label sources for the same category of label information, e.g. one or more recognition engines for truth labels and a human annotator for their validation.

hwData The hwData element allows hierarchical organization of annotation. It typically contains the root of the annotation hierarchy defined by the user, denoted by the element $H1$ Each level of hierarchy $H(i)$ contains a label element that captures annotation information at that level. $H(i)$ also contains either one or more $H(i+1)$ elements or hwTraces, the leaf element of the hierarchy that refers to raw ink traces represented using InkML.

The $H(n)$ elements are meant to be used to indicate the hierarchical structure of handwriting, and assigned meaningful names such as PARAGRAPH and WORD using the corresponding attributes of the hwData element.

The label element at each level can be used to capture alternative choices of label with confidence values if any, and the time of annotation. Although primarily intended to describe the truth value of a particular set of ink traces, it may also be used for describing other characteristics such as writing style, quality and script. The timestamp can be used to generate the history of annotation spanning different label sources of a particular unit of writing. The alternates can be used to facilitate the process of manual validation by prompting options for human validation.

Formally, the attributes of label are (a) *id* - identification of label (b) *labelSrcRef* - reference to label source defined earlier. This holds good for sub-levels of the current level except where explicitly overridden (c) *category* - category of label (e.g. truth, quality, script, style, etc) (d) *timestamp* - time when the act of annotation is performed.

2.6 Annotation Tool

2.6.1 Requirement Analysis

Standards Compliance

The toolkit complies to standard ink formats, such as the UNIPEN, InkML, and raw strokes separated by a delimiter. The toolkit produces the UPX format as the output which is in an XML format. The tool has to adhere and extract common attributes that are present in the UPX format and are available in the above mentioned input formats. The faithfully reproduces all the input file formats into UPX format without altering the original stroke information. The tool further supports batch operations for opening files for all the supported formats so as to enable a user to annotate files at one complete go. The tool gracefully handles unknown file formats, standard versions unfamiliar to the tool and maintains the compliance feature.

Functionality

The tool has many functionalities, and they have been enumerated below.

1. Segmentation: The tool provides the user with an important functionality of segmenting (grouping) chunks of ink according to the desired hierarchy (viz. Paragraph, Words, Characters or Strokes). There are some custom segmentation routines provided to the user to segment ink information at word and line level that will reduce the effort needed to be put in by annotators when annotating large data. The tool also provides for methods for manually correcting/editing errors that arise due to these segmentation routines.

2. Annotation: The tool supports easier and faster ways of annotation. This is the most desired functionality of tool wherein the user annotates the chunks of ink thereby tagging them depending on its current hierarchy.

3. Editing the UPX: The tool can open partially annotated UPX files, so that the use can start editing and annotating the files from the point it was previously left. This functionality helps users immensely as annotators need to annotate files completely, rather can annotate in sessions or fixed intervals of their choice.

4. Meta Data: The UPX format carries enough meta data that is part of its schema specifications. Some of these information are automatically populated while others need to be specifically supplied by users. The form based interface allows the user to enter mandatory data that is needed for the UPX format. The tool does not produce a UPX file unless all the necessary form fields necessary for the faithful reproduction of the UPX format is not supplied.

Statistical Analysis and Visualization

As the users keep annotating scores of pages, the statistical analysis provides information to the user on the number pages annotated during the current session, number hierarchies and its elements annotated for the current document and others statistical information as an information tab that helps the user in further analysis. These information snippets can also be visually seen. The tool also provides for a video demonstration that plays the manner in which the strokes were originally written by the writer and maintains the stroke order intact. This functionality helps the annotator to study the writing habits of the writer and his style.

User Categories

The tool has a login mechanism, with *Admin* having all the privileges. Only some functionalities are supported for *guest* or other annotators, and they cannot change the meta information. User categories are defined by the *Admin* who grants specific permissions to annotators and their capabilities. Usually one annotator cannot modify an another annotators meta or annotation information.

Interoperability

1. Common code: The tool has a common code base, so that the same code can be complied and executed in both Linux and Windows XP operating systems. This makes sure that the tool can accessed by a variety of users depending on their familiarity with the operating system and the same look and feel is adopted across operating systems. C++ has been used the programming language throughout.

2. Source code documentation The source code has been documented throughly, and the comments are written following the specifics of Doxygen [60], that generates documentation for the complete toolkit. The documentation is supplied in HTML and PDF with appropriate UML diagrams where ever necessary.

Usability

Usability of this tool has been widely and extensively studied as this is the crucial part that is critical from an end-user's perspective. The location of every GUI element (viz. buttons, drop-down boxes, check boxes, radio buttons, stroke drawing area) have been well thought and is the outcome after extensive deliberations and rationale.

1. Installation Procedures: Standard installers, like setup.exe, which is a wizard like interface for Windows XP was provided, and source and binary files in the form of RPM packages for Linux were provided. Static binaries and source code, compressed by ZIP was also provided for the end users. Users who wanted to compile and use the source code from scratch were provided with a standard *makefile* that can be used across operating systems. Utility such as *make* for Linux and *nmake* for Windows XP are required to make compile the sources, while QT-3.1 [61] or above needs to be installed prior to compiling of this code. Theses were also provided to users.

2. User Manual: The user manual consisted a step-by-step detailed presentation on how to use the tool and what every feature in the tool was meant for. Ranging from a novice user to an expert user, the content int he manual was appropriately divided and written.

3. UI Features: The GUI, which was primarily built on QT had loads of functionality to offer. On the usability front from the UI interface point of view, the drawing area, where the ink information was displayed always had to be a static area that was always to be seen and this occupies around 80% of the main display panel. The panel towards the right consisted of most frequently used functionalities, typically the segmentation, annotation, hierarchy views, and the annotator information in addition to some UI functionalities like Zoom and others. The Menu bar consisted of other functionalities that were less frequently used, such as the statistical information, language selection, stroke video, meta data and others.

4. Display: The display features, such as the Zooming facility gives the user that extra capability to view the strokes at sizes which he desires. Initially when a file is opened by the tool, all

the contents of the ink are scaled so that they fit well within the bounding box of the display canvas so that the user can quickly get a feel of how the contents in the file are arranged. Furthermore, the tool can reduce the width of the strokes, can flip them both ways (horizontal and vertical) and can smoothen them by applying some image processing routines over the entire range of ink strokes. These visual effects only are used during the display and the original content is never altered when the file is saved in the UPX format.

Testing

Standard testing procedures were adopted and followed, as the tool went through various modes of testing that included both the white-box testing and black box testing.

1. Crash Testing: The tool was subjected to severe crash testing procedures. As the stability of the tool was the top priority, several user's first priority was to crash test it and observations pertaining to the tool's behavior was recorded according to strict testing guidelines that emphasized on various behavioral aspects of th tool.

2. Use Cases: Several use case scenarios were presented, and the tools functionality was then checked with the desired functionality and was documented and reported accordingly. Typical use case scenarios include, handing over the tool to users from various backgrounds and different levels of expertise in computers, their reactions and suggestions, providing a set of input files and comparing the annotation outputs from various annotators.

3. Standard annotation use cases: Since annotation output, which is the UPX format was the most crucial, standard annotation use cases pertaining to annotation were made so that the output was clearly checked with a standard UPX schema so that basic things like conformity to the UPX format, annotation information in proper encoding, meta information and others were properly and completely verified. The input files typically included of files from various formats, various languages, different writing styles from different users across a varied age group.

4. Memory Profiling: Standard memory profile software (eg. *memprof*) were used to study and eliminate any memory leak the tool consisted. Profiling information also provides information on the overall performance of the tool and its stability.

5. Robustness: The Robustness and stability of the tool was ensured before it was deployed in both the operating systems and was tested in various hardware architectures.

2.6.2 Indian Language Support

The tool is primarily meant for annotating Indian language handwriting. Indian scripts are not easy or direct to annotate. The tool supports ITRANS [42] based annotation. Indian languages are phonetic in nature. Indian languages share a common alphabet and this is what ITRANS primarily exploits. ITRANS has a roman equivalent for every Indian language alphabet so as to enable users to type in Roman and visualize in the corresponding Indian language. In order to view the alphabet in the corresponding Indian language, we need a font. Instead of making the output of annotation dependent on Font for a particular Indian language, we have adopted the Unicode [62] encoding scheme. The primary advantage of using Unicode encoding is that Unicode is the *de facto* encoding standard which is primarily inherited from ISCII [63]. Also the fact that Unicode Fonts takes care of the compounded alphabets, all our ITRANS converters render output

in UTF-8 encoding. Figure 2.2 shows where the ITRANS and Indian language content appear on the GUI.

The toolkit displays the Indian language text in Unicode at the bottom as the user keeps typing in ITRANS and the conversion happens on the fly. On the other hand, users proficient in using standard keyboard layouts, like the INSCRIPT [59] keyboard, that allows the users to type in Unicode directly is also well supported by the toolkit.

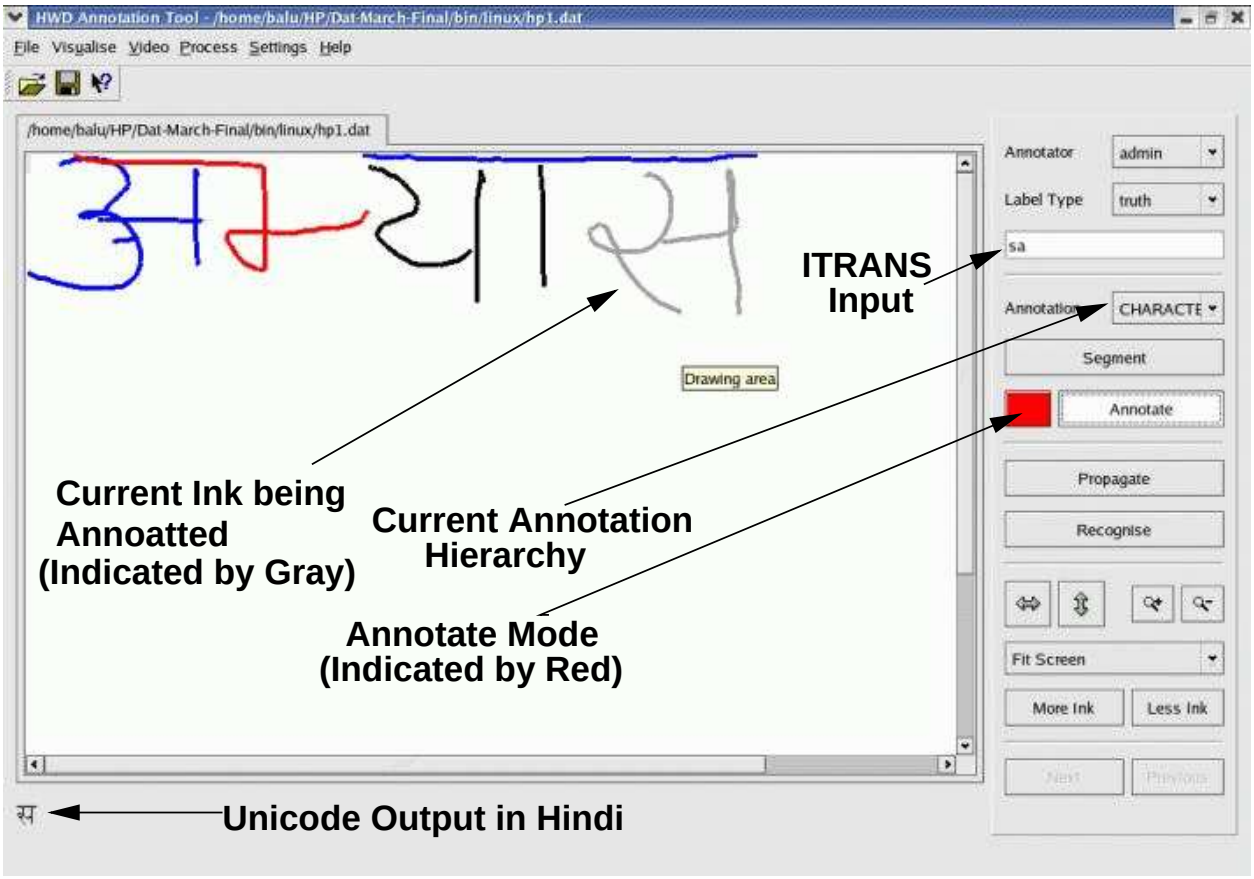


Figure 2.2: Figure Illustrating the Various Regions of Interest in Our Toolkit.

2.6.3 Design

Software Architecture

The software uses a plugin-based architecture, facilitating extensibility with the existing set of functionalities provided. The annotation process can be made semi-automatic or automatic by plugging in the corresponding recognizer of the script being annotated. The tool comes with a basic set of stroke signal processing routines. Users may write their own plugins using the simple interface provided and can see the outputs of the routines using the tool.

The tool is designed in such a way that the UI details are kept separate and independent of the core implementation. The *Document* class, as shown in Figure 2.3, offers the core implementation, that contains the data structure that entails the UPX schema, and contains sub-classes comprising of the *Stroke* and *Hierarchy* data structure. The rest of them form core implementation of the components that comprise the UPX format with the root node being the *HWDataSet*. The *Document*

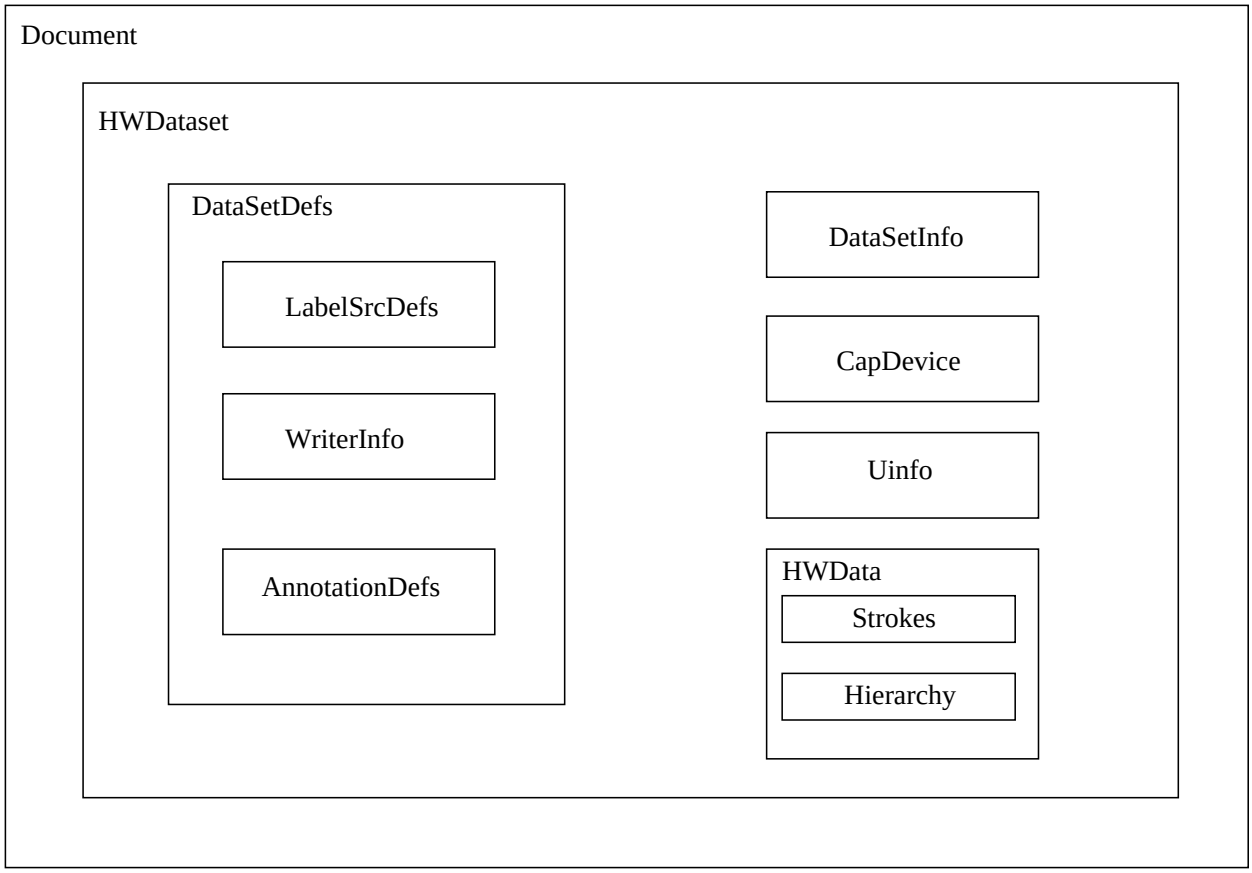


Figure 2.3: Figure Illustrating The Various Classes of the Document Class

class is basically a Singleton class, so that it can be used anywhere in the UI with essentially only one instance of its object existing.

User Interface

A screenshot of the tool's GUI is shown in the Figure 2.5. It has two components, working space is on the left side of the tool window occupying a major portion, and the right side has a toolbox with buttons for all the key functionalities. When a file is opened, false coloring of the hierarchical elements (like word, sentence, etc) is done based on the hierarchy selection made by the user, where neighboring elements are shown with different colors. If a hierarchy is already segmented (for example, automatic word segmentation and line segmentation), the elements of that hierarchy are false colored. If the selected level is not yet segmented either automatically or manually, stroke level false coloring is shown. This will allow the user to improve the segmentation, if needed.

Figure 2.4 shows the various components that comprise the GUI, with the base class being the *HWDisplay* that is derived from *QMainWindow*. All the class names that start with *Q* are essentially QT [61] classes. Qt sets the standard for high-performance, cross-platform application development. It includes a C++ class library and tools for cross-platform development and internationalization. As can be seen, various GUI styles like, Tabs, ScrollViews, Canvas, Plugins were used all throughout the GUI building. Figure 2.4 also contains the Singleton class, *Document*, which is used as an interface for accessing the core routines that are independent of the User Interface. Plugins such as Segmentation and Recognition provides a convenient mechanism for

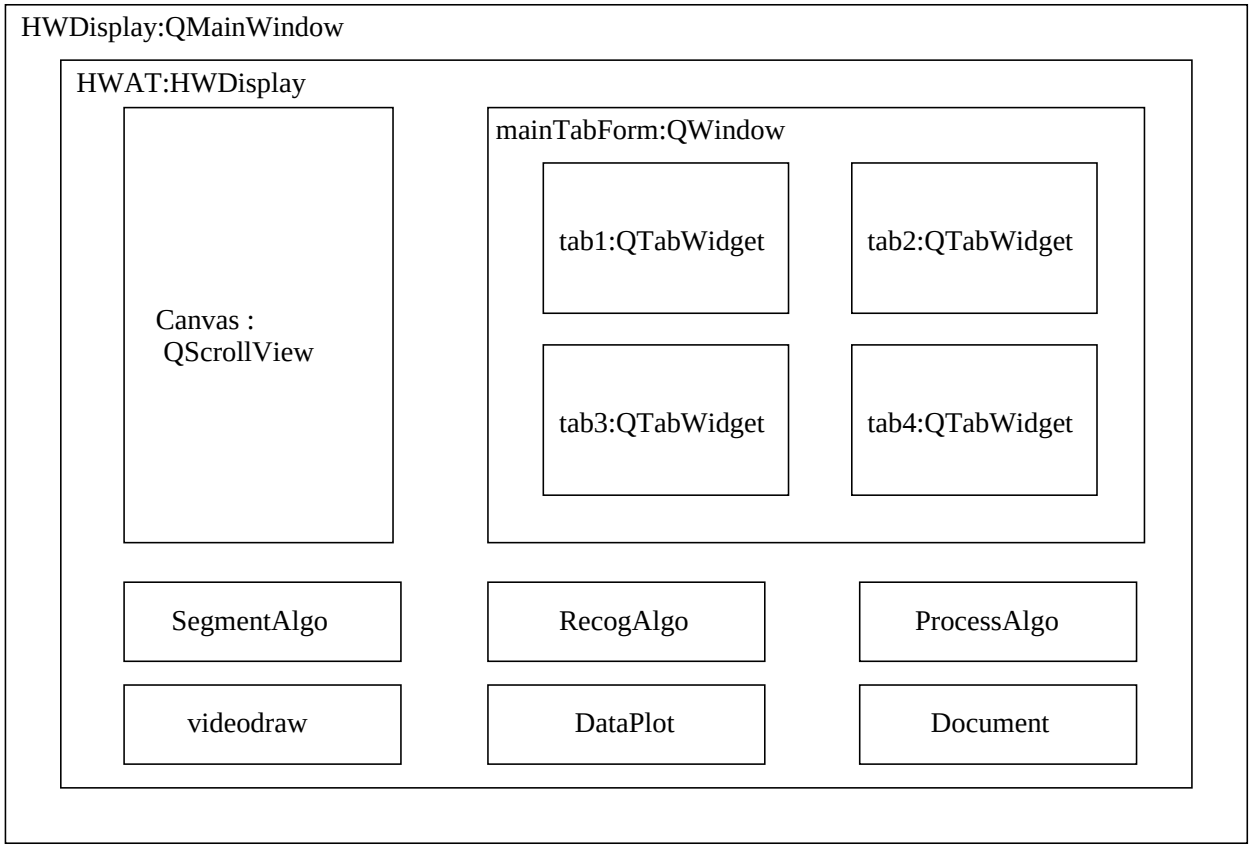


Figure 2.4: Figure Illustrating the Various Classes of the HWDisplay Class of the GUI

developers to add their own routines without actually recompiling or modifying the original source code. The *Canvas* (Figure 2.5 (b)) area is sued to draw the ink information, while the meta data is added through a Tabbed interface (Figure 2.5 (a)) that is divided in fashion that is convenient to annotator to add meta information in lieu with the UPX format.

Directory Structure The directory structure is structured so as to contain all files with similar functionalities at one place depending on the operating system it would be used. The *common* directory contains all files that are operating system independent, while *win* and *linux* contains files that are Windows XP and Linux dependent. The *lib* directory contains all the shared libraries (dynamic linked library, DLL) and plugins, and the *bin* directory contains all the executables. The *resources* directory contains the map files for Indian languages, the necessary Unicode fonts and other resource files useful for the tool.

2.6.4 Annotation Procedure

Annotation methods are provided with the tool with various levels of automation. Manual annotation can be done where the user may directly key in the annotation at the specified level. Semi-automatic annotation is achieved with designed text, wherein the annotations can be propagated. Also a recognizer can be plugged into the tool for automatic annotation. A generic interface is provided for this purpose.

Easy annotation at multiple levels

Automatic segmentation is provided for stroke, word and line levels. User can improve the segmentations if needed. For annotation, user could select the level of annotation-hierarchy. (eg. line level, word level or stroke level.) User selects next element with a selection-key and annotates the element. Note that the overhead of annotation is only one selection key press. The annotation is possible in various formats like ISCII, ASCII, ITRANS etc, although a display map has to be supplied to the system by the user in order to see what is being typed in the language in which it is being typed.

Annotation propagation for designed text

Annotation propagation is another key aspect of the tool. In cases of collecting the data with repetition (eg: signature verification dataset), the annotation for one element may be 'propagated' to the specified number of elements. The propagation may involve copying of the annotation at that particular hierarchy, or copying the whole tree of annotation involving all the lower levels. For example, propagation of word level annotation may mean propagation of word and stroke level annotations. Annotation propagation can be naively done by counting the number of strokes and assigning the labels corresponding to the strokes with same indices. More intelligent way of doing this would be to use sequence matching algorithms based on Dynamic Time Warping to properly identify the associations at stroke level between two words. Both of them are provided in the tool allowing the user to choose between them based on the requirements.

2.7 Discussions on UPX Vs UNIPEN

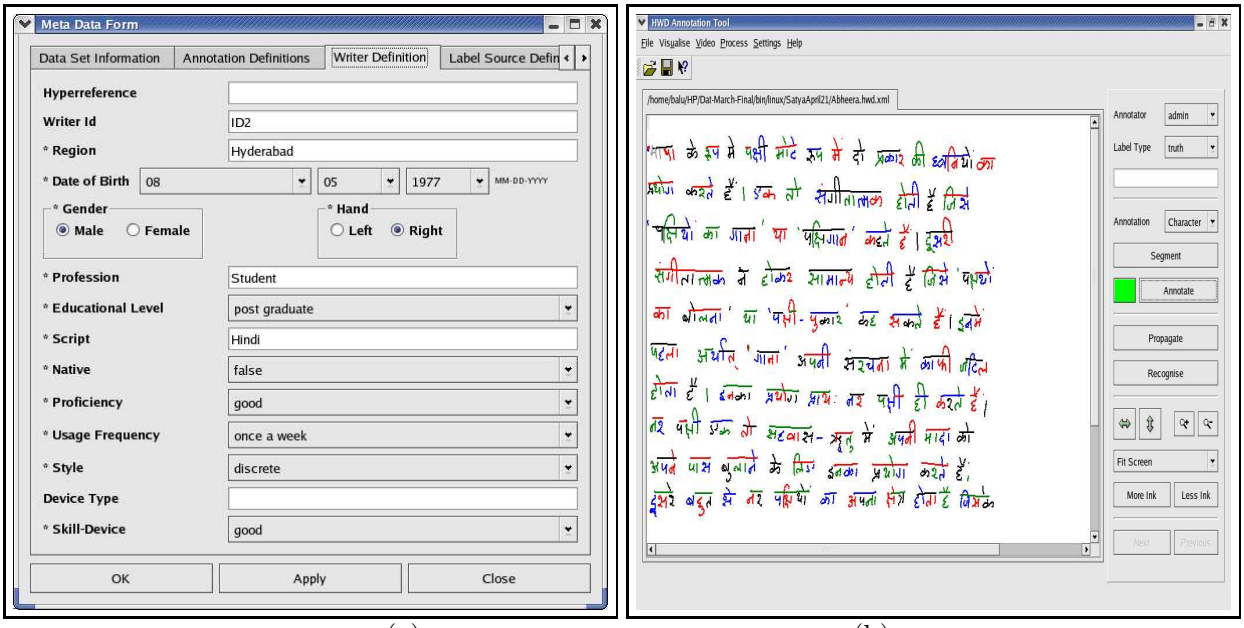
The hwDataset representation by design attempts to satisfy the requirements laid out in the first two sections. Script-independence is achieved by supporting different encoding standards for the truth values, and use of an XML based file format. The representation supports semantic interpretation of the writing at various user-defined logical levels and captures information about script, style, quality at these levels. In addition, these attributes may also be associated with the dataset as a whole, or with specific writers. The representation captures meta data related to capture conditions and the writer as part of writer and datasetInfo. The *label* and *labelsrc* elements provide the means to capture recognition alternates along with confidences, thus supporting the use of multiple sources of annotation for the automatic generation of annotation and subsequent manual validation.

2.7.1 Comparison

While the proposed representation is inspired by the UNIPEN standard, there are some important differences between the two. hwDataset is an XML representation (currently instantiated as a schema) unlike UNIPEN which uses a custom text format. Unlike UNIPEN, hwDataset contains only annotation and does not contain any information about the raw ink data or the digitizer used. These are left to the core InkML schema. This allows the separation of ink from annotation, which are combined in the same document in UNIPEN. hwDataset does not include support for evaluation of recognition engines, although it does support their use in the annotation process. As a consequence, UNIPEN elements relating to alphabet used, inline lexicons, characterization of dataset as being training, test or adaptation set and recognition results do not have an analog in hwDataset. UNIPEN is based on the ASCII format and has been used for significant data collection, primarily for cursive English. Though there have been attempts to use UNIPEN format

for creating datasets in Kanji and Arabic, such representations have limitations. The data design and word-lists used for planned data collection are supported only as references in the hwDataset representation.

Comparison of tools The *HWAT* (Hand Writing Annotation Toolkit is primarily an annotation tool, while the *Upviews* is meant to view annotation and hierarchy information. *HWAT* provides additional functionalities for segmenting ink data according to their hierarchies and view them, while *Upviews* can only display ink based on the hierarchy. In addition, annotation information in various encoding such as Unicode and font level can be viewed and annotated in *HWAT*. Meta information pertaining to ink can also be populated in *HWAT* which are typically unavailable in UNIPEN tools. *HWAT* also provides for annotation propagation across chunks of inks.



(a) (b)

Figure 2.5: (a) A Section of the Tabbed Interface for Collection of Metadata Related to the UPX (b) Telugu Text Being Annotated at Character Level After Automatic Character Segmentation.

2.7.2 Other Advantages

Additional advantages of the proposed representation include:

- The hwDataset uses predefined XML standard elements for defining certain parts of the standard wherever possible, eg. address of the writer, annotation software description (Eg. OSD [64]). This enables software not concerning to the handwriting applications also to understand certain portions of the data.
- Since the hwDataset document contains only the annotation of digital ink data, annotation is kept strictly separate from the digital ink that it describes. This arrangement allows different kinds of annotation referring to the same ink to be added over time
- The separation of ink and annotation also allows *casual* data collection involving one or more writers, i.e., annotation of ink generated in the course of an ink application such as ink chat.

- The representation also supports multiple hwData blocks of annotation distinguished by their *trialId* attribute - a feature designed to support planned data collection.
- Some of the elements can take two kinds of values – one being the value from a fixed set of strings, and the second being any textual description of the attribute as the value. A dual annotation scheme for certain attributes is thus suggested, where their quantified parts are machine understandable, and description parts store more information for the human understanding.
- Support for variable hierarchies is provided. The hierarchies can be defined by the user in the hierarchyDefs section according to the hierarchy required to represent the current document.
- The hwDataset has been designed to incorporate as much information as possible that can be obtained from a handwritten document keeping in view of futuristic applications.

2.8 Model-based Annotation

The overall approach is illustrated in figure 2.6. The input to the algorithm is a sequence of online handwritten data referred to as the input handwriting and the corresponding transcription, referred to as the text sequence. A synthesis module is used to convert the text sequence to the corresponding handwritten form using a model of handwriting of the writer. This forms the reference handwriting, where the segmentation and ground truth is accurately known. The input handwriting is then matched with the reference handwriting using a two-stage elastic matching module. Once the best match is identified, the ground truth is propagated to the words and characters of the input handwriting.

Once the annotation is available, the handwriting synthesis module can refine the parameters of the handwriting model of the writer and repeat the annotation process using the updated handwriting synthesizer. Thus the framework allows us to refine the annotation information over time. We will now describe the process formally.

2.8.1 Definitions

Online handwritten data is represented as a sequence of strokes: $\langle s_1, s_2, \cdots, s_K \rangle$, where a stroke is defined as the trace of the pen staring from a pen-down to the following pen-up. A word, w_i, forms a contiguous sub-sequence, $\langle s_j, s_{j+1}, \cdots, s_{j+k} \rangle$, where s_j is the starting stroke of the word w_i. Hence the data may also be represented as a sequence of words: $W = \langle w_1, w_2, \cdots, w_N \rangle$. The text corpus that corresponds to the handwritten data is a sequence of characters, that are separated by a blank character at the word boundaries: $T = \langle t_1, bl, t_2, bl, \cdots, t_M \rangle$. Each t_i corresponds to a text word, which is a sequence of characters: $\langle c_1, c_2, \cdots, c_J \rangle$.

The input handwriting, is a sequence of strokes, which could be grouped into a sequence of words. the problem of segmentation or grouping of strokes into words is often difficult because there might not be a clear spatial separation between the last stroke of a word and the first stroke of the next. We utilize the blank character information present in the text data to aid the word segmentation process. During the segmentation process, the stroke sequences are also aligned with the characters in the text, thus generating a word-level and character-level annotation of the handwritten data.

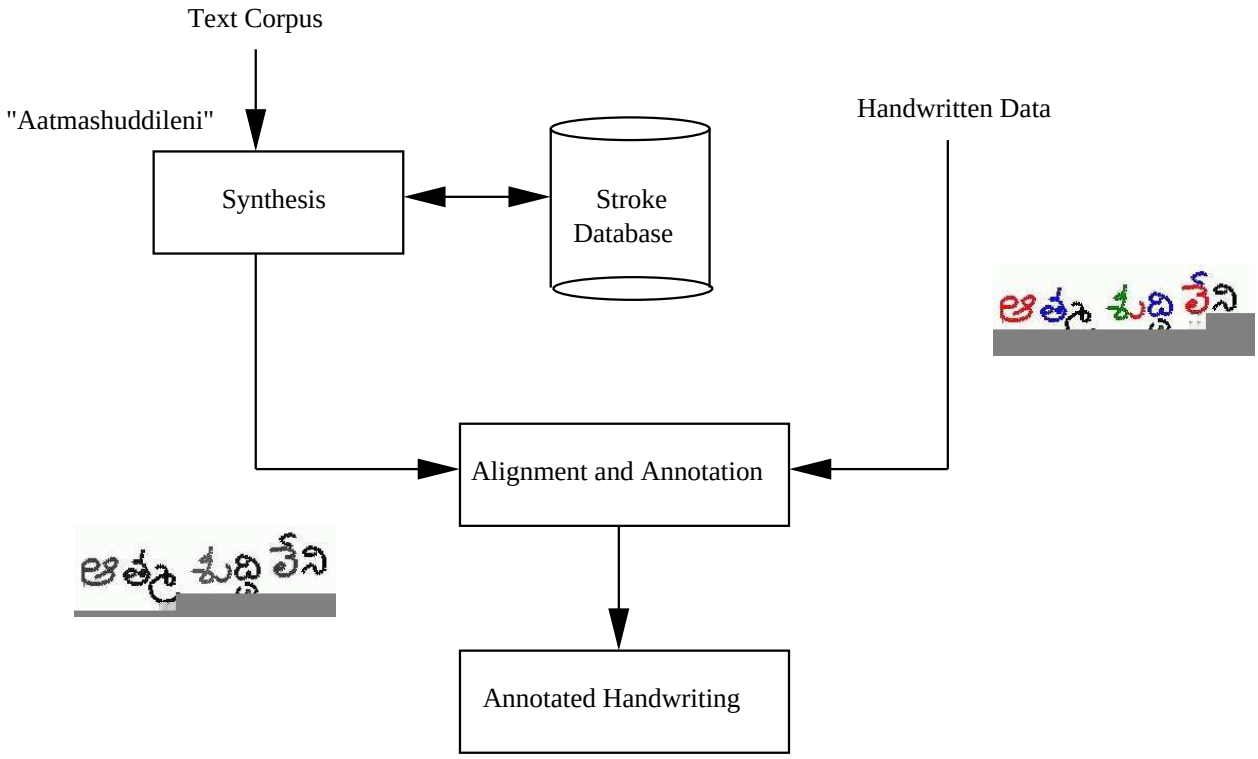


Figure 2.6: Block Diagram of the Model-based Annotation Process.

An important assumption in our implementation is that a character is always composed of an integral number of strokes. Thus the problem of character level annotation is reduced to finding out the set of strokes that correspond to a particular character in the text representation. The approach could also be extended to work with cursive writing, where a single stroke might span multiple character.

A primary requirement to do alignment, either at the word level or character level is that there is a way to match a text word/character with a handwritten stroke. However such a matching is not trivial due to the variations that are possible within a single character class in the handwritten data. The next section describes a handwriting generation approach that can be used to generate a handwritten word from text word, which is then used for matching with the handwritten sequence.

2.8.2 Handwriting Synthesis

The process of generating a handwritten equivalent of a given text word is referred to as handwriting synthesis. Depending on the application, the synthesis could generate a handwritten word in the writing style of a specific user or a generic one. Our approach to synthesis of handwritten words is as follows.

The synthesis module consists of two three parts: i) Conversion of a text word into a sequence of handwritten stroke classes, ii) Computation of candidate strokes that could be used for each stroke class, and iii) Computing the spatial layout of the candidate strokes to arrive at the final handwritten word.

The set of stroke classes that constitute a word or character is learned from a corpus of training samples that are annotated at the character and stroke levels. The learning could be writer specific or writer independent as the application mandates. Template strokes are also identified for each

stroke classes in this process. To synthesize the strokes for a particular word, we use a deformation model that takes the template strokes and modifies them within the parameters of the desired writing style. A third step learns the spatial bi-gram distribution of the strokes from the training corpus. This is used to combined the synthesized handwritten strokes into a single word. Details of the handwriting synthesis algorithm used in this work can be found in [65]. Figure 2.7 shows a sample word in *Telugu*, that is converted to handwriting using our approach.

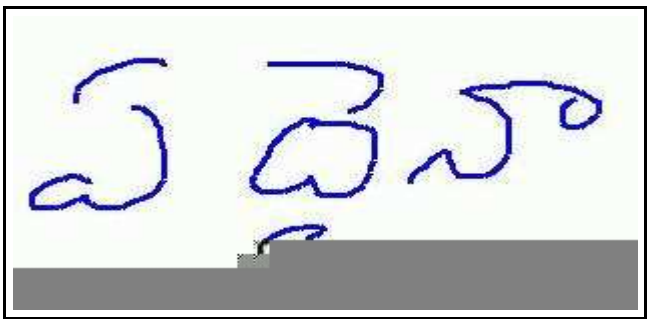


Figure 2.7: Figure Illustrating the Synthesis.

Once the handwritten word is generated, the strokes are mapped to a feature space representation for matching.

2.8.3 Matching Handwritten Strokes

Distance or dissimilarity between two scribbles (sets of strokes) is computed using a set of features extracted from the group of strokes. Each stroke consists of a sequence of sample points, (x, y) that describes the trace of the pen during writing. The strokes are first converted into a sequence of feature vectors, extracted from each of the sample points. The feature vector consists of:

1. The direction, θ, of the tangent to the stroke curve
2. The curvature, c, of the stroke at the sample point, and
3. The height, h, of the sample point from the word baseline

Figure 2.8 illustrates the computation of the three features.

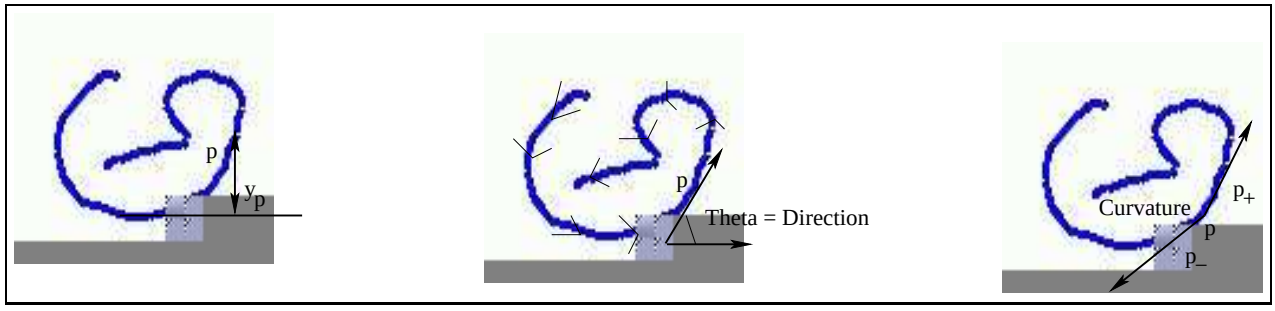


Figure 2.8: The Curvature, Height and Direction Features, Extracted from the Sample Points on the Curves.

The distance between two feature vectors $F_1 = \langle \theta_1, c_1, h_1 \rangle$ and $F_2 = \langle \theta_2, c_2, h_2 \rangle$ is defined as the weighted Euclidean distance between the two vectors:

$$D^2 = k_\theta * (\theta_1 - \theta_2)^2 + k_c * (c_1 - c_2)^2 + k_h * (h_1 - h_2)^2,$$

where ks are the weighting factors. A sequence alignment score computed using a Dynamic Time Warping (DTW) procedure. The use of the total cost of Dynamic Time Warping as a similarity measure is helpful to group together strokes that are related to their root character by partial match. Dynamic Time Warping is a dynamic programming based procedure to align two sequences of signals. This can also provide a similarity measure. Figure 2.9 shows the DTW based matching of two strokes in *Telugu*.

Let the strokes (say their curvatures) are represented as a sequence of vectors $F = \mathrm{F}_1,\mathrm{F}_2,\ldots,\mathrm{F}_M$ and $G = \mathrm{G}_1,\mathrm{G}_2,\ldots,\mathrm{G}_N$. The DTW-cost between these two sequences is $D_s\,(M, N)$, which is calculated using dynamic programming is given by:

$$D_s(i,j) = min \begin{cases} D_s(i-1, j-1) \\ D_s(i, j-1) \\ D_s(i-1, j) \end{cases} + \quad d(i,j)$$

where, $d\,(i,j)$ is the cost in aligning the i^{th} element of F with j^{th} element of G and is computed using squared Euclidean distance:

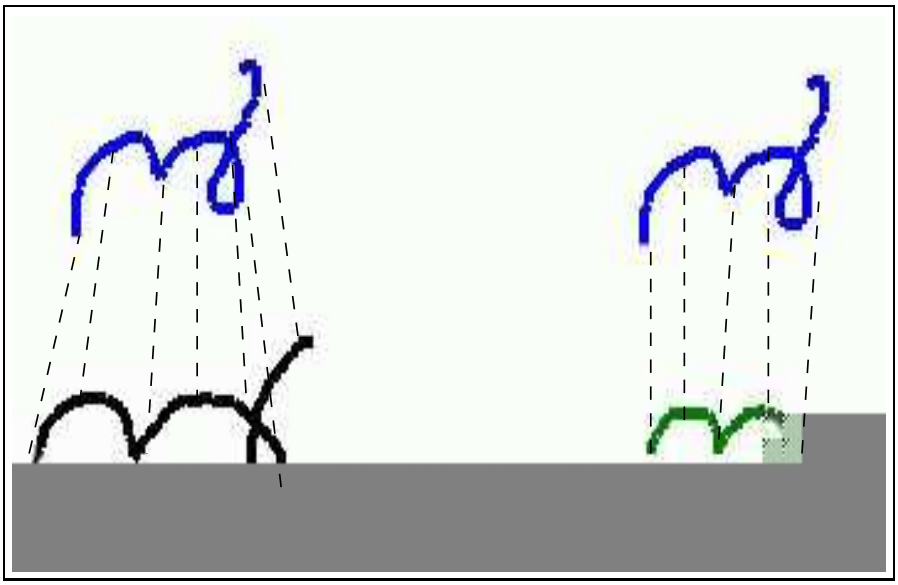


Figure 2.9: DTW-based Matching of Two Strokes.

Using the given three values $D_s(i, j-1)$, $D_s(i-1, j)$ and $D_s(i-1, j-1)$ in the calculation of $D_s(i,j)$ realizes a local continuity constraint, which ensures no samples left out in time warping. As in Figure 2.10, we also imposed global constraint using Sakoe - Chiba band [4] so as to ensure the maximum steepness or fatness of the DTW path. Score for matching the two sequences F and G is considered as $D_s\,(M, N)$, where M and N are the lengths of the two sequences.

2.8.4 Character and Word Level Annotation

We use the stroke matching module to come up with the best alignment of the strokes to the corresponding characters and hence words. As we pointed out earlier, we assume that each character in the text corresponds to one or more strokes in the input handwriting. Hence, annotation is the process of mapping a sequence of strokes from the input handwriting to the corresponding character. Once the characters are mapped, the segmentation and annotation of the words are straight forward. However, computing the best assignment of strokes to characters is not trivial as multiple strokes can form a character.

We employ a modifier version of the elastic matching or dynamic time warping (DTW) algorithm to solve this problem. With the assumption that multiple strokes might map to a single character, we formulate the problem as follows:

Let $S = \langle s_1, s_2, \cdots, s_n \rangle$ be the stroke sequence in a word and let $C = \langle c_1, c_2, \cdots, c_m \rangle$ be the corresponding handwritten characters synthesized from the transcript. The problem is to find the best alignment, where $n > m$. The cost of the best alignment is computed as:

$$D_w(c_i, s_j) = min \begin{cases} D_w(c_i, s_{j-1}) + Penalty(s_j) \\ D_w(c_{i-1}, s_j) + Penalty(c_i) \\ D_w(c_{i-1}, s_{j-1}) + D_s(c_i, s_{j,j}) \\ D_w(c_{i-1}, s_{j-2}) + D_s(c_i, s_{j-1,j}) \\ D_w(c_{i-1}, s_{j-3}) + D_s(c_i, s_{j-2,j}) \\ D_w(c_{i-1}, s_{j-4}) + D_s(c_i, s_{j-3,j}) \end{cases}$$

where, $D_s\left(c_i, s_{j,k}\right)$ is the matching score obtained from the stroke alignment routine in aligning the ith character of C with j^{th} through k^{th} strokes of S. The feature vectors of the j^{th} through k^{th} strokes are concatenated to compute this matching score. The score for matching the two sequences C and S is considered as $D_w\left(M, N\right)$, where M and N are the lengths of the two sequences.

Detecting Word Boundaries: It may be noted that the above algorithm assumes that the word boundaries are known for the input handwriting sequence. However, this is not the case as we described in the introduction. Hence we modify the above DTW algorithm to account for word boundaries. To compute the ending of a given word, w_i, in the text sequence, we augment the characters of w_i with one or more characters from the following word, w_{i+1}, at the end of the characters of w_i. Once the matching is performed, we identify the minimal distance of matching the augmented sequence with any sequence of strokes and then remove the strokes that mapped to the augmented characters. This method was found to be robust enough to detect the word endings in the experiments that we performed.

2.8.5 Updating the Handwriting Model

Once the annotation is performed at the character level, we update our handwriting model by replacing the stoke models for the writer with samples from the current document, where the matching scores are high. Note that the replacement can be performed only for those characters that are present in the document. However, this is sufficient as the updated model is only used for the document under consideration.

Once the handwriting model is updated, we use the updated model to carry out the annotation process. We repeat the cycle until there are no more changes to the result of annotation or a threshold is reached for the number of cycles.

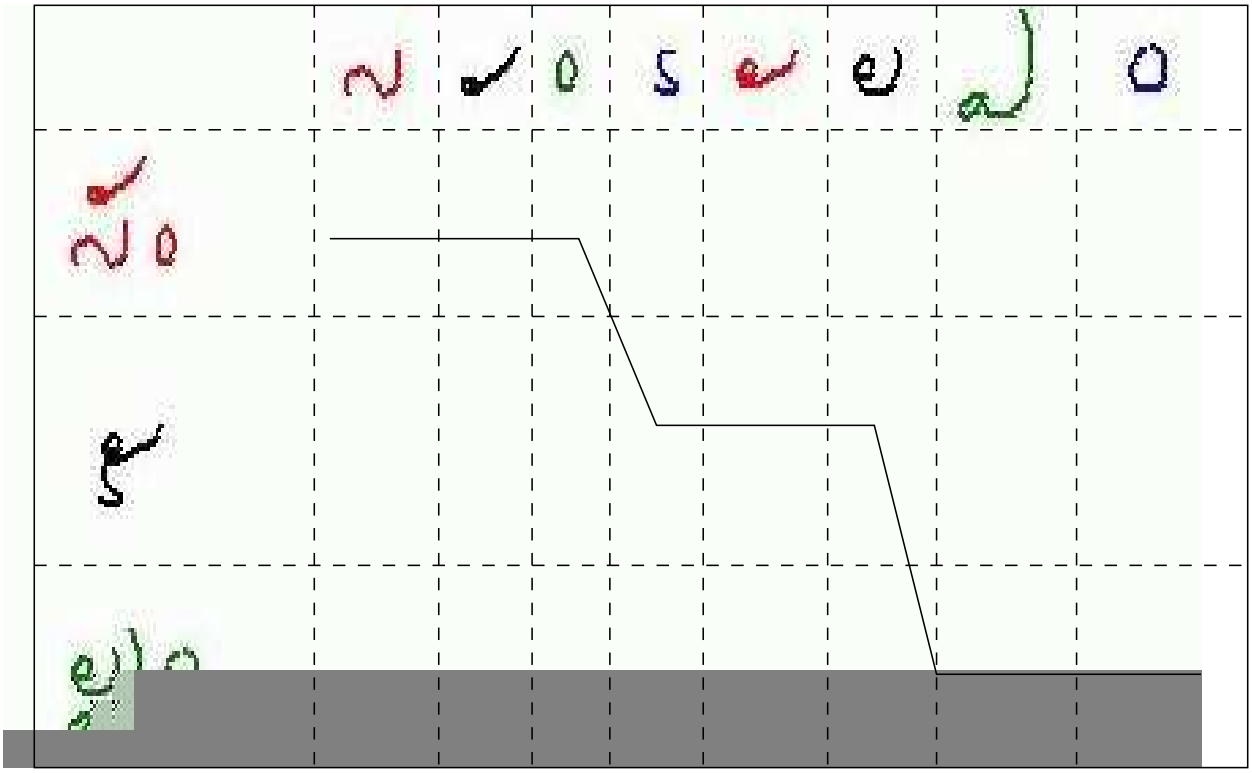


Figure 2.10: Figure (Matrix) Illustrating the Word-level DTW.

2.9 Experimental Results and Discussions

2.9.1 Performance of the Annotation Tool

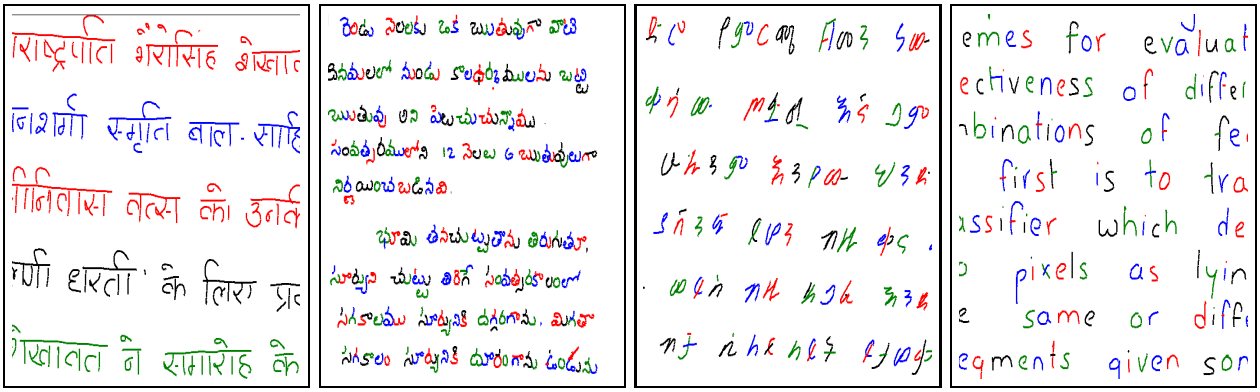
Performance of the tool is done using average number of key strokes made for annotating various documents at various hierarchical levels. Documents containing 200 characters on average are taken for the analysis. Average number of words in the documents taken is 50. Character level annotations took around 450 key strokes for annotation and word level annotation took around 250 key strokes on average. The time taken to annotate a page of 50 words at word level, was around 2 minutes for a person with average typing speed of 40-50 words per minute.

2.9.2 Access of the data

The data collection phase collects a large amount of information from an ink document, all of which may not be relevant to all the problems. One would like to have an access to the data for developing applications. The core function in the access library is a method that retrieves the information based on element and attribute names. This function is provided with a set of specific and generic wrapper functions for the convenience of the developers. Also facility is provided for the user to develop his own wrappers specific to his applications if the provided set is insufficient.

2.9.3 Annotation of non-Roman Scripts

Representing non-roman languages is a challenging task when building a standard. They are structurally very complex, and the number of strokes involved in making a character is very huge.



(a) (b) (c) (d)

Figure 2.11: Sections of Sample Documents (a) Devanagari(Indian script) Text Viewed at Line Level (b) Telugu(Indian Script) Text Viewed at Stroke Level (c) Amharic(Ethiopian Script) Text Viewed at Stroke level (d) English Text Viewed at Stroke Level

For example, in Devanagari script, *sirorekha*(a bar above the strokes) forms a part of the word, but independently does not have any meaning or existence. Also, it exists only at the word level in usual text. For many non-Roman scripts, the 'text' or 'codes' for annotation are either not properly defined or they are not as standard as the ASCII representation. This arises the need for support for various other annotation 'codes' and their display. The proposed schema could incorporate multiple encoding schemes. We have experimented with ASCII, Unicode and ISCII encodings. Some of the scripts tested for annotation are shown in Figure 2.11.

2.9.4 Results on Model-based Annotation

The experimental data for the annotation experiments were collected using a CrossPad and a TabletPC. The data collected included 25 pages of handwritten data in Telugu, Hindi, Tamil, Malayalam and Bengali scripts. The data was transcribed using ITRANS encoding to form a parallel text corpus. User models for synthesis were learned from part of the input data and the remaining were chosen for experiments on synthesis.

The text input is first converted into handwritten data using the synthesis module with the model of the user under consideration. The synthesized handwriting will contain the annotation information at the character level. The word is then aligned with the original handwritten input and the annotations are propagated from the synthesized word to the original input word. In the process, word boundaries are also identified (see Figure 2.12).

In addition to the word boundaries, the annotation is propagated at the character level from the synthesized to to the original data. Figure 2.13 shows examples of a Telugu words that were correctly annotated using our method.

The algorithm also allows for correction of errors that occur during the writing or transcription process. For example if the transcription was corrupted by a spurious character that changes the word form in the synthesized handwriting, the matching process often assigns no match to the

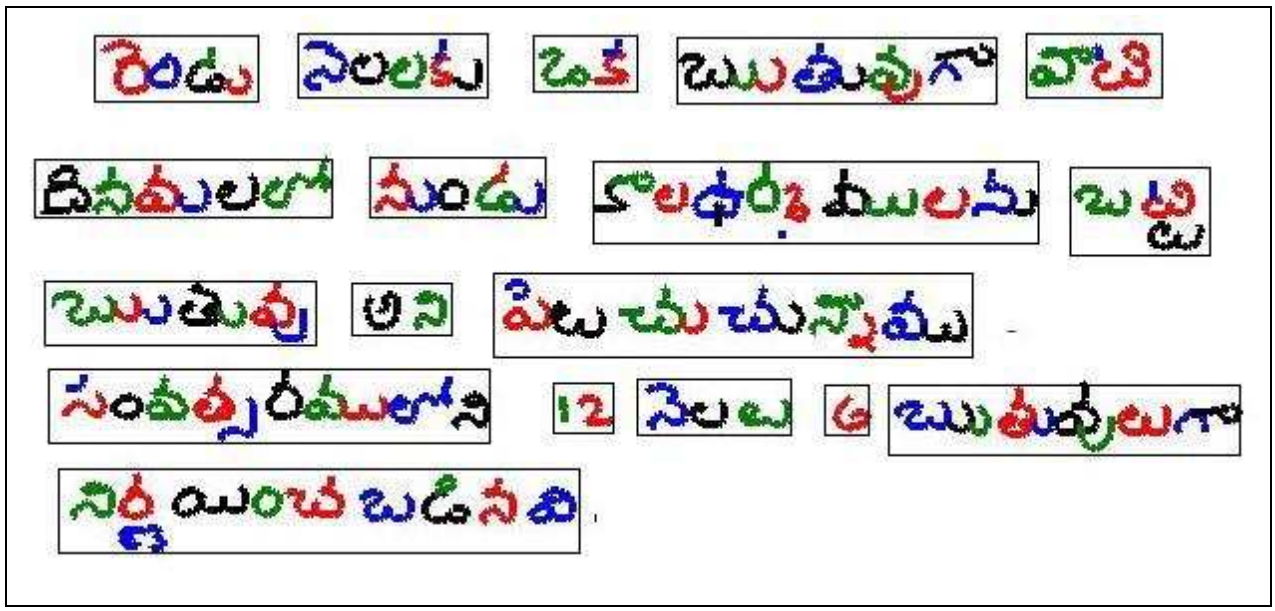


Figure 2.12: Word Boundaries Identified During the Matching Process.

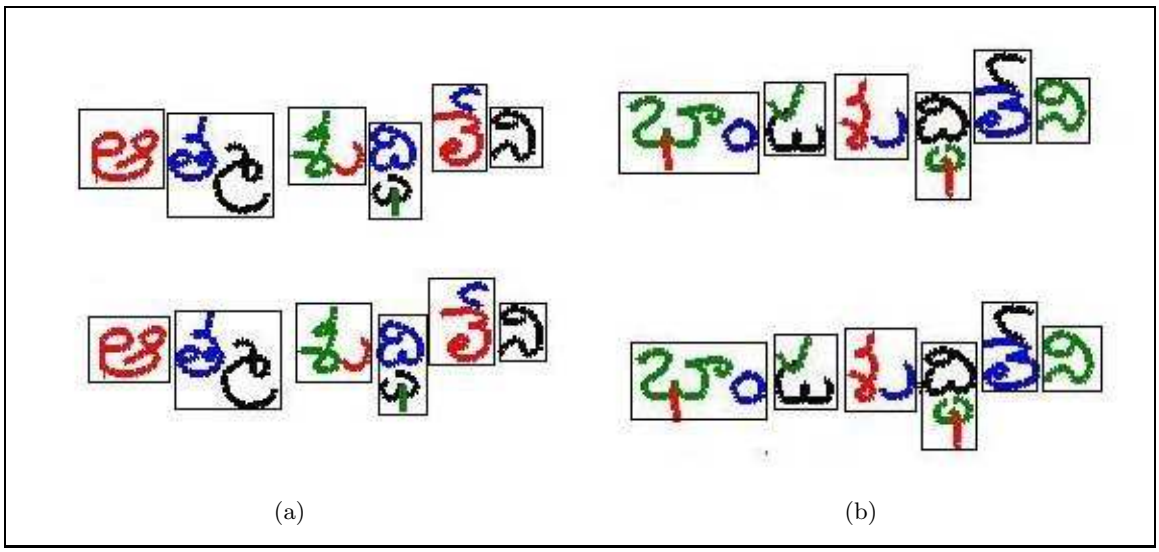
spurious character, effectively removing the transcription error from getting into the annotation. Figure 2.14 shows an example of an incorrectly transcribed word that was corrected during the matching process.

However, the matching process can also make errors in the case of similar looking strokes that are to be matched. Figure 2.15 shows an example of a word that is incorrectly matched.

To compute a quantitative measure of accuracy of the proposed annotation scheme, we manually annotated our training corpus using the annotation toolkit [3]. The propagated annotations from the synthesis module was then compared with the manual annotation to find out the error rate in transcription. The error rate at the word level was 26.5%, tested over 425 words. However, the number is slightly misleading due to fact that word-level errors often arise due to an error in a single stroke being mislabeled in the data. The character-level annotation gives a better picture of the accuracy. The error-rate of character level annotation was 3.6% when tested on a set of over 3500 characters. In other words, most of the word errors were essentially single character errors.

2.10 Discussions

In this chapter, we propose a model-based framework for annotation of non-cursive online handwritten data when a parallel text corpus is present for the data. The approach employs a handwriting synthesis scheme that generates the handwritten equivalent of the transcription. An elastic matching is used to propagate the annotation from the synthesized words to the original handwritten words. The annotation can be improved further by using the current annotation (partially correct) to refine the handwriting model of the writer under consideration, and then repeating the



(a) (b)

Figure 2.13: Annotation at Character Level Achieved through Matching.

annotation process. The algorithm achieves an annotation accuracy of 96.4% on a set of 3500 characters.

Currently we are extending this work to incorporate partially cursive scripts under the same framework. Work is also being done to extend the labeling to stroke level. This would enable one to handle various stroke orders within the handwriting of a person. Currently it is assumed that a person has a consistent stroke order.

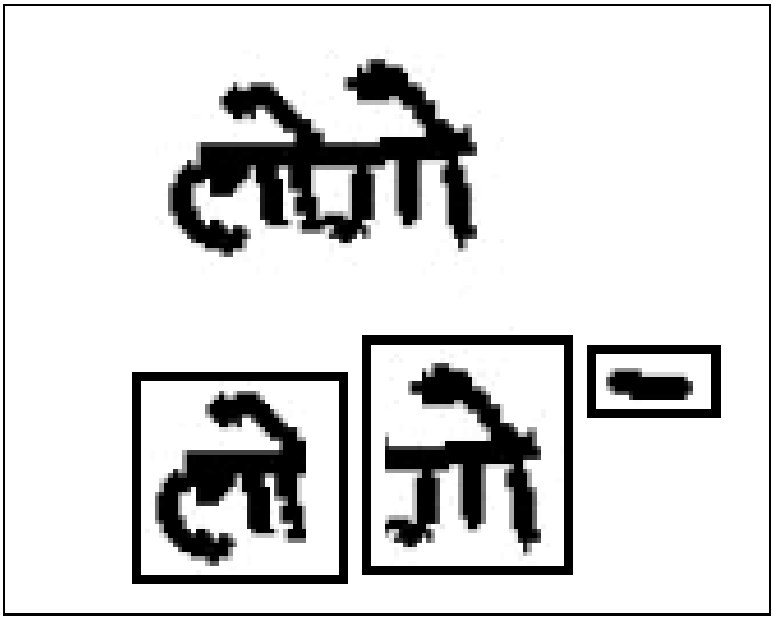
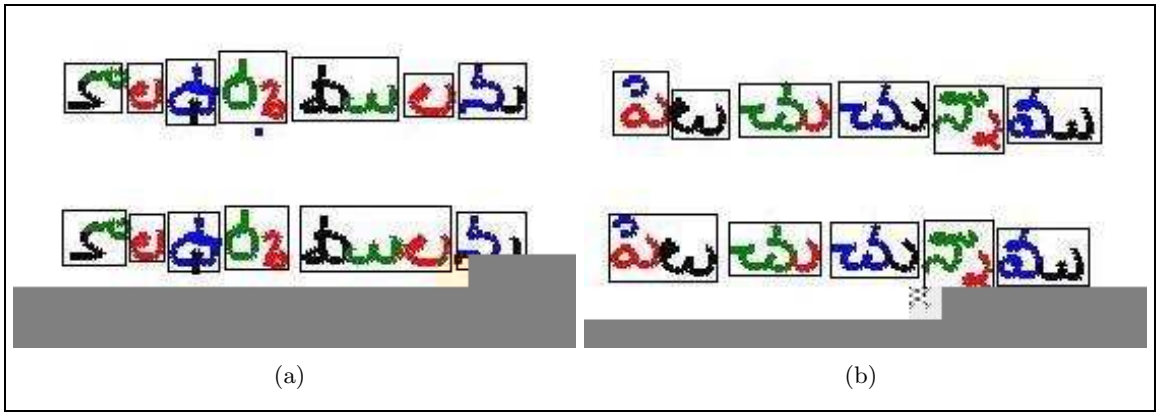


Figure 2.14: Correction of Transcription: The Synthesized Word in Hindi at the Bottom Contains an Extra Stroke, which is Discarded by the Matching.



(a) (b)

Figure 2.15: Errors in Matching: a) and b) Shows Examples of Two Words, where Strokes in the Synthesized Word (Top) were Not Matched With any in the Original Handwriting (Bottom).

Chapter 3

Retrieval of Online Handwriting by Synthesis and Matching

3.1 Introduction to Retrieval of Online Handwriting

Pen-based interfaces are gaining popularity due to the flexibility and popularity of pen as an interface as well as compactness of online handwriting that enables efficient storage and communication. Moreover, handwriting has more expressive power as compared to typed text due to the possibility of annotations and sketches, that makes it an effective medium of communication. The pen technologies have also matured over the last twenty years starting from touch screen based sensors with limited resolution to highly accurate and robust technologies such as electromagnetic and sonic sensors. Due to its capabilities, applications that treat handwriting or *ink* as the primary data type are also on the rise. However, as the amount of data available in the form of handwriting increases, access to specific documents becomes an issue due to the inability of current indexing and retrieval algorithms to efficiently handle such data.

The primary solution to handling online handwritten data is to employ a *HWR* (handwriting recognizer) to convert the ink into text, and use the results to search and retrieve the documents. However, this approach is suited only where the handwritten data is purely text and where a robust HWR is available for the language contained in the document. An alternate solution is to do matching and retrieval in the ink domain itself. The problem is very challenging due to the large amount of variation that is present in online handwriting. The sources of these variations include: i) differences in writing styles of various users, and inconsistency in writings of a single user, ii) differences in writing surfaces and capabilities of different digitizers, iii) noise introduced by the digitizers and representation, etc. In addition to the above, online handwriting also contain variations due to differences in writing speeds as the temporal information is also captured in the digitization process. Figure 3.1 shows search for relevant documents from a collection of handwritten documents. The input could be a handwriting using a pen/stylus, a sample word from a document, or even text typed from a keyboard. Users expect the most relevant documents to be retrieved from a database and presented to them in a meaningful way.

The level of complexity of the search increases as the diversity and size of the document collection increases. However, the range of applications also varies accordingly.

1. **Single Writer Collections:** These are typical in scenarios like the archived notes taken by the user of a pen-based device. The matching and retrieval of online documents are not easy even with such a collection of homogeneous writings [31]. Such collections usually contain a single script or language.

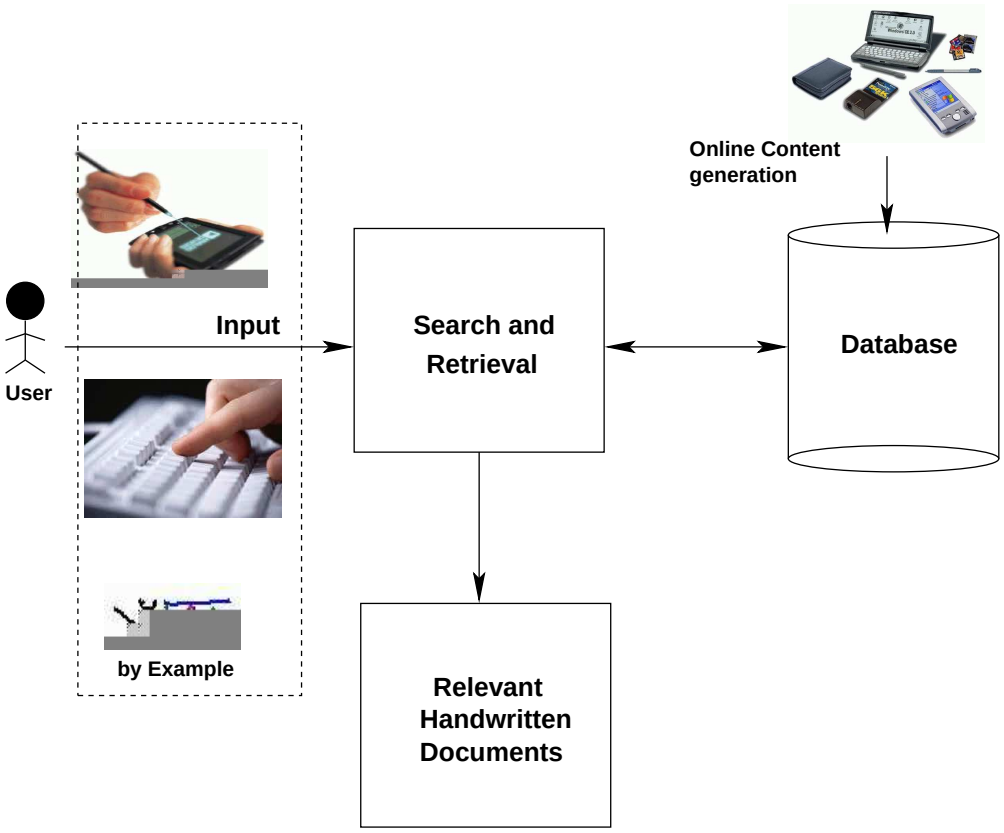


Figure 3.1: An Effective Online Handwritten Database Should Accept Queries based on Keyboard, Pen or a Sample Handwritten Word.

2. **Multi-Writer Single Script:** As the number of writers increase, the variability of handwriting also increases dramatically. Such document collections come about in applications that communicate handwritten documents across users, such as a mailing application.

3. **Multi-Writer Multi-Script:** As digital communications span across continents, the documents transmitted are likely to contain a large variety of languages and scripts. Dealing with such documents would be essential for email processing applications such as spam filters or searches in the archives of an organization. The search in such large collections have to be efficient in addition to being able to handle the different languages used in the documents. The problem is commonplace in a country like India, where there are 18 official languages, most having their own scripts.

4. **Digital Libraries:** Applications such as digital libraries add one more level of complexity to the problem due to an increase in the order of magnitude of the database as well as the varieties in devices that are used for digitization of handwriting.

3.2 Overview of the Previous Work

Approaches to searching handwritten documents can be divided into recognition-based and recognition-free approaches. Depending on the application and digitization process, the data could be either word images (offline) or traces of pen motion (online). Recognition of online data has the advantage

Work	Data	Approach	Pros	Cons	Applications
Rath *et al* [29]	Offline, Historic	Word Image Matching	Accuracy	Single Writer	Single writer document collections
Srihari *et al* [32]	Offline	Writer Matching	Multi-User	Lower accuracy	Forensic document retrieval
Russell *et al* [30]	Online	Recognition results	Multi-User, Fast	Needs Recognizer	Indexing and retrieval for multi-user and single script collections
Kamel [66]	Online	KLT-based, RTree	Multi-User, Fast	Limited Datasize, Lower accuracy	Online document retrieval
Jain and Anoop[31]	Online	Ink Matching	Accurate	Single User	Search in single-writer document collections
Balasubramanian *et al* [33]	Offline, printed	Word Matching	Accurate, Robust	Slow to Index	Search in large printed document collections
Current Work	Online	Synthesis and Matching	Multi-user, Accurate	Slow to Index	Search, Index multi-user document collections

Table 3.1: Overview of Existing and Current Work in the Area of Document Retrieval.

of using the additional temporal information present in the data. Recognition of handwritten data has received a lot of research attention in the past. Initial attempts in this direction were to recognize images of handwritten characters (offline handwriting recognition) and is useful in applications such as postal address recognition [67], handwritten form recognition [68], etc. However, in the case of HCI for pen-based devices, one can utilize the additional writing order, direction and velocity information to aid the recognition process. This is referred to as online handwriting recognition. The strokes in the word could be modeled using statistical models such as HMMs [69]. Such models are often tuned for a particular writer's handwriting. On the other hand, writer independent handwriting recognition is an extremely difficult problem due to the amount of variation between writing styles of different writers. Such variations require the recognition engines to be trained to a particular user's handwriting style. Adaptation of a recognizer to a specific writers handwriting [70] has been the most promising solution in this regard.

A second approach is to represent the handwritten words using a set of features and match two words by comparing the corresponding feature vectors. This approach is referred to as *word spotting*, and is quite effective when compared to recognition-based search for poor quality documents. Rows 1 and 5 of Table 3.1 show examples of the word spotting technique applied to offline and online handwritten documents.

The approach proposed here consists of three major steps: i) Synthesis of handwritten data from the query, ii) Matching of the query to words in the database, and iii) Computation of relevance

scores of documents to order the results of matching. The synthesis is primarily aimed at Indian scripts, which are non-cursive in nature. The synthesized handwritten word is then matched with all the words in the database and those with scores below a threshold qualifies as matches. The matching documents are then ranked according the the frequency of the search term in the document (TF) and the inverse of the number of documents containing the word (IDF), and presented to the user.

The following are the characteristic differences of the current work, when compared to other approaches to retrieval of online handwriting data.

1. **Use of Synthesis:** Our approach employs a handwriting synthesis module to map the query text into the ink domain. Matching is done at the ink level rather than on the recognition output. This enables the use of a keyboard, pen or example word for query input.

2. **Use of IR Measures:** The proposed approach extends the concepts of TF/IDF and word-form variations into the ink domain and integrates it with the search and retrieval module. This enables us to have better page rank computations. This part is an extension of our work that uses a similar approach for printed documents [33].

3. **Cross-Lingual Retrieval:** The synthesis module, when combined with the transliteration properties of Indian languages enables us to do cross-lingual retrievals for a single query word.

4. **Indian Language Search:** The state of both OCR and handwriting recognition for Indian languages are still in its infancy and there are no commercial OCR or HWR programs available even on platforms like the Tablet PCs. A recognition-free approach is extremely useful in this context. To the best of our knowledge, this is the first attempt at the problem of document retrieval from Indian language online data.

The bottleneck of most recognition-free approaches is the fact that one is trying to match the content of the query word with that of the words in a document. The matching algorithm proposed by Jain and Namboodiri [31] achieves excellent results for word matching. However, the performance of the algorithm drops drastically when one tries to compare the words written by two different writers. In general, ink-matching based algorithms work well only for single user document collections. Moreover, the user interface for the method described in [31] is restricted to pen-based input and hence is not convenient in all settings. We solve both of the problems by generating a handwritten sample of the query word in the writing style used by the document. In other words, matching is done using a synthesized sample of the query in the style of the writer under consideration, which is generated from typed text.

Figure 3.2 describes the entire process of synthesis and retrieval of the handwritten documents. The input query is entered either in ITRANS or in Unicode. The query word is then synthesized and the corresponding handwritten word is rendered by the Word Rendering module. Features are then extracted from the rendered word and is matched against all the feature values of the handwritten words that are stored in the database. Then the handwritten documents are appropriately fetched based on their relevance scores.

3.3 Handwriting Synthesis

Given an input text, the problem of handwriting synthesis is to generate data that is close to how a human would write the text. The characteristics of the generated data could be that of a specific writer or that from a generic model. Even with a given model, the synthesis method should not be

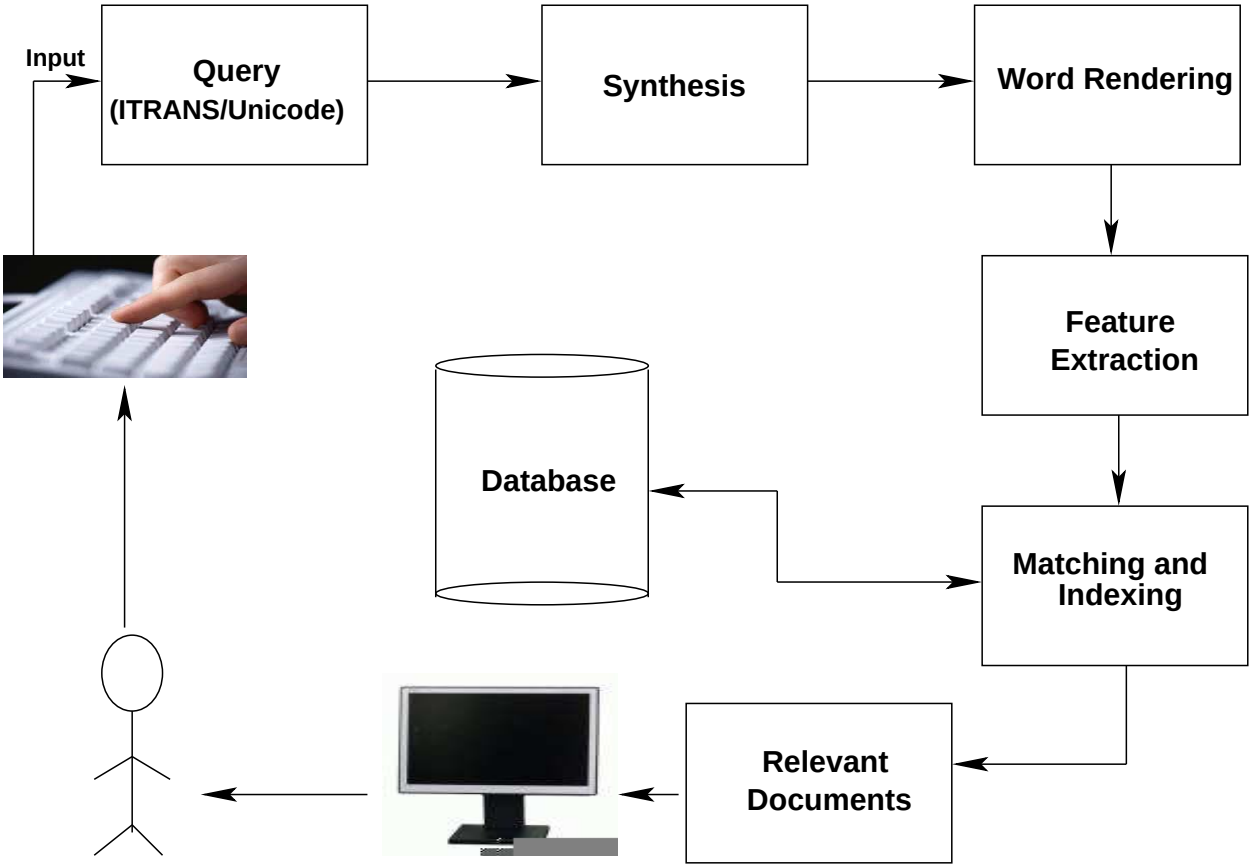


Figure 3.2: Block Diagram Explaining the Entire Process of Synthesis and Retrieval

deterministic since the variations that are found in human handwriting are inherently stochastic. However, if we need to generate data that is similar to a particular writer's handwriting, we need to identify, model, and preserve the basic characteristics of his/her handwriting. The problem of maintaining the writing style while introducing variability [71] makes the problem of synthesis very difficult. A handwriting synthesis solution has a variety of applications including automatic creation of personalized documents [21], generation of large quantities of annotated handwritten data for training recognizers [72][73], and writer-independent matching and retrieval of handwritten documents.

Traditionally, handwriting synthesis has been dealt within the realm of offline handwriting [25], where the handwritten data is a scanned image of a paper document. Online handwriting is stored as a sequence of *strokes*, where each stroke is defined as the trace of the pen tip from a pen-down to the next pen-up. Devices with pen-based interfaces facilitate storing of the handwritten ink in the digital format and thus enabling a variety of applications such as search and retrieval of large sets of handwritten notes [74] as well as efficient communication across the Internet. In the context of digital ink, the technique of handwriting synthesis is extremely useful as it leads to applications that preserve the compactness of online data, while being natural.

3.3.1 Characteristics of Indian Scripts

The problems associated with handwriting synthesis are different depending upon the nature of the script that one is trying to synthesize. Languages that use the Roman script contain a small

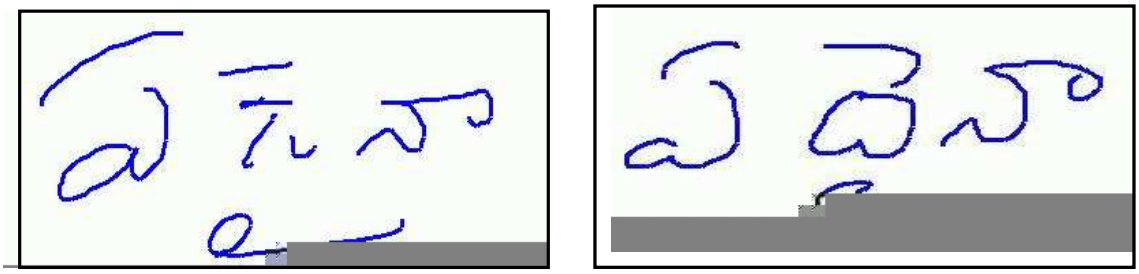
set of symbols that are arranged in a linear fashion to create a word. The complexity of these scripts arises due to the cursive nature of the script, where the individual characters are connected together. In fact, real-world handwriting is a mixture of cursive and non-cursive parts, which makes the problems of recognition and synthesis, more difficult.

Indian language scripts are fundamentally non-cursive in nature, where the *aksharas* (equivalent to characters) are written independently, separated by space or pen-lifts. An *akshara* can be combination of one or more (upto 3) consonants with a vowel. However, these scripts often contain a large number of characters that have complex spatial layout of strokes. Indian scripts have compound characters, which are combinations of multiple consonants and vowels. The handwriting synthesis process should hence model all the possible variations of characters and their combinations to be able to generate any given text. This makes the problem of synthesis, extremely complex in the case of Indian language scripts.

There are many other properties of the Indian scripts that are not seen in Roman. This arises due to a variety of factors:

- Alphabets of Indian scripts have far more complex shapes and varied writing styles.
- The size of the alphabets is typically high. In addition, the presence of *samyuktaksharas* (compound characters) makes modeling of Indian scripts more difficult.
- The basic stroke shapes in Roman scripts are often unambiguous in their meaning. However, this is not the case with the Indian scripts, where a single stroke shape can acquire different meanings depending on its position and size.
- Indian scripts are non-cursive in nature and thus the available models need not be the most suitable.
- The spatial location of an akshara is dependent on the previous akshara in some Indian language scripts.

Along with the spatial complexity, the variance in writing styles also increases for Indian scripts. For example, Figure 3.3 shows the handwritten Telugu word EdainA (means *anything*) written by two different writers. As it is evident from Figure 3.3, the second writer's handwriting (Figure 3.3 (b)) is readable and clean when compared to that of the first writer (Figure 3.3 (a)), whose handwriting sample is composed of several pen lifting movements, stroke discontinuities, slant and other deformities. Also note the fact that the stroke order is not unique for every writer for a given word and sometimes even the same writer has different stroke order when writing the same word.



(a) (b)

Figure 3.3: Telugu Word EdainA Written by (a) Writer 1, and (b) Writer 2.

As pointed out before, the scripts of Indian languages are more complex than Roman script. Following is a list of interesting characteristics of Indian scripts that are relevant to the synthesis problem.

- Words in many of the Indian language scripts have *Shirorekha* (horizontal bar on the top). Figure 3.4 (a) shows how characters of *Bangla* script are joined on the top with the *Shirorekha* to form a word. After writing the individual characters, *Shirorekha* is drawn to connect them and describe the word boundaries. This need not be considered as a sign of cursiveness. Some of the Indian languages that have *Shirorekha* are Hindi, Bangla, Marathi, and Gurumukhi (Punjabi).

- Figure 3.4 (b) shows the cast of the vowel modifiers in *Telugu* and *Hindi*. The consonant *ka* when combined with the vowel *e* gives a compound akshara *ki*. Vowels often get converted as augmented shape modifiers to consonants in most Indian scripts. Vowels could also appear in isolation.

- Figure 3.4 (c) is an example from the *Devanagari* script which shows how the word *raastr* is formed from the basic consonants and vowels. Here the variant is further more complex with the last akshara *str* being formed by the following three consonants *sh* , *tt* and *ra*. This is called the *Samyuktakshara* which could contain multiple consonants and a vowel.

- Figure 3.4 (d) shows an example from the *Tamil* script where we have a vowel *u* and a consonant *lla*. The vowel sound *uu* shown at the right side is a concatenation of the symbols of *u* and *lla*.

- In Figure 3.4 (e) we have the consonant *ka*, which when combined with the vowel *uu* results in *kuu* and when combined with the vowel *e* results in *ke*. One has to note that both the vowel modifiers for *uu* and *e* have similar looking strokes, however their positions are different.

One may observe that for Indian scripts, spatial positioning of the strokes is equally important as their shapes.

3.3.2 Synthesis of Non-Roman Scripts

Broadly there are two basic approaches in synthesizing the handwriting. Earlier attempts employ the motor model based synthesis of handwriting [75]. Singer and Tishby [76] model the handwriting process as modulation of oscillatory motions of the pen. The modulation parameters decide the shape and variations of the generated data. The Delta LogNormal Model by Guerfali and Plamondon [75] is based on movement simulation techniques, and may be defined as a curvilinear stroke generator made up of two parallel neuromuscular networks, which control the agonist and the antagonist [77] activities associated with a specific movement. The second approach is based on the mathematical model of an algebraic curve whose shape is controlled by parameters, typically direction and time [21][75][24]. There have been other attempts like the vector-matrix of successive strokes by Kondo [78], and use allograph codes as input by Schomaker *et al.* [79].

Roman script was the primary focus in the motor model based synthesis techniques [75][77]. On the other hand there has been no concrete model available for the oriental languages. Some oriental characters need many pen-tip lifting steps due to the presence of large number of short segments. Scripts such as the Korean has been studied before for the purpose of synthesis [80, 81]. The Beta-Velocity model was proposed for Kanji scripts by Lee and Cho [80] to simulate cursiveness with a letter or a word. This model was an improvised version of the Delta LogNormal model, the

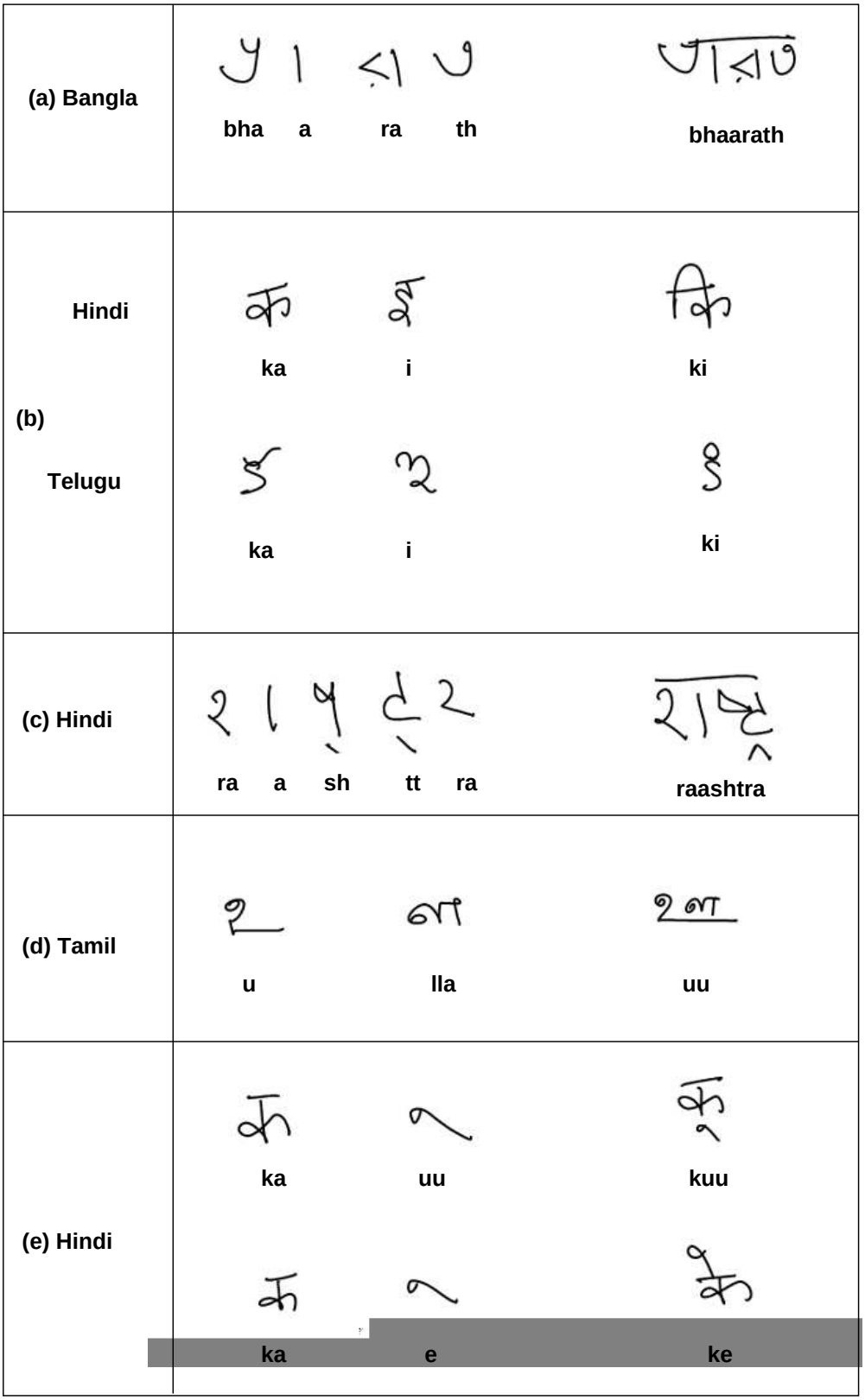


Figure 3.4: Some of the Special Cases in Indian Language Scripts.

main difference being that the Beta-Velocity model uses asymmetric curves, whose skewness can be controlled by variables.

The work presented in this chapter is primarily aimed at Indian language scripts, which are different from both western and other oriental scripts. As per our knowledge, this is the first such attempt on the synthesis of Indian language scripts. We develop a stroke shape and layout model for Indian language scripts that can be learnt from labeled samples. Hence we can use the model for generation of a specific writer's handwriting or develop a generic writer model. The proposed model captures both the shape and temporal aspects of the strokes as well as their order information. Hence we can generate either online or offline data using the learnt parameters.

3.4 Modeling Handwritten Data

Figure 3.4 gives the outline of the learning and synthesis steps of our method. The handwriting model consists of two parts, a **stroke model**, which captures the shapes and variations in the basic strokes that form the characters, and a **layout model**, which controls the spatial layout of the individual strokes. The individual models can be learnt from online handwritten data that is collected from a writer or multiple writers. The training data needs to be annotated manually. Examples of strokes corresponding to each stroke class is used to learn the stroke model. The spatial distribution of strokes are learnt by the layout model.

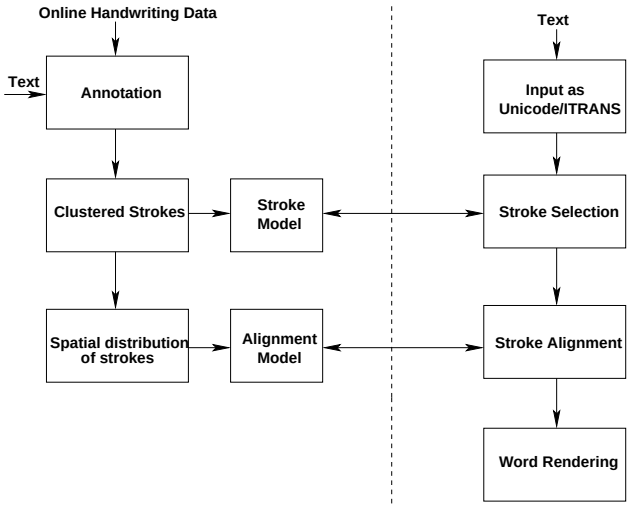


Figure 3.5: Block diagram for Handwriting Synthesis. The Modules to the Left of the Dotted Line Forms the Training Phase, while those to the Right Form the Synthesis.

3.4.1 Stroke Model

Each script contains a set of basic strokes that are used to form all the characters in the script. For example, Hindi consists of around 200 basic strokes, while Telugu consists of more than 300 basic strokes. The stroke model consists of a representation for each of these basic strokes. We have used two different models for representation of the strokes:

1. **Normalized Template Model:** In this model, we represent each basic stroke class using a set of training strokes, that are normalized in size. The stroke selector will randomly select

one of the samples for the required class and scale it to the appropriate size determined by the position of the stroke in the character. This model is appealing in many applications due to its simplicity and is useful in cases where the training data is limited. It can also provide high quality synthesis in presence of limited training samples. However the range of variations that are present in the results could be limited with smaller training samples. The model M, could be represented by a set of k stroke models, each containing a set of stroke samples:

$$M = \{S_1, S_2, .., S_k\}$$
$$S_i = \{s_1, s_2, .., s_{n_i}\}$$
$$s_i = [(x_1, y_1)(x_2, y_2)..(x_m, y_m)],$$

where S_i represents the model of the i^{th} stroke class containing n_i stroke samples, and each s_i is a sample that consists of a sequence of (x, y) coordinate pairs.

2. **Mean Trace Model:** A more complex model that is capable of generating a large set of stroke samples is the mean trace model. Here we compute the mean of all the traces (or strokes) of each of the given stroke classes during the training phase. To compute the mean trace, the strokes are normalized and the points are aligned using an elastic matching technique. The distribution of the samples of the strokes are estimated using the means and covariance matrices of the aligned sample points and are stored in the order in the trace. The stroke selector module will generate random samples from each of the distributions to create a new stroke of a given class, assuming a Gaussian distribution. The alignment and estimation steps make sure that outliers are not included in the training phase, so that the learnt model is close to the supplied handwriting.

$$M = [\bar{X}_1, \bar{X}_2, .., \bar{X}_k]^T$$
$$\bar{X}_{ip} = \frac{1}{n_i} \sum_{j=1}^{n_i} (x_{pj}, y_{pj}),$$

where $\bar{X}_i$ represents the mean of the p^{th} point of the i^{th} stroke class containing n_i stroke samples, and each sample point of the strokes model is the mean of the corresponding points ((x, y) coordinate pairs) after alignment and outlier removal. The mean model estimate is hence the maximum likelihood estimate of the sample sequence, assuming each sample comes from a multivariate Gaussian distribution.

One could also employ generative models such as HMMs to learn the stroke structure and generate samples of a person's handwriting. However, we noticed that such models tend to allow a lot of variation in the writing style, and the individuality information is lost in the training phase. Hence they are not best suited for synthesis applications in their basic form.

Note that the stroke models that we employ are relatively simple compared to the motor model based approaches employed for Roman scripts. However, since the strokes themselves are only parts of characters in the Indian language scripts, their spatial distribution, captured by the layout model, can generate realistic renderings of handwriting.

3.4.2 Layout Model

The most important part of our synthesis model is the layout model that is capable of capturing the spatial relationship between stroke classes. As noted before the spatial layout between the strokes in an akshara can be very complex. A straight forward approach is to learn the spatial distribution of strokes within each akshara. However, the number of possible aksharas in Indian languages are very large when considering the samyuktaksharas or compound characters. The total number of aksharas can run into many thousands. Moreover learning such a model does not exploit the redundancy in information present between similar characters. For example, all the modifications of a multi-stroke character will have the basic set of strokes repeated in some form. To exploit this redundancy, we model the spatial layout as a set of pairwise spatial distributions between stroke classes.

Let ω_i and ω_j be two stroke classes that are modeled using the stroke model defined in Section 3.4.1. We describe the layout model as the spatial relationship between the two strokes ω_i and ω_j in terms of distance, r, and direction, θ. The distance and direction could be measured using the centroids of the strokes or based on the bounding boxes. In this experiment we use the top left corner of the bounding boxes to compute the relative distance and direction.

Let $p(r, \theta|\omega_i\omega_j)$ represent the spatial distribution of the class ω_j with respect to the class ω_i, where r is the radial distance and the θ is the angle between the two classes. We represent the spatial layout $D_{\mathcal{L}}$ of a language $\mathcal{L}$, as a set of such pairwise spatial distributions of the strokes:

$$D_{\mathcal{L}} = \{p(r, \theta|\omega_i\omega_j)|\omega_i \text{ and } \omega_j \text{ are neighbors in } \mathcal{L}\}$$

The total number of possible stroke pairs in Indian languages can be very high since the number of stroke classes range from 200 to 350. However, not all stroke pairs will appear in proximity to each other in any character in the language. Hence the typical number of parameters that we need to estimate is around 1000. Here we assume that the parameters r, θ form a distribution for each stroke class pair, which can be modeled as a multivariate Gaussian distribution. The mean and covariance matrices are estimated using MLE, similar to that in the stroke model.

Each character in the script could compose of multiple strokes and its number can differ based on the writing style. Hence, in addition to the spatial layout, we also need to learn the set of stroke classes that are used to write a particular character by specific a writer. This information is identified and stored for each character class during the training phase. In Figure 3.4, the estimation of the model is depicted to the left of the dotted line. Once the model parameters are estimated from the annotated samples, we can synthesize any given text based on the synthesis procedure (shown to the right of the dotted line in Figure 3.4).

3.5 Synthesis from Spatial Layout

Let $\omega_1, \omega_2, ..., \omega_k$ be the k primitive classes that compose the entire script. These primitive classes consist of strokes extracted from the stroke model. Let x_i be a particular stroke in the training samples of ω_j. Conventional algorithms model $p(x_i|\omega_j)$ (denoted as $p_{\omega_j}(x_i)$) based on a set of parameters that control the shape of x_i [24], and the primitives used are characters. In our model, $p_{\omega_j}(x_i)$ is controlled by the stroke model.

The word is synthesized using the individual primitive classes, and the distribution of samples within the classes. The synthesis proceeds as follows:

- Given a text word, create a sequence of stroke classes that constitute the handwriting equivalent of the input word. This information was learnt during the training of the layout model.

- For each stroke class, we select/generate a sample stroke using the stroke model for the writer under consideration.
- The layout model is used to arrange the sample strokes to generate the final word.
- The word could be rendered in appropriate form, depending on the application or could be passed to the next phase in applications such as retrieval and training of recognizers and also can be used for other applications.

Stroke Selection: The stroke selection module selects a sample stroke from the set of training samples in the case of normalized template model. For the mean trace model, we generate a sample stroke based on the learnt distribution of the sample points within the stroke. One can introduce variations in the synthesis by generating random samples from each sample distribution. Alternately, the mean of the sample points would give the most likely stroke sample for each stroke class.

Stroke Alignment: Given two consecutive primitives, We compute the most likely direction between the two classes using the layout model. To introduce writing variations, one could generate random sample from the spatial relation distribution that was estimated in the training phase.

The probability of the synthesized word with m stroke classes can be computed according to the stroke and layout models as follows:

$$P(word) = p(x_1|\omega_1) \prod_{i=2}^{m} p(r_i, \theta_i|\omega_{i-1}\omega_i) p(x_i|\omega_i),$$

where r_i and θ_i are the distance and direction, generated by the layout model for the i^{th} stroke sample.

3.6 Matching Handwritten Words

Once a handwritten word is synthesized from the query word, we can use it to search the database of handwritten documents using elastic matching. Since every online handwritten word is a collection of strokes, we need to define a matching technique to compare two strokes as well as to compare two words. Distance or dissimilarity between two strokes is computed using a set of features extracted from the group of strokes. Each stroke consists of a sequence of sample points, (x, y) that describes the trace of the pen during writing. The strokes are first converted into a sequence of feature vectors, extracted from each of the sample points. The feature vector consists of:

1. The direction, θ, of the tangent to the stroke curve
2. The curvature, c, of the stroke at the sample point, and
3. The height, h, of the sample point from the word baseline

Figure 3.6 illustrates the computation of the three features. The angle, curvature and height of a point on the stroke completely characterizes the local neighborhood. Moreover, these features have been demonstrated to be effective for online word spotting for single user datasets [31].

The distance between two feature vectors $F_1 = \langle \theta_1, c_1, h_1 \rangle$ and $F_2 = \langle \theta_2, c_2, h_2 \rangle$ is defined as the weighted Euclidean distance between the two vectors:

$$D^2 = k_\theta * (\theta_1 - \theta_2)^2 + k_c * (c_1 - c_2)^2 + k_h * (h_1 - h_2)^2,$$

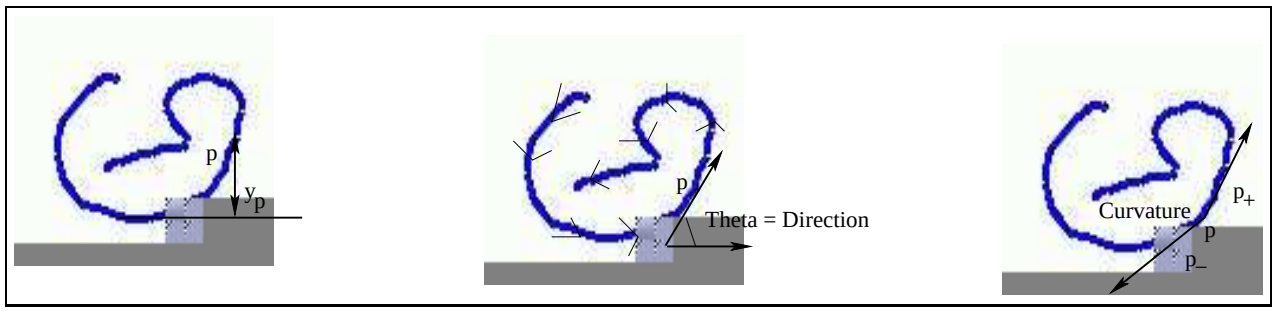


Figure 3.6: Features Computed for Stroke Matching: Direction, Height and Curvature.

where ks are the weighting factors. A sequence alignment score is then computed using a Dynamic Time Warping (DTW) procedure. The use of the total cost of Dynamic Time Warping as a similarity measure is helpful to group together strokes that are related to their root character by partial match. Dynamic Time Warping is a dynamic programming based procedure to align two sequences of signals. This can also provide a similarity measure.

Let the strokes (say their curvatures) are represented as a sequence of vectors $F = \mathrm{F}_1,\mathrm{F}_2,\ldots,\mathrm{F}_M$ and $G = \mathrm{G}_1,\mathrm{G}_2,\ldots,\mathrm{G}_N$. The DTW-cost between these two sequences is $D\,(M,N)$, which is calculated using dynamic programming is given by:

$$D(i,j) = min \begin{cases} D(i-1,j-1) \\ D(i,j-1) \\ D(i-1,j) \end{cases} + \quad d(i,j)$$

where, $d\,(i,j)$ is the cost in matching the ith element of F with jth element of G and is computed using a weighted Euclidean distance:

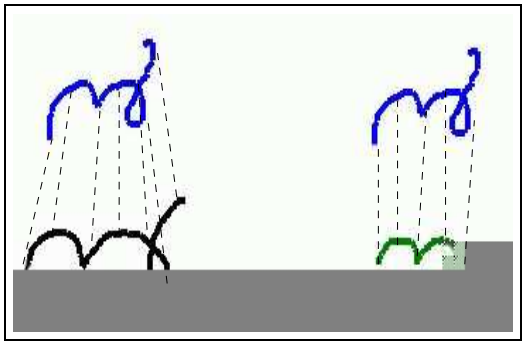


Figure 3.7: The Dynamic Time Warping Matching.

Using the given three values $D(i,j-1)$, $D(i-1,j)$ and $D(i-1,j-1)$ in the calculation of $D(i,j)$ realizes a local continuity constraint, which ensures no samples left out in time warping.In Figure 3.7, we also imposed global constraint using Sakoe - Chiba band [4] so as to ensure the maximum steepness or fatness of the DTW path. Score for matching the two sequences F and G is considered as $D\,(M,N)$, where M and N are the lengths of the two sequences.

The distance between two words are computed similar to the inter-stroke distance. However, in this case, the primitives becomes strokes (unlike feature vectors in stroke matching) and $D(i,j)$ is used as the distance measure between strokes. The DTW algorithm used at the word level allows us to handle spurious or missing strokes in a word. It also allows us to do partial matching of words

as will be seen in the next section.

3.7 Information Retrieval Measures for Document Retrieval

Most of the document retrieval algorithms mentioned in Section 2, focus on spotting of similar looking word and retrieving documents containing such instances. However, with widespread use of textual search engines with powerful information retrieval techniques, one naturally look for mimicking these ideas, even in the absence of an explicit recognition engine. We have developed a computationally efficient procedure for the indexing and retrieval of offline printed documents in [33]. Here, we extend the same idea for the online data.

In the case of ASCII encoded text documents, the retrieved documents are ranked according to a relevance score. The relevance scores are computed as follows: i) stop words in the query are removed to avoid spurious documents being retrieved, ii) morphological variations of the word are identified and discarded, and iii) relevance of the document to a query rank is computed based on factors such as *term frequency* (TF) and *inverse document frequency* (IDF). An important process in speeding up the retrieval of documents is that of indexing. During indexing, the unique words in a document are identified and a table is created with them, where each index term points to all the instances of that term in the document. Hence one can search only in the index table to locate any document containing the query. For efficient indexing, we first hierarchically cluster the words using a scheme such as Minimum Spanning Tree based clustering. The clusters form sets of words that are similar to each other. The statistical distribution of the words across clusters is used for detecting stop words and also for computing a pseudo-TF/IDF. Once similar words are clustered, we analyze the clusters for their relevance. A measure of the uniformity of the presence of similar words across the documents is computed. This acts as an inverse document frequency. If a word is common in all the documents, this word is less meaningful to characterize any of the documents. Given a query, the corresponding handwriting is synthesized and the cluster corresponding to this word is identified. In each cluster, documents with highest occurrence of similar words are ranked and listed. The process of computation of relevance of the word as well as the counting of word frequencies does help to index document images similar to TF/IDF(Term frequency inverse document frequency) for text indexing.

One of the main problems in document search using keywords is that of word-form variations. The exact word that is present in the document being searched is a variant of the keyword. There are two types of variations that are possible for a word: i) variation of the form of the word and ii) alternate word with the same meaning. For example, let the keyword being searched is *compute*. Word-form variants include words such as *computer*, *computing*, *computation*, etc., where as, alternate words could include *calculate* and *determine* in specific contexts. Addressing the second type of variation needs knowledge of the meaning of the word and its context and is beyond the scope of this work. We address the problem of dealing with the first class of variations, namely word-form variations. To match variants of a word, we note that word form varies mostly as changes in the prefix or suffix of a word, both in English and in Indian languages. Figure 3.8 shows an example of partial matching for the Hindi words having root word "gyan" (meaning "knowledge"). As can be seen, the word "Agyanta" (on the top of Figure 3.8) is matched with the word "gyan" (to the left of Figure 3.8). We thus modify our matching algorithm to minimize the penalty for not matching the initial and final parts of a word.

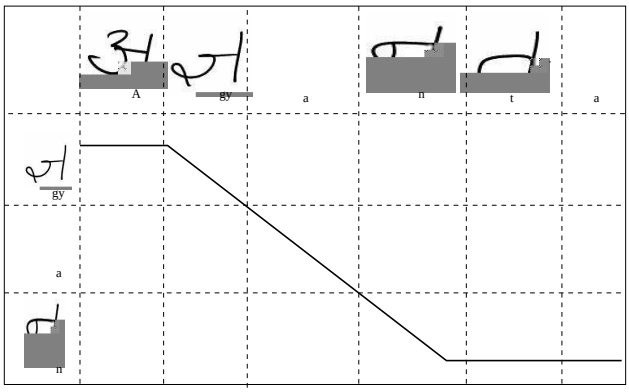


Figure 3.8: Figure Illustrating Partial Matching.

3.8 Experimental Results

The proposed search and retrieval scheme is tested on online handwriting data sets in Indian scripts, primarily in Telugu. The data for the experiments were collected using an IBM Crosspad and a Tablet PC. For convenience of description, we divide our dataset into two: *Dataset 1*, consists of 100 pages of data from 20 different writers in Telugu script. The writers were chosen from varied backgrounds based on attributes such as script familiarity, educational qualification, age, and gender. The data contains over $12,000$ words. *Dataset 2*, consists of 15 pages each in Bangla, Hindi, Malayalam, Tamil, and Telugu scripts, collected from a total of 5 writers. The Telugu pages are common in both sets. The scripts mentioned above were selected for the following reasons: i) Telugu, which has one of the most complex layouts among all of the Indian languages, ii) Hindi, which has shirorekha, iii) Malayalam which has long and complex strokes, and iv) Bangla, which is similar to Hindi and v) Tamil, which has the smallest set of alphabets in Indian languages. A larger Telugu dataset (Dataset 1) was collected due to its complexity in spatial layout of characters.

3.8.1 Modeling and Synthesis of Handwriting:

The first set of experiments were to evaluate the performance of the synthesis algorithm. The experimental framework consists of i) an annotation tool that can annotate the handwritten data at the stroke level, ii) a handwriting model and synthesis module and iii) an ITRANS-based encoding module for processing the text input. The data was annotated manually using the annotation toolkit [82] in ITRANS [42]. In this toolkit, handwritten data can be displayed simultaneously with the annotation, which is dynamically updated as the user types the annotation.

Figure 3.9 shows two original samples of the Telugu word EdainA from two different writers and the corresponding synthesized words. As can be seen, the synthesized words for two different writers (Figure 3.9 (b) and (d)) are very similar to their original forms (Figure 3.9 (a) and (c)). As noted from the example, our model is able to generate natural and realistic words that are very close to the original ones. Also shown in Figure 3.9 (e)-(j) the word "manushya" in their synthesized and original forms in Malayalam, Bengali and Tamil scripts respectively.

Figure 3.10 shows the synthesis results of the same set of words written by three different writers. Characteristics such as the inter-character spacing, relative size of loops and other matras (diacritical marks) of the three writers vary greatly. The mean trace model allows us to generate variations in handwriting of a specific writer, while maintaining the characteristics traits of the individual.

During synthesis the user input is typically in the form Unicode or ITRANS [42] encoded text,

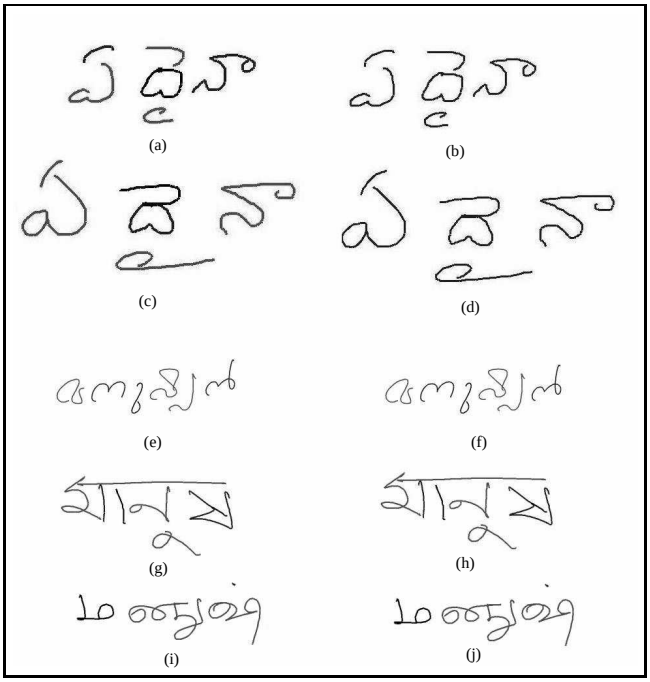


Figure 3.9: Some Synthesized Words in and their Original Forms Written by Various Writers in Telugu, Malayalam, Bengali and Tamil

which are the two popular encodings for Indian language scripts. The input encoder converts the ITRANS or Unicode text string into a sequence of stroke classes. The specific stroke classes that are generated depend on the character in the input as well as the writing style of the writer under consideration. The stroke selector module generates a sequence of strokes using the stroke model that are needed to generate the input text. Stroke alignment module is then used to compute the spatial positioning of the generated strokes based on the information from the alignment model and finally the word is rendered.

A quantitative evaluation of the synthesis algorithm is not feasible without a similarity measure between the generated word and the writing style of the writer under consideration. However, the writing style is a subjective quantity, that has not yet been quantified in the literature. Hence we use the effectiveness of the synthesis module in the retrieval experiments to gauge the correctness of our synthesis algorithm.

To run the retrieval experiments, we identify a set of query words that are present in the documents in the database, and the corresponding online handwriting samples are synthesized. The features from the synthesized word are extracted as described before and compared against the words in the document. Those words with a matching (distance) score that is lower than a prespecified threshold were assumed to match the query. Retrieved documents are ranked based on the relevance to the query using TF/IDF scores.

Figure 3.11 shows the retrieved documents with the matching words placed inside the bounding box for the query in Figure 3.12. We note that all the documents that contained the word were retrieved and the top three matches were all relevant documents. Figure 3.11(a) has three matches where the first two matches are for the word *"roojulaku"* a variant of *"roojulu"*, while the last match is a direct match with the word *"roojulu"*. The match with the variant occurs due to the first sequence of strokes in all the three matches are similar and constitute the root word *"rooju"*

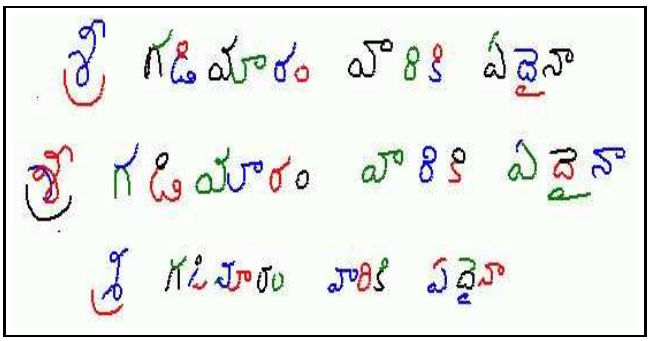
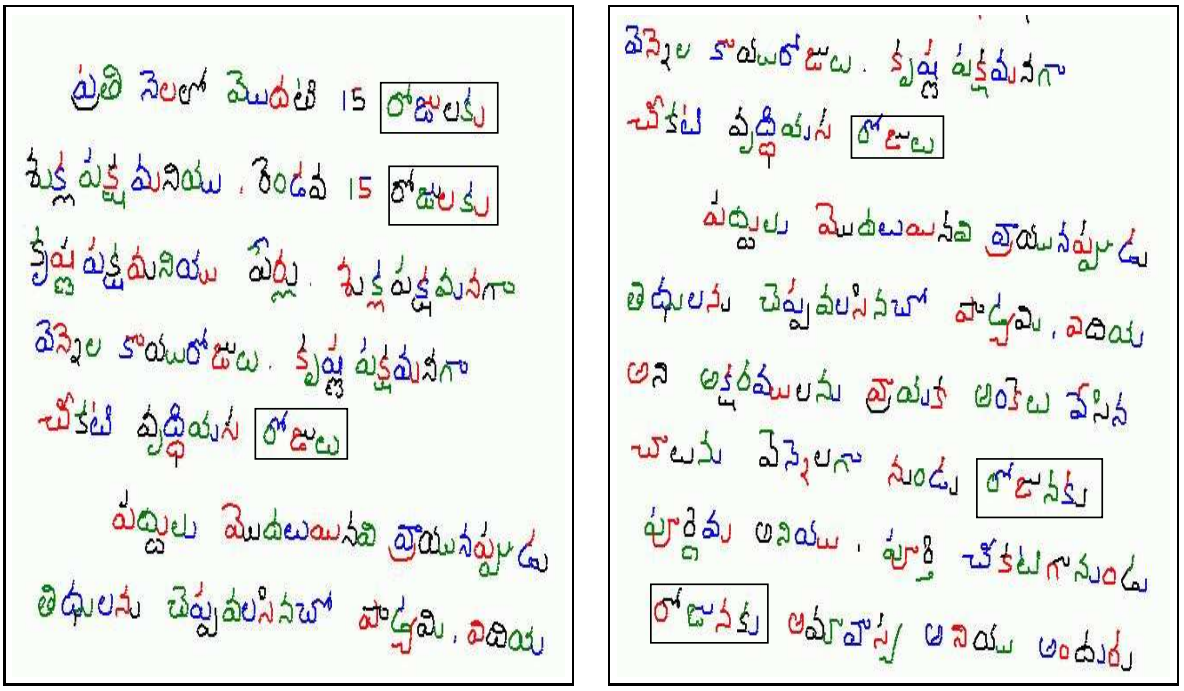


Figure 3.10: Sample Words Synthesized for three Different Writers

(meaning *"day"*). Figure 3.11 (b) also has similar results for the word *"roojulu"* written by a different writer. As can be seen from the result, the search is able to retrieve words written by different writers based on the synthesized words, which shows the effectiveness of the synthesis based retrieval scheme.



(a) (b)

Figure 3.11: Search Result for the Input Word "roojulaku" in Telugu Written by (a) Writer 1, (b) Writer 2.

Figure 3.12 takes a closer look at the retrieved words, which contain both writer differences and word form variations. The results clearly indicate that the partial matching scheme can effectively handle word form variations.

We also quantified the overall performance of the system on the complete database using the measures of precision and recall. The precision refers to the proportion of the retrieved entities that are relevant, while recall refers to the proportion of relevant entities in the database that were retrieved. As the threshold used for matching is varied, one gets various values for precision and

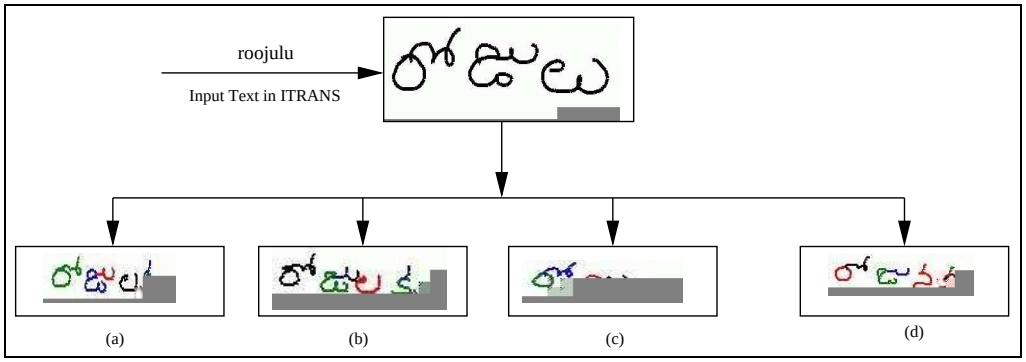


Figure 3.12: Synthesized Telugu Word *"roojulu"* and Retrieved words (a,b) *"roojulaku"* by Writer 1, (c) *"roojulu"* by Writer 2, and (d) *"roojulaku"* by Writer 2.

recall. However, it may be noted that as the threshold is made tighter, the precision increases, while the recall decreases, and vice versa.

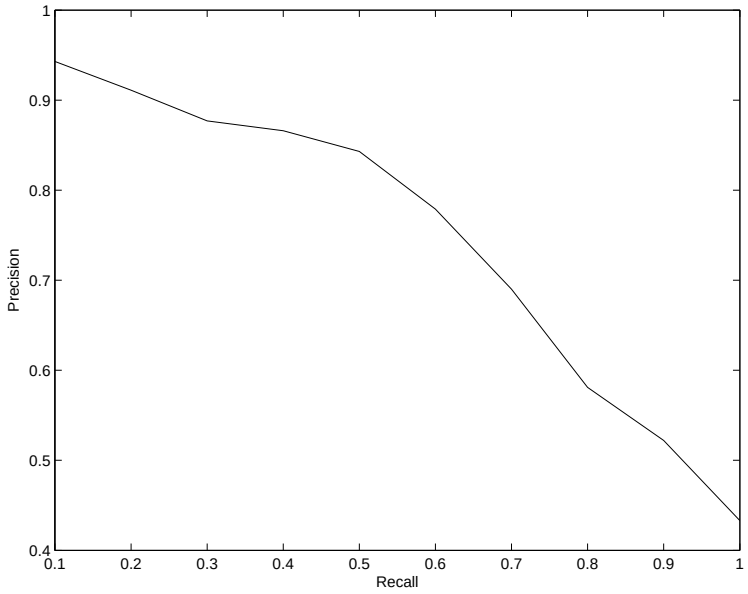


Figure 3.13: Precision-vs-Recall Plot for the Experiment.

Figure 3.13 shows the results of the retrieval experiment as a precision-recall curve. It is clear that even in a multi-writer databases, our algorithm is able to achieve very good retrieval performances. Comparable algorithms are those by Lopresti and Tompkins [83] and Jain and Namboodiri [31], both of which do not handle the multi-writer cases well. The equal error rate is the point at which the precision equals the recall, which is 25.2% on Dataset 1. A direct comparison of the numbers are meaningless as the databases are not the same. However, based on the dataset size and its complexity, the performance is quite comparable to the reported work, while adding the capability of searching in multi-writer databases.

3.8.2 Multi-Script Synthesis and Retrieval

In addition to the multi-writer case, our algorithm is also able to handle multi-script databases using a single query, especially in Indian scripts. For multi-script synthesis, the input data is either translated or transliterated into the target script. Transliteration is possible for proper nouns in Indian languages due to the fact that all the scripts share a common alphabets although the shape of scripts themselves are very different (see Figure 3.14), which is used for cross-lingual search as shown in Figure 3.14.

Figure 3.14: Shared Alphabets of Multiple Scripts.

Once the query word is converted to the representation in the required script, a synthesizer in the corresponding script is employed to generate the corresponding handwritten data (see Figure 3.15).

Given an input word in ITRANS, one can generate the corresponding strokes in any of the Indian scripts and use the corresponding language model to generate the handwriting for a particular user. This capability is essential for applications such as cross-lingual search, where one would like to search for a word in different scripts, simultaneously.

Each of the generated query string is then compared with the words in the database (or index) to retrieve the corresponding documents. The results are identical to providing separate queries for each script, and hence is very effective. The query results on the multi-script database yields a precision of 94.3% at a recall rate of 89.1%. The results are comparable to the single script case and hence shows the effectiveness of the framework.

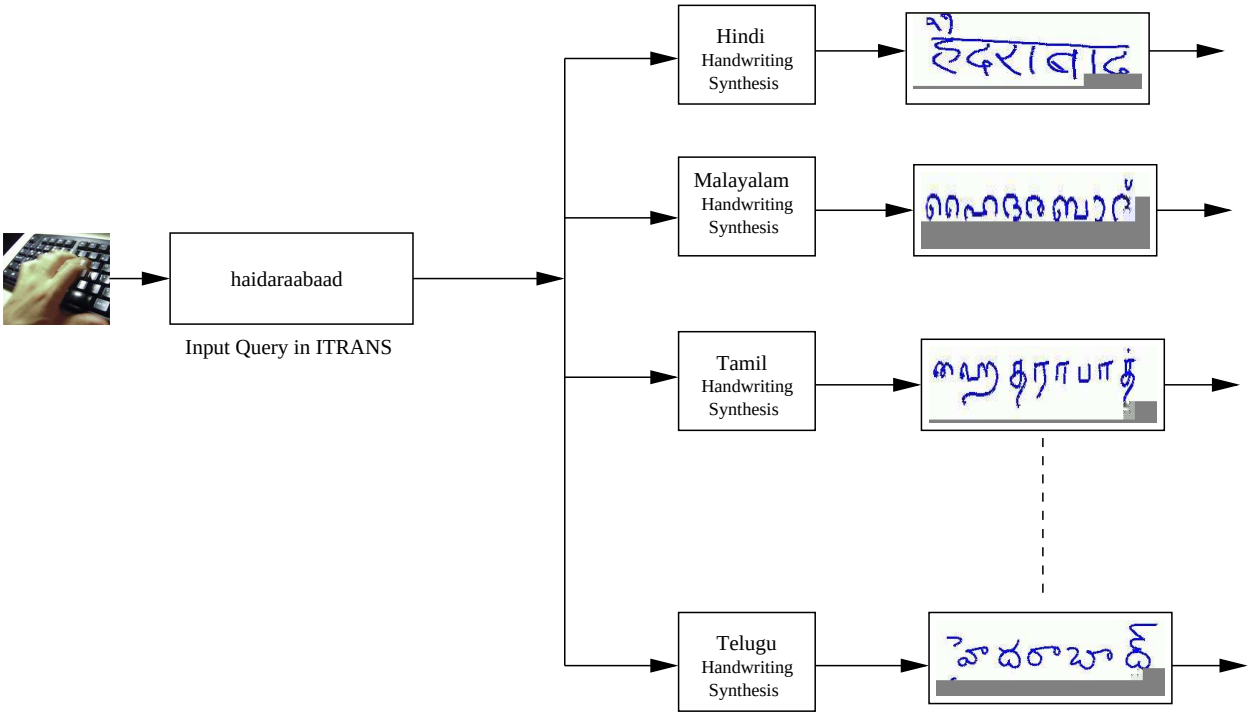


Figure 3.15: Multi-script Query Expansion.

3.9 Discussions

We have proposed a writer-independent recognition-free approach for retrieval of handwritten data in Indian language scripts. The proposed approach uses handwriting synthesis to do matching in the ink domain as opposed to the use of a recognizer. Synthesis results using the model learnt from training samples produces natural looking words. The framework also incorporates information retrieval measures such as TF/IDF to rank the retrieved documents. The approach also supports multi-lingual queries, which is especially useful for Indian languages. On the other hand, there needs to be further investigation on writer independence, stroke ordering, a complex stroke model and a way to quantify performance. We are currently working on modeling the handwritten data as a mixture distribution, which will be able to incorporate more writing variations within the data of a single user. Another interesting directions that we are currently pursuing is the study of stroke shape variations when it is in the proximity of other strokes or when the position of the stroke in the word changes.

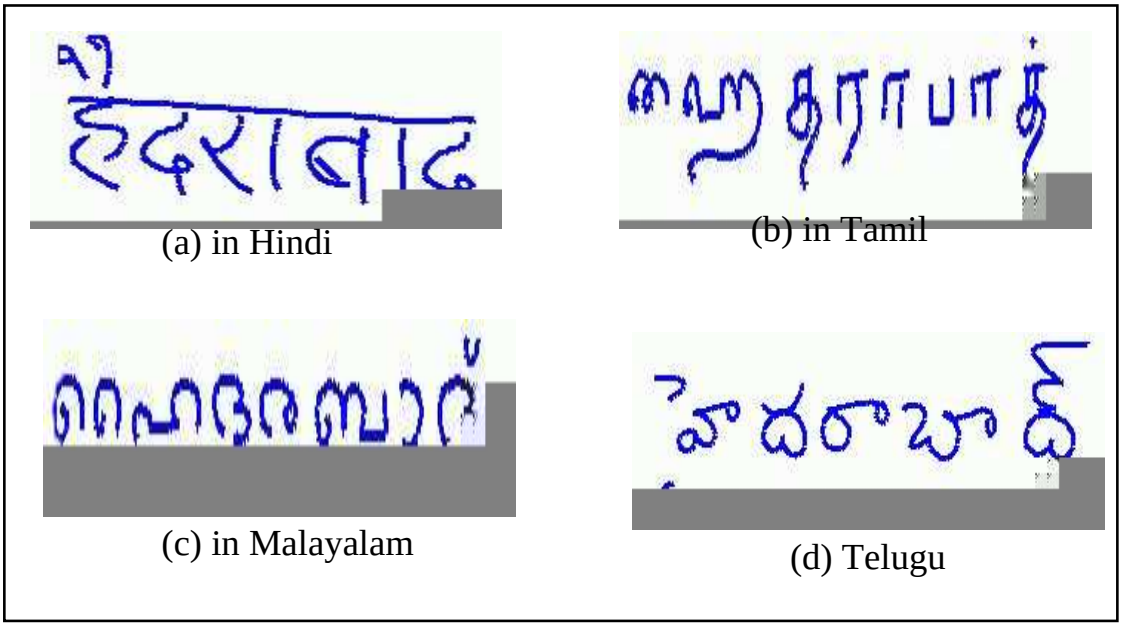


Figure 3.16: The Word *haidaraabaad* Synthesized in Various Indian Languages

Chapter 4

Development of Document Retrieval Systems

4.1 Introduction to Document Retrieval Systems

Document retrieval is about matching of some user stated query against useful parts of text records and presents the retrieved results to the user. These records could be any type of mainly unstructured text, such as bibliographic records, newspaper articles, or paragraphs in a manual, images, audio and video records and other multimedia data. User queries could range from multi-sentence full descriptions of an information need to a few words.

4.1.1 Adapting Search Engines for Document Images

A search engine or search service is a program designed to help find information stored on a computer system such as the World Wide Web, inside a corporate or proprietary network or a personal computer. The search engine allows one to ask for content meeting specific criteria (typically those containing a given word or phrase) and retrieves a list of references that match those criteria. Search engines use regularly updated indexes to operate quickly and efficiently. Without further qualification, search engine usually refers to a Web search engine, which searches for information on the public Web. Other kinds of search engine are enterprise search engines, which search on intranets, personal search engines, which search individual personal computers, and mobile search engines. However, while different selection and relevance criteria may apply in different environments, the user will probably perceive little difference between operations in these. Search engines do not always provide the right information, but rather often subject the user to a deluge of disjointed irrelevant data.

Search engines in the context of digital libraries have assumed an even more critical role. The number of users accessing a digital library search are increasing constantly at a higher rate. Digital libraries consist of millions of scanned document images and it is of utmost importance that each and every page in a digital library is searchable. Typical search engines are designed for textual documents and needs special care to adapt to document images. Scalability of these search engines to adapt to the ever growing size of digital libraries, adaptation of language processing modules in the context of document images and many other search areas still remain in an open area.

4.1.2 Document Viewer and Portable Document Format

There are a variety of document formats that are available and each document format has a different file structure and has to be treated individually. Some of the widely used document formats are HTML, PDF, DOC, RTF, etc. A document viewer typically might support many formats to render them faithfully or there may be document viewers that are built specifically for a particular type of document format, for instance, the Xpdf [41] viewer supports only PDF documents, while the MS Word can handle many different formats, ranging from DOC and RTF to HTML.

The following are some of the important roles of a document viewer.

- Faithful reproduction of the content in the document
- To provide/display content in a conceivable manner to the end-user
- Ease of navigation within and among documents
- Ability to locate (find) parts of a document and present them to the user

Portable Document Format, the PDF, is the default standard for electronic exchange of documents, and also an industry standard for intermediate representation of printed material. The goal for developing PDF is to enable users exchange and view electronic documents easily and reliably, independently of the environment in which they were created. Digital libraries contain a variety of documents in different storage formats, and PDF files are available in large amounts, especially most of the scientific literature gets published in PDF format which the digital libraries indexes.

Limitations Though digital libraries contain a plethora of PDF files, search in PDF documents, especially within the graphics stream that contain document images is currently not possible. Most of the PDFs are generally a scanned a copy of the original hard copy, and since these are stored as images within a PDF file, their textual content is inaccessible. Also, Indian language content cannot be searched within a PDF file in both text and graphics stream. This is because most of Indian language content in PDF files are encoded in a particular font-encoding and the text stored inside images cannot be OCR'ed as we do not have an OCR for Indian languages that claims a high accuracy. This leads us to a recognition-free approach to solve the search in document images by matching directly at the word image level.

4.2 Scalable Search Engines for Document Images

Large digital libraries, such as Digital Library of India (DLI) [34] are emerging for archiving large collection of printed and handwritten documents. The DLI aims at digitizing all literary, artistic, and scientific works of mankind so as to create better access to traditional materials, easier preservation, and make documents freely accessible to the global society. More than 25 scanning centers all over India are working on digitization of books and manuscripts. The mega scanning center we have, has around fifty scanners, each one of them capable of scanning approximately 5000 pages in 8 hours. As on September 2005, close to 100 thousand books with 25 million pages were digitized and made available online by DLI (*http://dli.iiit.ac.in*) as document images.

Building an effective access to these document images requires designing a mechanism for effective search and retrieval of textual data from document image collections. Document image indexing and retrieval were studied with limited scope in literature [35]. Success of these procedures mainly depends on the performance of the OCRs, which convert the document images into text. Much of the data in DLI are in Indian languages. Searching in these document image collections based on

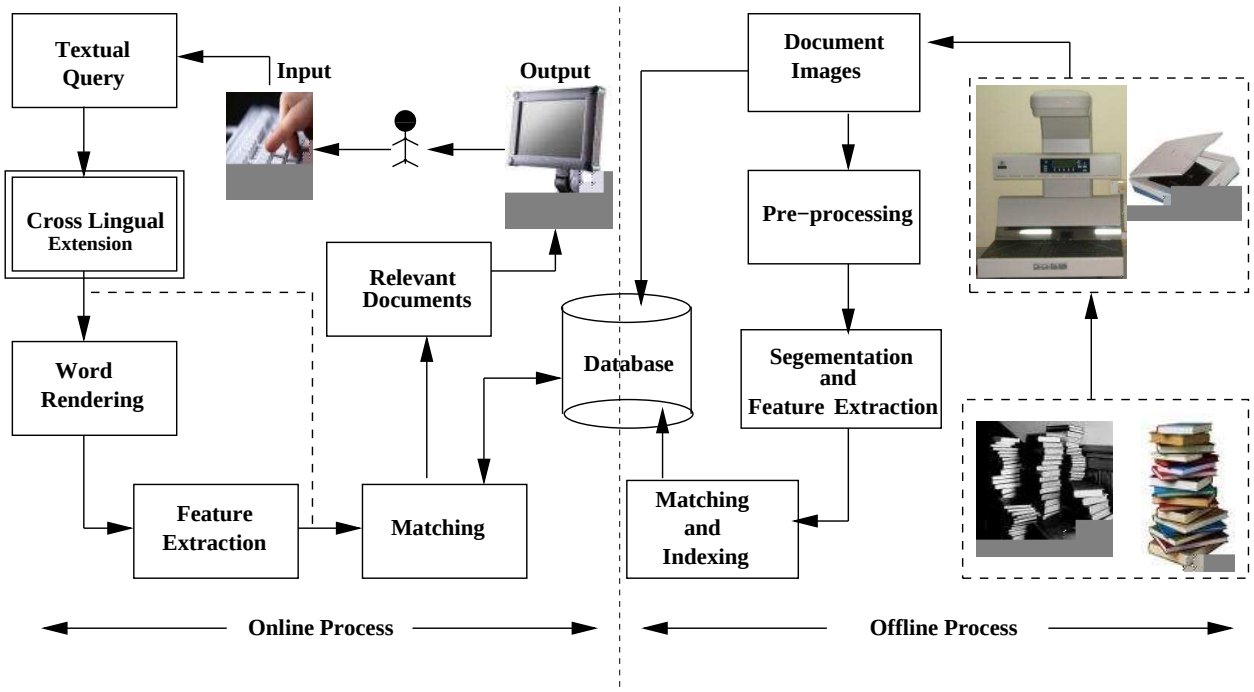


Figure 4.1: Conceptual Diagram of the Searching Procedure from Multilingual Document Image Database.

content, is not presently possible. This is because OCRs are not yet able to successfully recognize printed texts in Indian languages. We need an alternate approach to access the content of these documents [37]. A promising alternate direction is to search for relevant documents in image space without any explicit recognition. We have been motivated by the successful attempts on locating a specific word in handwritten English documents by matching image features for historical documents [11, 36].

We have already addressed algorithmic challenges for effective search in document images [84] . This chapter describes the issues associated with the implementation of a scalable system for Indian language document images. A conceptual block diagram of our prototype system is shown in Figure 4.1. Our system accepts textual query from users. The textual query is first converted to an image by rendering, features are extracted from these images and then search is carried out for retrieval of relevant documents. Results of the search are pages from document image collections containing queried word sorted based on their relevance to the query.

4.2.1 Greenstone

Greenstone [38] is a suite of software for building and distributing digital library collections. It provides a new way of organizing information and publishing it on the Internet or on CD-ROM. Greenstone is produced by the New Zealand Digital Library Project at the University of Waikato, and developed and distributed in cooperation with UNESCO and the Human Info NGO.

The core of our system is the Greenstone, an open source search engine for digital libraries, which indexes and stores text information about all documents. Greenstone is a comprehensive system for constructing and presenting collections of thousands or millions of documents. including text, images, audio and video.

Collections

A typical digital library built with Greenstone will contain many collections, individually organized though they bear a strong family resemblance. Easily maintained, collections can be augmented and rebuilt automatically. There are several ways to find information in most Greenstone collections. For example, you can search for particular words that appear in the text, or within a section of a document. You can browse documents by title or you can browse documents by subject and many other categories.

Indexing

Greenstone constructs full-text indexes from the document text that is, indexes that enable searching on any words in the full text of the document. Indexes can be searched for particular words, combinations of words, or phrases, and results are ordered according to how relevant they are to the query.

Users can browse interactively around lists, and hierarchical structures, that are generated from the metadata that is associated with each document in the collection. Metadata forms the raw material for browsing. It must be provided explicitly or be derivable automatically from the documents themselves. Different collections offer different searching and browsing facilities. Indexes for both searching and browsing are constructed during a "building" process, according to information in a collection configuration file. Greenstone creates all index structures automatically from the documents and supporting files: nothing is done manually. If new documents in the same format become available, they can be merged into the collection automatically. Indeed, for many collections this is done by processes that awake regularly, scout for new material, and rebuild the indexes all without manual intervention.

Document formats

Source documents come in a variety of formats, and are converted into a standard XML form for indexing by "plugins". Plugins distributed with Greenstone process plain text, HTML, WORD and PDF documents, and Usenet and E-mail messages. New ones can be written for different document types. To build browsing structures from metadata, an analogous scheme of "classifiers" is used. These create browsing indexes of various kinds: scrollable lists, alphabetic selectors, dates, and arbitrary hierarchies. Again, Greenstone programmers can create new browsing structures.

Multimedia and multilingual documents

Collections can contain text, pictures, audio and video. Non-textual material is either linked into the textual documents or accompanied by textual descriptions (such as figure captions) to allow full-text searching and browsing. Unicode, which is a standard scheme for representing the character sets used in the world's languages, is used throughout Greenstone. This allows any language to be processed and displayed in a consistent manner. Collections have been built containing Arabic, Chinese, English, French, Maori and Spanish. Multilingual collections embody automatic language recognition, and the interface is available in all the above languages.

4.2.2 Challenges in Design and Implementation of the System

Search and retrieval from large collection of document images is a challenging task, specially when there is no textual representation available. To design and implement a successful search engine in image domain, we need to address the following issues.

Search in images: Search in image space requires appropriate representational schemes and similarity measures. Success of content-based image retrieval(CBIR) schemes were limited by the diversity of the image collections. Digital libraries primarily archive text images, but of varying quality, script, style, size and font. We need to come up with appropriate features and matching schemes, which can represent the content (text), while invariant to the popular variations.

Degradations of documents: Documents in digital libraries are extremely poor in quality. Popular artifacts in printed document images include (a) Excessive dusty noise, (b) Large ink-blobs joining disjoint characters or components, (c) Vertical cuts due to folding of the paper, (d) Cuts of arbitrary direction due to paper quality or foreign material, (e) Degradation of printed text due to the poor quality of paper and ink, (f) Floating ink from facing pages etc. We need to design an appropriate representational scheme and matching algorithm to accommodate the effect of degradation.

Need for cross-lingual retrieval: Document images in digital libraries are from diverse languages. Relevant documents that users need may be available in different languages. Most educated Indians can read more than one language. Hence, we need to design a mechanism that allows users to retrieve all documents related to their queries in any of the Indian languages.

Computational speed: Searching from large collection of document images pass through many steps: image processing, feature extraction, matching and retrieval of relevant documents. Each of these steps could be computationally expensive. In a typical book, there could be around 90,000 words and processing all of them online is practically impossible. We do all computationally expensive operations during the offline indexing (Section 4.2.4) and do minimal operations during online retrieval (Section 4.2.5).

Indian languages: Indian languages pose many additional challenges [85]. Some of these are: (i) lack of standard representation for the fonts and encoding, (ii) lack of support from operating system, browsers and keyboard, and (iii) lack of language processing routines. These issues add to the complexity of the design and implementation of a document image retrieval system.

4.2.3 Representation and Matching of Word Images

Word images extracted from documents in digital libraries are of varying quality, script, font, size and style. An effective representation of the word images will have to take care of these artifacts for successful searching and retrieval. We combined two categories of features to address these effects: word profiles and structural features. We use many features that are used for matching the word images that cater to font size, style and script variations. Explicit definitions of these features may be seen in [84]. These features typically are the *Word Profiles*, *Structural Features* and others. A detailed description of these features are explained in Appendix E.3.

Spotting a word from handwritten images is attempted by pairwise matching of all the words [36]. However for proper search and retrieval, one needs to identify the similar words and group them based on their similarity, and evaluate the relative importance of each of these words and word clusters. Matching is used to compute dissimilarity between word images. We use a simple squared Euclidean distance while computing the dissimilarity. For matching word images we use Dynamic Time Warping (DTW) that computes a sequence alignment score for finding the similarity of words [84]. The use of the total cost of DTW as a distance measure is helpful to cluster together word images that are related to their root word, which is discussed in Section 4.2.4.

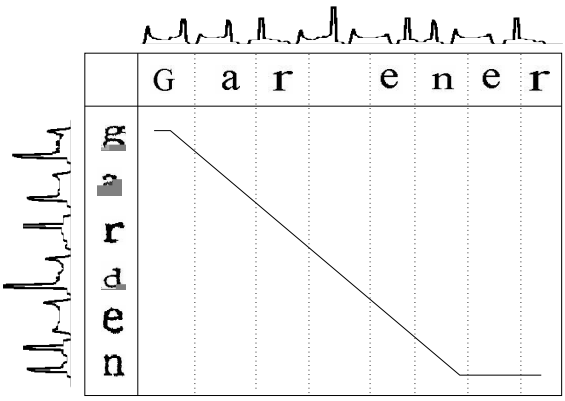


Figure 4.2: Dynamic Time Warping (DTW) Plot during Matching.

DTW is a dynamic programming based procedure [36] to align two sequences of signals and compute a similarity measure. Let the word images (say their profiles) are represented as a sequence of vectors $\mathcal{F} = \mathbf{F_1}, \mathbf{F_2}, \ldots \mathbf{F_M}$ and $\mathcal{G} = \mathbf{G_1}, \mathbf{G_2}, \ldots, \mathbf{G_N}$. The DTW-cost between these two sequences is $D(M, N)$, which is calculated using dynamic programming is given by:

$$D(i,j) = \min \begin{cases} D(i-1, j-1) \\ D(i, j-1) \\ D(i-1, j) \end{cases} + d(i,j)$$

where $d(i,j) = \sum_{k=1}^{N} (F(i,k) - G(j,k))^2$ (the cost in aligning the ith element of $\mathbf{F}$ with jth element of $\mathbf{G}$). Using the given three values $D(i, j-1), D(i-1, j)$ and $D(i-1, j-1)$ in the calculation of $D(i,j)$ realizes a local continuity constraint, which ensures no samples are left out in time warping. Score for matching the two sequences $\mathcal{F}$ and $\mathcal{G}$ is considered as $D(M, N)$, where M and N are the lengths of the two sequences. Structural features can also be incorporated into the framework by computing them for the vertical strips. Detailed discussion of the algorithms is available in [84].

4.2.4 Offline Indexing

The simple matching procedure described in Section 4.2.3 may be efficient for spotting or locating a selected word-image. However the indexing process for a good search engine is more involved than the simple word-level matches. A word usually appears in various forms. Variation of word forms may obey the language rules. Text search engines use this information while indexing. However for text-image indexing process, this information is not directly usable.

We take care of simple, but very popular, word form variations taking place at the beginning and end. For this, once sequences are matched, we backtrack the optimal cost path. During the backtracking phase, if the dissimilarity in words is concentrated at the end, or in the beginning, they are deemphasized. For instance, for a query "garden", the matching scores of the words "Gardener" and "garden" are only the matching of the six characters, 'g-a-r-d-e-n', of both words. Once an optimal sub-path is identified, a normalized cost corresponding to this segment is considered as the matching score for the pair of words. With this we find that a large set of words get grouped into one cluster. We expect to extend this for more general variations of words.

The optimal warping path is generated by backtracking the DTW minimal score in the matching space. As shown in Figure 4.2, extracted features (using upper word profile) of the two words 'Gardener' and 'garden' are aligned using DTW algorithm. Also, note that the starting letter, **"g"** and **"G"**, at the image level are different. It can be observed that for word variants the DTW path deviates from the diagonal line in the horizontal or vertical direction from the beginning or end of the path, which results in an increase in the matching cost. In the example Figure 4.2, the path deviates from the diagonal line at the two extreme ends. This happened during matching the two words, that is, the root word (garden) and its variant (Gardener). Profiles of the extra character ('er') have minimal contribution to the matching score and hence subtracted from the total matching cost so as to compute the net cost. Such word form variations are very popular in most languages.

For the indexing process, we propose to identify the word set by clustering them into different groups based on their similarities. This requires processing the page to be indexed for detection of relevant words in it. Many interesting measures are proposed for this. We propose the following steps for effective retrieval at image level.

Detection of Common Stop Words: Once similar words are clustered, we analyze the clusters for their relevance. A very simple measure of the uniformity of the presence of similar words across the documents is computed. This acts as an inverse document frequency. If a word is common in most of the documents, this word is less meaningful to characterize any of the document.

Document Relevance Measurement: Given a query, a word image is generated and the cluster corresponding to this word is identified. If a cluster is annotated, matching query word is fast and direct. For other clusters, query word image and prototype of the cluster are compared in the image domain. In each cluster, documents with highest occurrence of similar words are ranked and listed.

Clustering: Large number of words in the document image database are grouped into a much smaller number of clusters. Each of these clusters are equivalent to a variation of the single word in morphology, font, size, style and quality. Similar words are clustered together and characterized using a representative word. We follow a hierarchical clustering procedure [86] to group these words. Clusters are merged until the dissimilarity between two successive clusters become very high. This method also provides scope for incremental clustering and indexing.

Annotation: After the clustering process has been completed offline, we have a set of similar words in each cluster. These clusters are annotated by their root word to ease searching and retrieval. Suppose a cluster contains words such as 'programmer', 'programmers', 'programming', 'programs' and 'program'. Then, we annotate the cluster with the root word "program". Likewise all clusters are manually annotated. If the annotation is not available, we identify an image-representative for the cluster. However, presence of image-prototype can slow down the search process. During searching, cluster prototypes are accessed and checked for their similarity with the query word. This makes sure that search in image domain is as fast as search in text domain.

4.2.5 Online Retrieval

A prototype web-based system for searching in document images, is also developed. This is presently available at http://cvit.iiit.ac.in/wordsearch.html. The system has many basic features as discussed below.

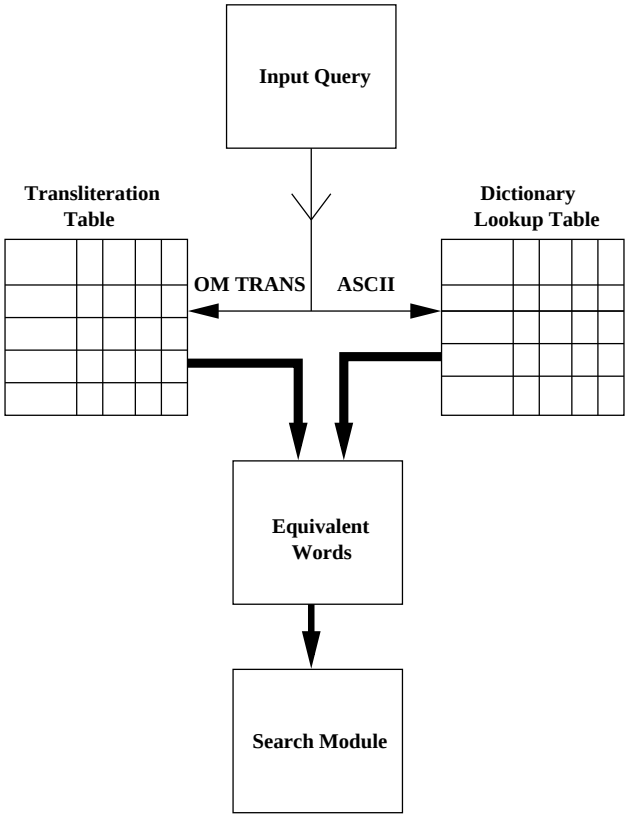


Figure 4.3: A Conceptual Diagram that Shows Document Searching in Multiple Languages using Transliteration and Dictionary Based Approach

Alphabet	a	aa	i	ii	u	uu	....
Hindi	अ	आ	इ	ई	उ	ऊ	....
Telugu	అ	ఆ	ఇ	ఈ	ఉ	ఊ	....
:	:	:	:	:	:	:	
:	:	:	:	:	:	:	
:	:	:	:	:	:		

Figure 4.4: Sample Entries of the Transliteration Map Built for Cross-lingual Retrieval in English, Devanagari and Telugu

Web-based GUI: The web interface allows the user to type in Roman text and simultaneously view the text in one of the Indian languages of his choice. Users can also have the option to search with cross lingual retrieval and can specify the kind of retrieval they want to use. Many retrieval combinations are also provided in the advanced search options such as case insensitivity, boolean searching using !, &, | *and parenthesis*, displaying up to 50 search results per page, and various others. There is on the fly character transliteration available. The user can first choose a particular language (such as Hindi, Telugu, etc.) and then see the text in the corresponding language as he keeps typing the query in Roman (OM-Trans).

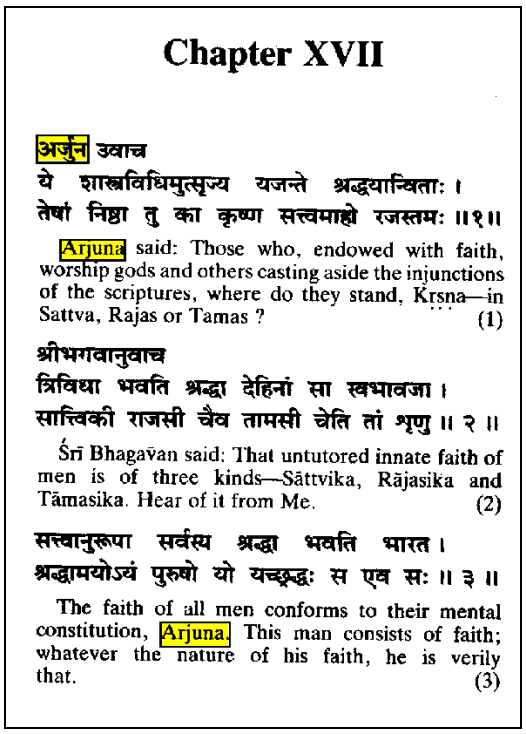
Delivery of Images: In order to facilitate users access to the retrieved document images, there is a need to control image size and quality. When a book is typically scanned at a resolution of 600 dpi, the original scanned size of a single page is around 12MB as a PNG file. Viewing such page is too slow and needs network resources. It is wise to make these images available in a compressed form. We compress the above image to a size ranging between 30 to 40 KB in TIFF format, by reducing the size of the image. This makes sure that the delivery of images are faster over the Internet. TIFF image format helps us in general for achieving the trade-off between image size and quality. It keeps the quality of the image during the compression process over JPG and BMP formats.

Speculative Downloading: Our system also supports speculative downloading, where some related pages with the currently retrieved page are prefetched for quick viewing during searching and retrieval as per users query. This mechanism is helpful especially when the user is viewing a collection, page by page, with the assumption that he might view the next page also. Speculative downloading is a background process.

Dynamic Coloring: When a user searches for relevant pages to a given query, our system searches and displays the result with dynamic coloring of all the words in the page that are similar to the queried word. This helps users to easily evaluate relevance of the retrieved page to their need. We adopt false coloring mechanism such that each word in a query carries a unique color in a document image. All this coloring happens at runtime (Figure 4.5) at image level.

Scalability: DLI is a one million book scanning project. Hence it archives huge collection of document images. Searching in this situation raises the question of scalability.

The current prototype system searches in three books, that are a mixture of English and other Indian Languages (Hindi and Telugu). Each book on the average consists of 350 pages, and each



Chapter XVII

अर्जुन उवाच
ये शास्त्रविधिमुत्सृज्य यजन्ते श्रद्धयान्विताः ।
तेषां निष्ठा तु का कृष्ण सत्त्वमाहो रजस्तमः ॥१॥

Arjuna said: Those who, endowed with faith, worship gods and others casting aside the injunctions of the scriptures, where do they stand, Krsna—in Sattva, Rajas or Tamas ? (1)

श्रीभगवानुवाच
त्रिविधा भवति श्रद्धा देहिनां सा स्वभावजा ।
सात्त्विकी राजसी चैव तामसी चेति तां शृणु ॥ २ ॥

Śrī Bhagavān said: That untutored innate faith of men is of three kinds—Sāttvika, Rājasika and Tāmasika. Hear of it from Me. (2)

सत्त्वानुरूपा सर्वस्य श्रद्धा भवति भारत ।
श्रद्धामयोऽयं पुरुषो यो यच्छ्रद्धः स एव सः ॥ ३ ॥

The faith of all men conforms to their mental constitution, Arjuna. This man consists of faith; whatever the nature of his faith, he is verily that. (3)

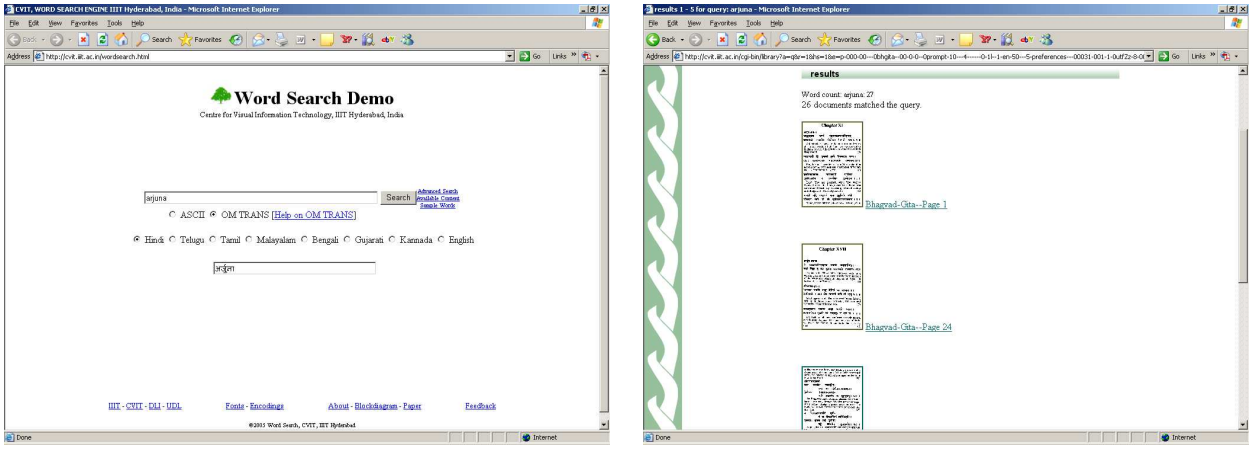
Figure 4.5: Search Result with Dynamic Coloring for Query Word 'Arjuna' seen Both in English and Devanagari

page with 300 words. This brings the total number of words to 360,000. This is relatively a small number. The system should aid in searching the huge one million book collection and thus the scalability issues come to forth. Indexing this large collection takes immense time. For us indexing is an offline activity. Searching and retrieval is the only online process. That is why the system manages to run fast in the above sample database. Because it only checks keywords of the index to search for similar words with the query. Even with an increase in the size of document images we do not expect much increase in the number of clusters. Because, words are limited in every language and they are only the morphological variants of the root word. The system handles addition of new books without re-indexing every time. This saves much time and creating new indexes will be a smooth process. However, we need to deal with delivery of the document images. An increase in the number of pages viewed may slow the transfer process. A good compression technique needs to be applied.

Cross-lingual Search Our system can also search for cross-lingual documents for a given query. As shown in Figure 4.3 we achieve this in two ways: transliteration and dictionary-based approaches. Figure 4.3 is an expanded view of the cross-lingual block diagram presented in Figure 4.1.

Since Indian scripts share a common alphabet (derived from *Brahmi*), we can transliterate the words across languages. This helps us to search in multiple languages at the same time. We use OM-Transliteration scheme [43]. In OM-Trans scheme, there is a Roman equivalent for all the basic Indian language characters.

Figure 4.4 shows a sample transliteration map built for this purpose. For Example, "Bhim " can be typed in its Roman equivalent using OM-trans as "Bhima". Then the transliteration table is looked up for searching in Hindi (Devanagari script) and Telugu pages. Their cumulative result



(a) (b)

Figure 4.6: Screenshots of Implementation Results for Cross-lingual Search. (a) An Interface where Users Enter their Query, (b) View of Result of the Search as Thumbnails

is finally displayed back to the user. A screenshot showing implementation result is presented in Figure 4.6.

We also have a dictionary-based translation for cross-lingual retrieval. In this approach, every English word has an equivalent word in the corresponding Indian and other oriental languages. If a user queries for the word 'India', the dictionary lookup points to 'भारत' in Hindi for searching relevant documents across languages. This table is extended also for other Indian languages. The result of the search are documents that contain the query word 'India' in all the languages.

We tried searching in scanned documents from the book 'Bhagavat-Gita'. Pages from this book contain English and Devanagari text. These pages are of poor quality. We search for the occurrences of the word 'arjuna'. It fetched pages which contain 'Arjuna' in both English and Devanagari. Sample results are shown in Figure 4.7 (b). In this respect, we need to exploit the available technology at WordNet Project [87] and Universal Language Dictionary Project [88]. WordNet is a lexical database that has been widely adopted in artificial intelligence and computational linguistics for a variety of practical applications such as information retrieval, information extraction, summarization, etc. The Universal Language Dictionary is an attempt to create a list of concepts along with words to express those concepts in several "natural" and "artificial" (constructed) languages.

4.2.6 Discussions

We have a prototype system for retrieval of document images. This system is integrated with 'Greenstone search engine' for digital libraries [38]. Greenstone is a suite of software for building and distributing digital library collections via the Internet. Given a textual query, we convert it to image by rendering. Features are extracted from these images and then search is carried out for retrieval of relevant documents in image space. We extend the search to cross-lingual retrieval by transliteration among Indian languages and a table-lookup translation for other languages. Results of the search are presented to the user in a ranked manner based on their relevance to the query word.

We evaluated the performance of the system on data sets from languages such as English and Hindi. Pages of Hindi and English are taken from digital library of India collections. The system is extensively tested on all these data sets. Sample words retrieved are shown in Figure 4.7 (a). We measure the speed of the system so as to see its practicality. The system takes 0.16 seconds to search and retrieve relevant documents from image databases and 0.34 seconds to transfer that page for viewing by users over the intranet. In comparison, Greenstone text search takes 0.13 seconds

Figure 4.7: Results: (a) Sample Word Images Retrieved for the Queries Given in Special Boxes. Examples are from English and Hindi Languages. The Proposed Approach Takes Care of Variations in Word-form, Size, Font and Style Successfully. (b) Example Result for Cross-lingual Search from Bhagavat Gita Pages.

to search and retrieve relevant documents from image databases and 0.31 seconds to transfer that page for viewing by users over the intranet. The speed of our system is almost comparable with the Greenstone text search. This shows the effectiveness of the system. The strategy we followed is to perform text processing and indexing offline. The search then takes place on the representative words indexed. Compressing the image (to a size of few KB) also help us a lot during the transfer of the document image for viewing.

Quantitative performance of the matching scheme is computed on sample document image databases of size more than 2500 words. Around 15 query words are used for testing. During selection of query words, priority is given to words with many variants. We computed recall and precision for these query words. Percentage of relevant words which are retrieved from the entire collection is represented as recall, where as, percentage of retrieved words which are relevant is represented as precision. It is found that a high precision and recall (close to 95%) is registered for all the languages. High recall and precision is registered in our experiment. This may be because of the limited dataset we experimented with, that are similar in font, style and size. We are working towards a comprehensive test on real-life large datasets. Our existing partial matching module controls morphological word variants. We plan to make the module more general so that it addresses many more variations of words encountered in real-life documents. We are also working on avoiding the manual annotation and still retaining the same performance.

4.3 Textual Search in Graphics Stream of PDF

4.3.1 Portable Document Format

More than 200 million Portable Document Format (PDF) [39] documents on the web today serve as evidence of the number of organizations that rely on PDF to store information. Today, PDF is the *de facto* standard for electronic exchange of documents, and also an industry standard for intermediate representation of printed material. The goal for developing PDF was to enable users to exchange and view electronic documents easily and reliably, independent of the environment in which they were created. Origins of the PDF [39] dates back to the nineties. During those days, PostScript page description language was the widely accepted standard for printing purposes. PDF

was built on top of the PostScript, so that not only it supported printing but also supported viewing capability. PDF *document* is a collection of *objects*. These *objects* can be located in a PDF file in any arbitrary order, but are connected to each other by a reference mechanism. Therefore, a viewer application should process a PDF file by following references from *object* to *object*, instead of processing *object* sequentially. Hence, every PDF file contains a *cross-reference table*, stored at the end of the file. Cross-reference makes sure that a PDF file containing very large number of documents can be accessed efficiently with almost no time constraints. A PDF document can be regarded as a hierarchy of objects contained in the body section of a PDF file. At the root of the hierarchy is the document's catalog dictionary. Most of the objects in the hierarchy are objects named *dictionaries*. For example, each page of the document is represented by a *page object*, a dictionary that includes references to the page's contents. The individual page objects are tied together in a structure called *Page Tree*.

4.3.2 Challenges in Design and Implementation

The challenges in using an existing opensource PDF viewer are manifold. In order to understand how a document viewer (Xpdf in this case) supports a particular document format (PDF in this case), one also should have insights and understanding of that particular document format. For example, a movie player software must understand and decipher how a typical movie file would be structured, so as to play the movie and give access controls to the user for navigation. As a software designer, one must also understand how the frames are arranged in a standard movie file, its resolution and other specifics so as to render the movie faithfully onto the display. Similarly, one must understand the nuances of the PDF file format before one could start using an existing opensource implementation of the same. We extend the conventional textual search to graphics (image) representation of documents. Conventional text search is based on matching or comparison of textual description (say in UNICODE). These techniques can not be used to access content at the image/graphics level, where text is represented as pixels but not as UNICODE. We have implemented the proposed solution in an open source PDF reader (Xpdf) to demonstrate that the textual search is possible in the graphics stream. We illustrate the challenges involved, starting from the understanding of the PDF file format till its implementation using the Xpdf viewer.

- Understanding the PDF file format: The PDF file format is an open format and a vast one that is being constantly upgraded. Unlike other document file formats, PDF has some programming language constructs, though not in its entirety and must not be read sequentially but by following references using a cross-reference table present in it. One has to clearly understand each of the PDF file format details and map them to the Xpdf source code. This mapping is not a direct one as it involves finding code fragments and stubs that are split across various source files across directories. There are 106 header files and 87 source files that comprise of the Xpdf application. The display has been developed using *Motif*, an X-windows based User Interface builder to achieve faithful rendering of a PDF file onto the display.

- The Graphics framework: The graphics stream hitherto has only been used for rendering purposes. Locating images, and more importantly document images in the graphics stream was of utmost priority. All the images had to be extracted in its entirety (original size) so that they can be subjected to document processing operations such as thresholding, segmentation and feature extraction. We had to accomplish this task during a PDF file read operation without disturbing the rendering activity. Fragments of code were added appropriately so as

to fork out the activity of extracting images and their features, and then the feature vectors were stored in a separate data structure that was always contained in the memory.

- Transparency in search: Transparency in search was one of the main motivations, because from a user's perspective he is finding a "word" irrespective of this result comes from a text region or a graphics region. The search in graphics region is little more elaborate as opposed to text search which is handled well within the existing opensource viewer Xpdf. We had to combine the existing text search framework into the graphics stream search framework in order to achieve this transparency.

- Indian language support: One of the main goals is to search Indian language content present in the graphics stream. Providing an intuitive interface to type in Indian language text was necessary. Our solution also allows the query word to be entered as Roman, using the ITRANS notation, and simultaneously view the chosen Indian language content that is being typed. This text must then be converted to an image so that matching in the graphics stream takes place at the word image level. Opensource image library, ImageMagick [89] was used to convert the input text into an image using a standard font in that particular Indian language. All the image reading and writing operations involved in this entire process was performed by using this library effectively. Present implementation supports Indian languages documents in Hindi and Telugu.

4.3.3 Word Search in a PDF file

A PDF file encapsulates a complete description of the document that includes the text, fonts, images, and 2D vector graphics. Importantly, PDF files do not encode information that is specific to the application software, hardware, or operating system used to create or view the document. This feature ensures that a valid PDF will render exactly the same regardless of its origin or destination. A PDF document is a data structure essentially made of *Objects*. A *Content Stream* is a PDF stream object whose data consists of sequence of instructions describing the graphical elements to be painted on a page. Each page object is represented by one or more content streams. Content streams are also used to package sequence of instructions as self-contained graphical elements, such as forms, patterns, certain fonts, and annotation appearances. PDF serves purpose for fundamentally two kinds of applications: the Producer (PDF generator) and the Consumer (PDF reader). Today, we have many opensource PDF viewers available, the popular among them are the Xpdf, Ghostscript and KPDF for Linux platforms and proprietary viewer such as the Adobe Acrobat for the Windows operating system. Each of these applications implement the textual query search, with additional functionalities such as searching a sequence of pages, specific selected pages, case-sensitive search, searching as a regular expression and provide navigating functionalities such as forward and backward search.

Figure 4.8 shows the entire procedure that a typical PDF reader application does when handling a PDF file for viewing and searching purposes. As shown in Figure 4.8, the PDF file structure consists of a one-line header identifying the version of the PDF specification to which the file conforms, a body containing Text and Graphic stream objects that make up the PDF, a Cross-reference table containing references to indirect object, a trailer giving the location of the cross-reference table and certain other special objects. A user typically presents a search string, and the *Text Search* module looks into all the *Text Stream* objects in a PDF file and displays the results to the user. This is how conventional PDF search works. We used an existing PDF reader application and modified it so as to implement the *Word Image Search* (within double rectangle) that looks into the *Graphics Stream* objects in a PDF file to match word images. The modules indicated by a double rectangle

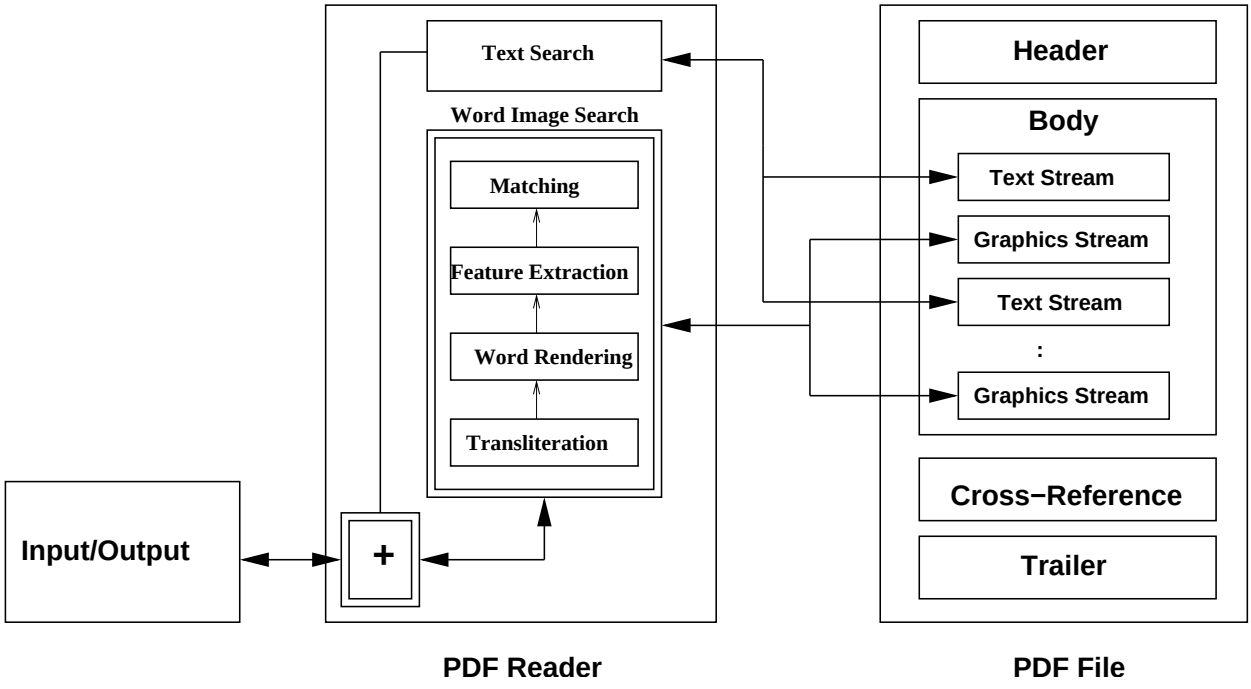


Figure 4.8: PDF Search Procedure. Our Proposed Search Scheme Integrates the Conventional Textual Search in a Transparent Manner.

in Figure 4.8 are the routines plugged inside a conventional PDF reader application to handle word image search. The results from both the search modules are integrated and is presented to the user making the whole process of searching transparent to the user.

4.3.4 Text Search in a PDF file

Most of the PDF readers/viewers support search (find) of query text. This search mechanism is handled *page* by *page* in a PDF file providing a means for inline searching. Whenever the user searches for a particular word, the word has to be searched in all the available pages of a PDF file. Typically, the search starts from the current page till the last page in the file and then continues from the first page. Every block (*Text Block*) in a page, is searched in a top-down sequential order. Text from every line in a block is extracted and is then compared with the input search string. The comparison is not straightforward, as the text extracted from these lines are read character by character (including spaces) from a line and then matched with the input search string. Additional information on search is handled appropriately during the search, like ignoring the case while searching. In this case, both the input search string and the text characters in the line are both converted to lower case (or upper case) and then compared. Information such as searching forward in a page or backward in page are also explicitly handled by the PDF reader applications. These search operations are PDF reader application dependent as some of them are capable of handling search at multiple lines and across blocks, while some of them handle only at the line level. Once a match has been found, the search string in the PDF file is shown in reverse video. The display related functionalities are handled separately and they are dependent on the *user coordinates*, which typically is dependent on the resolution of the display, the current zoom of the reader application and other such device dependent features.

4.3.5 Searching in the Graphics Stream

OCR-based Search: We often find PDF files that contain document images stored in the graphics stream. Document image contains textual information in the form of an image. Most of these images are scanned copies of technical reports, papers, journals or books. A PDF reader cannot extract text from an image and thus making it impossible to search within document images embedded in the graphics stream. During a typical text search in PDF documents, the image area is now ignored as it is not the region of interest, and is handled only during display related operations. It is sometimes in the interest of the user to search words within an image. One solution to the above problem is to use an Optical Character Recognizer (OCR) to convert the image data into text so that this text is available to the user to search. OCR indeed looks like a veritable solution and some applications (Adobe Acrobat 7.0 Professional) have used such techniques to search text inside images in a PDF file. This solution is better suited for languages that use the Latin Script. The fact that we have commercial OCRs for English with high accuracies makes the above approach possible. OCRs for Indian languages and other oriental scripts are not known for high accuracies and this makes the approach less effective for word search in PDF files that contain document images in Indian or oriental languages. Also the fact that Indian language text is non-standard (represented in custom fonts) makes even the textual search a difficult problem.

Recognition-Free Search Word level matching has been attempted for printed [10] documents. They are useful for locating similar occurrences. None of these matching schemes are designed to do partial matching, which is very important for addressing word-form variations. There have been successful attempts on locating a specific word in a handwritten text by matching image features for historical documents [11]. Word Spotting [13] is a technique wherein word images are matched using various image matching techniques. Dynamic Time Warping is a dynamic programming based procedure [11] to align two sequences of feature vectors. This can also provide a similarity measure. This is a popular technique in speech analysis and recognition. Indexing and retrieval from document image collections were studied by converting the images to text [35]. Success of these procedures depends on the performance of the OCRs, which convert the document images to text. Manmatha *et al.* [47] built a search engine for historical manuscript images, wherein the retrieval system was trained using an annotated set of 100 pages of George Washington's manuscripts and is used to query a dataset containing images from the same collection. The current approach is to use meta data or indices, which are manually created. This makes automatic approaches to searching and accessing the content very attractive. Another approach to such a problem is to use handwriting recognizers followed by a text search engine. However, in real life, the documents, especially the historical documents, are of poor quality which makes the handwriting recognizers vulnerable to poor results. Manmatha *et al.* [47] used an alternative approach bypassing explicit recognition. We [33] employed a similar methodology to retrieve printed document images using word matching techniques without recognition.

4.3.6 Recognition-Free Search in PDFs

We employ a similar idea for searching within the graphics stream of PDF files. This involves the following steps

- **Extraction of graphic streams**: Document images are extracted from the graphics stream of a PDF file in its entirety.
- **Word Segmentation**: The pre-processed image is then subjected to word-level segmentation and all the word images in a document image are then extracted.

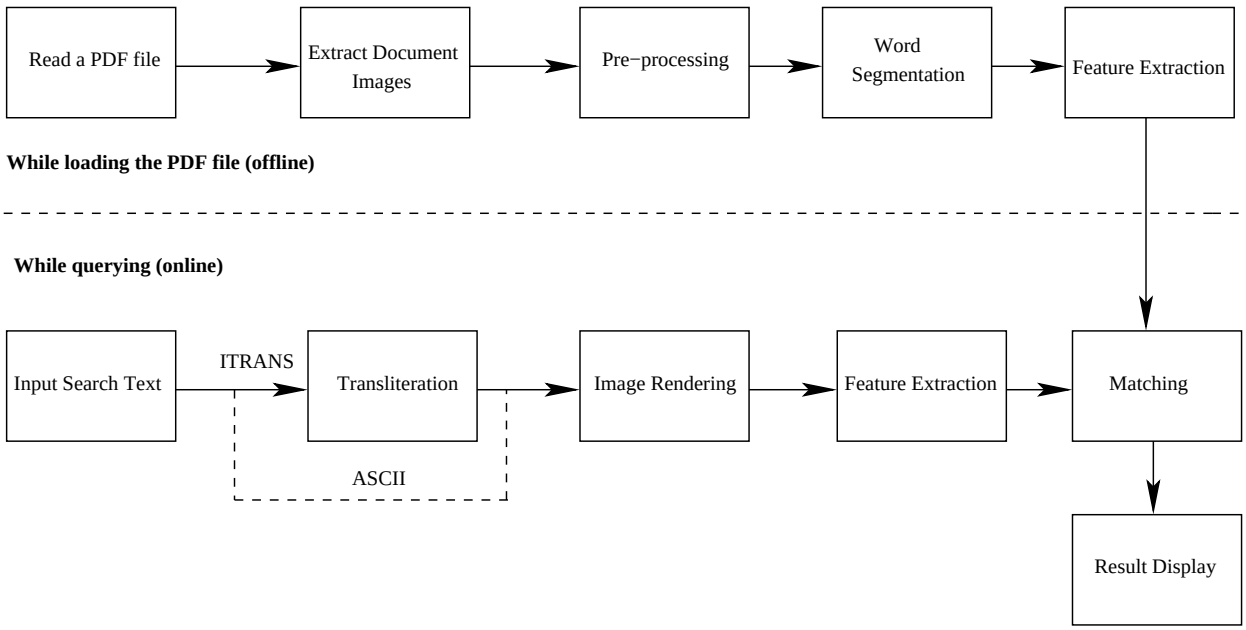


Figure 4.9: Block Diagram Illustrating our Approach

- **Feature Extraction**: Feature values are extracted from these segmented word images that are used for matching purposes.
- **Matching**: Word Image matching techniques are employed to match word images using their feature values.

4.3.7 Structure of the Implementation

The graphics stream of a PDF file has been only used for viewing purposes and thus making it inaccessible to search. We have used the existing text search framework in PDF files to locate and then search graphic stream objects (refer Figure 4.8). The entire process of locating the graphics stream objects in a PDF file and then searching and matching is explained below. As can be seen from Figure 4.9, there are two stages: the PDF file load stage and immediately after loading, the search stage, separated by the horizontal dashed lines at the middle. During the load stage, a PDF file is read as it is normally read. All images in the PDF file are then extracted in the *Read PDF file* module. In this routine we extract all images in its entirety (full size) and some additional information such as, its width, height, location in the *page* of a PDF document, and page number are extracted and stored in a separate data structure. Since every image in a PDF file need not be a document image, the *Extract Document Image* module determines whether the current image is a document image or not. If the current image is not a document image, then it is ignored while document images are subjected to further processing. The extracted document image then goes through the *Pre-processing* module that does various preprocessing operations like Skew-correction, Thresholding and Binarization. This pre-processed image is then sent to the *Word Segmentation* module that segments the pre-processed image into word images. These word images are then sent to the *Feature Extraction* routine that extracts feature values from the word images. We have used features [9] such as the horizontal profile, the vertical profile, and the background to ink transition. All these features are normalized so that variations due to font size are taken into account. This

process is repeated for all the document images in the PDF file. This entire offline process should not interfere with the PDF readers loading and displaying contents of a PDF file, which is very fast. Keeping this in mind, we fork out the entire offline activity in order to facilitate the user to query the text content until the features are extracted for all the document images in a PDF file.

During the search phase, the input text is processed along with the language information. There are two kinds of input, the ASCII and the ITRANS [42]. In the ASCII mode, the English text is handled (Latin scripts) as is. While in the ITRANS input mode, the user chooses a target Indian language in which he has to search. ITRANS is a transliteration scheme wherein the user types input search text in Roman characters while the output is in the particular Indian language. After determining the input mode (ASCII or ITRANS) and language, the input text is rendered as a word image. This word image is then subjected to *Feature Extraction* process. In the *Matching* stage, the features of the rendered input word image is matched with all the features of the word images that were extracted during the load process. We use the Dynamic Time Warping (DTW) [13] based matching technique to match the input word image with every other word image in the PDF file. Partial matching [9] is accommodated so that word form variations are also searched and taken care of. The resultant matches are restricted according to a threshold and the matched word images are grouped according to their page numbers. The display is also in reverse video, with appropriate conversion from the image coordinate space to the display coordinate space. The load process is initiated every time a user opens a new PDF file or updates the PDF document. This search is integrated within the text search (refer Figure 4.8), so that the search result is transparent to the user irrespective of whether the search result was from a Text stream or Image stream.

4.3.8 Open Source Implementation and Indian Language Support

We integrated our algorithm into an opensource PDF reader (Xpdf) [41]. Implementation integrates the textual and image (graphics) search in a transparent manner. Xpdf is designed to be small and efficient. The Xpdf source code implementation clearly separates out the front-end (User Interface) from its core which is the back-end. This independence of the User Interface from the core is of major advantage to users who are interested to extract text and images from a PDF file without having to view it. The User Interface has been developed using *Motif*, an X Windows based user interface builder in Linux, while the core implementations concerning the PDF file has been implemented using C++. All the above operations are achieved using Xpdf by appropriately adding and modifying code snippets into the Xpdf source code. The textual search operations are handled well within the Xpdf code, while our module of word image search is integrated within the text search module of the Xpdf source code. Integration process is explained in Figure 4.8. Indian language scripts have complex layout. That is why OCR for Indian languages do not claim of high accuracies similar to Latin scripts. Indian language content is often stored as images (graphics) in the PDF files. To search we need an input mechanism which can be compatible with the Roman script. To enter an Indian language text as UNICODE, ISCII or font is a cumbersome process. The alphabet set for Indian languages are typically high when compared to English. The presence of *Samyuktakshar* (compounded letter) in Indian languages makes the rendering process of the word images all the more difficult. It is very difficult to enter a search string for Indian languages following a specific font encoding for that particular language. This makes it contingent for the user to be well-versed with font encoding for every Indian language. This process is not intuitive and ITRANS [42] fits in as a perfect workaround. ITRANS is a transliteration scheme wherein the user enters the text in Roman such that the scheme is common across Indian languages. ITRANS text is then converted to UNICODE, and compatible fonts are used to render word images for the UNICODE text. Though there have been attempts for Indian language keyboard layout, known as

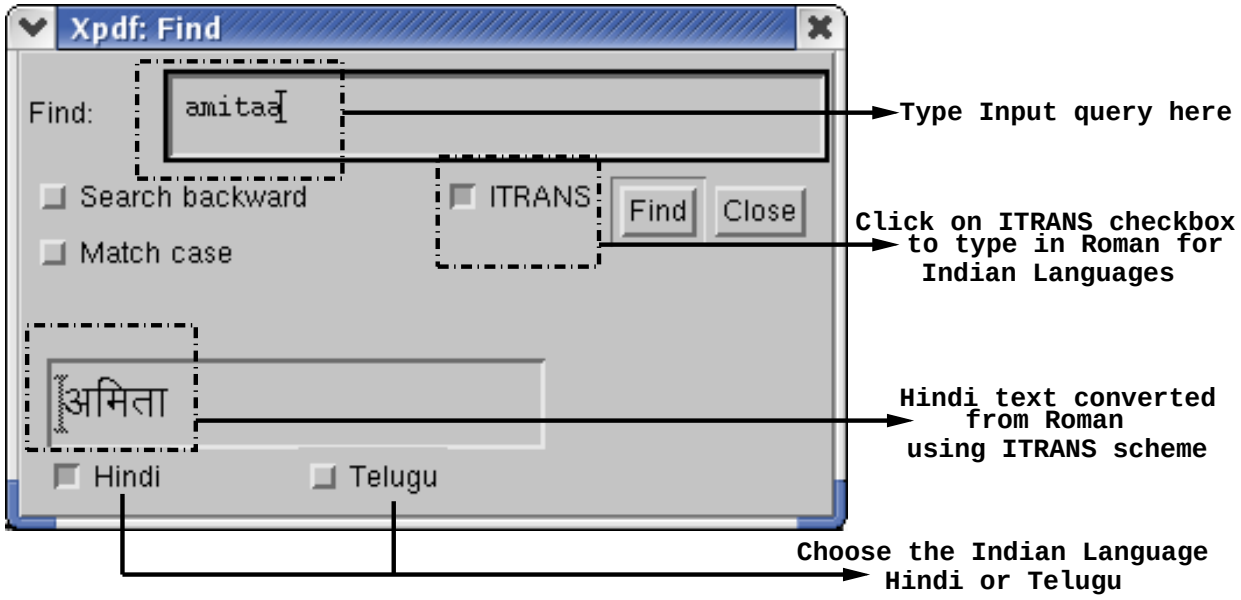


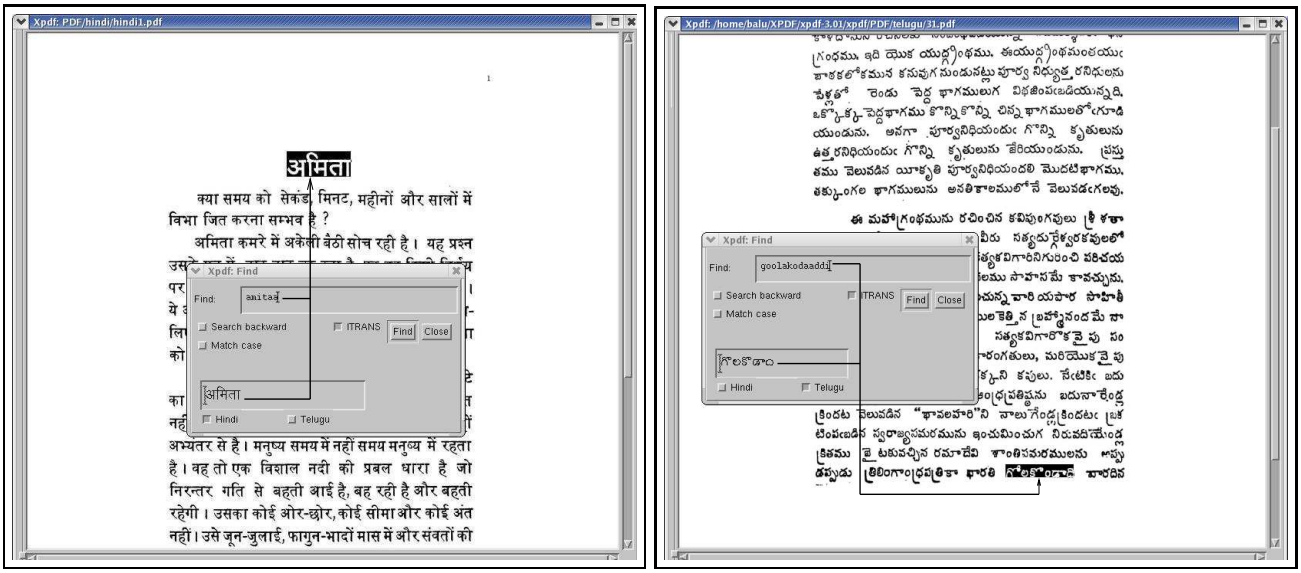
Figure 4.10: Screenshots of Find Dialog Box in Xpdf. User can Enter the Query Word in Roman (ITRANS), View the Script and See the Results in Reverse Video.

INSCRIPT, its support at the operating system level is not satisfactory. One also needs mapping information to convert the ITRANS text to a specific language UNICODE. Word image rendering is handled by an image rendering routine, which takes a font-encoded text, its font name, its size, and its color as input and then renders the word image.

Figure 4.10 shows the *Find* dialog box in Xpdf, which has been modified to show Indian language content depending on the language chosen (e.g.,: Hindi or Telugu). The user chooses the *ITRANS* check box in the *Find* dialog in order to search Indian language content. As the user keeps typing in the search *Text Box*, the corresponding text in the chosen Indian language (Hindi in the above example) will appear. As can be seen from Figure 4.10, the example shows the search word "amitaa" that was queried. Figure 4.11 (a) shows the results highlighted in the reverse video. One has to take note of the fact that the content displayed in the PDF is a document image, but not a textual (ASCII) content. The modified Xpdf code also contains facility to search in Telugu document images (Figure 4.11 (b)) other than Hindi. All that the user has to do is to select the Telugu check box in order to search Telugu PDF files. The converters have been written from ITRANS to a specific font. The same interface is used to search both inline text and word images. Essentially all the available and existing functionalities of Xpdf are preserved.

4.3.9 Discussions

Here we have demonstrated a word spotting based PDF search for document word images using an existing open source PDF viewer, Xpdf. We have effectively handled the issues arising out of Indian languages and have supported all the functionalities well within the existing source code of Xpdf and have demonstrated it for PDF files containing Hindi and Telugu document images. This solution can help the readers of Digital library by giving access to the textual content stored in the graphics stream. For other Indian languages, appropriate fonts and converter maps have to be added appropriately.



(a) (b)

Figure 4.11: Screenshots of Word Image Search results in Xpdf in Hindi and Telugu PDF Files.

Chapter 5

Conclusions

5.1 Summary and Conclusions

We have presented techniques for document annotation and retrieval. Our contributions are in building over a proof-of-concept approach in building scalable search systems and addressing Indian language issues involved in the entire process.

We proposed and demonstrated a search system for retrieval of relevant documents from large collection of document images. Our search is accomplished by matching word images without explicit recognition. This method of search will be important in using large digitized manuscript data sets in Indian languages. Our system is capable of searching across languages for retrieving relevant documents from multilingual document image database. Also, our retrieval mechanism for online handwriting can handle different writing styles and variations.

We proposed a model based framework for annotation of non-cursive online handwritten data when a parallel text corpus is present for the data. The approach employs a handwriting synthesis scheme that generates the handwritten equivalent of the transcription. An elastic matching is used to propagate the annotation from the synthesized words to the original handwritten words.

In lieu with our work for *printed* documents, we extended our work to online handwriting documents and proposed a writer-independent recognition-free approach for retrieval of handwritten data in Indian language scripts. The proposed approach uses handwriting synthesis to do matching in the ink domain as opposed to the use of a recognizer. The synthesis model that learns the handwriting of a writer from the trained samples produces natural looking words. The framework also incorporates information retrieval measures such as TF/IDF to rank retrieved documents and also supports multilingual queries, which is especially useful for Indian languages. The system for retrieval of relevant documents from large document image collections is developed by adapting an existing search engine.

5.2 Future Scope

This work opens up many interesting problems in document understanding in the Indian language context.

- The Model-based framework for annotation of online handwriting data can be extended for *printed* documents, taking care of font variations in order to create huge ground truth data for the OCRs.

- One direction is to Model the handwritten data as a mixture distribution, which will be able to incorporate more writing variations within the data of a single user.
- Another interesting direction that can be pursued is the study of the stroke shape variations when it is in the proximity of other strokes or when the position of the stroke in the word changes.
- Many other opensource image/PDF viewer applications can extend this word image search without explicit recognition.

Appendix

Appendix A

Fonts and Indian Language Specific Details

A.1 Fonts

What is a Font?

A font is a complete set of characters in a particular size and style of type. This includes the letter set, the number set, and all of the special character and diacritical marks you get by pressing the shift, option, or command/control keys.

Types of Fonts?

There are essentially two types of fonts.

- BITMAP FONTS: Also known as RASTER fonts. These typically contain a bitmap (image) for every character.
- OUTLINE FONTS: Also known as TRUE TYPE FONTS and the OPEN TYPE FONTS.

The advantage of using a Vector font (Outline Font) is enlisted below.

- Vector fonts implemented Vector Graphics, i.e. it has information on line segments and curves that have to be rendered.
- This gives a better scaling and small file size.
- Unlike Bitmap fonts , no image is stored, instead points are stored.

Difference between TTF and OTF?

TTF and OTF both are reffered to as TrueType fonts, with OTF also referred to as OpenType fonts. OTF are 16 bit fonts with extra space to accomodate more glyphs and have hidden hints for rendering certains set of glyphs.

8-bit and 16-bit fonts

TTF are generally 8-bit fonts while OTF are 16-bit fonts. Note that both of them have the same file extension (.ttf).

Type I and Type II fonts

Type I and Type II are fonts are specifically used by PDF generating applications. Their strucutre is little different from the TTF fonts and have different mechanism for rendering.

Unicode fonts ?

Most of the 16-bit fonts (OTF) are Unicode fonts. These fonts typically render fonts for Unicode (UTF8) range to which it belongs to. For example the Unicode font Raghindi (raghu.ttf) renders Hindi glyphs for Hindi (Devanagiri) Unicode range.

CDAC Fonts (ISFOC fonts)

CDAC Fonts, also popularly known as ISFOC (Indian Standard Font Coding), was an attempt to standardize Indian language fonts. They are TrueType (8 bit) fonts.

Installing Fonts

One of the easiest way to install fonts in Linux (Redhat 9 and above) is to create a directory ".fonts" in a user's home directory (typically /home/USER/.fonts) and copy all the TrueType font (*.ttf) files to that directory. This will make the fonts available only to the USER. In order to make the fonts available for use to all the users in that machine, do the following below

- Login as "root"
- cd /usr/X11R6/lib/X11/fonts/ (or /usr/share/fonts)
- One can create a directory inside the fonts directory so as to group all the related fonts. For example create a directory called "indic" (mkdir indic) and then copy all the *.ttf files into indic directory.
- open the "XF86Config" file (usually in /etc/X11/) and the following line in the FontPath. FontPath "/usr/X11R6/lib/X11/fonts/indic" (or add /usr/share/fonts)
- Next, in order to make use of the OpenType font you have, load the "freetype" module at startup. You can achieve this by adding the following line in the Module section of XF86Config file. Load "freetype"
- Type mkfontdir in the /usr/X11R6/lib/X11/fonts/indic directory and then type xset fp rehash
- Typically one might have to restart the X for the effects to take place.

In case you want to place your new Indic fonts in some other directory, you must use xset to add the new FontPath. Please see the xset man-page for further assistance. You can check the new installed fonts by running the xlsfonts command. In case you don't see any Indic fonts using this command, you may need to restart X.

How do I see Unicode fonts (UTF-8) in a Linux console

The user need to install LOCALE in order to view Indian language content in a Linux console. (*http://www.cse.iitk.ac.in/users/isciig/downloads/downloads.html#locales*) LOCALE support Unicode (UTF8). The user after the font instalaltion, has to set the envirnoment variables, typically the LANG variable. The default would like this LANG=en_US.UTF-8

For Hindi, the user has to set, `LANG=hi_IN.UTF-8`. From now on the "console" would display it in Hindi. This can be repeated for other Indian langugaes in a similar manner.

How to do it in Windows ?

Most of the Windows XP operating systems come with pre-installed Indian Language fonts (Unicode fonts). Optionally one might download fonts and would have to copy them into the c:
windows
fonts directory.

Fonts for Latex

Using TTF in latex so as to use fonts for Indian Languages
(Reference link: *http://www.radamir.com/tex/ttf-tex.htm*)

- Create a temporary directory (say, mkdir tmp) and copy the TTF file (say times.ttf) into it
- and Encoding file, T1-WGL4.enc there. (This file is availble in most of the Linux instalations typically)
- Creating TeX Font Metrics (tfm)
- Please download the ttf2tfm converter, as this utility will be used to convert the ttf files to tfm
- ttf2tfm times.ttf -q -T T1-WGL4.enc -v ectimes.vpl rectimes.tfm >> ttfonts.map
- This will create a tfm files of raw fonts and vpl files of virtual fonts. To generate slanted versions of regular and bold font, command should be extended like this: ttf2tfm times.ttf -q -T T1-WGL4.enc -s .167 -v ectimeso.vpl rectimeso.tfm ¿¿ ttfonts.map
- Creating Virtual Fonts (vf)
- vptovf ectimes.vpl ectimes.vf ectimes.tfm
- This creates vf and tfm files for virtual fonts. After that you can delete vpl files: rm *.vpl
- Locate the texmf directory (/usr/share/texmf/) and move all vf files to /usr/share/texmf/fonts/vf/times/. , and nd all tfm files to /usr/share/texmf/fonts/tfm/times/ (create times directory if it doesn't exist at both the places). The times directory is not mandatory, you can leave them in their parent directories.
- Now open the ttfonts.map in /usr/share/texmf/ttf2tfm/ and copy all the contents into it from the ttfonts.map that was created in the tmp directory.
- This is sufficient for generating DVI file, but to view or print it, there should be a raster (pk) font.

- Latex will attempt to generate it automatically with new utility: ttf2pk. For successful generation ttf2pk should find both TrueType font and encoding file. It happens that ttf2pk can find ttf files, if they are installed and are in a system fonts directory, and it looks for encoding files in a directory /usr/share/texmf/pdftex (among others). Therefore you have to copy T1-WGL4.enc to /usr/share/texmf/pdftex.
- Testing the installation in TeX
- tex testfont
- It will ask about a font to test. Enter "ectimes". It will ask about a command. Enter "
  table
  eject
  init". After all fonts are tested, enter a command "
  bye". This will create a file testfont.dvi in current directory.
- dvips testfont.dvi
- Using new fonts in TeX
- ```
  {\font\myfont=ectimes
  \font\mybigfont=ectimes at 36pt
  \myfont Hello, I am being typeset in Times New Roman
  \mybigfont Me too...}
  ```

Unicode in latex

Unicode can be used in latex by using a Unicode complaint font, and being subjected to the above process.

A.2 Encoding

What is encoding?

A set of characters (letters, logograms, digits, punctuation, symbols, control characters, and so on) that have been mapped to numeric values (called code points) that can be used by computers. The code points are assigned to the characters in the character set by applying an encoding method. Some examples of encodings are wlatin1, wcyrillic, and shift-jis. Below is a summary of the complete rendering process from a character to a glyph.

- Character is an entity used in a data exchange
- Glyph is a particular shape of a character
- Character set, ordered set of characters
- Font, ordered collection of glyphs
- Encoding, ordered collection
  ```

Different Types of Encoding

- ASCII - 7 bit
- ISO-8859-1(Extended Ascii) - 8 bit (Western Characters). Latin-1,2,3,..
- ISCII- 8 bit - Indian Script code for Information interchange
- EBCDIC - IBM (before ASCII)
- UNICODE - 16 bit (now up to 21 bits)
- UCS -32 bit (Universal Character set)

What is a character set?

These above encoding schemes are also referred to as Character set.

Unicode font for all Unicode values

Arial MS Unicode is one huge font that typically covers almost all the sripts in the Unicode range, and is single font that can display all the glyphs within the range of Unicode.

UTF-8?

Below is the UTF-8 sequence for the Unicode ranges given. The one represented by "x" are to be filled by the bits of the UNICODE value.

```
0x00000000 - 0x0000007F: 0xxxxxxx

0x00000080 - 0x000007FF: 110xxxxx 10xxxxxx

0x00000800 - 0x0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx

0x00010000 - 0x001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

0x00200000 - 0x03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

0x04000000 - 0x7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

The Unicode character copyright sign character

0xA9 = 1010 1001 is encoded in UTF-8 as: 11000010 10101001 = 0xC2 0xA9

"not equal" symbol character
0x2260 = 0010 0010 0110 0000
11100010 10001001 10100000 = 0xE2 0x89 0xA0

ISCII ?

INDIAN SCRIPT CODE FOR INFORMATION INTERCHANGE - ISCII. This Indian Standard was adopted by the Bureau of Indian Standards fter the draft finalied by the Computer Media Sectional Committee has been approved by the Electronics and Telecommunication Division Council. 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. There are 15 officially recognized languages in India: Hindi, Marathi, Sanskrit, Punjabi, Gujarati, Oriya, Bengali, Assamese, Telugu, Kannada, Malayalam, Tamil, Urdu, Sindhi and Kashmiri. Out of these, Urdu, Sindhi and Kashmiri are primarily written in Perso-Aabic scripts, but get written in Devanagari too (Singhi is also written in the Gujarati script). Apart from Perso-Arabic scripts, all the othe 10 scripts used for Indian languages have evolved from the ancient Brahmi script and have a common phonetic structure, making a common character set possible. The Northern scripts are Devanagari, Punjabi, Gujarati, Oriya, Bengali and Assamese, while the Southern script are Telugu, Kannada, Malayalam and Tamil. The official language of India, Hindi is written in the Devanagari script. Devanagari is also used for writing Marathi and Sanskrit. It is also the official script of Nepal. As Perso-Arabic scripts have a different alphabet, a different standard is envisaged for them. 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 (with diacritic marks for non-Devanagari alphabets) ha been used in the standard. For notational simplicity, elsewhere, the term Indian scripts implies Brahmi-based Indian scripts. Figure A.1 shows an ISCII chart for Devanagiri script.

Unicode

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. Table A.1 shows the unicode ranges for Indic Scripts.

A.3 INSCRIPT - Keyboards Layouts

Figure A.2 shows Inscript keyboard layout for Devanagiri (Hindi). Similarly keyboard layouts for other Indian scripts can be designed. Figure A.3 and Figure A.4 are for Telugu and Malayalam respectively.

A.4 Transliteration

Figure A.5 showd the transliteration table for ITRANS for Hindi. The same Roman notation can be extended to other Indian languages. Figure A.6 displays the vowels in some Indian languages

Hex	Dec	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex	Dec	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
0	0	NUL	DLE	SP	0	@	P	`	p				ओ	ढ	ऱ	े	EXT
1	1	SOH	DC1	!	1	A	Q	a	q			ॅ	औ	ण	ल	े	०
2	2	STX	DC2	"	2	B	R	b	r			ं	ऑ	त	ळ	ै	१
3	3	ETX	DC3	#	3	C	S	c	s			ः	क	थ	ळ	ॅ	२
4	4	EOT	DC4	$	4	D	T	d	t			अ	ख	द	व	ो	३
5	5	ENQ	NAK	%	5	E	U	e	u			आ	ग	ध	श	ो	४
6	6	ACK	SYN	&	6	F	V	f	v			इ	घ	न	ष	ौ	५
7	7	BEL	ETB	'	7	G	W	g	w			ई	ङ	ऩ	स	ॉ	६
8	8	BS	CAN	(	8	H	X	h	x			उ	च	प	ह	्	७
9	9	HT	EM	)	9	I	Y	i	y			ऊ	छ	फ	INV	़	८
A	10	LF	SUB	*	:	J	Z	j	z			ऋ	ज	ब	ा	।	९
B	11	VT	ESC	+	;	K	[	k	{			ऌ	झ	भ	ि		
C	12	FF	FS	,	<	L	\	l	|		ए	ञ	म	ी		
D	13	CR	GS	-	=	M	]	m	}			ऐ	ट	य	ु		
E	14	SO	RS	.	>	N	^	n	~			ऍ	ठ	र	ू		
F	15	SI	US	/	?	O	_	o	DEL			ऑ	ड	ऱ	ृ	ATR	

Figure A.1: ISCII chart for Devanagiri.

and Figure A.7 displays some consonants in some Indian languages for ITRANS transliterations scheme.

A.5 Font Converters

Font converters play an important role, especially when there are so many vendor-specific fonts available for Indian languages. Since there has been no specific or standard scheme which these vendors adhere to, Indian language text is available in plenty but in a non-standard form. Also there is a lot of Indian language content available in either ISCII and other encoding which are not viewable. In order to view this content one needs to convert ISCII to a font encoding. Also its intutive to type in english and deriable to view the Indian langauge content in a font encoding. This also requires a converter and we have developed custom font converters from ISCII and ITRANS encodings to CDAC-fonts such as Hemalatha (Telugu), Yogesh (Hindi) and Karthika (Malayalam).

This is a laborious activity as it involves mapping of every possible Akshara to a font glyph or group of font glyphs. The presence of Samyuktakshara (compound characters) also makes this mapping even more combersome, as typically these Samyuktaksharas run into several thousands and there are no general rules to map them in the font domain. Figure A.8 is a typical example of a font conveter from ITRANS to Unicode (Unicode compliant font for Devanagiri).

Script	Unicode Range	Major Languages
Devanagari	U+0900 to U+097F	Hindi,Marathi,Sanskrit
Bengali	U+0980 to U+09FF	Bengali,Assamese
Gurumukhi	U+0A00 to U+0A7F	Punjabi
Gujarati	U+0A80 to U+0AFF	Gujarati
Oriya	U+0B00 to U+0B7F	Oriya
Tamil	U+0B80 to U+0BFF	Tamil
Telugu	U+0C00 to U+0C7F	Telugu
Kannada	U+0C80 to U+0CFF	Kannada
Malayalam	U+0D00 to U+0D7F	Malayalam

Table A.1: Indic Scripts in Unicode

Figure A.2: INSCRIPT keyboard layout for Devanagiri.

A.6 Existing text editing tools

Some of the exisiting text editors that support Unicode are Yudit and Gedit in Linux, while for Windows XP MS word and Wordpad have in built support.

A.7 Browsers

Mozilla, Opera, Firefox all typically support Unicode. Mozilla and firefox have rendering issues for Unicode for Indian languages. When integrated with pango engine (a font rendering engine), the rendering issues are solved, though this adds additional burden on the CPU. On the other hand, Internet Explorer (IE) has the best support for Unicode in Windows XP.

A.8 Font based Image Libraries

Imagemagick is a standard library that is used for many Image related operations, such as read, write, and other image editing operations. It can also be used for rendering text images, given a

Figure A.3: INSCRIPT keyboard layout for Telugu.

Figure A.4: INSCRIPT keyboard layout for Malayalam.

font-encoded text and its font file to a specific size and thickness. *QT* can also be used for many Image related operations using the *QImage* and *QPixmap* class.

a	अ	k.h	क्	p	प	.D	ड़	
aa or A	आ or ा	k	क	ph	फ	.Dh	ढ़	
i	इ or ि	kh	ख	b	ब	1	१	
ii or I	ई or ी	g	ग	bh	भ	2	२	
u	उ or ु	gh	घ	m	म	3	३	
uu or U	ऊ or ू	~N	ङ	y	य	4	४	
R^i	ऋ or ृ	ch	च	r	र	5	५	
R^I	ॠ or ॄ	chh or Ch	छ	l	ल	6	६	
L^i	ऌ or ॢ	j	ज	v	व	7	७	
L^I	ॡ or ॣ	jh	झ	sh	श	8	८	
e	ए or ॅ	JN or ~n	ञ	shh or Sh	ष	9	९	
ai	ऐ or ै	T	ट	s	स	0	०	
o	ओ or ो	Th	ठ	h	ह	.n or M	ं	anusvāra
au	औ or ौ	D	ड	ld or L	ळ	.N	ँ	chandrabindu
a.n or aM	अं	Dh	ढ	x or kSh	क्ष	H	ः	visarga
aH	अः	N	ण	j~n or GY or dny		.	।	danda
a.c	ऍ	t	त	q	क़	R	ऽ	Marathi r
aa.c or A.c	ऑ	th	थ	K	ख़	.. or ;	॥	paragraph end
OM or AUM	ॐ	d	द	G	ग़	.a	ऽ	avagraha
.a	ऽ	dh	ध	z	ज़	.c	ॅ	a like in at
Ra	र	n	न	f	फ़	{\rm .}	.	period

Figure A.5: ITRANS Encoding Table.

name	itrans	dev-1	dev-2	ben-1	ben-2	guj	kan	tel	gur	tam	rom
A	a	अ	অ	অ	অ	અ	ಅ	అ	ਅ	அ	a
AA	aa	आ	আ	আ	আ	આ	ಆ	ఆ	ਆ	ஆ	ā
I	i	इ	ই	ই	ই	ઇ	ಇ	ఇ	ਇ	இ	i
II	ii	ई	ঈ	ঈ	ঈ	ઈ	ಈ	ఈ	ਈ	ஈ	ī
U	u	उ	উ	উ	উ	ઉ	ಉ	ఉ	ਉ	உ	u
UU	uu	ऊ	ঊ	ঊ	ঊ	ઊ	ಊ	ఊ	ਊ	ஊ	ū
V. R	RRi	ऋ	ঋ	ঋ	ঋ	ઋ	ಋ	ఋ	–	ரு	ṛ
V. RR	RRI	ॠ	–	–	ৠ	–	ೠ	ౠ	–	–	ṝ
V. L	LLi	ऌ	–	–	–	–	ಌ	ఌ	–	–	ḷ
V. LL	LLI	ॡ	–	–	–	–	ೡ	ౡ	–	–	ḹ
E	e	ए	এ	এ	এ	એ	ಎ	ఎ	ਏ	எ	e
EE	E	–	–	–	–	–	ಏ	ఏ	–	ஏ	E
AI	ai	ऐ	ঐ	ঐ	ঐ	ઐ	ಐ	ఐ	ਐ	ஐ	ai
O	o	ओ	ও	ও	ও	ઓ	ಒ	ఒ	ਓ	ஒ	o
OO	O	–	–	–	–	–	ಓ	ఓ	–	ஓ	O
AU	au	औ	ঔ	ঔ	ঔ	ઔ	ಔ	ఔ	ਔ	ஔ	au
VISARGA	aH	ः	ঃ	ঃ	ঃ	ઃ	ಃ	ః	ਃ	◌ஃ	ḥ

Figure A.6: ITRANS Encoding Table for vowels for some Indian languages.

name	itrans	dev-1	dev-2	ben-1	ben-2	guj	kan	tel	gur	tam	rom
KA	k*a*	क	क	ক	ক	ક	ಕ	క	ਕ	க	ka
KHA	kh*a*	ख	ख	খ	খ	ખ	ಖ	ఖ	ਖ	க	kha
GA	g*a*	ग	ग	গ	গ	ગ	ಗ	గ	ਗ	க	ga
GHA	gh*a*	घ	घ	ঘ	ঘ	ઘ	ಘ	ఘ	ਘ	க	gha
NGA	~N*a*	ङ	ङ	ঙ	ঙ	ઙ	ಙ	ఙ	ਙ	ங	ṅa
CA	ch*a*	च	च	চ	চ	ચ	ಚ	చ	ਚ	ச	ca
CHA	chh*a*	छ	छ	ছ	ছ	છ	ಛ	ఛ	ਛ	ச	cha
JA	j*a*	ज	ज	জ	জ	જ	ಜ	జ	ਜ	ஜ	ja
JHA	jh*a*	झ	झ	ঝ	ঝ	ઝ	ಝ	ఝ	ਝ	ஜ	jha
NYA	~n*a*	ञ	ञ	ঞ	ঞ	–	ಞ	ఞ	ਞ	ஞ	ña
TTA	T*a*	ट	ट	ট	ট	ટ	ಟ	ట	ਟ	ட	ṭa
TTHA	Th*a*	ठ	ठ	ঠ	ঠ	ઠ	ಠ	ఠ	ਠ	ட	ṭha
DDA	D*a*	ड	ड	ড	ড	ડ	ಡ	డ	ਡ	ட	ḍa
DDHA	Dh*a*	ढ	ढ	ঢ	ঢ	ઢ	ಢ	ఢ	ਢ	ட	ḍha
NNA	N*a*	ण	ण	ণ	ণ	ણ	ಣ	ణ	ਣ	ண	ṇa
TA	t*a*	त	त	ত	ত	ત	ತ	త	ਤ	த	ta
THA	th*a*	थ	थ	থ	থ	થ	ಥ	థ	ਥ	த	tha
DA	d*a*	द	द	দ	দ	દ	ದ	ద	ਦ	த	da
DHA	dh*a*	ध	ध	ধ	ধ	ધ	ಧ	ధ	ਧ	த	dha
NA	n*a*	न	न	ন	ন	ન	ನ	న	ਨ	ந	na
PA	p*a*	प	प	প	প	પ	ಪ	ప	ਪ	ப	pa
PHA	ph*a*	फ	फ	ফ	ফ	ફ	ಫ	ఫ	ਫ	ப	pha
BA	b*a*	ब	ब	ব	ব	બ	ಬ	బ	ਬ	ப	ba
BHA	bh*a*	भ	भ	ভ	ভ	ભ	ಭ	భ	ਭ	–	bha
MA	m*a*	म	म	ম	ম	મ	ಮ	మ	ਮ	ம	ma

Figure A.7: ITRANS Encoding Table for vowels for some Indian languages.

bai बै
bau बौ
be बे
bi बि
bii बी
bo बो
bu बु
buu बू
bh भ्
bh.h भ्
bhA भा
bhA.c भॉ
bhI [redacted]

Figure A.8: Converter map file from ITRANS to Unicode Devanagiri.

Appendix B

Details of the PDF format

B.1 The PDF format

Origins of the PDF [39] dates back as early as the nineties, and at that time PostScript [40] page description language was the widely accepted standard for printing purposes.

PDF was built on top of the PostScript, so that not only it supported printing but also supported viewing capability. Today, PDF is the *de facto* standard for electronic exchange of documents, and also an industry standard for intermediate representation of printed material. The goal for developing PDF is to enable users exchange and view electronic documents easily and reliably, independently of the environment in which they were created.

PDF *document* is a collection of *objects*. These *objects* can be located in a PDF file in any arbitrary order, but are connected with each other by a reference mechanism. Therefore, a viewer applications should process a PDF file by following references from *object* to *object*, instead of processing *object* sequentially. Hence, every PDF file contains a *cross-reference table*, stored at the end of the file. Cross-reference makes sure that a PDF file containing very large number of documents can be accessed efficiently with almost no time constraints.

In addition, PDF also includes objects, such as annotation and hypertext links, and can also contain interactive form fields which all are useful for interactive viewing and document interchange. These annotations are useful for storing applications specific content. PDF serves purpose for fundamentally two kinds of applications, the Producer (PDF generator) and Consumer (PDF reader). Today you will find many open-source PDF viewers (Consumer applications) available, and XPDF is one among the many popular PDF viewers in the open-source community. PDF file is a sequence of 8-bit bytes. It is advised to create PDF file as a binary one instead of text, to avoid end-of-line and other custom conventions that are operating system dependent.

Interestingly, PDF allows modifications to be appended to a file. This makes sure that, any new changes to the current PDF need not require the whole document to be reprocessed or saved again, saving us valuable time. This is a useful feature, if your PDF file typically runs into thousands of pages, and to add or modify a particular page shouldn't warrant for the whole PDF to be saved from the first page to the last page. So essentially the time consumed is more or less the time one spends on editing or modifying the PDF document. Below shows a very basic example of a PDF file.

```
%PDF-1.5
%âãÏÓ
%EOF
```

PDF file contains binary data, and it is recommended that the header line be immediately followed by comment line containing at least four binary characters (ASCII code greater than 127) so as to ensure proper behaviour of file transfer applications that inspect the data in the near beginning of a file to determine whether to treat the file's content as text or binary as shown in the second line of the above example. Comments in a PDF file starts with %, while comments other than the two (PDF-1.5 and EOF) in the above example have no semantics.

B.2 PDF Syntax

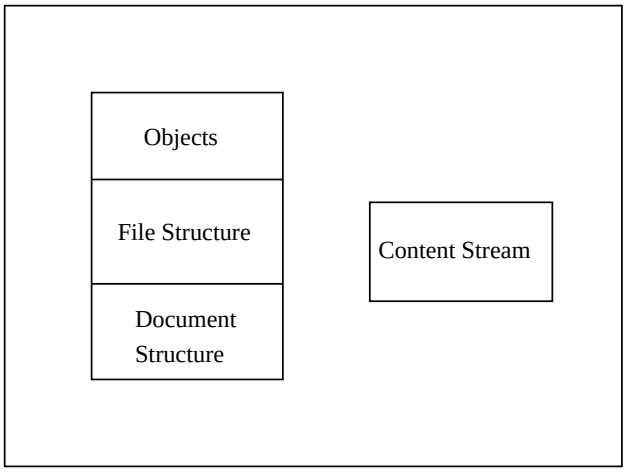


Figure B.1: Components of PDF.

Figure B.1 shows the various components that make up a PDF file.

Objects

A PDF document is a data structure essentially made of Objects. PDF implicitly has a close resemblance to a programming language paradigm, though not in entirety as PostScript. PDF supports eight boolean objects; Boolean values, Integer and real numbers, Strings, Names, Arrays, Dictionaries, Streams and the Null Object. Objects may be labeled so that they can be referred by other objects. A labeled object is called an indirect object. This gives the object a unique identifier by which other objects refer to it. The definition of an Object in a PDF file consists of its object number and generation number, followed by the values of the object itself within braces between the keywords obj and endobj.

```
12 0 obj
(IIIT Hyderabad)
endobj
```

Here the object number is 12, with generation number 0 with value "IIIT Hyderabad". This Object can be referred to from elsewhere from the file as shown below.

```
12 0 R
```

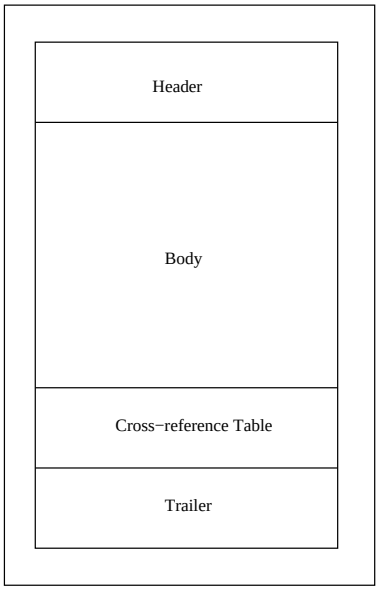


Figure B.2: Structure of PDF.

Figure B.2 shows the PDF file structure. A PDF file consists of the following:

- A one-line header identifying the version of the PDF specification to which the file conforms
- A body containing objects that make up the PDF
- A Cross-reference table containing references to indirect object
- A trailer giving the location of the cross-reference table and certain other special objects within the body of the while

Cross-reference table Cross-reference table contains information that permits random access to indirect objects within the file, so that entire file need not be read to locate any particular object. The table contains a one line entry for each indirect object, specifying the location of that object within the body of the file. The cross-reference table is the only part of a PDF file with a fixed format. The table contains one or more *cross reference sections* that begin with the keyword *xref*. Following it are one or more cross-reference subsections which may appear in any order. This subsection is useful for incremental updates. So for a file that has never been updated, the cross-reference section contains only one subsection whose object numbering begins at 0. The subsection begins with a line containing two numbers separated by a space. Example,

```
29 3
```

which means that it is a subsection that contains three objects, numbered consecutively from 29 to 31. Following this line is the cross-reference entries themselves, one per line and exactly 20 bytes long that includes the end-of-line marker.

Example:

```
nnnnnnnnnn ggggg n eol
```

where,

nnnnnnnnnn is a 10-digit byte offset
ggggg is a 5-digit generation number
n is a literal keyword identifying this as an in-use entry
eol is a 2-character end-of-line sequence

The cross-reference entry for a free object has the keyword f instead of n.
nnnnnnnnnn ggggg f eol

The byte offset is a 10-digit number, padded with leading zeros if necessary, giving the number of bytes from the beginning of the file to the beginning of the object.

Example:

```
xref
0 6
0000000003 65535 f
0000000017 00000 n
0000000081 00000 n
0000000000 00007 f
0000000331 00000 n
0000000409 00000 n
```

The above example shows an example of a single subsection with six entries starting from 0. Four of them are in use (objects number 1,2,4,and 5) and two of them are free (objects number 0 and 3). Object number 3 has been deleted, and the next object created with that object number will be given a generation number of 7.

The below example shows a cross-reference section with four subsections,containing a total of five entries. The first subsection contains one entry, for object number 0, which is free. The second subsection contains one entry,for object number 3, which is in use. The third subsection contains two entries,for objects number 23 and 24, both of which are in use. Object number 23 has been reused, as can be seen from the fact that it has a generation number of 2. The fourth subsection contains one entry, for object number 30, which is in use.

```
xref
0 1
0000000000 65535 f
3 1
0000025325 00000 n
23 2
0000025518 00002 n
0000025635 00000 n
30 1
0000025777 00000 n
```

Trailer Trailer section quickly enables a PDF to locate the cross-reference table and certain special objects.

```
trailer
$$<<$$ key1 value1
   key2 value2
   ...
   keyn valuen
$$>>$$
startxref
{Byte\_offset\_of\_last\_cross-reference\_section}
%%EOF
```

Here the *startxref* keyword is preceded by the trailer dictionary consisting of the keyword *trailer* followed by a series of key-value pairs enclosed within << and >>. The line just before the EOF marker, has the byte offset of the last cross-reference section from the beginning of the file until the xref keyword of the last cross-reference section.

Incremental Updates The contents of the PDF can be updated incrementally, without rewriting the entire file. Changes are appended to the end of file leaving its original content intact. The main advantage being that the contents of large file can be updated quickly.

Every time there are new objects added or changes, a cross-reference section is added, and trailer is inserted. Deleted objects are left unchanged in the file, but are marked as free via their cross-reference entries. The added trailer contains all the entries from the previous trailer, as well as a *Prev* entry giving the location of the previous cross-reference section. Also note that each trailer is terminated by its own end-of-file (%%EOF) marker.

That is why, applications must read a PDF file from the end.

Document Structure

A PDF document can be regarded as a hierarchy of objects contained in the body section of a PDF file. At the root of the hierarchy is the document's catalog dictionary. Most of the objects in the hierarchy are dictionaries. For example, each page of the document is represented by a *page object*, a dictionary that includes references to the page's contents. The individual page objects are tied together in a structure called *Page Tree*.

Page Tree

The pages of a document are accessed through a structure known as the page tree, which defines their ordering within the document. The tree structure allows PDF viewer applications to quickly open a document containing thousands of pages using only limited memory. The tree contains nodes of two types–intermediate nodes, called page tree nodes, and leaf nodes, called page objects.

Page Objects

The leaves of a page tree are page objects - each of which is dictionary specifying the attributes of a single page of a document.

Entries in a page object Type is *Page* for a Page Object. *Page Tree Node* is its immediate Parent. MediaBox is of type *Rectangle*, defining the boundaries of the physical medium on which the page is intended to be displayed or printed , while CropBox is also of the type *Rectangle*, wherein the contents of a page are clipped to this rectangle useful for output medium for use in some implementation defined manner.

Thumb is of the type *stream* and used to display the thumbnail image for a page. Dur is of the type *number* and is used as duration time in seconds in presentations, where a particular page will displayed, while "Trans" is used for transition effects and PZ is the page's preferred zoom.

Content Streams

They are the primary means of describing the appearance of pages and graphical objects. A content stream depends on information contained in an associated resource dictionary; in combination, these two objects form a self-contained entity. This section describes these objects.

A content stream is a PDF stream object whose data consists of a sequence of instructions describing the graphical elements to be painted on a page.The instructions are represented in the form of PDF objects,using the same object syntax as in the rest of the PDF document. However, whereas the document as a whole is a static, random-access data structure, the objects in the content stream are intended to be interpreted and acted upon sequentially.

Each page of a document is represented by one or more content streams. Content streams are also used to package up sequences of instructions as self-contained graphical elements, such as forms, patterns, certain fonts, and annotation appearances A content stream, after decoding with any specified filters,is interpreted according to the PDF syntax rules. It consists of PDF objects denoting operands and operators.

The operands needed by an operator precede it in the stream. An operand is a direct object belonging to any of the basic PDF data types except a stream. Dictionaries are permitted as operands only by certain specific operators. Indirect objects and object references are not permitted at all.

An operator is a PDF keyword that specifies some action to be performed, such as painting a graphical shape on the page. An operator keyword is distinguished from a name object by the absence of an initial slash character (/).Operators are meaningful only inside a content stream.

Operand and Operators

Operand is a direct PDF data type (except streams , Object references and indirect objects)

Operator specifies some action to be performed, such as painting a graphical element onto the page. Operators assume role within a content stream.

B.3 Graphics

Graphic state operators includes a *Current Transformation Matrix* (CTM), which maps a user space coordinates used within a PDF content stream into output device coordinates.

Path Construction and Painting

Paths define shapes, trajectories, and regions of all sorts. They are used to draw lines, define the shapes of filled areas,and specify boundaries for clipping other graphics. The graphics state includes

a current clipping path that defines the clip- ping boundary for the current page. At the beginning of each page, the clipping path is initialized to include the entire page.

A path is composed of straight and curved line segments, which may connect to one another or may be disconnected. A pair of segments are said to connect only if they are defined consecutively, with the second segment starting where the first one ends. Thus the order in which the segments of a path are defined is significant. A path is made up of one or more disconnected subpaths, each comprising a sequence of connected segments. The topology of the path is unrestricted, i.e. it may be concave or convex,may contain multiple subpaths representing disjoint areas, and may intersect itself in arbitrary ways. There is an operator, h, that explicitly connects the end of a subpath back to its starting point;such a subpath is said to be closed. A subpath that has not been explicitly closed is open.

This also includes Path construction and painting operators.

1. Path construction operators define the geometry of a path.A path is constructed by sequentially applying one or more of these operators.

2. Path-painting operators end a path object, usually causing the object to be painted on the current page in any of a variety of ways.

3. Clipping path operators,invoked immediately prior to a path painting operator, cause the path object also to be used for clipping of subsequent graphics objects.

In content stream, the operands and operators are written sequentially in postfix notation.

Coordinate system

This defines the canvas on which all the painting occurs. Coordinate space encompasses

1. Location of Origin

2. The Orientation of X and Y axes

3. The length of the units along each axis

If the coordinates in a PDF file were specified in a *device space*, the file would be device dependent and would be rendered differently on different environments. *User Space* is a device independent space. Here the length of unit along both the X and Y axes in 1/72 inch. (also a point, term widely used in printing industry).

1. The transformation from user space to device space is defined by Current Transformation Matrix (CTM). The coordinates of text are specified in *text space* and the transformation is through a text matrix (text positioning operators)

2. Character glyphs in a font are defined by the *glyph space*. Glyph space to text space conversion is defined by font matrix.

3. All sampled images are defined by *image space*. Transformation from image space to user space is predefined and cannot by changed.

4. Similarly we have, *Form space* for XObject and form matrix for form space to user space conversion, and *Pattern Space* for patterns

Graphics state

An internal data structure that hold the current graphics state or graphics controlled parameters. The graphics state is initialized at the beginning of each, using default values of graphic state parameters that are device independent. A well-structured PDF document typically contains many graphical elements that are essentially independent of each other and sometimes nested to multiple levels. The *graphics state stack* allows these elements to make local changes to the graphics state without disturbing the graphics state of the surrounding environment.

The q operator pushes the copy of the entire graphics state onto the stack while the Q operator restores the entire the graphics state to its former value by popping it from the stack.

Images

PDF's painting operators include general facilities for dealing with sampled images, which is a two dimensional array for values representing the color information. In PDF, images can be specified by an XObject or as an inline image (generally for very small images).

To paint an image, four interrelated items must be specified:

1. the format of image - number of coloumns, number rows and number color components per sample.
2. the sample data constituting the image
3. user space and image internal coordinate space.
4. Color component mapping

All these are specified by an image XObject or an inline image. Sample data, is represented by a stream of bytes, interpreted as 8-bit unsigned integers in the range 0-255. Image Coordinate system is known as the image space. Here Coordinate origin (0,0) is at the upper left corner. The below example shows an XObject based declaration.

```
/Type /XObject
/Subtype /Image
/Width
/Height
```

Inline Images are for small images (typically 4KB or less). It is delimited in the content stream by operators BI (begin image), ID (image data), and EI (end image). Below example shows an inline declaration.

```
BI
/W 17
/H 17
/CS /RGB
ID
EI
```

B.4 Text

A Glyph is a graphical shape and is subject to all graphical manipulations. (eg Coordinate Transforms)

How Glyphs from Fonts are painted on the page ?

A Character is an abstract symbol, while glyph is a specific graphical rendering of a character. Glyphs are organized into fonts. Fonts define a glyphs for a particular character set; like Helvetica and Times for the Latin Character set.

The below example illustrates the most straightforward use of a font. It places the text ABC 10 inches from the bottom of the page and 4 inches from the left edge,using 12-point Helvetica.

```
BT
/F13 12 Tf
288 720 Td
(ABC)Tj
ET
```

BT means, Begin a Text Object. Tf operator specifies the name of a font resource (identified by F13). Here 12 is the font size. Td operator adjusts the current text position. Tj operator takes a string operand and paints the corresponding glyphs using the current font and other text related parameters present in the graphics state. ET meaning, End the Text Object.

```
/Resources
$$<<$$/Font $$<<$$/F13 23 0 R $$>>$$
$$>>$$
23 0 obj
$$<<$$/Type /Font
/Subtype /Type1
/BaseFont /Helvetica
$$>>$$
endobj
```

The above is a sample Font Resource which specifies Font related parameters.

Default colour is black, but other color specific operator can be used for font colors and other graphic effects.

Tr is for rendering mode that outlines whether the glyph boundaries are to be filled, stroked or other such effects.

Glyph Positioning and Metrics

A glyph's width, is the amount of space it occupies along baseline of a line of text that is written horizontally. In some fonts, the width is constant, i.e all glyphs have the same width, and these are called fixed-pitch or monospaced. Fonts having variable glyph width are known as proportional or variable-pitch fonts. The Tj operator positions the consecutive glyphs according to the width operator (or the widths of the glyphs)

Transformation from a Glyph space to a text space is specified by a *Font Matrix*. The Glyph displacement is the distance from the glyph's origin to the point at which the origin of the next glyph should normally be placed when painting consecutive glyphs.

Writing mode instructs whether the text has to be written in a Horizontal(mode 0) or Vertical manner. Spacing adjustments between glyphs based on context.

Text State Parameters, such as

1. Tc is used for character spacing,
2. Tw Word Spacing
3. Th Horizontal scaling
4. Tmode Text rendering mode
5. Tf text font
6. Tfs text font size

Text Rise (Trise) parameters are useful for superscript and subscript text placement.

Text Space is the coordinate system, in which the text is shown, defined by the *Text Matrix* Tm and text state parameters Tfs, Tf and Trise that together determine the transformation from text space to user space. Text Positioning operators include Td, Tm and Text Showing operator includes Tj.

Multiple bytes can represent a single character. The Code and lengths and mappings from code to glyphs are defined in a data structure called *CMap*. Type 0 are called Composite fonts, and others are called simple fonts.

Composite fonts are consists of multiple bytes, that refer to a set of glyphs of a single character.

B.5 PDF and PostScript Language

1. Unlike PostScript, PDF is not a full-scalable programming language
2. Difference
   (a) Contains Font information for viewing facility
   (b) PDF enforces a strictly defined file structure that can be accessed in arbitrary order
   (c) Can contain additional information such hyperlinks etc. for interactive viewing
3. Though the conversion is not direct into PS format, but can be achievable.

Appendix C

Details of UNIPEN and UPX

C.1 Unipen Representation

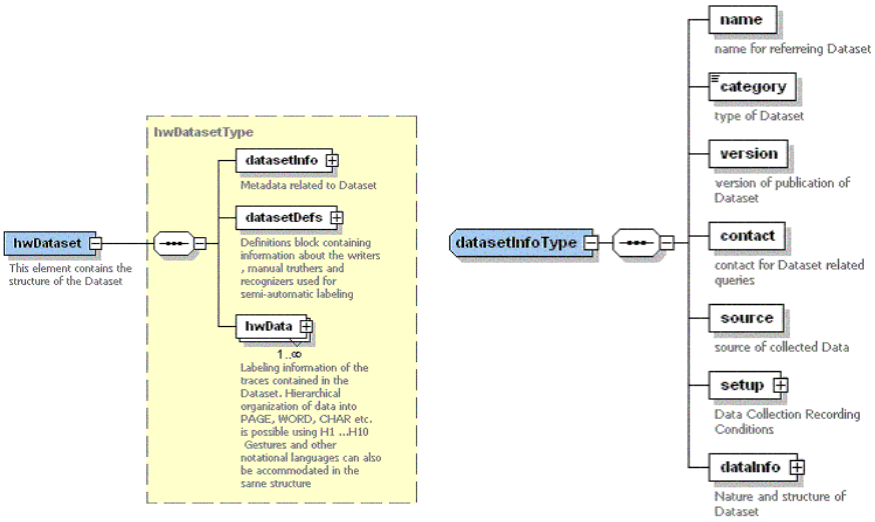
Below shows a sample Unipen file. The stroke (ink) information is stored between in .PEN_DOWN and .PEN_UP.

```
.VERSION                0.6
.DATA_SOURCE            ATT
.DATA_ID                SAMPLE
.COMMENT                ####################
### INFORMATION ###
####################
.DATA_CONTACT
Name: Dept 11359 - Adaptive Systems Research
Affiliation: AT&T Bell Laboratories
Address: 101 Crawfords Corner Rd
Holmdel, NJ, 07733 , USA
Phone: 1 (908) 949-2783 (secty.)
Fax: 1 (908) 949-7722
Tech Contacts: Don Henderson    908-949-4591
Isabelle Guyon 415-442-5937
.DATA_INFO
Alphabet: Latin, uppercase, lowercase, punct
Lexicon: The text from "Alice In Wonderland"
Quantity: very small sample
Quality: ?
Distribution: Given by .LEXICAL_FREQ
Number of writer(s): 1
Writing style: Natural - no constraints....
Segmentation: Words written in boxes in
upper area of form. Last line on
the form is a unseg'd sentence.
.SETUP
Site: AT&T Bell Laboratories
Holmdel cafeteria, New Jersey, USA
Time:
```

```
Conditions: Supervised recording conditions
People sitting at a desk
Writers: Volunteer staff members
Form layout: 1 form containing 1 segmented
word area (1-12 words) and a second
area having a sentence made up of these
words.
.PAD
Machine name: WACOM HD-648A LCD Digitizer &
386 PC
Display: Backlit LCD

Matrix 640 x 480 points
VGA standard, black on white, monochrome
Sensor: Electromagnetic resonance sensor
Pen: Untethered pen, tip switch and
side button
Driver: PC driver developed in house..
.COMMENT                ####################
### DATA LAYOUT ###
####################
.X_DIM                  4160
.Y_DIM                  3200
.H_LINE                 450 1900 2300
.COMMENT                ####################
### UNIT SYSTEM ###
####################
.X_POINTS_PER_INCH      508
.Y_POINTS_PER_INCH      508
.POINTS_PER_SECOND      200
.COMMENT                ####################
### DECLARATIONS ###
####################
.COORD                  XY
.HIERARCHY              PAGE TEXT_CHUNK WORD
=============== End file ATT.DOC ==================
=============== File ATT.DAT ======================
.COMMENT   Most of the point have been
removed to shorten the
example.
.INCLUDE                ATT.DOC
.DATE                   9 20 93
.WRITER_ID              08171408_14804
.STYLE                  MIXED
.START_BOX
.SEGMENT PAGE           0-58
.SEGMENT TEXT_CHUNK     0-29 ?
"that nothing more happened ,"
```

```
.SEGMENT WORD            0-6 ? "that"
.PEN_DOWN
707 2417
707 2424
590 2319
.PEN_UP
.DT 151
.PEN_DOWN
588 2377
586 2377
695 2393
.PEN_UP
.DT 231
.PEN_DOWN
772 2456
771 2456
745 2343
.PEN_UP
.DT 231
.PEN_DOW
827 2384
826 2384
827 2340
.PEN_UP
.DT 201
.PEN_DOWN
818 2362
819 2362
871 2361
.PEN_UP
.DT 231
.PEN_DOWN
929 2411
929 2411
902 2340
.PEN_UP
.DT 151
.PEN_DOWN
882 2370
882 2370
995 2384
.PEN_UP
.SEGMENT WORD 7-15 ? "nothing"
.PEN_DOWN
1778 2366
1778 2366
1778 2366
.PEN_UP
```



(a) (b)

Figure C.1: (a) The document root element (hwDataset) and its sub-elements (b) datasetInfo element capturing metadata about the dataset

C.2 UPX Representation

Our representation for annotated datasets of handwriting is called hwDataset, and it includes several elements for detailed annotation of handwriting, which extend the basic traceRefGroup element of the core InkML. The hwDataset element is the root of the XML document and captures meta-data about the dataset under datasetInfo, various definitions as part of datasetDefs, and hierarchical annotation of handwritten data under hwData (Fig C.1(a)). These elements are described briefly in the following paragraphs.

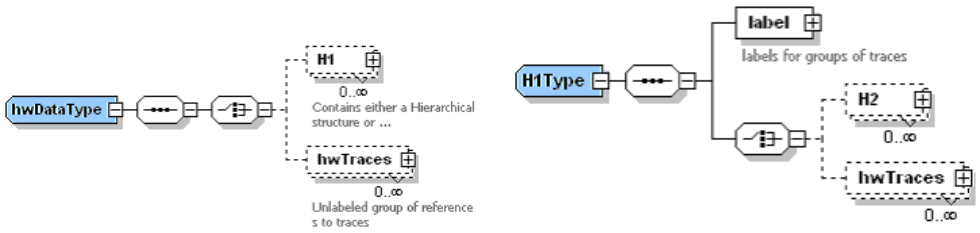
DatasetInfo: The datasetInfo element (Fig C.1(b)) captures metadata related to the dataset as a whole. It contains the following elements: (a) *name* - name for referring to the dataset (b) *category* - type of dataset (c) *version* - version number and/or datestamp of publication (d) *contact* - contact info for dataset-related queries (e) *source* - source of collected data (f) *setup* - physical conditions in which data was collected (g) *dataInfo* - information about the data

The dataInfo element in turn contains the following sub-elements: (a) *script* - language/script captured in dataset (b) *quality* - quality of handwritten data captured in dataset (c) *truth* - truth of what is captured (d) *methodology* - design of data and collection procedure (e) *annotationScheme* - description of annotation scheme

datasetDefs The datasetDefs element (Fig C.3(a)) captures information about different writers and sources of labels (annotation) represented in the dataset, and provides the means for referring to them later in the document. It contains the following elements:

- writerDefs - declarations of writers as a sequence of writer elements
- labelSrcDefs - declarations of sources of annotation as a sequence of labelSrc elements

The writer element in turn contains the following elements: (a)*date* - date when writing occurred (meant as a coarse description as opposed to the trace timestamps in the core InkML) (b) *personal-*



(a) (b)

Figure C.2: (a) hwData element capturing annotation (b) Annotation hierarchy

(a) (b)

Figure C.3: (a) datasetDefs Element capturing dataset definitions (b) label element with alternates

personal information including (c) *hand* - left/right handedness (d) *gender* - gender (e)*age* - age at the time of capture (f) *skill* - level of skill with script (g) *style* - predominant writing style (h) *region* - native region

The labelSrcDefs element contains the following elements: (a) *name* - name of the human or automated source of labels (b) *source* - organization that this label source represents (c) *time* - date and time of annotation (d) *contact* - contact details of label source (e) *labelTypes* is an attribute and describes the categories of labels generated by the source (e.g. truth, quality, script, style, etc) and their character encoding (e.g. Unicode).

The above elements provide a mechanism for representing the writing of different writers in the same dataset, as well as multiple sources and categories of annotation for the same handwritten data. An algorithm for script identification might be used as a source of script labels, while a human annotator may provide labels for truth as well as script, style and quality of writing. Of course, the representation can also accommodate multiple label sources for the same category of label information, e.g. one or more recognition engines for truth labels and a human annotator for their validation.

hwData The hwData element allows hierarchical organization of annotation. It typically contains the root of the annotation hierarchy defined by the user, denoted by the element $H1$ (Fig C.2(a)). Each level of hierarchy $H(i)$ contains a label element that captures annotation information at that level. $H(i)$ also contains either one or more $H(i+1)$ elements or hwTraces, the leaf element of the hierarchy that refers to raw ink traces represented using InkML (Fig C.2 (b)).

The $H(n)$ elements are meant to be used to indicate the hierarchical structure of handwriting, and assigned meaningful names such as PARAGRAPH and WORD using the corresponding attributes of the hwData element.

The label element (Fig C.3(b)) at each level can be used to capture alternative choices of label with confidence values if any, and the time of annotation. Although primarily intended to describe

the truth value of a particular set of ink traces, it may also be used for describing other characteristics such as writing style, quality and script. The timestamp can be used to generate the history of annotation spanning different label sources of a particular unit of writing. The alternates can be used to facilitate the process of manual validation by prompting options for human validation.

Formally, the attributes of label are (a) *id* - identification of label (b) *labelSrcRef* - reference to label source defined earlier. This holds good for sub-levels of the current level except where explicitly overridden (c) *category* - category of label (e.g. truth, quality, script, style, etc) (d) *timestamp* - time when the act of annotation is performed.

A sample UPX file

```
<hwDataset schemaVersion="0.5">
<datasetInfo id="ID0" datasetId="1">
<name>Thanigai</name>
<category>Skilled</category>
<version pubDate="2006-01-21">1.0</version>
<contact>HP Bangalaore</contact>
<source>HP Bangalaore</source>
<setup>JNU Data</setup>
<dataInfo>
<contentDesc>Hindi Word</contentDesc>
<numWriters>1</numWriters>
</dataInfo>
</datasetInfo>

<datasetDefs>
<writerDefs id="ID1">
<writer id="ID2">
<personal>
<hand>r</hand>
<educationLevel>high school</educationLevel>
<gender>m</gender>
<profession>Student</profession>
<region>Bangalore</region>
<dateofBirth>1977-08-05</dateofBirth>
</personal>
<skillDevice>average</skillDevice>
<skillScript script="Hindi" native="false">
<style>discrete</style>
<usageFreq>everyday</usageFreq>
<proficiency>average</proficiency>
</skillScript>
</writer>
</writerDefs>
<labelSrcDefs id="ID3">
<labelSrc id="ID4" type="human">
<name>Balu</name>
```

```
<source>HP Bangalaore</source>
<contact>IIIT</contact>
<desc>Hindi Word level data</desc>
<labelTypes>
<labelType encoding="unicode">truth</labelType>
</labelTypes>
</labelSrc>
</labelSrcDefs>
<annotationDefs id="ID5">
<annotationScheme id="ID6">
<annotationLevel rank="1" name="WORD">WORD definition </annotationLevel>
<annotationLevel rank="2" name="CHARACTER">CHARACTER definition </annotationLevel>
</annotationScheme>
</annotationDefs>
</datasetDefs>

<hwData id="0" annotationSchemeRef="/descendant::annotationScheme[attribute::id=ID6]">

<H1 id="ID7" writerRef="/descendant::writer[attribute::id=ID2]">
<label id="ID8" labelSrcRef="/descendant::labelSrc[attribute::id=ID4]"
labelType="truth" timestamp="2006-04-13T11:03:52.0Z">
<alternate rank="1" score="100">abhyaas</alternate>
</label>
<H2 id="ID9" writerRef="/descendant::writer[attribute::id=ID2]">
<label id="ID10" labelSrcRef="/descendant::labelSrc[attribute::id=ID4]"
labelType="truth" timestamp="2006-04-13T11:03:58.0Z">
<alternate rank="1" score="100">a</alternate>
</label>
<hwTraces>
<traceref>/descendant::trace[attribute::id=ID0]</traceref>
<traceref>/descendant::trace[attribute::id=ID1]</traceref>
<traceref>/descendant::trace[attribute::id=ID2]</traceref>
<traceref>/descendant::trace[attribute::id=ID3]</traceref>
</hwTraces>
</H2>
<H2 id="ID11" writerRef="/descendant::writer[attribute::id=ID2]">
<label id="ID12" labelSrcRef="/descendant::labelSrc[attribute::id=ID4]"
labelType="truth" timestamp="2006-04-13T11:04:00.0Z">
<alternate rank="1" score="100">bh</alternate>
</label>
<hwTraces>
<traceref>/descendant::trace[attribute::id=ID4]</traceref>
</hwTraces>
</H2>
<H2 id="ID13" writerRef="/descendant::writer[attribute::id=ID2]">
<label id="ID14" labelSrcRef="/descendant::labelSrc[attribute::id=ID4]"
labelType="truth" timestamp="2006-04-13T11:04:02.0Z">
<alternate rank="1" score="100">ya</alternate>
```

```
</label>
<hwTraces>
<traceref>/descendant::trace[attribute::id=ID5]</traceref>
<traceref>/descendant::trace[attribute::id=ID6]</traceref>
<traceref>/descendant::trace[attribute::id=ID7]</traceref>
</hwTraces>
</H2>
<H2 id="ID15" writerRef="/descendant::writer[attribute::id=ID2]">
<label id="ID16" labelSrcRef="/descendant::labelSrc[attribute::id=ID4]"
labelType="truth" timestamp="2006-04-13T11:04:04.0Z">
<alternate rank="1" score="100">sa</alternate>
</label>
<hwTraces>
<traceref>/descendant::trace[attribute::id=ID8]</traceref>
<traceref>/descendant::trace[attribute::id=ID9]</traceref>
</hwTraces>
</H2>
<H2 id="ID17" writerRef="/descendant::writer[attribute::id=ID2]">
<label id="ID18" labelSrcRef="/descendant::labelSrc[attribute::id=ID4]"
labelType="truth" timestamp="2006-04-13T11:04:07.0Z">
<alternate rank="1" score="100">---</alternate>
</label>
<hwTraces>
<traceref>/descendant::trace[attribute::id=ID10]</traceref>
</hwTraces>
</H2>
</H1>
<uiInfo yDim="0" xDim="0" yOrigin="Text" xOrigin="Text">
<hLines>0</hLines>
<vLines>0</vLines>
</uiInfo>

</hwData>

</hwDataset>
```

Sample InkML file

```
<ink>
<traceFormat id="ID00">
<regularChannels>
<channel name="X" type="decimal" />
<channel name="Y" type="decimal" />
<channel name="T" type="decimal" />
</regularChannels>
</traceFormat>
<captureDevice id="ID01" manufacturer="" model="pad" sampleRate="sampleRate" uniform="" latency="
```

```
 <channelList id="ID02">
 <channelDef id="ID03">
 </channelDef>
 </channelList>
</captureDevice>

<trace id="ID1">
783 707 0
783 707 0
783 707 0
689 1199 0
658 1187 0
637 1178 0
</trace>

<trace id="ID2">
1023 1044 0
1026 1044 0
1038 1040 0
1059 1037 0
1092 1034 0
1132 1028 0
</trace>
<trace id="ID3">
1402 748 0
1399 741 0
1390 1047 0
1393 1084 0
1393 1118 0
</trace>
</ink>
```

Figure C.4 displays the Hindi word "abhyaas" in the tool for the above UPX files. The Hindi word present in the UPX files is "abhyaas", annotated in ITRANS and displayed in Unicode as shown in Figure C.4.

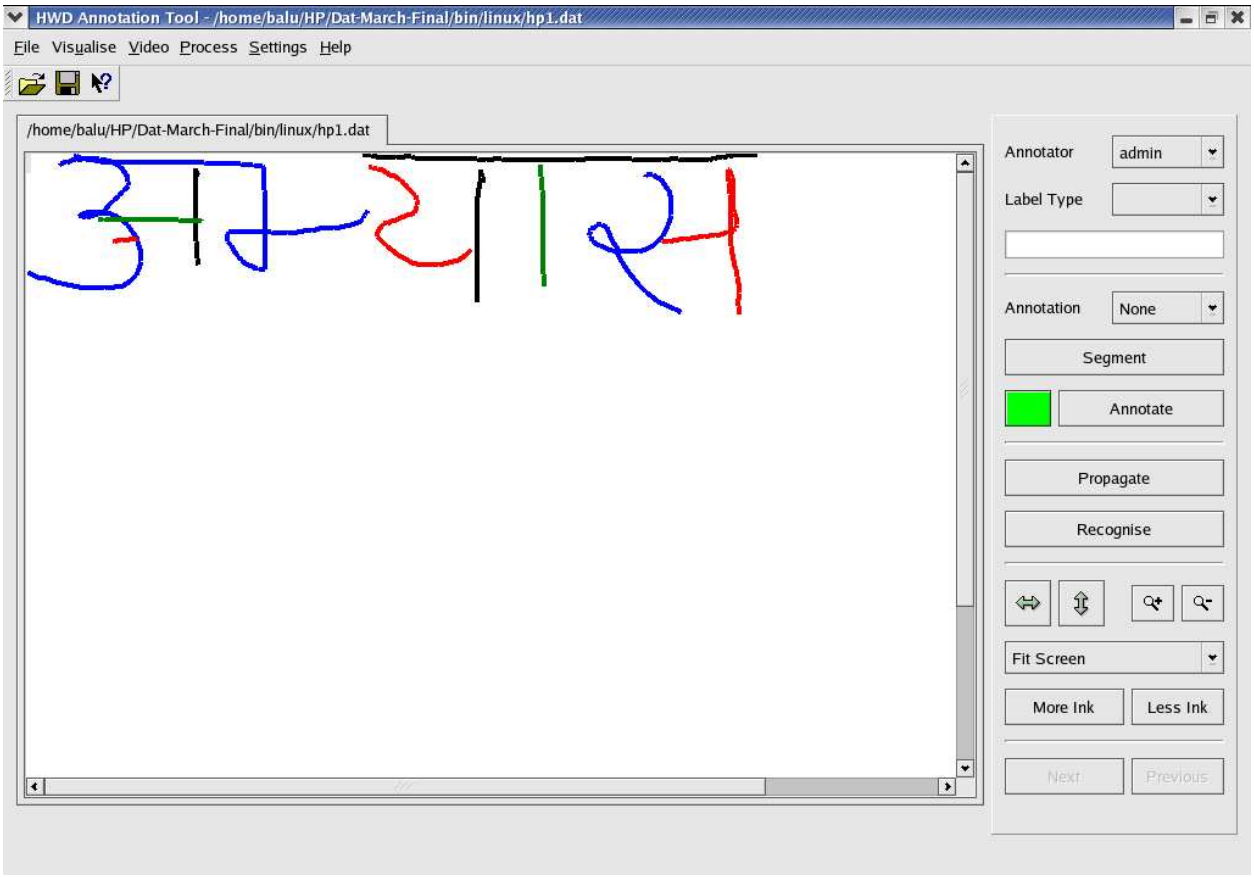


Figure C.4: Toolkit displaying the Hindi word *abhyaas*

Appendix D

Greenstone, an Open Source Search Engine

D.1 Greensotne

Greenstone is a suite of software for building and distributing digital library collections. It provides a new way of organizing information and publishing it on the Internet or on CD-ROM. Greenstone is produced by the New Zealand Digital Library Project at the University of Waikato, and developed and distributed in cooperation with UNESCO and the Human Info NGO. It is open-source software, available from *http://greenstone.org* under the terms of the GNU General Public License.

D.2 Overview of Greenstone

Greenstone is a comprehensive system for constructing and presenting collections of thousands or millions of documents, including text, images, audio and video.

Collections

A typical digital library built with Greenstone will contain many collections, individually organized though they bear a strong family resemblance. Easily maintained, collections can be augmented and rebuilt automatically. There are several ways to find information in most Greenstone collections. For example, you can search for particular words that appear in the text, or within a section of a document. You can browse documents by title: just click on a book to read it. You can browse documents by subject. Subjects are represented by bookshelves: just click on a bookshelf to look at the books. Where appropriate, documents come complete with a table of contents: you can click on a chapter or subsection to open it, expand the full table of contents, or expand the full document into your browser window (useful for printing). The New Zealand Digital Library website (nzdl.org) provides numerous example collections.

On the front page of each collection is a statement of its purpose and coverage, and an explanation of how the collection is organized. Most collections can be accessed by both searching and browsing. When searching, the Greenstone software looks through the entire text of all documents in the collection (this is called full-text search). In most collections the user can choose between indexes built from different parts of the documents. Some collections have an index of full documents, an index of paragraphs, and an index of titles, each of which can be searched for particular words or phrases. Using these you can find all documents that contain a particular set of words (the words

may be scattered far and wide throughout the document), or all paragraphs that contain the set of words (which must all appear in the same paragraph), or all documents whose titles contain the words (the words must all appear in the document's title). There might be other indexes, perhaps an index of sections, and an index of section headings. Browsing involves lists that the user can examine: lists of authors, lists of titles, lists of dates, hierarchical classification structures, and so on. Different collections offer different browsing facilities.

Finding information

Greenstone constructs full-text indexes from the document text that is, indexes that enable searching on any words in the full text of the document. Indexes can be searched for particular words, combinations of words, or phrases, and results are ordered according to how relevant they are to the query. In most collections, descriptive data such as author, title, date, keywords, and so on, is associated with each document. This information is called metadata. Many document collections also contain full-text indexes of certain kinds of metadata. For example, many collections have a searchable index of document titles. Users can browse interactively around lists, and hierarchical structures, that are generated from the metadata that is associated with each document in the collection. Metadata forms the raw material for browsing. It must be provided explicitly or be derivable automatically from the documents themselves. Different collections offer different searching and browsing facilities. Indexes for both searching and browsing are constructed during a "building" process, according to information in a collection configuration file.

Greenstone creates all index structures automatically from the documents and suppporting files: nothing is done manually. If new documents in the same format become available, they can be merged into the collection automatically. Indeed, for many collections this is done by processes that awake regularly, scout for new material, and rebuild the indexes all without manual intervention.

Document formats

Source documents come in a variety of formats, and are converted into a standard XML form for indexing by plugins. Plugins distributed with Greenstone process plain text, HTML, WORD and PDF documents, and Usenet and E-mail messages. New ones can be written for different document types (to do this you need to study the Greenstone Digital Library Developer's Guide). To build browsing structures from metadata, an analogous scheme of classifiers is used. These create browsing indexes of various kinds: scrollable lists, alphabetic selectors, dates, and arbitrary hierarchies. Again, Greenstone programmers can create new browsing structures.

Multimedia and multilingual documents

Collections can contain text, pictures, audio and video. Non-textual material is either linked into the textual documents or accompanied by textual descriptions (such as figure captions) to allow full-text searching and browsing. Unicode, which is a standard scheme for representing the character sets used in the world's languages, is used throughout Greenstone. This allows any language to be processed and displayed in a consistent manner. Collections have been built containing Arabic, Chinese, English, French, Mori and Spanish. Multilingual collections embody automatic language recognition, and the interface is available in all the above languages (and more).

Distributing Greenstone

Collections are accessed over the Internet or published, in precisely the same form, on a self-installing Windows CD-ROM. Compression is used to compact the text and indexes. A Corba protocol supports distributed collections and graphical query interfaces. The New Zealand Digital Library (nzdl.org) provides many example collections, including historical documents, humanitarian and development information, technical reports and bibliographies, literary works, and magazines. Being open source, Greenstone is readily extensible, and benefits from the inclusion of GNU-licensed modules for full-text retrieval, database management, and text extraction from proprietary document formats. Only through international cooperative efforts will digital library software become sufficiently comprehensive to meet the world's needs with the richness and flexibility that users deserve.

D.3 Understanding the collection-building process

End users of Greenstone can build collections using the Collector. This makes it very easy to build collections modelled after existing ones but with new content. However, it is not really feasible to use the Collector to create collections with completely new structures. It does invite you to edit the collection configuration file, which governs the collection's structure, but you need to know quite a lot about Greenstone to make radical yet effective changes. This section tells you what you need to know to do this. It also describes the Greenstone directory structure and the format in which documents are stored internally. We assume throughout this manual that you have installed Greenstone on your computer, be it Windows or Unix. If you have not yet done this you should consult the Greenstone Digital Library Installer's Guide. The name GSDLHOME is used throughout to refer to the Greenstone home directory, which is called $GSDLHOME on Unix ones. You set this directory during the installation procedure.

Building collections from the command line

the commands typed to produced the bhgita collection (a BhagvadGita collection) are:

```
cd ~/gsdl # assuming default Greenstone in home directory
source setup.bash # if you're running the BASH shell
source setup.csh # if you're running the C shell
mkcol.pl creator balu@students.iiit.net dlpeople
cd $GSDLHOME/collect/bhgita
mount  /cdrom # assuming this is where CD-ROM is mapped to
cp -r /cdrom/collect/bhgita/* import/
umount /cdrom
import.pl bhgita
buildcol.pl bhgita
rm -r index/*
mv building/* index
```

Configuration file

```
creator balu@students.iiit.net
maintainer balu@students.iiit.net
```

```
public true
beta true
indexes document:text
defaultindex document:text
plugin ZIPPlug
plugin GAPlug
plugin TEXTPlug
plugin HTMLPlug
plugin EMAILPlug
plugin ArcPlug
plugin RecPlug
classify AZList -metadata "Title"
collectionmeta collectionname "dlpeople"
collectionmeta iconcollection ""
collectionmeta collectionextra ""
collectionmeta .document:text "documents"
```

A sample meta file

```
<?xml version="1.0" ?>
<!DOCTYPE GreenstoneArchive SYSTEM
"http://greenstone.org/dtd/GreenstoneArchive/1.0/GreenstoneArchive.dtd">
<Section>
<Description>
<Metadata name="gsdlsourcefilename">ec158e.txt</Metadata>
<Metadata name="Title">Freshwater Resources in Arid Lands</Metadata>
<Metadata name="Identifier">HASH0158f56086efffe592636058</Metadata>
<Metadata name="gsdlassocfile">cover.jpg:image/jpeg:</Metadata>
<Metadata name="gsdlassocfile">p07a.png:image/png:</Metadata>
</Description>
<Section>
<Description>
<Metadata name="Title">Preface</Metadata>
</Description>
<Content>
This is the text of the preface
</Content>
</Section>
<Section>
<Description>
<Metadata name="Title">First and only chapter</Metadata>
</Description>
<Section>
<Description>
<Metadata name="Title">Part 1</Metadata>
</Description>
<Content>
```

```
This is the first part of the first and only chapter
</Content>
</Section>
<Section>
<Description>
<Metadata name="Title">Part 2</Metadata>
</Description>
<Content>
This is the second part of the first and only chapter
</Content>
</Section>
</Section>
</Section>
```

Appendix E

Word Spotting and Searching

E.1 Overview of Word Spotting

Text has always been the primary source of information in conventional, and more recently digital libraries. If the text is in machine readable form (ASCII), it can be indexed using standard text retrieval engines. However, much of the text in both historical and even current collections is contained in paper documents. One solution is to use Optical Character Recognition (OCR) to convert scanned paper documents into ASCII. Existing OCR technology works well with standard machine printed fonts. It works poorly if the documents are of poor quality or if the text is handwritten. Indexing and Searching such documents thus becomes difficult. Word Spotting idea provides an alternative approach. Word Spotting was first applied in the context of indexing handwriting data [47] and it is analogous to the word spotting in the speech processing. Word spotting is a content-based information retrieval task to find relevant words within a repository of scanned document images. The retrieval of matching words finds applications in several areas like forensics, historical documents, personal records, etc. In these applications it is of interest to retrieve words from a large database of documents based on the visual and the textual content.

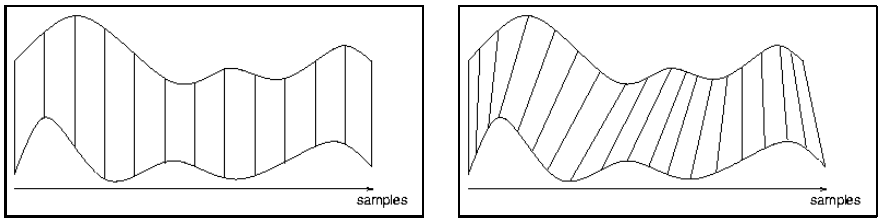
The word spotting technique involves the segmentation of each document into its corresponding lines and then into words. Each document is indexed by the visual image features of its words. Below outlines the a word spotting procedure.

1. A scanned greylevel image of the document is obtained.
2. The graylevel image is then binarized by thresholding the image.
3. The binary image is then segmented into words.
4. A given word image (i.e. the image of a word) is used as a template, and matched against all the other word images. This is repeated for every word in the document.
5. Matching is done against the features extracted from the word images.

E.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is used to compute a distance between two time series. A time series is a list of samples taken from a signal, ordered by the time the respective samples were obtained. A naive approach to calculating a matching distance between two time series could be to resample one of them and then compare the series sample-by-sample. The drawback of this method

is that it does not produce intuitive results, as it compares samples that might not correspond well (Figure E.1 (a)). Dynamic Time Warping solves this discrepancy between intuition and calculated matching distance by recovering optimal alignments between sample points in the two time series. The alignment is optimal in the sense that it minimizes a cumulative distance measure consisting of "local" distances between aligned samples. Figure E.1 (b) shows such an alignment. The procedure is called Time Warping because it warps the time axes of the two time series in such a way that corresponding samples appear at the same location on a common time axis. DTW has been widely used in speech processing, bio-informatics, and also in the online handwriting communities to match the 1D signal.



(a) (b)

Figure E.1: Alignment of two time series using (a) linear stretching and (b) Dynamic Time Warping.

The DTW-distance between two time series $x_1, x_2, \ldots, x_M$ and $y_1, y_2, \ldots, y_N$ is $D(M, N)$, which is calculated using dynamic programming is given by:

$$D(i,j) = min \begin{cases} D(i-1, j-1) \\ D(i, j-1) \\ D(i-1, j) \end{cases} + \quad d(x_i, y_j)$$

where, $d(i, j)$ is the cost in matching the ith element of X with jth element of Y and is computed using a weighted Euclidean distance. Using the given three values $D(i, j-1)$, $D(i-1, j)$ and $D(i-1, j-1)$ in the calculation of $D(i, j)$ realizes a local continuity constraint, which ensures no samples left out in time warping. Backtracking along the minimum cost index pairs (i, j) starting from (M, N) yields the DTW warping path. We use the Sakoe-Chiba band constraint [90] to ensure this path stays close to the diagonal of the matrix which contains the D(i, j).

A worked out example

Let us consider the problem of comparing two strings. For example, how do we compare "exercise" and "exirsais"? First of all, they are not the same. They might sound similar, but we are talking about how they are written.

As can be seen, Figure E.2 shows the DTW path for the words *exercise* and *exirsais*. The local dsitance here is a simple string comaprision operator, i.e., whenever there is a character level match, the edit ditance ($d(i, j)$) will be zero, and if there is no match the distance is given a value a one. Similarly for word images, the feature values are compared using Eucledian edit distance.

Below we sumamrize some of the characterisitics of the DTW:

- Dynamic time warping (DTW) has been suggested as a technique to allow more robust distance calculations, however it is computationally expensive.
- Dynamic time warping (DTW) is the favoured technique for non-linear time alignment.
- DTW uses dynamic programming to find the optimal alignment of two sequences of data of different lengths.

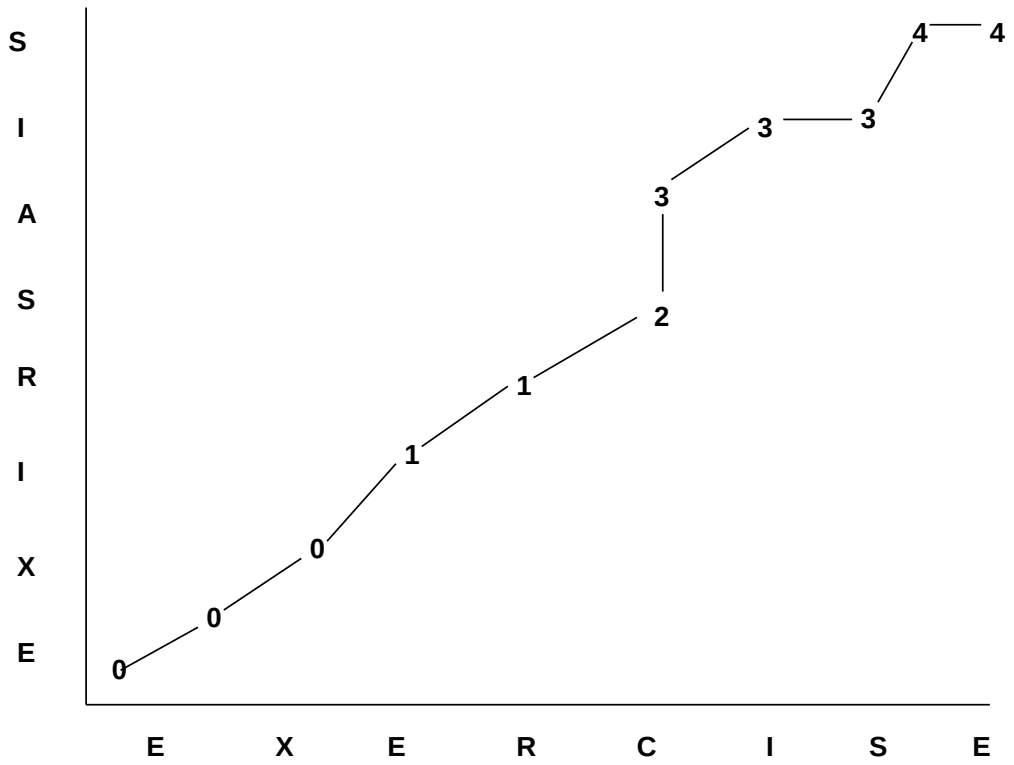


Figure E.2: A DTW example for the words *exercise* and *exirsais*.

- The DTW algorithm used at the word level allows us to handle spurious or missing strokes in a word.
- It also allows us to do partial matching of words.
- DTW can also handle variations, such as the font-style for a word image.

E.3 Word Image Matching

Representation of Word Images

Building an appropriate description is critical to the robustness of the system against signal noise. Generic content-based image retrieval systems use colour, shape or/and texture features for characterising the content. In the case of document images, features can be more specific to the domain as they contain image-description of the textual content in it. We find that many of the popular structural features work well for good quality documents. Word images, particularly from newspapers and old books, are of extremely poor quality. Common problems in such document database will have to be analysed before identifying the relevant features.

Popular artifacts in printed document images include (a) Excessive dusty noise, (b) Large ink-blobs joining disjoint characters or components, (c) Vertical cuts due to folding of the paper, (d) Cuts in arbitrary direction due to paper quality or foreign material, (e) Degradation of printed text due to the poor quality of paper and ink, (f) Floating ink from facing pages. An effective representation of the word images will have to take care of these artifacts for successful indexing and retrieval.

We found that three categories of features are effective to address these artifacts. The features could be a sequential representation of the word shape or a structural representation of the word image.

Word Profiles: Profiles of the word provide a coarse way of representing a word image for matching. Upper word, lower word, projection, density and ink-to-background transition profiles are used here for word representation. Upper and lower word profiles capture part of the outlining shape of a word, while projection and transition profiles capture the distribution of ink along one of the two dimensions in a word image.

Structural Features: Structural features of the words are used to match two words based on image similarities. Features employed in image processing literature for shape recognition are adopted for this. Normalised moments, such as first-order moments (M_{00}, M_{01}), central moment (CM_{01}), and statistical moments (mean, standard deviaition, skew) are employed in this work for describing the structure of the word. For artifacts like pepper and salt noise, structural features are found to be very robust.

Transformed Domain Representations: A compact representation of a series of observations (such as profiles) is based on Fourier Transform. Fewer set of coefficients are enough to represent a word robustly in a transformed domain, and these coefficients are matched at a coarse level for recognition. We used five predominant Fourier coefficients for the word representation.

E.4 Matching and Grouping of Words

For proper search, we need to identify the similar words, group them and evaluate the relative importance of each of these words and word clusters. For the indexing process, we identify the similar word set by clustering them into different groups based on similarities between words. Distance or dissimilarity between words are computed using the features discussed in the previous section. For this, we use a simple Euclidean distance.

In this paper, we find the similarity of words based on two components:

- A sequence alignment score computed using a Dynamic Time Warping (DTW) procedure.
- Structural similarity of word images by comparing the shapes.

The use of the total cost of Dynamic Time Warping as a distance measure is helpful to cluster together word images that are related to their root word by partial match as explained in the next section.

Dynamic Time Warping is a dynamic programming based procedure [11] to align two sequences of signals. This can also provide a similarity measure. This is a popular technique in speech analysis and recognition.

Let the word images (say their profiles) are represented as a sequence of vectors $\mathcal{F} = \mathbf{F_1}, \mathbf{F_2}, \ldots \mathbf{F_M}$ and $\mathcal{G} = \mathbf{G_1}, \mathbf{G_2}, \ldots, \mathbf{G_N}$. The DTW-cost between these two sequences is $D(M, N)$, which is calculated using dynamic programming is given by:

$$D(i,j) = \min \begin{cases} D(i-1, j-1) \\ D(i, j-1) \\ D(i-1, j) \end{cases} + d(i,j)$$

where, $d(i,j)$ is the cost in aligning the ith element of $\mathbf{F}$ with jth element of $\mathbf{G}$ and is computed using a simple squared Euclidean distance:

$$d(i,j) = \sum_{k=1}^{N} (F(i,k) - G(j,k))^2$$

Using the given three values $D(i,j-1), D(i-1,j)$ and $D(i-1,j-1)$ in the calculation of $D(i,j)$ realizes a local continuity constraint, which ensures no samples left out in time warping. As shown with red line in Figure 4.2 we also imposed global constraint using Sakoe - Chiba band [90] so as to ensure the maximum steepness or flatness of the DTW path. Score for matching the two sequences $\mathcal{F}$ and $\mathcal{G}$ is considered as $D(M,N)$, where M and N are the lengths of the two sequences.

E.5 Information Retrieval (IR) from Word Image Collection

Addressing Word Form Variations: The simple matching procedure described above may be efficient for spotting or matching a selected word-image. However the indexing process for a good search engine is more involved than the simple word-level matching. A word, with similar meanings, usually appears in various forms. This requires a method to conflate such different morphological variants of a word to a common stem/root [91]. Stemming controls word variations such that words with the same underlying stem are grouped together. In the text domain, variation of word forms may obey the language rules. Text search engines use this information while indexing. However for text-image indexing process, this information is not directly usable.

For instance, for a query "direct", the matching scores of the words "directed" and "redirected" are only the matching of the six characters, 'd-i-r-e-c-t', of both words. Once an optimal sub-path is identified, a normalized cost corresponding to this segment is considered as the matching score for the pair of words. With this we find that a large set of words get grouped into one cluster. We expect to extend this for more general variations of words.

The optimal warping path is generated by backtracking the DTW minimal score in the matching space. As shown in Figure E.3, extracted features (using upper word profile) of the two words 'direct' and 'redirected' are aligned using DTW algorithm. It is observed that features of these words are matched in such a way that elements of 're' at the beginning as well as 'ed' at the end of the word 'redirected' get matched with characters 'd' and 't' of the word "direct".

This additional cost is identified and removed while backtracking. It can be observed that for word variants the DTW path deviates from the diagonal line in the horizontal or vertical direction from the beginning or end of the path, which results in an increase in the matching cost. In the example Figure E.3, the path deviates from the diagonal line at the two extreme ends. This happened during matching the two words, that is, the root word (direct) and its variant (redirected). Profiles of the extra characters ('re' and 'ed') have minimal contribution to the matching score and hence subtracted from the total matching cost so as to compute the net cost. Such word form variations are very popular in most languages.

Detection of relevant words for indexing a page is very important for effective retrieval. Many interesting measures are proposed for this [35, 91]. We propose a measure in the same direction to do the similar job at image level.

Detection of Stop Words: Once similar words are clustered, we analyse the clusters for their relevance. Inverse document frequency is computed as a measure of the uniformity of the presence of similar words across the documents. If a word is common in all the documents, this word is less meaningful to characterize any of the document and hence considered as a stop word. Flagging such words from the list of word representations can have a significant impact on the search process [91].

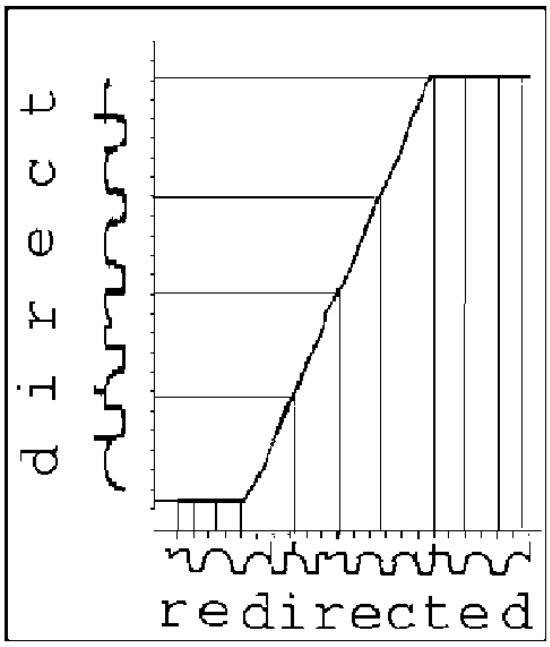


Figure E.3: Plot Demonstrating Matching of Two Words 'direct' and 'redirected' using Dynamic Time Warping and the Optimal Matching Path. Similar Word Form Variations are Present in Indian Languages.

Word Frequencies for Retrieval: Given a query, a word image is generated and the cluster (group of similar word images) in which this word will fall is identified. In each cluster, documents with highest occurrence of similar words are ranked offline using term frequency/inverse document frequency (TFIDF) weights defined at image level. Thus, TFIDF is applied for weighting the frequency of occurrence of a word in a document by the number of total documents containing that word in the collection and search results are displayed in descending order of TFIDF scores consequently.

Weight of the jth term (cluster) in the ith document is computed as

$$W_{ij} = f_{ij} \cdot (\log_2 N - \log_d d_j)$$

where f_{ij} is the frequency of the jth term (cluster representative) in ith document. N is the total number of documents in the collection and d_j is the number of documents containing the jth term. The TF/IDF weights computed shows the relevance of the documents to a given query.

Related Publications

- A. Bhaskarbhatla, S. Madhavanath, M. Pavan Kumar, A. Balasubramanian and C. V. Jawahar, "Representation and Annotation of Online Handwritten Data", in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 136-141.

- C. V. Jawahar, A. Balasubramanian and Million Meshesha, "Word-Level Access to Document Image Datasets", in *Proceedings of the Workshop on Computer Vision, Graphics and Image Processing (WCVGIP)*, 2004, pp. 73-76.

- C. V. Jawahar, Million Meshesha and A. Balasubramanian, "Searching in Document Images", in *Proceedings of the Indian Conference on Vision, Graphics and Image Processing*, 2004, pp. 622-627.

- A. Balasubramanian, Million Meshesha and C. V. Jawahar, "Retrieval from Document Image Collections", in *Proceedings of Seventh IAPR Workshop on Document Analysis Systems , 2006 (LNCS 3872)*, pp.1-12.

- Sachin Rawat, K. S. Sesh Kumar, Million Meshesha, Indineel Deb Sikdar, A. Balasubramanian and C. V. Jawahar, "A Semi-Automatic Adaptive OCR for Digital Libraries", in *Proceedings of Seventh IAPR Workshop on Document Analysis Systems , 2006 (LNCS 3872)*, pp.13-24.

- C. V. Jawahar and A. Balasubramanian, "Synthesis of Online Handwritten Data for Indian Languages", in *Proceeding of International Workshop on Frontiers in Handwriting Recognition*, 2006.

- Anand Kumar, A. Balasubramanian, Anoop Namboodiri and C. V. Jawahar, "Model-Based Annotation of Online Handwritten Datasets", in *Proceeding of International Workshop on Frontiers in Handwriting Recognition*, 2006.

Bibliography

[1] Ajay S Bhaskarabhatla and Sriganesh Madhvanath, "An xml representation for annotated handwriting datasets for online handwriting recognition," in *Proceedings of the Fourth International Conference on Linguistic Resources and Evaluation*, 2004.

[2] Mudit Agrawal, Kalika Bali, Sriganesh Madhvanath, and Louis Vuurpijl, "Upx: A new xml representation for annotated datasets of online handwriting data," in *Proceedings of International Conference on Document Analysis and Recognition*, pp. 1161–1165, 2005.

[3] A. Bhaskarbhatla, S. Madhavanath, M. Pavan Kumar, A. Balasubramanian, and C. V. Jawahar, "Representation and annotation of online handwritten data," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, pp. 136–141, 2004.

[4] Million Meshesha, "Recognition and Retrieval from Document Image Collections," in *Ph.D. Proposal for the Thesis*, 2006.

[5] A. K. Ray and T. Acharya, *Information Technology, Principles and Applications.* India: Prentic-Hall, 2004.

[6] I. Guyon, L. Scholamker, R. Plamondan, M. Liberman, and S. Janet, "UNIPEN Project of online data exchange and recogniser benchmarks," *Proc. ICPR*, pp. 29–33, 1994.

[7] The Multimodal Interaction Working Group 2003 Ink Markup Language (InkML) *http://www.w3.org/2002/mmi/ink.*

[8] Digital Library of India *http://dli.iiit.net.*

[9] C. V. Jawahar, Million Meshesha, and A. Balasubramanian, "Searching in document images," in *Proceedings of the Indian Conference on Vision, Graphics and Image Processing*, pp. 622–627, 2004.

[10] Santanu Chaudhury, Geetika Sethi, Anand Vyas, and Gaurav Harit, "Devising interactive access techniques for indian language document images," in *Proceedings of International Conference on Document Analaysis and Recognition*, pp. 885–889, 2003.

[11] T. Rath and R. Manmatha, "Features for Word Spotting in Historical Manuscripts," in *Proc. of the Seventh International Conference on Document Analysis and Recognition (ICDAR)*, pp. 218–222, 2003.

[12] A. Jain and A. M. Namboodiri, "Indexing and Retrieval of On-line Handwritten Documents," in *Proc. of the Seventh International Conference on Document Analysis and Recognition (ICDAR)*, pp. 655–659, 2003.

[13] T. Rath and R. Manmatha, "Word Image Matching Using Dynamic Time Warping," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 521–527, 2003.

[14] "Jot : A specification for an ink storage and interchange format," *Unpublished Standards Document, http://hwr.nici.kun.nl/unipen/jot.html*, 1993.

[15] Guyon.I, Schomaker.L, Plamondon.R, Liberman.M, and Janet.S, "Unipen project of on-line data exchange and recognizer benchmarks," in *Proceedings of International Conference on Pattern Recognition*, pp. 29–33, 1994.

[16] Dave Elliman and Nasser Sherkat, "A truthing tool for generating a database of cursive words," in *Sixth International Conference on Document Analysis and Recognition*, pp. 1255–1262, 2001.

[17] Sherif Yacoub, Vinay Saxena, and Sayeed Nusrulla Sami, "Perfectdoc: A ground truthing environment for complex documents," in *Proceedings of International Conference on Document Analaysis and Recognition*, pp. 452–457, 2005.

[18] M. Zimmermann and H. Bunke, "Automatic segmentation of the iam off-line database for handwritten english text," in *Proceedings of the 16th International Conference on Pattern Recognition*, pp. 35–39, 2000.

[19] E.M. Kornfield, R. Manmatha, and J. Allan, "Text alignment with handwritten documents," in *Proceedings of the International Workshop on Document Image Analysis for Libraries*, pp. 195–209, 2004.

[20] Anand Kumar, A. Balasubramanian, Anoop Namboodiri, and C. V. Jawahar, "Model-based annotation of online handwritten datasets," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, 2006.

[21] Isabelle Guyon, "Handwriting synthesis from handwritten glyphs," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, (Colchester, England), pp. 140–153, 1996.

[22] I. Guyon, M. Schenkel, and J. Denker, "Overview and synthesis of online cursive handwriting recognition techniques," in *Handbook of Character Recognition and Document Image Analysis*, May 1997.

[23] Jue Wang, ChenyuWu, Ying-QingXu, and Heung-Yeung Shum, "Combining shape and physical models for on-line cursive handwriting synthesis," in *International Journal on Document Analysis and Recognition*.

[24] Jue Wang, Chenyu Wu, Ying-Qing Xu, Heung-Yeung Shum, and Liang Ji, "Learning-based cursive handwriting synthesis," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, pp. 157–162, 2002.

[25] Y. Zheng and D. Doermann, "Handwriting matching and its application to handwriting synthesis," in *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 861–865, 2005.

[26] C. V. Jawahar and A. Balasubramanian, "Synthesis of online handwritten data for indian languages," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, 2006.

[27] R. Manmatha, C. Han, and E. Riseman, "Word spotting: A new approach to indexing handwriting," in *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, (San Francisco, CA), June 1996.

[28] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: A comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63–84, 2000.

[29] T. Rath and R. Manmatha, "Word image matching using dynamic time warping," in *Technical Report MM-38, Center for Intelligent Information Retrieval, University of Massachusetts Amherst*, 2002.

[30] G. Russell, M. Perrone, Y.-M. Chee, and A. Ziq, "Handwritten document retrieval," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, (Korea), pp. 233–238, Aug. 2002.

[31] A. K. Jain and A. M. Namboodiri, "Indexing and retrieval of on-line handwritten documents," in *Proceedings of the International Conference on Document Analysis and Recognition*, (Edinburgh, Scotland), pp. 655–659, Aug. 2003.

[32] S. N. Srihari and Z. Shi, "Forensic handwritten document retrieval system," in *Proceedings of the International Workshop on Document Image Analysis for Digital Libraries*, (Palo Alto, CA), pp. 188–192, Jan. 2004.

[33] A. Balasubramanian, M. Meshesha, and C. Jawahar, "Retrieval from document image collections," in *International Workshop on Document Analysis Systems*, (Nelson, NewZealand), pp. 1–12, 2006.

[34] Digital Library of India at: `http://www.dli.gov.in`.

[35] D. Doermann, "The Indexing and Retrieval of Document Images: A Survey," *Computer Vision and Image Understanding (CVIU)*, vol. 70, no. 3, pp. 287–298, 1998.

[36] T. Rath and R. Manmatha, "Word Image Matching Using Dynamic Time Warping," *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 521–527, 2003.

[37] S. Chaudhury, G. Sethi, A. Vyas, and G. Harit, "Devising Interactive Access Techniques for Indian Language Document Images," in *Proc. of the Seventh International Conference on Document Analysis and Recognition (ICDAR)*, pp. 885–889, 2003.

[38] Greenstone Digital Library Software at: `http://www.greenstone.org`.

[39] Adobe Systems Inc., PDF Reference, Fourth Edition, "http://adobe.com/,"

[40] Adobe Systems Inc, PostScript Language Reference, Third Edition, "http://adobe.com/,"

[41] XPDF - an open source PDF viewer, "http://www.foolabs.com/xpdf/,"

[42] "ITRANS - Indian Language Transliteration Package," *www.aczoom.com/itrans/*.

[43] OMTRNAS - Indian Language Transliteration at: `http://www.cs.cmu.edu/~madhavi/Om/`.

[44] A. P. Lenaghan and R. R. Malyan, "XPEN: An XML Based Format for Distributed Online Handwriting Recognition," *Int. Conference on Document Analysis and Recognition(ICDAR)*, pp. 1270–1274, 2003.

[45] K. Franke, L. Schomaker, C. Veenhuis, L. Vuurpijl, M. van Erp, and I. Guyon, "WANDA: A common ground for forensic handwriting examination and writer identification," *ENFHEX news - Bulletin of the European Network of Forensic Handwriting Experts*, no. 1/04, pp. 23–47, 2004.

[46] The UNIPEN Project *http://www.unipen.org*.

[47] Toni M. Rath, R. Manmatha, and Victor Lavrenko, "A search engine for historical manuscript images," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 369–376, 2004.

[48] Xian-Sheng Hua, Liu Wenyin, and Hong-Jiang Zhang, "An automatic performance evaluation protocol for video text detection algorithms," in *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 498–507, 2004.

[49] Berrin A. Yanikoglu and Luc Vincent, "Ground-truthing and benchmarking document page segmentation," in *Proceedings of the Third International Conference on Document Analysis and Recognition*, pp. 601–604, 1995.

[50] Louis Vuurpijl, Lambert Schomaker, and Egon van den Broek, "Vind(x): using the user through cooperative annotation," in *Proceedings of International Workshop on Frontiers of Handwriting Recognition*, pp. 221–226, 2002.

[51] C. Tomai, B. Zhang, and V. Govindaraju, "Transcript mapping for historic handwritten document images," in *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition*, pp. 413–418, 2002.

[52] J. Rothfeder, T.M. Rath, and R. Manmatha, "Aligning transcripts to automatically segmented handwritten manuscripts," in *Proceedings of the Seventh International Workshop on Document Analysis Systems*, 2006.

[53] J. Jeon, V. Lavrenko, and R. Manmatha, "Automatic image annotation and retrieval using cross-media relevance models," in *Proceedings of the 26th International ACM SIGIR Conference*, pp. 119–126, 2003.

[54] V. Lavrenko, S. L. Feng, and R. Manmatha, "Statistical models for automatic video annotation and retrieval," in *Proceedings IEEE International Conferene on Acoustics, Speech and Signal Processing*, pp. 17–21, 2003.

[55] Thorsten Brants, "Tnt: a statistical part-of-speech tagger," in *Proceedings of the sixth conference on Applied natural language processing*, pp. 224–231, 2000.

[56] "Sphinx - The Carnegie Mellon Sphinx Project," *http://cmusphinx.sourceforge.net*.

[57] "The Festival Speech Synthesis System," *http://www.cstr.ed.ac.uk/projects/festival/*.

[58] M. Pavan Kumar, S. S. Ravikiran, Abhishek Nayani, C. V. Jawahar, and P. Narayanan, "Tools for Developing OCRs for Indian Scripts," *DIAR'03 in connection with Int. Conf. on Computer Vision and Pattern Recognition*, 2003.

[59] Inscript (Indian Script) Keyboard *http://tdil.mit.gov.in/keyoverlay.htm*.

[60] "Doxygen - a documentation generator for C++, C, and others," *http://www.stack.nl/ dimitri/doxygen/*.

[61] "QT - a C++ graphics library," *http://www.trolltech.com/*.

[62] "The Unicode Consortium," *http://www.unicode.org*.

[63] "ISCII - The Indian Script Code for Information Interchange," *http://tdil.mit.gov.in/standards.htm*.

[64] "The Open Software Description Format (OSD)," *http://www.w3.org/TR/NOTE-OSD.html*.

[65] A. Balasubramanian and C. V. Jawahar, "Synthesis of online handwritten data for indian languages," in *Proceeding of International Workshop on Frontiers in Handwriting Recognition*, 2006.

[66] I. Kamel, "Fast retrieval of cursive handwriting," in *Proceedings of the 5th International Conference on Information and Knowledge Management*, (Rockville, MD), pp. 91–98, 1996.

[67] S. N. Srihari, "Recognition of handwritten and machine-printed text for postal address interpretation," *Pattern Recognition Letters*, vol. 14, pp. 291–302, 1993.

[68] C. Cracknell and A. Downton, "A handwritten form reader architecture," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, (Korea), pp. 67–76, 1998.

[69] J. Hu, M. Brown, and W. Turin, "Hmm based online handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 1039–1045, Oct. 1996.

[70] S. D. Connell and A. K. Jain, "Writer adaptation for online handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 329–346, Mar. 2002.

[71] S. Srihari, H. Cha, S.-H. Arora, and S. Lee, "Individuality of handwriting," *Journal of Forensic Sciences*, vol. 47, no. 4, pp. 1–17, 2002.

[72] Tamas Varga and Horst Bunke, "Generation of synthetic training data for an HMM-based handwriting recognition system," in *Proceedings of the International Conference on Document Analysis and Recognition*, (Edinbugh, Scotland), pp. 618–622, 2003.

[73] Muriel Helmers and Horst Bunke, "Generation and use of synthetic training data in cursive handwriting recognition," in *Proc. 1st Iberian Conference on Pattern Recognition and Image Analysis*, pp. 336–345, 2003.

[74] D. Lopresti and A. Tomkins, "On the searchability of electronic ink," in *Proc. of IWFHR*, pp. 156–65, 1994.

[75] Wacef Guerfali and R. Plamondon, "The delta lognormal theory for the generation and modeling of cursive characters," in *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 495–498, 1995.

[76] Y. Singer and N. Tishby, "Dynamical encoding of cursive handwriting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 341–346, 1993.

[77] R. Plamondon, "A Kinematic theory of rapid human movements, Part I. Movement representation and generation," in *Biological Cybernetics*, vol. 72, pp. 295–307, 1995.

[78] S. Kondo, "A model of handwriting process and stroke structure of character figures," in *Computer Recognition and Human Production of Handwriting*, pp. 103–118, 1989.

[79] L. Schomaker, A. Thomassen, and H. Teulings, "A computational model of cursive handwriting," in *Computer Recognition and Human Production of Handwriting*, pp. 119–130, 1989.

[80] D.-H. Lee and H.-G. Cho, "A new synthesizing method for handwriting Korean scripts.," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 12, no. 1, pp. 46–61, 1998.

[81] Ondrej Velek, Cheng-Lin Liu, and Masaki Nakagawa, "Generating realistic kanji character images from on-line patterns," in *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 556–560, 2001.

[82] A. Bhaskarbhatla, S. Madhavanath, M. Pavan Kumar, A. Balasubramanian, and C. V. Jawahar, "Representation and Annotation of Online Handwritten Data," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, pp. 136–141, 2004.

[83] D. Lopresti and A. Tomkins, "On the searchability of electronic ink," in *Proceedings of the International Workshop on Frontiers in Handwriting Recognition*, (Taipei), pp. 156–165, 1994.

[84] C. V. Jawahar, Million Meshesha, and A. Balasubramanian, "Searching in Document Images," *Proc. of the 4th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)*, pp. 622–627, 2004.

[85] Department of Information Technology, "Technology Development for Indian Languages." at: `http://tdil.mit.gov.in`.

[86] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York: John Willey & Sons, 2001.

[87] P. Vossen and C. Fellbaum, "The Global WordNet Association." at: `http://www.globalwordnet.org`.

[88] Universal Language Dictionary Project at: `http://ogden.basic-english.org`, 2003.

[89] An Image library, "http://www.imagemagick.org/,"

[90] H. Sakoe and S. Chiba, "Dynamic Programming Optimization for Spoken Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 26, pp. 623–625, 1980.

[91] R. Korfhage, *Information Storage and Retrieval*. New York: John Willey, 1997.