



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

UNCONSTRAINED HANDWRITING RECOGNITION
APPLIED TO THE PROCESSING OF BANK CHEQUES

DIDIER GUILLEVIC

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY AT
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

SEPTEMBER 1995
© DIDIER GUILLEVIC, 1995



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-05079-3

Canada

Abstract

Unconstrained handwriting recognition applied to the processing of
bank cheques

Didier Guillevic, Ph.D.

Concordia University, 1995

A method for recognizing unconstrained handwritten words belonging to a small static lexicon is proposed. Previous approaches typically attempt to recognize characters or parts of characters in order to recognize words. Our approach, in its first step, bypasses the notion of characters. In addition to language independence, our method is more context oriented and should prove to be more robust against poor handwriting, spelling mistakes, noise and the like. Our computational theory is based on a psychological model of the reading process of a fast reader. First a few graphical clues such as ascenders, descenders and their relative positions are extracted from the word. If these prove not to be sufficient to clearly identify the word, then details (secondary features including first and last characters of words) are extracted to enhance the word recognition.

We designed and collected a database of bank cheques both in English and French. This resulted in a *one of its kind* database in a university setting dealing with handwritten information from bank cheques, both in terms of the size of the database as well as the number of different writers involved. We further designed an innovative, simple yet powerful *in place* tagging procedure for our database. It enables us to extract at will not only the bitmaps of words, characters, digits, lines, commas, etc... but also all kinds of contextual information.

We developed a fully trainable word recognizer with the requirement that the switch to a different database and/or language shall not require any redesign nor any extensive retraining time. The number of parameters within the system has

been kept to a minimum and whenever possible we designed algorithms that require no parameters and therefore no training. Such an example is our slant correction algorithm that shines by its simplicity and robustness. Whenever parameters might need to be adjusted to a specific database, it is done automatically by running some genetic algorithms.

We tested the generality and adaptability of our system on 2 different databases of bank cheques (respectively English and French). We noticed that the system's parameters did not need to be readjusted for it to perform satisfactorily when the switch was made from one database to the other. At the time of this dissertation, our survey indicates that this research is the only one in the literature which can handle English cheques and our results are comparable to those published on the processing of French cheques. Our preliminary results on the French database report a recognition rate of 98.9% and 94.3% on the word and the full legal amount respectively among the top 5 choices.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Ching Y. Suen for having created a very stimulating and well-equipped lab environment, through many years of dedicated efforts.

I am grateful to all the people who helped me one way or another during those years at CENPARMI. Special thanks go to Steve Malowany, Stan Swiercz, Michael Assels and William Wong for their technical competence, for maintaining our computer network, and their great help in mastering the Unix operating system. Thanks to Christine Nadal for her help in collecting the database of cheques. I am also thankful to Raymond Legault and Liu Ke for helpful exchanges of ideas.

I am thankful to "my Pal" Peiying Zhu for her great help in many ways and her friendship.

I am indebted to Nick Strathy for *many* fruitful discussions. His help in all the aspects of conducting research, including structuring and managing my code, has been of a significant importance in making that research possible.

I would like to express my gratitude to my external examiner Dr. Theo Pavlidis as well as to the other members of my examining committee: Dr. R. Bhat, Dr. C.Y. Suen, Dr. R. Gurnsey, Dr. T.D. Bui and Dr. Tony Kasvand. Their comments helped to improve the quality of the final version of this thesis.

Special thanks to my coffee breaks pals, Peiying, Juliette, Nick and Louisa, for sharing such relaxing and fun moments and to other members of CENPARMI and other friends for 'being there' such as Patrice, Henry, Daniel and Satomi.

I would like to thank my early supervisors back in England, Dr. Andy Downton and Dr. Graham Leedham. Their friendliness motivated me to pursue my studies.

Last but not least, j'aimerais remercier ma famille; mes parents, Nicole et Gilbert ainsi que mon frère, Serge.

A Nicole, Gilbert et Serge.

En souvenir de mes années tripartites:

- Mes années à l'ESIEE.
- Karlsruhe, Anfang meiner internationalen unvergeßlichen Erfahrungen.
- My final year in Colchester.

Contents

List of Tables	xii
List of Figures	xv
1 Introduction	1
1.1 The challenge	1
1.2 Motivation	2
1.3 Previous work	4
1.4 The proposal	5
1.5 Contributions	6
2 Psychological Models	7
3 Database	11
3.1 Previous databases	11
3.2 Design and printing of cheques	13
3.3 Generating the cheque amounts	13
3.4 Collection of the database	18
3.5 Storage of information	20
3.6 The truthing procedure	20
3.7 The final database	24
3.8 French database	26
3.9 Summary	26

4	Legal Amount Processing	27
4.1	Preprocessing	30
4.1.1	Baseline skew correction	30
4.1.2	Broken image repair	32
4.1.3	Slant correction	33
4.1.4	Smoothing	37
4.1.5	Noise removal	39
4.1.6	Average stroke thickness	41
4.2	Segmentation of the amount into words	41
4.3	Word recognition	43
4.3.1	Global word features	44
4.3.2	Feature vector	48
4.3.3	Probabilistic classifier	50
4.3.4	Nearest neighbour classifier	54
4.3.5	Confidence measure for the classifier	58
4.3.6	Word details extraction	60
4.3.7	Character recognition	66
4.3.8	Integrating character results with word results	76
4.4	Parsing module	82
4.4.1	English parser	82
4.4.2	French parser	86
5	Experimental Results	91
5.1	Global features recognizer	91
5.2	Character recognition	95
5.3	Integrating character and word recognition	96
5.4	Legal amount recognition	97
5.5	Extra: Processing of French cheques	98
5.5.1	French word recognition results	99
5.5.2	French legal amount recognition results	101
5.6	Comparison of results	102

5.7	Novel word spelling	103
5.8	Demonstration's interface	104
6	Conclusion	106
6.1	Summary of contributions	106
6.2	Strengths and weaknesses of the method	107
6.3	Future work	108
	References	110
A	Handwriting Samples	117
A.1	English legal amounts	117
A.2	English words	134
A.3	Characters	142
A.4	French legal amounts	148
B	Baseline Skew Correction	156
B.1	Determination of the baseline skew	156
B.1.1	Principal axis decomposition	156
B.1.2	Least square method	158
B.1.3	Local minima	166
B.2	Rotation of the original image	167
B.3	Summary	170
C	Slant Correction Algorithm	171
C.1	Computation of the slanted histograms	171
C.2	Shear transformation	172
D	Mathematical Morphology	175
D.1	Dilation and erosion	175
D.1.1	Dilation	176
D.1.2	Erosion	177
D.2	Opening and closing	178

D.3 Gray scale morphology	179
D.4 Implementation	180
E Parameters	182

List of Tables

1	Example of randomly generated legal amounts in English	14
2	Grammar for generating the digit amounts	15
3	Meaning of each state of the grammar	16
4	Values taken by the terminal symbols	16
5	Example of randomly generated digit amounts	17
6	Estimate of samples per class generated with a set of 3,000 cheques .	17
7	Number of samples per word class in our database of legal amounts .	25
8	Number of single characters in our database of legal amounts	25
9	Character sets for the first and last position of words	68
10	Character features: 0 = <i>NO</i> , 1 = <i>YES</i> , 2 = <i>MAYBE</i>	73
11	Grammar for the English legal amount	83
12	English final states: solely full dollar amounts	85
13	English final states: dollar + cents amounts	85
14	Classes of terminal symbols for the English grammar	86
15	Grammar for the French legal amount	88
16	Classes of terminal symbols for the French grammar	90
17	English word recognition results (% correct in top <i>N</i> choices)	92
18	English word recognition results per class (% correct in top <i>N</i> choices)	93
19	Character recognition results (% correct in top <i>N</i> choices)	95
20	English word recognition results when merged with character recognition	97
21	English word recognition results when using the English parser	98
22	English legal amount recognition results	98
23	Word recognition results (% correct in top <i>N</i> choices)	99

24	French word recognition results per class with AD features	100
25	French word recognition results when using the French parser	102
26	Legal amount recognition results (% correct in top N choices)	102
27	Comparison of word recognition results (% correct in top N choices) .	103
28	Novel word spelling	104

List of Figures

1	Various handwriting styles	3
2	Definition of terms referring to handwritten cheque information . . .	12
3	Sample of cheque used for the generation of the 'information' database	14
4	Example of a Concordia cheque filled by a student	18
5	Legal amount written solely with capital letters	18
6	Samples of legal amounts	19
7	User interface for the tagging of the database	21
8	Image message after the tagging	22
9	Hierarchical diagram of the cheque processing system	28
10	Processing of the legal amount	28
11	A baseline-skewed image and its desired skew-free version	31
12	Baseline skewed words within "skew-free" legal amounts	31
13	(a) Original image, (b) Reconnected broken strokes	32
14	Side effect of the reconnection module	33
15	Slant correction	34
16	Vertical histogram: (a) original word (b) slant-corrected word	35
17	Derivative of vertical histogram: (a) original word (b) slant-corrected word	36
18	Example of a word for which our current slant heuristic fails	37
19	Situation for the removal of black pixel	38
20	Situation for the removal of white pixel	39
21	(a) Original image (b) Smoothed image	40
22	Contour: (a) original (b) smoothed	40

23	Examples of legal amounts to be segmented	41
24	Line removal: (a) Original(b) Lines removed	42
25	Hierarchical diagram of the word recognizer	44
26	Lower and upper reference lines	45
27	Ascender and descender bodies	46
28	Ascender and descender thresholds and buckets	47
29	Ascender, descender and loop features	47
30	Stroke features: (a) original image (b) vertical (c) horizontal (d-e) diagonal	49
31	Creating the feature vector for a given class	51
32	Probabilistic classifier: probability computation	53
33	Nearest Neighbour Classifier	55
34	Vector distance computation	56
35	Example: Distance between 2 ascender position vectors	56
36	Disconnected first and/or last characters of words	61
37	Word represented as an ordered list of connected components	62
38	Locating the first character of a word	62
39	Character extraction: first component overlapping the main body	63
40	Character extraction: character width cw	64
41	Character extraction: Thresholds for the character width	65
42	Character segmentation	67
43	Dynamic character sets	69
44	Information available for each character	69
45	Character shape variations according to the position within a word	70
46	Example of the distance map computation	71
47	Non-characters extracted from the first and last positions of words	75
48	Selection of the top character(s) for subsequent processing	77
49	Shifting up and down of word solutions	79
50	Shifting of the top <i>TOP_PERMUTATION</i> word solutions	80
51	Parsing of the English legal amount	84
52	Parsing of the French legal amount	89

53	Example of legal amount recognition results	92
54	User interface for demonstration purposes	105
55	Eigenvectors e_1, e_2 : Principal axis decomposition	158
56	Eigenvectors: Not appropriate for baseline skew estimation	159
57	Linear regression	160
58	Polar coordinate system: (θ, ρ)	161
59	Polar coordinate system: positive distance of a point to a line	162
60	Polar coordinate system: negative distance of a point to a line	163
61	Local minimas: (a) original image (b) local minimas (c) pruned local minimas (d) outliers removed	166
62	Local minimas: (a) original image (b) pruned set of local minimas (c) least square fitting line (d) resulting set of minimas	168
63	Baseline skew normalization	169
64	Computation of the slanted histograms	172
65	Shear transformation by a given angle θ	174
66	Shear transformation of an image initially slanted at 60 degrees	174

Chapter 1

Introduction

The reading of characters by computer known as Optical Character Recognition (OCR) is a topic that has been investigated for many years [SBM80, MSY92]. Nowadays both machine printed and handwritten characters are recognized by machines. The extension of OCR to the reading of words has also been investigated. Most solutions to this problem rely on a process that segments the input into individual characters. These characters are then exhaustively recognized. Systems are now in wide commercial use for the processing of printed materials. However this technique has not yet been successfully applied to read cursive handwriting. The problem remains extremely challenging due to the great variability in handwriting styles, handwriting devices, etc. . . . Current research aims at developing systems for limited domain applications such as the processing of bank cheques. The defined lexicon plus a well constrained syntax help provide a feasible solution to the problem.

1.1 The challenge

The processing of handwritten words is extremely challenging due to the great variability in handwriting styles (Fig 1), handwriting instruments, etc. . . . The problem of recognizing handwritten words is so difficult that in order to get feasible solutions, one needs to set some constraints. The *3 major components* that influence the complexity of a handwriting recognition problem are the *size of the lexicon, the type of*

handwriting and the number of writers. The more the constraints on these components, the easier it gets to produce a solution; but do not misunderstand those words, “easier” does not mean that the problem is a trivial one. It simply means that better overall recognition rates can be attained.

Most of the studies in the literature deal with a limited number of writers, a small lexicon, and sometimes some constraints on the handwriting. The more ambitious (by necessity) studies are focusing on a database with “unlimited” writers, unconstrained handwriting and a small lexicon, dynamic or static. As an example, people working on postal applications (e.g. [Sri92]) are dealing with a problem of unlimited writers, unconstrained handwriting and a “small” dynamic lexicon of city, state and street names. The lexicon of city names, street names is a large one but can be reduced dynamically with the help of the zip code recognition results prior to being passed to the word recognition module. Another such application domain with “unlimited” writers and unconstrained handwriting is the processing of bank cheques [Mor91, GL93, GS95]. In this latter problem domain, the lexicon is small and static.

1.2 Motivation

Utility companies have great interest in a system which is able to process handwritten cheques reliably. At the present the processing of payments is still done in a semi-automatic fashion with operators keying in the amount read on each cheque. This slow and costly process is a major bottle-neck for those companies and it would seem that it is to stay that way for still quite a while. Indeed a recent Gallup poll commissioned by the Financial Stationers Association (FSA) [Cas94] once again demonstrated that as much as 83% of Americans still favor the cheque as the most convenient form of monthly bill payment. For that reason and other factors research has been undertaken at various centres and companies over the world in an attempt to bring a solution to that problem. Even though progress has been made, no reliable system has reached the market yet.

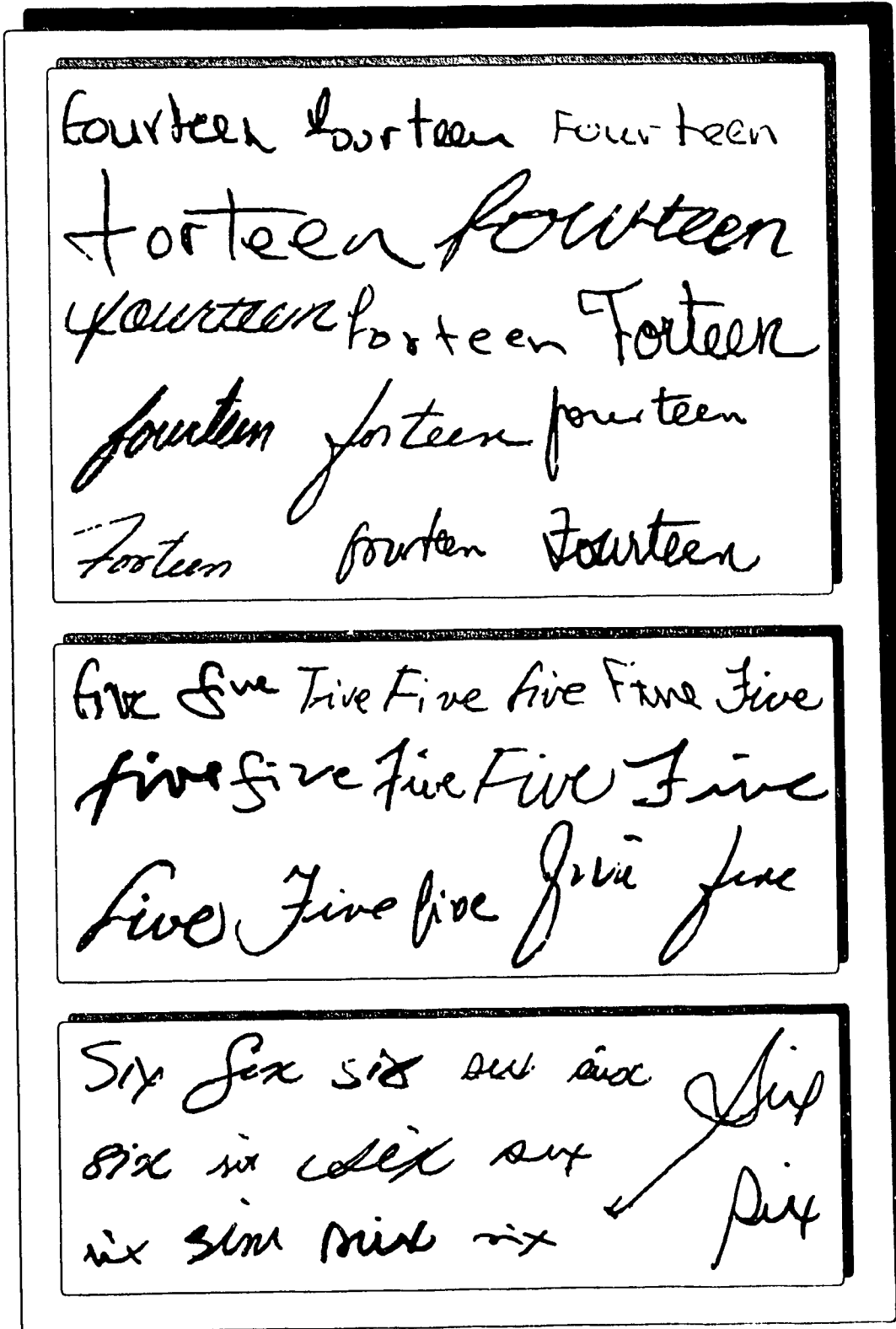


Figure 1: Various handwriting styles

1.3 Previous work

We shall discuss here in general terms the research that has taken place to tackle the problem of handwritten word recognition. A full detailed review would only be duplicating other people's work, so that the interested reader is referred to [Fav93] for an in depth review of the research field.

Previous studies on word recognition typically apply some technique to segment the word into individual characters [Say73, EK75, BS89, BHD91]. These characters are then sent to a character recognizer and the combination of the recognition results produced a ranked list of possible words. Such technique is commonly referred to as *analytical approach*. While this approach can handle an "unlimited" lexicon size, it has the drawback of relying on the results of the difficult and unreliable segmentation stage.

Some other techniques [Far79, BG80, FS90, Mor91, GS95], known as *global approach* look at the word as a whole, and extract some global features such as ascenders, descenders, loops, etc. . . . Individual characters are not recognized as such but features are extracted indicating which characters may be present in the word. This approach has the advantage of avoiding the difficult segmentation stage.

The analytical approach is theoretically stronger in handling a large vocabulary. Indeed with a constant number of classification classes (e.g. the number of letters in the alphabet), it can handle any string of characters and therefore an unlimited number of words. The wholistic approach on the other hand, must generally rely on an established vocabulary of acceptable words. Its number of classification classes increases with the size of the lexicon. The "whole word" scheme is potentially faster when considering a relatively small lexicon. It is also more accurate having to consider only the legitimate word possibilities. One disadvantage of a whole word recognizer is its inability to identify a word not contained in the vocabulary. On the other hand, it has greater tolerance on the presence of noise, spelling mistakes, missing characters, unreadable part of the word, and the like. For a small vocabulary, it is feasible to recognize cursive words using the wholistic approach.

An alternate approach coming from the speech recognition field is to scan words as

a whole and make use of Hidden Markov Models (HMM) [GL93, GBL93]. The HMM approach seems to be well suited for applications with limited dynamic lexicons such as the reading of city names for postal sorting. An HMM system has the ability to recognize word classes for which no samples are present in the training set.

In the field of cheque processing, we are dealing with a limited static lexicon and therefore alternate, less general but no less robust approaches can be investigated. Our computational theory is based on the psychological model of reading of the fast reader [GS93].

Some researchers seem to agree that the future of reliable recognition systems lies in the combination of multiple classifiers [SNL⁺92, HHS94]. It is our intent to test the feasibility of having a system based on the psychological model of the fast reader as one of the basic word classifiers used for the recognition of legal amounts found in bank cheques.

1.4 The proposal

If we were to analyze the work done by psychologists in the effort to understand the human reading process, one could classify the generated models into two main kinds. The model of the fast reader is for those people who manage to read faster than they can speak, while the model of the reading process for a slow reader matches the rest of us. Slow readers are known to read no faster than they can speak. At the extreme low end of the reading speed is the child who learns how to read. He analyzes each character, then makes up words, then sentences.

Most of the algorithms for word recognition published in the literature do match the model of the slow reader. We propose to investigate how a computational theory based on a model of the reading process for a fast reader would perform in an application like the processing of cheques.

1.5 Contributions

The first contribution of this work is to myself. Not only did I learn a lot technically but I also gained insights into terms like *patience* and *wisdom*.

In a more practical sense, we did contribute a great amount of time designing, collecting and then truthing the database of cheques that we later used for training and testing our system. This resulted into a *one of its kind* database in a university setting dealing with handwritten information from bank cheques. The *in-place* truthing procedure is also innovative and unique of its kind in the literature. Its simplicity, yet power, makes it a real asset for further generations of researchers working on our database. It can also be easily applied to the tagging stage of other databases.

This work introduced yet another simple, but no less efficient, algorithm dealing with the slant correction of handwritten words. Its power comes not only from its simplicity but also from its robustness versus the numerous handwriting types found in our database.

Last but not least, we designed a fully trainable word recognizer that should be able to handle any small static lexicon of words written with the Roman alphabet. Our computational theory based on a psychological model of the reading process for a fast reader shines by its simplicity and still compares equally with other published schemes. It is to be noted that this work is the sole study in the literature reporting results on the recognition of handwritten English bank cheques tested on a significant database. This work also compares equally with the sole other study reporting results on the processing of handwritten French bank cheques tested on a significant database.

Chapter 2

Psychological Models

Extensive work has been carried out by psychologists in the effort to understand the human reading process. Waters [Wat77] conducted an extensive survey of the literature discussing the psychology of reading and word recognition. As early as the beginning of this century, psychologists such as Huey [Hue08] pointed out that a full understanding of the processes involved in reading would represent quite an achievement. However, as of today, a comprehensive model of reading yet remains to be developed.

In psychological studies, a distinction is made between children who start reading and adults who have already gained some experience in this process. Adults are further subdivided into slow and fast readers. Young children and slow readers are known to read no faster than they can speak. Studies have shown that fast readers read much too fast to be able to identify every single character of a word [NB65].

Shebilske [She75] reports on the eye movement while reading. The reader's eye movement across a line is not continuous, but rather occurs in a series of short jumps called 'saccades'. The fixation time between saccades is roughly ten times longer than the saccade. The typical saccade of 1 to 2 degrees requires 20 to 30 msec, whereas fixation time averages 250 msec. Initial processing of the visual stimulus must occur during the fixation time because the stimulus pattern is blurred during a saccade and its duration is too short for sufficient processing to occur.

In some studies [Smi69], words were found to be identified under conditions in

which their component letters could not be identified. These findings have led several researchers to suggest that readers may in fact sample the visual data in reading rather than process every element of the text.

Goodman [Goo67] portrays reading as a *psycholinguistic guessing game* in which the reader processes and coordinates simultaneously three types of information: graphic, syntactic and semantic. Prior to analysing the graphical input, the reader holds syntactic and semantic expectations about the information residing there. He samples the graphic cues in accordance with his expectation and uses this information to generate a guess. If the guess proves consistent with subsequent information, then it is accepted. If not, the reader recognizes that he has misread the stimulus and he reinspects it to correct his error.

The proficient reader is thought to ignore many of the features contained in the graphic display. Goodman, in fact, asserts that the more efficient the reader, the fewer printed cues he needs to derive meaning accurately from text. Furthermore, this text sampling process is governed primarily by the adequacy of the meaning being extracted. If the reader misreads a word but produces a semantically acceptable substitute, he fails to recognize the discrepancy. However if his construction does not make sense, then inaccuracy is detected.

Current research efforts have shown that fluent readers read much too quickly to be identifying in succession the letters or words on the page, and that, fluent readers need not in fact, process every letter or phoneme of the material they are reading into a full perceptual representation in order to abstract meaning.

Massaro [Mas75] illustrates the notion of *between letters* (orthographic), *between word* (syntactic), and *within meaning* (semantic) redundancies with the following example:

Suppose the reader encounters the following sentence: 'With the bases loaded the boy hit the lll over the fence'. Assume that the reader completely resolved all of the letters in the sentence except for the two underline positions. Accordingly, it is necessary to identify the two missing letters in the four letter word. Partial visual information defines a vertical line at the first letter position and no feature information is registered for the second position. The one visual feature in the first

position eliminates all vowel alternatives for that position, exemplifying the use of visual information. Having determined that the first and last two letters are consonants, orthographic constraints (orthographic redundancy) dictate that the second letter is a vowel. At this point many possible four letter words remain, for example, {tell, tall, ball, bull, hill, fill}. Syntactic information given by the surrounding words (syntactic redundancy) eliminates all of the alternatives except nouns. Finally the meaning of the other words in the sentence also provide contextual information (semantic redundancy). It would make no sense to say 'The boy hit the bull over the fence', therefore ball is the only remaining alternative in the list given above.

There is a considerable amount of evidence which shows that redundancy is fairly high in the English language. This evidence has led several researchers to hypothesize that readers may utilize this redundancy to supplement visual information in reading.

This conceptualization of the nature of the English orthography along with studies showing that readers need not read on a letter by letter basis, has led several researchers [Goo67, Mas75] to propose models of reading in which the reader is thought to sample the visual data and capitalize on the redundancy of the English language as an aid in fluent reading.

The fluent reader is thought to be involved in an active process in which he selectively samples the visual data and combines this information with his knowledge of semantic, syntactic and within word orthographic constraints to construct hypotheses as to the meaning of larger units.

Waters [Wat77] reports on some studies which attempted to find out where the eyes move next after a prediction. It is supposed that since the first letter of a word contains much information about the identity of the word, the fixations of the eye could happen at or near letters that follow blank spaces. In order to test this hypothesis, in one study they had the blank spaces between words filled with X's. The reading speed of slow readers was not affected by this manipulation of the text, while fast readers showed a significant drop in reading speed. This experiment tends to confirm that letters near blank spaces are important for the fast reader.

The skilled reader is thought to have learned guessing techniques, to utilize redundancy in the text, to read for meaning, and not be concerned with the specific

stimulus pattern that confronts his eyes but rather with what information it gives him about what he is reading and where to look next. The main task of the skilled reader is then seen as one of extracting information from an array of redundant symbols.

All these models suggest that the fluent reader samples the visual data and in doing so capitalizes on the redundancy of the English language as an aid to fluent reading.

So far, most of the studies that have been proposed to tackle the problem of cursive script understanding are based upon the model of the child that learns how to read. Indeed, studies intend to recognize each and every single character of a word, and then utilize some high level knowledge (such as a lexicon) to produce a valid output word.

It would seem that a computational theory based on the model of the fast reader could prove to be more efficient. In this model, prior to reading, the reader holds syntactic and semantic expectations about the input text. He samples the input accordingly and uses the information to produce a guess. Hence, since such a model is mainly based on non-visual information such as orthography, syntax and semantics, the theory should prove to be more robust against spelling mistakes, missing letters, unreadable letters and the like. Whereas the slow reader model is suitable for applications dealing with printed material, the fast reader model would seem more appropriate for the imperfect nature of cursive script.

Chapter 3

Database

In this chapter, we describe an image database for research in cheque processing. We begin by presenting the design involved in the collection of the database. An elaborate tagging scheme has been semi-automated which allows for the tagging to the legal amount of all kinds of information relating to words, characters, dashes, commas, etc. . . . Complete description of a legal amount can thus be stored in the image message. Words, characters, punctuation marks, and the like can be extracted automatically whenever needed. This in-place tagging technique does not only facilitate the truthing process but also enables the recognition system to take advantage of the context in the design and training of various recognition modules. When designing a particular recognition module, one itemizes the kind of contextual information that one might want to use. The extraction of images with the desired contextual information is then performed in a fully automatic way. Throughout this chapter, I might refer to the terms *date*, *courtesy amount*, *legal amount* and *signature* when describing the handwritten information found on cheques. These terms are defined in Fig. 2.

3.1 Previous databases

We report here the design, collection and truthing procedure of a database of information contained in handwritten cheques. This project has been undertaken to provide a significant number of handwriting samples for the training of a Cheque



Figure 2: Definition of terms referring to handwritten cheque information

Processing System currently being developed at the CENTre for PATtern Recognition and Machine Intelligence (CENPARMI).

Significant databases of handwritten digits and Roman characters are now available from various institutions such as NIST [SBSV94], CEDAR [Hul94], and CENPARMI [SNL+92]. This enables any research team to adequately train their recognition algorithms as well as to form a basis for comparison of results. However, a similar database for cheque processing research is not available at the moment. The collection of a database of cheque information is a non-trivial task. One of the main problems that we may encounter is getting a financial institution to provide us with real cheques. For security reasons and protection of the customer and the institution, this is usually not possible without an explicit contract between the research centre and the institution. This latter step can turn out to be a stumbling block.

This may explain why most of the research publications on the recognition of handwritten words for cheque processing report results on small databases with a total of only 5 to 25 different writers. To our knowledge, there are only two studies which have reported results on a significant database, namely Gilloux and Leroux [GL93] from the French Post Office Research Centre as well as Moreau [Mor91] from Matra MS2I. These teams are linked to the French Post Office (which also provides Banking Services) which provides them with a database with a large number of writers.

In the framework of our project, we decided to experiment with the design, collection and truthing procedure of cheque handwritten information, while some negotiation was being conducted for the access to 'real-life' cheques. The extraction of information from real life data also implies that an extraction module should be readily available. However, this is usually not the case in an on-going project where modules are being developed in parallel.

For the reasons mentioned above, we decided to create our own database from scratch. This included (a) the designing of cheques that would make it easy to extract the information without the explicit use of an extraction module, (b) the generation of the amounts to be written, (c) the collection of a meaningful database, as well as (d) the design of a simple yet very efficient truthing procedure. These steps are described in the following sections.

3.2 Design and printing of cheques

The 'Bank of Concordia' cheques that we designed (Fig. 3) are similar in appearance to those from banks in the Montréal area. The major difference being that these cheques have a white background and lines are printed in a light blue colour instead of black or dark ink. Using such a colour facilitates the removal of the lines from the written information. We gave those prototypes to a printing company and had 15,000 cheques printed.

3.3 Generating the cheque amounts

Simply asking people to fill out cheques without specifying the amount they should write would most probably result in a database having a selected few amounts coming back over and over again (e.g. "One hundred dollars"). For example, we would not like to have a database with a given word (say 'eleven') appearing only once or twice. Otherwise this would make the training of our algorithm for this specific word rather difficult due to the lack of training samples.

Thus such a scheme would not serve our purpose since we need a significant and

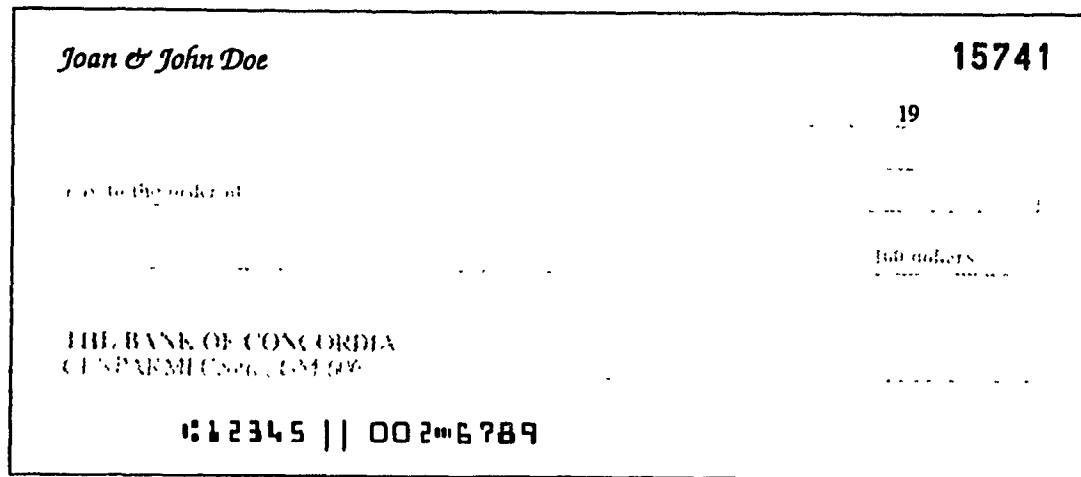


Figure 3: Sample of cheque used for the generation of the ‘information’ database

sixteen hundred dollars five dollars and twenty seven cents eight twelve hundred sixteen fifty six dollars and fourteen

Table 1: Example of randomly generated legal amounts in English

somewhat equal number of samples from each class we are to recognize. For this reason, we decided to generate the amounts ourselves. This allowed us to control the approximate number of samples from each class that we would collect.

In order to control the exact number of words produced for each class, one might be tempted to specify the amounts in full (Table 1) using some parser for the English cheque amounts. While this is attractive, it limits the freedom of people to write down amounts as they ‘feel’, and this may introduce some biases into the data. For a given amount of 120\$, one person might write down “One hundred and twenty dollars and 00 cents”, while another might simply write “One hundred twenty”. With concerns to loosening the constraints, we opted for generating the amounts in digits as they would usually appear on a phone or electricity bill. This scheme relaxes the constraints on

S	→	\$ De S1 S1
De	→	(digit zero) (digit zero) dot
S1	→	zero T1 digit T
T1	→	zero H1 digit H
T	→	T1 e
H1	→	zero Th1 digit Th
H	→	H1 e
Th1	→	comma (zero TTh1 digit TTh)
Th	→	Th1 e
TTh1	→	digit
TTh	→	TTh1 e

Table 2: Grammar for generating the digit amounts

the writers and still allows us to somewhat estimate the number of words generated for each class. For the example of 120\$ given above, we would compute one occurrence for each of the following 3 classes ‘one’, ‘hundred’ and ‘twenty’.

We wrote a simple grammar (Table 2) to produce numerical amounts in the range [0.00\$ – 99,999.99\$]. The meaning of each state of the grammar (non-terminal symbols) is explained in Table 3. The terminal symbols digit, zero, dot and comma are described in Table 4. Intentionally we limited the range of the amounts to a maximum of 99,999.99\$. For increased reliability of the processing system and protection of the financial institution, greater amounts are to be rejected from the automated process so that they can be handled ‘safely’ by a human operator.

While generating amounts, the transition from one step to the next within the grammar is chosen randomly. Statistics are computed on the number of occurrences of each word class as well as on the number of amounts which are within the ranges [0.00 – 9.99], [10.00 – 99.99], [100.00 – 999.99], [1,000.00 – 9,999.99] and [10,000.00 – 99,999.99]. In order to be representative of the kind of cheques a utility company may receive, we would like to have more samples belonging to the first four intervals than to the last one. Taking these considerations into account, we adjust some weights within each state of the grammar in order to produce more samples belonging to the first intervals as well as to have a somewhat equal number of occurrences for each class

S	=	Start symbol
S1	=	Start1: produce the first digit
De	=	Decimal: to produce a decimal value (e.g. 0.40\$)
T	=	Tens: produce the second digit
T1	=	Tens1: produce second digit when previous digit was a zero
H	=	Hundreds: produce the third digit
H1	=	Hundreds1: produce third digit when previous digit was a zero
Th	=	Thousands: produce the fourth digit
Th1	=	Thousands1: produce fourth digit when previous digit was a zero
TTh	=	Tens of Thousands: produce the fifth digit
TTh1	=	produce the fifth digit when the previous one was a zero

Table 3: Meaning of each state of the grammar

digit	=	{ '1', '2', '3', '4', '5', '6', '7', '8', '9' }
zero	=	{ '0' }
dot	=	{ '.' }
comma	=	{ ',' }

Table 4: Values taken by the terminal symbols

\$446.84	\$624	\$434.80
\$70	\$89.29	\$93.77
\$75.97	\$19	\$408

Table 5: Example of randomly generated digit amounts

one	441	eleven	148	ten	157	hundred	1758
two	386	twelve	126	twenty	197	thousand	727
three	419	thirteen	133	thirty	167		
four	423	fourteen	126	forty	176		
five	433	fifteen	138	fifty	171		
six	405	sixteen	120	sixty	181		
seven	429	seventeen	143	seventy	216		
eight	405	eighteen	130	eighty	176		
nine	406	nineteen	136	ninety	190		

Table 6: Estimate of samples per class generated with a set of 3,000 cheques

in our lexicon. An example of some digit amounts generated following the rationale described above is shown in Table 5. The estimated number of samples from each word class that one might expect to collect with a set of 3,000 cheques is shown in Table 6.

As we can see from the statistics of Table 6, we get at least 120 samples for each word class with a set of 3,000 cheques. It is generally agreed that a minimum of 1,000 samples per class is needed to train a digit recognizer in a satisfactory fashion. If the same 'rule' was to be applied to our research problem, then a minimum of 25,000 cheques would be needed. Ideally the database should be collected from 'real-life' cheques which might not have such a convenient random nature as our computer generated amounts. As a result as many as 100,000 cheques might be needed for training to attain the goal of a minimum of 1,000 samples per class. This represents a major task and it is important to experiment with the design and procedures involved in the building of such a database to insure its quality and usefulness.

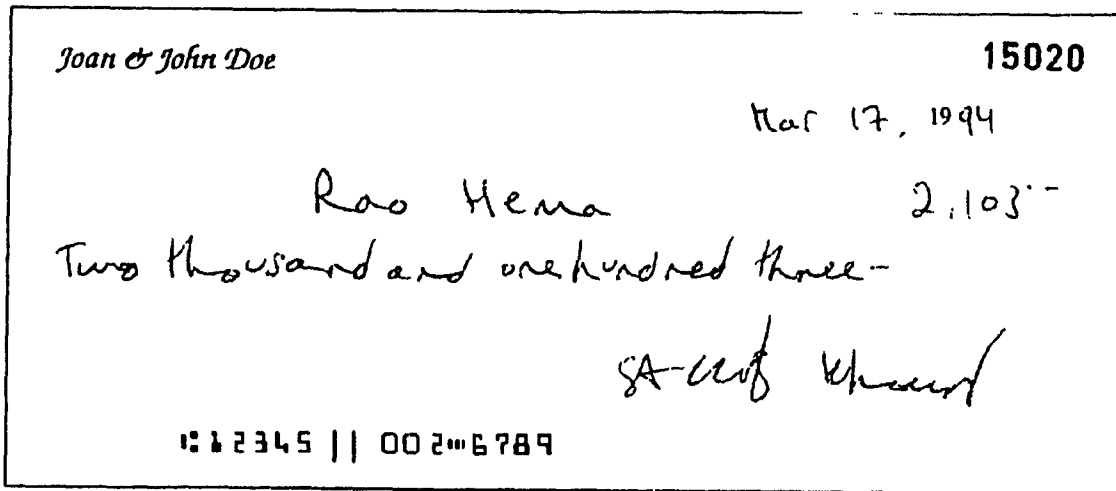


Figure 4: Example of a Concordia cheque filled by a student

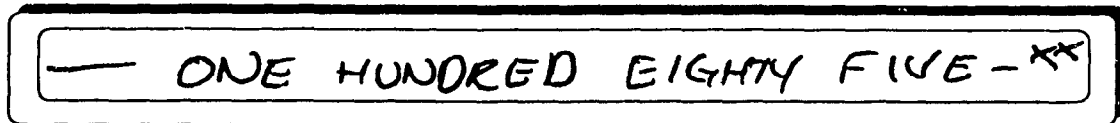


Figure 5: Legal amount written solely with capital letters

3.4 Collection of the database

In order to have our cheques filled, we visited classes given at Concordia University in the departments of Computer Science and Accountancy. Students in those classes were asked to write 3 cheques each. The amounts (in digits) for each cheque were predefined but the students were free to fill in any information they pleased in the fields 'date', 'Pay to the order of', and 'signature'. An example of a cheque filled by some student is shown in Fig. 4.

Since we are interested in developing a cursive script recognition module, as opposed to recognizing single characters, we introduced one constraint in the collection process. Students were shown a slide of "full capital letters" style of handwriting, and were asked to avoid if possible such style in favor of a more natural style of writing. Even so, approximately 5% of the cheques we collected were written solely in capital letters, one example of which is shown in Fig. 5.

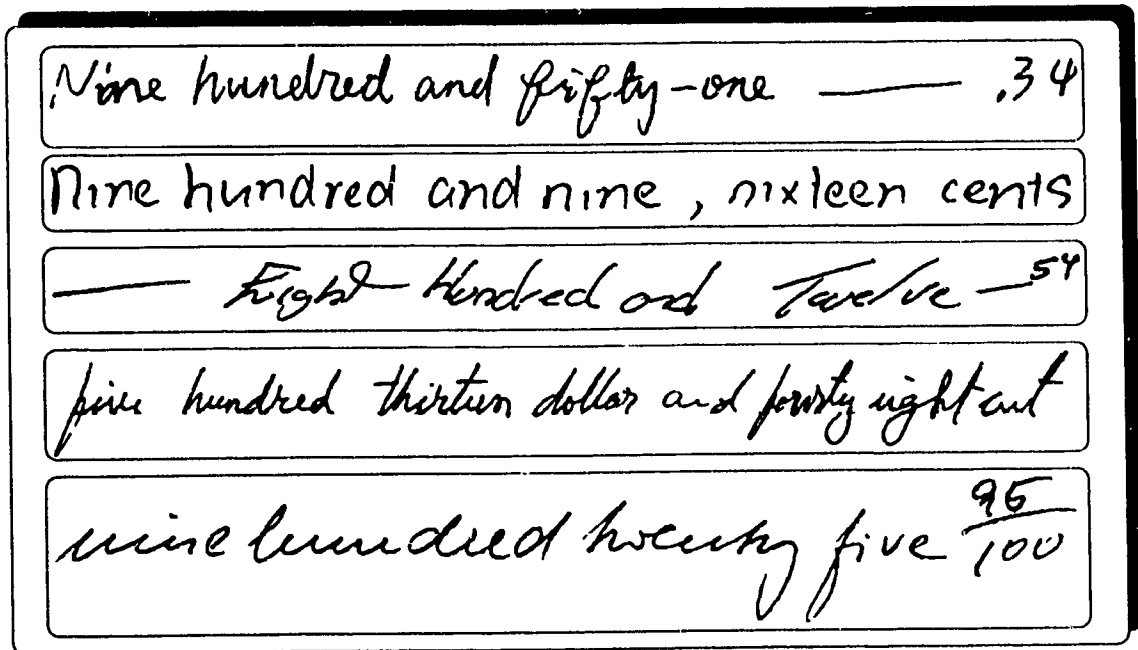


Figure 6: Samples of legal amounts

As mentioned by Hull [Hul94], the awareness by subjects that their handwriting would be used to develop automatic recognition algorithms might have introduced biases into the data. The desire of some students to perform well, may yield samples that are abnormally neat, while others may try to “fool the computer” by making their handwriting unusually sloppy. While the notion of ‘neat’ or ‘sloppy’ might be hard to define, we feel that the vast majority of the data we collected does not belong to any of these two extremes. Some typical images from our database are shown in Fig. 6.

As of today, close to 2,500 cheques written in English have been collected. The number of different writers is estimated to be about 800. We are also in the process of collecting cheques written in French. At present, the French database is made of close to 1,900 cheques corresponding approximately to 600 different writers.

3.5 Storage of information

The cheques from our database have been scanned at 300 pixels per inch (ppi) in 8-bit greyscale. We designed a user interface to extract and binarize the written information in a semi automatic fashion. For each cheque, the date, courtesy and legal amounts have been stored into their own files.

3.6 The truthing procedure

We decided to investigate a simple yet powerful way of truthing the images of legal amounts. One easy but tedious way might have been to extract manually each and every single word from the images and store them into their individual files. The same procedure would have had to be applied to the single characters as well as the numerals. Such an approach has the disadvantage not only to be inefficient but also to lose the contextual information in which the words were placed. While one might attach a message containing contextual information to each file, it is not obvious to think from the start the extent of contextual information that one might need in the future.

Since our ultimate goal is to extract the information contained in the entire legal amount image, we decided to have truthing the legal amount as a whole, rather than creating manually a multitude of smaller files. The procedure we adopted not only simplifies the truthing but also has the advantage of keeping intact the entire contextual knowledge.

In order to facilitate the tagging procedure, we developed a user interface running under the X window environment. This tool is presented in Fig. 7. The entire process of truthing a legal amount is strictly mouse-driven. Upon clicking on the image, a point is displayed on the image and its coordinates are entered in the image message. The final tagging information for the amount in Fig. 7 is shown in Fig. 8.

In the message of a tagged legal amount, there are at the moment 2 main fields, one called "WORDS" and the other "CHARACTERS". The characters ';;', ',' and ':' in the message are simply field separators that are used by some extraction algorithm

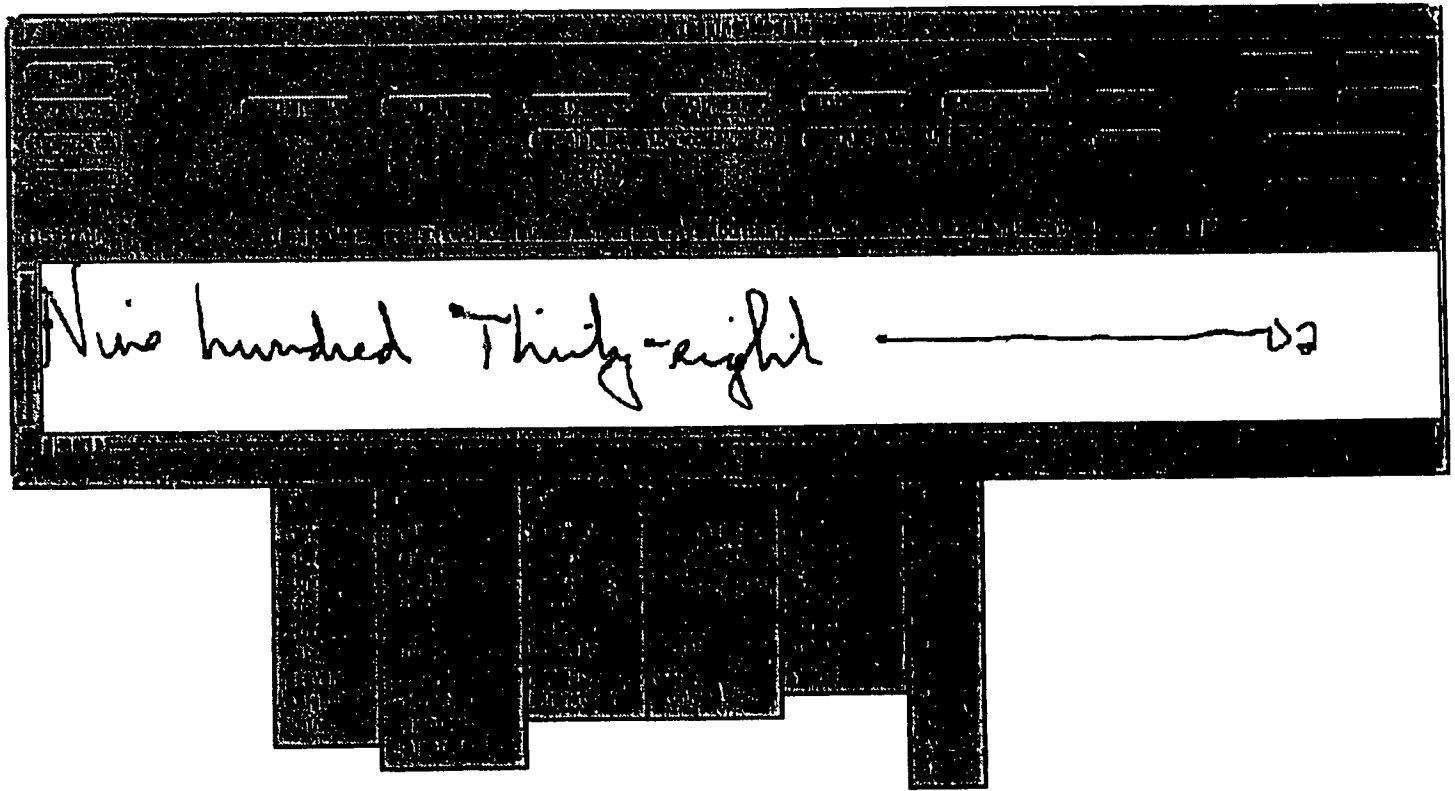


Figure 7: User interface for the tagging of the database

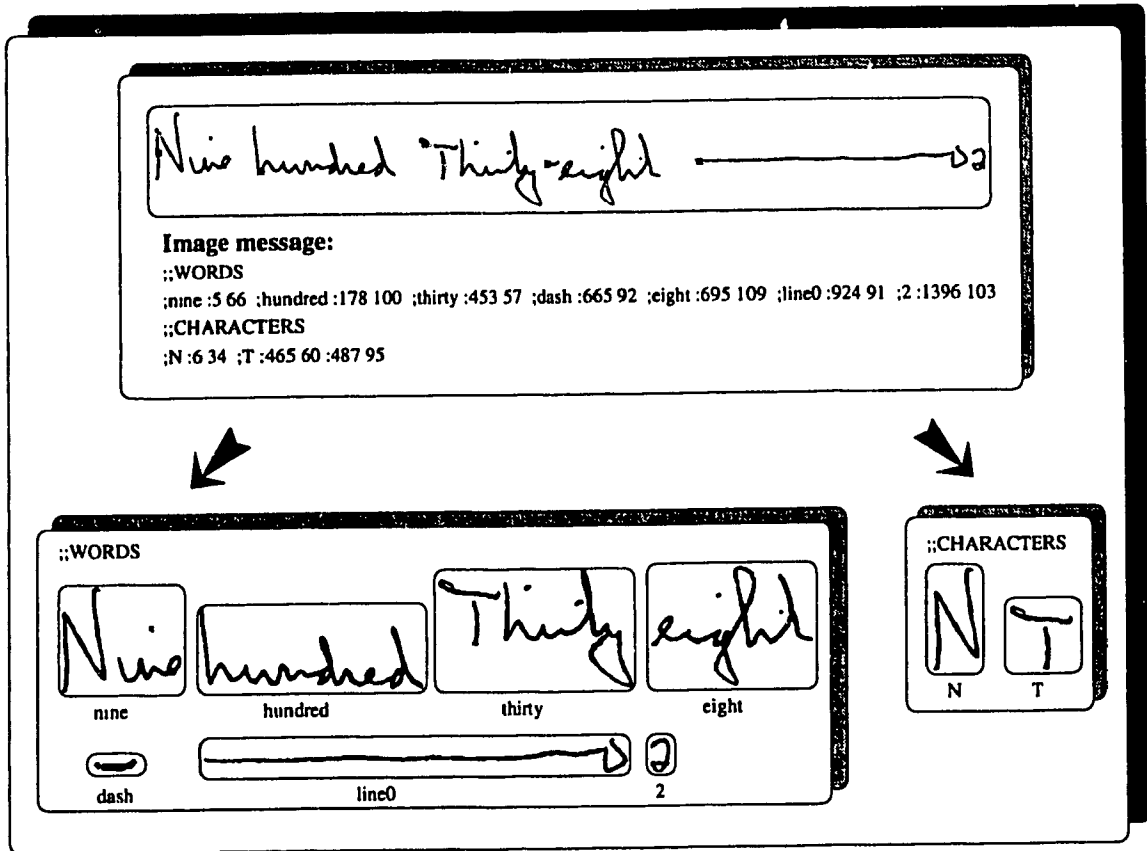


Figure 8: Image message after the tagging

to access the words and characters in the image. Under WORDS, we tag the words belonging to our lexicon as well as the lines, dashes, commas, digits, etc. . . . The exclusive set is displayed in the pop-up menus of Fig. 7. Under CHARACTERS, we tag single characters that are disconnected from words.

From a tagged image, one can extract easily any part of the entity. First the image is represented as a linked list of contours using a contour tracing procedure developed at our centre [Str93]. Then the words or characters can be extracted by using the information stored in the image message. As an example, in Fig. 8, the word 'nine' is composed of all the connected components located between the tag points (5,66) and (178,100). Similarly 'hundred' is composed of the connected components located between (178,100) and (453,57). The single characters are tagged slightly differently. For a character, all of the connected components that make up the character must be specified with a tag point. For example, the 'T' in Fig. 8 is composed of 2 connected components; the one closest to the point at coordinates (465,60) plus the one closest to the point (487,95).

Note that being able to specify more than a single connected component is relatively important in the case of alphanumerals. Indeed, 'T' often has a disconnected hat, as does the digit '5'. This specific point was a drawback in the tagging procedure adopted in [Hul94], because with their scheme, the alphanumerals could only be composed of just a single component.

Apart from its obvious advantages of simplicity and completeness, the current scheme proves to be extremely useful for training various classifiers. For example, to design a classifier that would recognize and remove dashes, lines and commas from the legal amount, in addition to their bitmap, one can extract various contextual properties, such as the location in the sentence, the proximity to other connected components or the existence of other components above or below. Similarly for the design of a character recognizer, one can not only extract the bitmap (a list of connected components) but also among other things the relative position of the character within the word to which it belongs, the position of reference lines (section 4.3.1) computed both at the sentence and at the word level and the slant computed at the level both of the entire sentence as well as of the word containing the character. The

adopted scheme will also prove useful when training an algorithm to perform the splitting of the legal amount into individual words.

The contextual information is resident within the legal amount. One can extract as much of it as one wishes at any time. Had the tagging procedure been similar to the one in [Hul94], then one would have had to know at the time of tagging the exact list of contextual information that one needed.

3.7 The final database

At the moment, we have built a database of approximately 2,500 handwritten (non printed) cheques written in English. The number of writers is estimated to be close to 800. The legal amounts from our cheques have been fully tagged with the procedure described in the previous section. From these legal amounts, for each class in our lexicon, we can extract the number of samples shown in Table 7. Additionally, we can extract the single characters represented in Table 8. In Appendix A, we show some data extracted from our testing sets. The first set corresponds to the handwriting of students of the Computer Science department, whereas the second set is taken from students in the Accountancy department. Occasionally the images of characters, words and legal amounts had to be scaled down in order to fit them into a single page in the appendix. Therefore the differences in sizes for a given character or word are actually greater than what is implied from the figures in the appendix.

In addition to the words and characters mentioned in Tables 7 & 8, one can also extract a significant number of commas, dashes, lines, etc. . . . These latter components will be used to design some classifiers based on information such as the bitmap, the location within the sentence and the proximity of other components in order to extract the punctuation marks and facilitate the segmentation of the sentence into individual words. Additionally, digits could also be extracted from our database.

one	345	eleven	110	ten	112	hundred	1207
two	287	twelve	81	twenty	149	thousand	486
three	316	thirteen	95	thirty	135	and	797
four	309	fourteen	93	forty	135	dollars	474
five	312	fifteen	97	fifty	132	only	84
six	297	sixteen	82	sixty	126		
seven	317	seventeen	96	seventy	156		
eight	283	eighteen	94	eighty	140		
nine	287	nineteen	91	ninety	112		

Table 7: Number of samples per word class in our database of legal amounts

a	237	o	362	A	6	S	178
c	18	r	359	C	2	T	269
d	1007	s	303	D	38		
e	786	t	559	E	114		
f	218	u	250	F	215		
g	152	v	191	H	94		
h	400	w	113	L	2		
i	601	x	198	N	135		
l	197	y	212	O	78		
n	729			R	3		

Table 8: Number of single characters in our database of legal amounts

3.8 French database

Using the same procedure and tools described in the previous sections, French cheques have been collected. The current database of French cheques consists of 1,861 cheques for an estimated number of 600 different writers. Some samples of legal amounts belonging to our testing database are shown in appendix .3.

3.9 Summary

We have described the design of an image database that has been collected for the training and testing of a cheque processing system. To our knowledge, this represents the only publication in the field describing a database for legal amounts extracted from cheques. When compared with previous schemes for the tagging of postal address databases, this design demonstrates a simple yet powerful procedure to keep all the contextual information intact. Moreover we are able to isolate characters composed of several distinct connected components, and this overcomes the problem of other alphanumeric character databases.

The data were scanned at 300 ppi in 8-bit greyscale. This allows for experimentation with greyscale recognition techniques.

Chapter 4

Legal Amount Processing

The current research is only one of the numerous modules needed to process a cheque successfully. A hierarchical diagram of the whole recognition system is shown in Fig. 9. Our work fits into the module dealing with the processing of the cheque amount. Even so, it is only a subsystem handling specifically the recognition of the legal amount. The hierarchical diagram of our system along with examples of the type of input image we are working with is shown in Fig. 10. The 4 different sub-modules from the legal amount processing system (Fig. 10) will be discussed in the following sections.

The proposed reading algorithm will take its roots into the results of the psychological studies presented in Chapter 2. We will attempt to develop a computational theory based on the model of the fast reader as defined by Goodman [Goo67]. Not to depart from these intentions, we shall remember the advice given by Pitkin [Pit29] in his book entitled "The art of rapid reading; a book for people who want to read faster and more accurately". He recommends us to "Read wholes, not parts. Read sentences, not words." and finally to "Read for the broadest meaning first, then for the details later if necessary".

The present project is to recognize the legal amount from cheques. Before starting to look at the stimulus, we will summarize the knowledge that we already possess. We know that we want to recognize some amount. Hence we can define the words that do belong to our lexicon. We also possess some knowledge of orthography, syntax

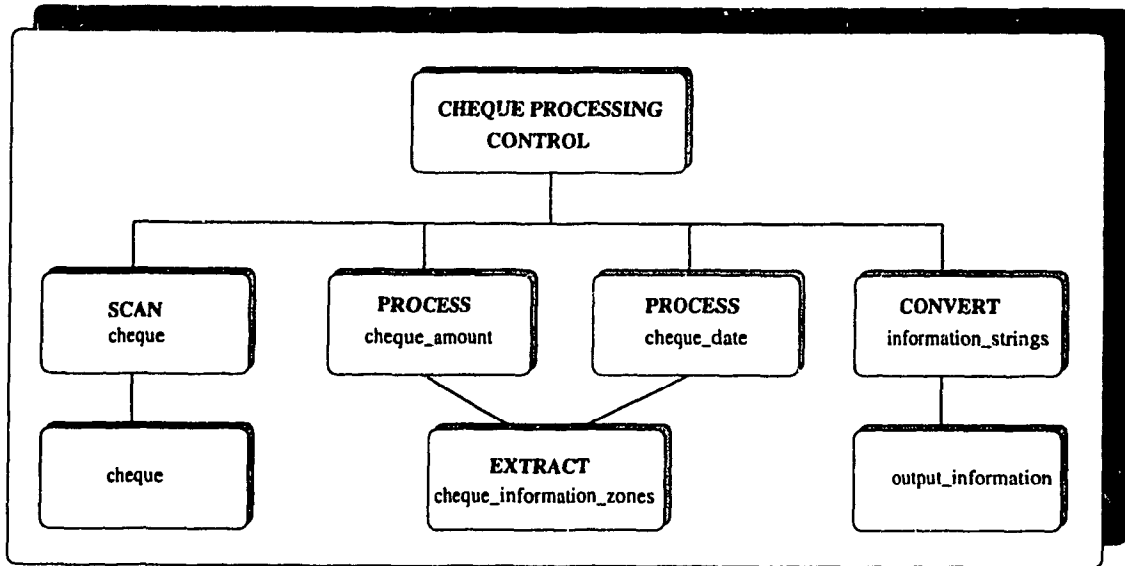


Figure 9: Hierarchical diagram of the cheque processing system

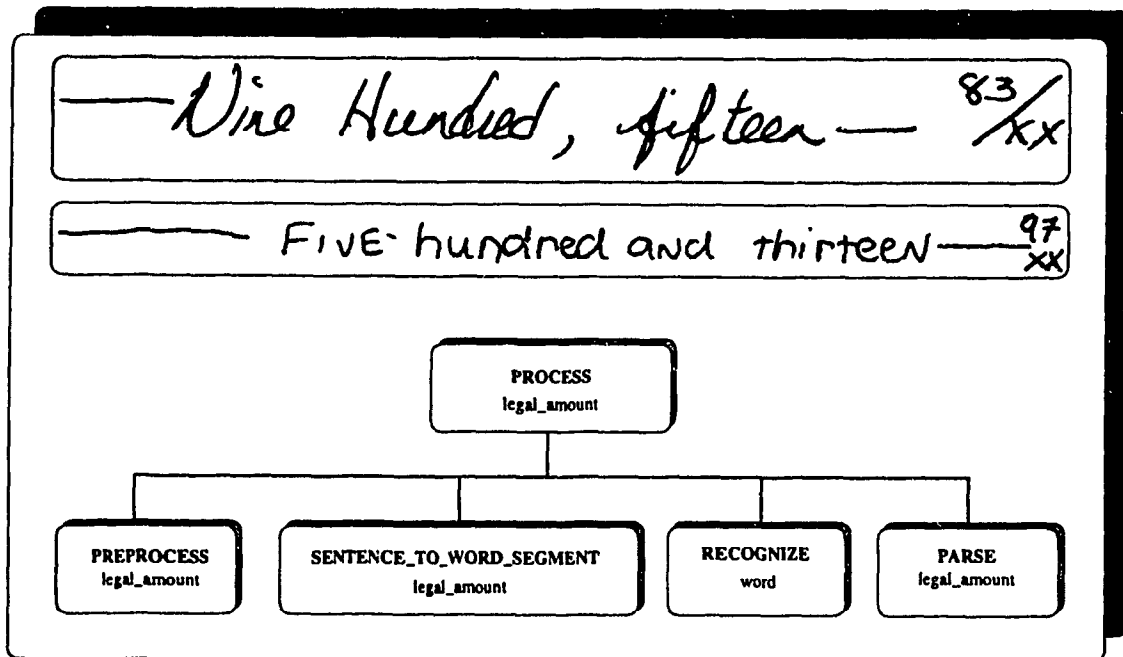


Figure 10: Processing of the legal amount

and semantics. A natural language parser specific to our lexicon, will guarantee that the syntax and semantics will not be violated. The parser is defined so that it gives some indication as to what kind of word we should expect next.

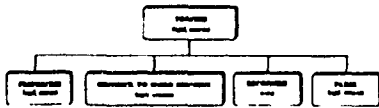
Our goal is to read sentences, therefore we shall extract some features of the whole amount, rather than attempt to segment first the input line into individual words. The primary features that we extract should be invariant enough so that they can be present in any type of handwriting style. At the same time, they are to be discriminative enough to enable acquisition of useful information. This type of features is referred to as 'character dependent' as opposed to 'writer' and 'time dependent' features [BG80]. This implies that these features are to be a characteristic of the character, and it ought to be present each time the character is present in the input. We chose to extract loops, ascenders, descenders, an estimate of the word length and strokes as our *primary features*. The length of an input word would represent an estimation of the number of characters in the word. By combining this graphical knowledge with our current expectation about the input, we attempt to construct the words. Knowledge combination will be performed by a flexible matching scheme, allowing for missing characters, misspelling, and the like.

If these preliminary features prove to be insufficient to recognize the input sentence, we can then decide to read for the *details*, since it becomes necessary. We recall that Waters [Wat77] reported studies which showed that the letters next to blank spaces were important clues for the fast reader. Therefore in our search for details, we may decide to attempt to recognize characters on the left and right of blank spaces. These characters are somehow easier to recognize than those within a word. Indeed the position of at least one of the lateral sides of the character is perfectly defined.

We recall that previous studies focus on recognizing the presence of some character classes in their attempt to recognize words. This process can translate into segmenting the words into characters or parts of characters ([Fav93]) or extracting some global features to identify the presence of certain character classes ([FS90, GL93]). Therefore all previous studies would fall into the psychological model of the reading process of a slow reader. In other words, attention is focused on characters to make up words.

The first step of our proposed word recognition approach bypasses the notion of characters. It therefore implements the psychological model of the fast reader where attention is purely focussed on words rather than characters. Our proposed scheme implies strong contextual information or an equivalent small static lexicon. Indeed we recall that a fast reader might fall into the slow reader category when confronted with text of an unfamiliar topic.

4.1 Preprocessing



Preprocessing modules are needed to compensate for the limitation of some modules in the extraction step such as binarization. Additionally we wish to normalize the input image as much as possible in an attempt to reduce the great variability in handwriting styles. Some of the processes performed are described in the following sections.

4.1.1 Baseline skew correction

An image might be skewed with respect to the horizontal axis as shown in the purposely created example of Fig. 11. Such an extreme case is not likely to happen in our application since legal amounts are generally written on the pre-drawn horizontal lines found on cheques. Moreover when scanned in an industrial setting, some mechanism forces the cheques to touch an horizontal flat bed.

Even though extreme cases are not likely to occur in our problem domain, some words within a legal amount might be found to be skewed with respect to the horizontal axis. In the examples of Fig. 12, the words “three” and “hundred” are found to be skewed when other words in the legal amounts do not exhibit any baseline skew.

For such cases, a baseline skew correction module might be desirable. Accordingly we did spend some time investigating various schemes to estimate the baseline skew

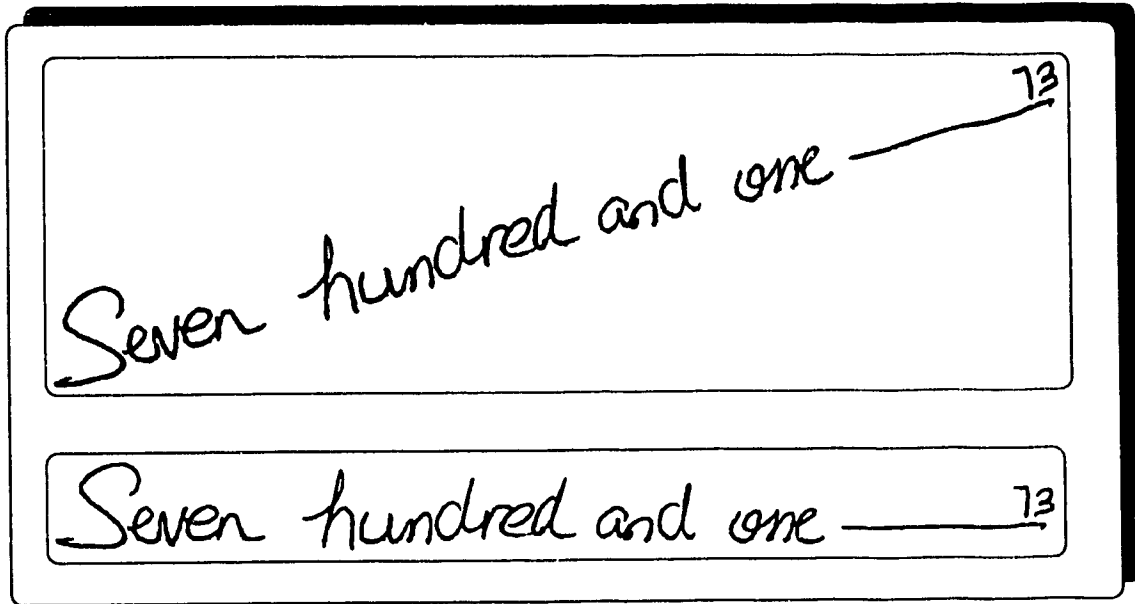


Figure 11: A baseline-skewed image and its desired skew-free version

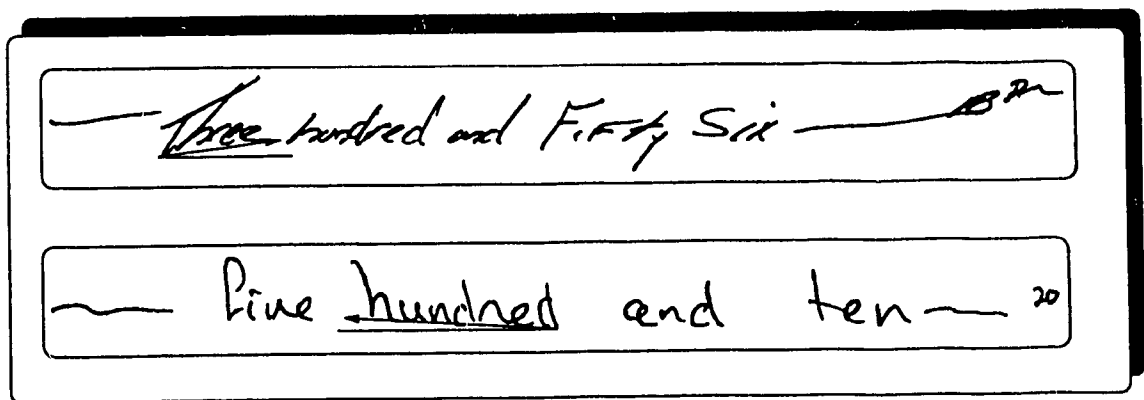


Figure 12: Baseline skewed words within "skew-free" legal amounts

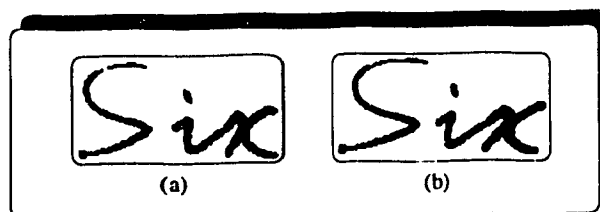


Figure 13: (a) Original image, (b) Reconnected broken strokes

of a given word. The various methods implemented as well as insights as to their suitability and performance are presented in Appendix B.

Since the average skew of our word images was rarely greater than ± 5 degrees, we decided not to 'plug' any skew correction algorithm into our system. It is to be noted that at the cost of extra time needed to further design the deskewing module as well as extra processing time for each word image, better recognition results might be obtained if a deskewing algorithm was added.

4.1.2 Broken image repair

The conversion of a grey level image into a binary image is a non-trivial task. Strokes of characters or words are not always uniform in their grey level values. Parts of strokes may be lighter than the remaining of the character or word due to the writing instrument, the writing styles, the paper background, etc.... This causes problems for the binarization module. Once in a while a fully connected character or word appears to be broken into several components by the binarization process. This can generate errors for the feature extraction and the character or word recognition. Experimentally we notice that when a break does occur, it is rarely wider than a few pixels.

We implemented a module that checks whether 2 distinct strokes are 'close' to each other. If this is the case, then the strokes are connected at their point of minimal distance. An illustration of this process is shown in Fig. 13. After the reconnection of strokes, the image is smoothed.

While this process is effective in doing what it is supposed to do, it might also connect, as a side effect, isolated letters to the rest of the word as shown in Fig. 14.

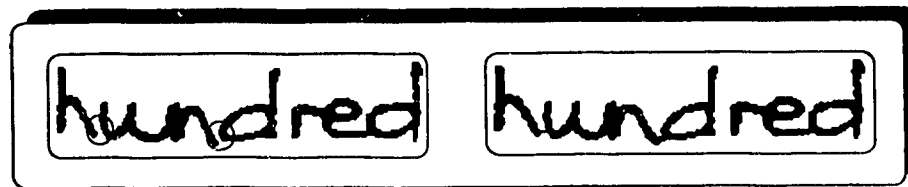


Figure 14: Side effect of the reconnection module

This is undesirable for the second step of our word recognition module (section 4.3.6) where the first and last letters of words are sought to be extracted effortlessly. The connection of isolated letters to the rest of the word might result in an increased difficulty in the extraction of those first and last letters of words.

4.1.3 Slant correction

Slant correction is the process which tries to normalize the slant of the handwriting to the vertical. Some people write with the words slanted to the right of the vertical, while others write more or less slanted to the left. Slant correction aims to normalize the handwriting to get a word with a slant of zero to the vertical. An example of original words and their slant corrected version is shown in Fig. 15. First the average slant of a word is determined by some algorithm; then the word is transformed through a shear transformation (appendix C.2).

Previous approaches

Several studies have been made to remove slant in words. The basic approach used is to look for straight line segments in words. The average orientation of these line segments is then taken as the average slant of the word. In order to find these line segments, several techniques can be used. Some people might use Hough Transform [DH72, Fav93, LL94] or analyze the chain code of a word contour [KSN93]. Still other people [Zac84, BS89] might operate by removing horizontal lines containing at least one run of length greater than one parameter *maxrun* and additionally removing all horizontal strips of height less some parameter *stripheight*. After these operations, slant is estimated in some subwindows and then averaged. In earlier work [BG83],

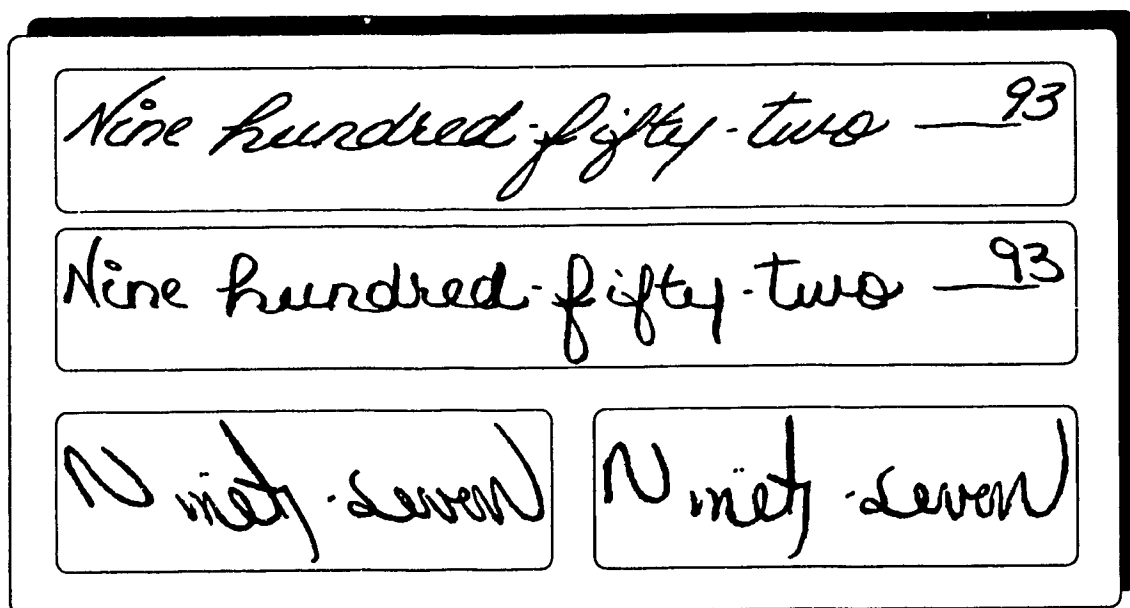


Figure 15: Slant correction

the slant was estimated by placing two horizontal thresholds through the centre of the script. Crossover points where the script crosses the thresholds are determined and an associated slope is computed from the crossover coordinates. The average of these slopes is used as the measure slant of the script.

The approaches described above seemed appropriate in the application where they were used. It was doubtful however that some of these techniques could perform reasonably well on our database due to its great variability in writing styles. Indeed for some words in our database, the centre of the script is either non-existent or carries very little information. We were also attracted into designing a scheme that would require no training and therefore no parameters to adjust.

The slanted histogram approach

Leedham and Friday [LF89] used angled histograms as a means of locating letter boundaries in real hand-printed text. We extended that approach for the slant correction of handwritten words. Experimentally one can notice several properties of

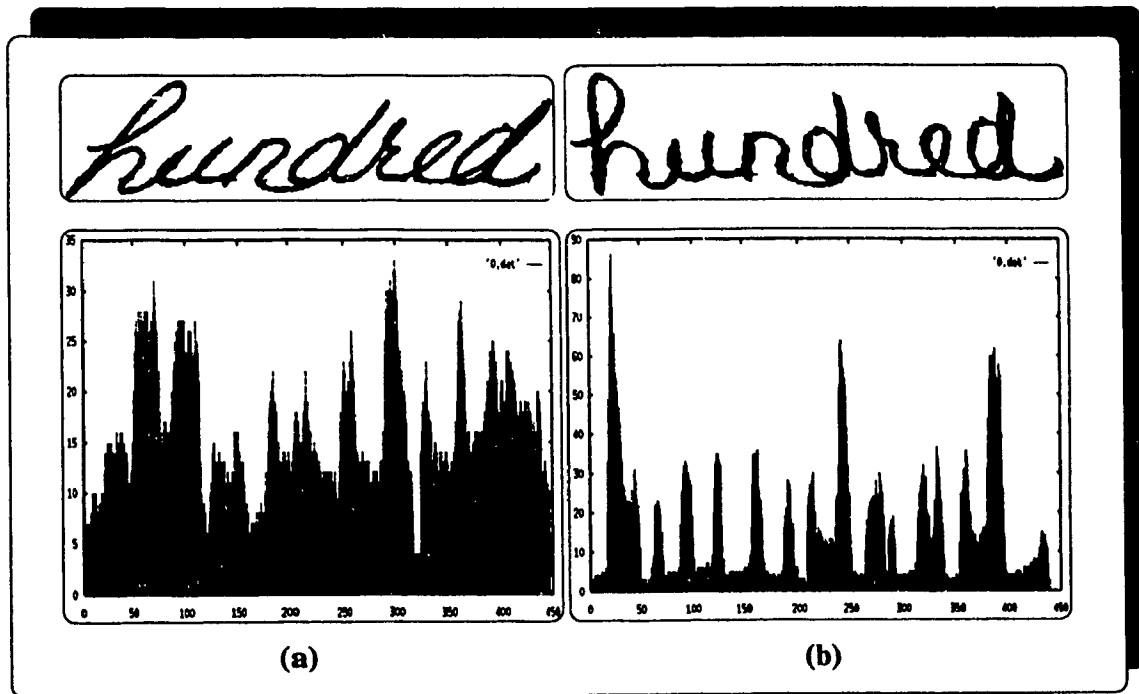


Figure 16: Vertical histogram: (a) original word (b) slant-corrected word

the vertical histogram for a slant-free word. To illustrate that point, we shall refer to Fig. 16 which shows the vertical histogram both for an original word and its slant-corrected version.

For a slant-free word, we notice that the vertical histogram shows the following properties:

- The histogram shows a certain property of periodicity. The periodicity lies in the succession of low values corresponding to the ligature between letters, the horizontal part of the letters, and the like, followed by high values corresponding to vertical strokes.
- The values for the lows correspond approximately to one or two times the stroke thickness. One time the stroke thickness corresponds to a ligature. A value of twice the stroke thickness should correspond to the middle part of a letter such as 'o', 'c', etc. . .

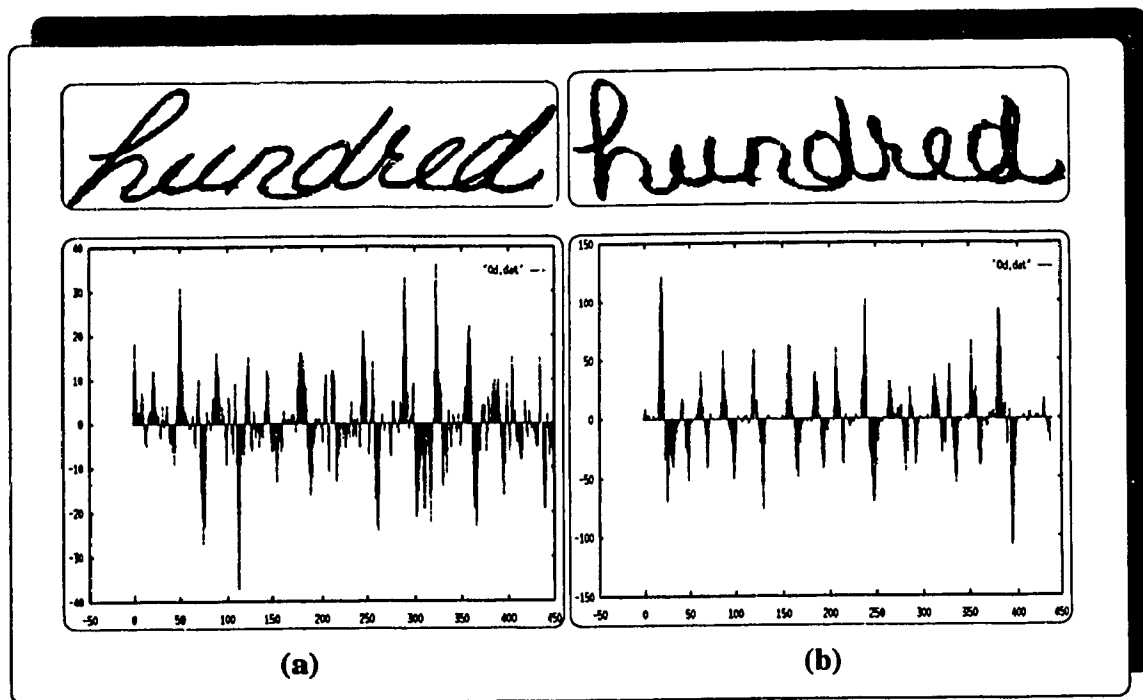


Figure 17: Derivative of vertical histogram: (a) original word (b) slant-corrected word

- There are two kinds of 'high' value in the histogram. A large value corresponding to long vertical strokes (ascenders, descenders), and a smaller value corresponding to short vertical strokes from 'main-body' letters such as 'n', 'r'.

Additional properties can be derived from the analysis of the histogram of the derivatives (Fig. 17). The latter shows a succession of sharp (great) positive value followed by a sharp negative value. A large positive value corresponds to the transition between a horizontal part of the word to a vertical part.

From these properties defined empirically, one can design several schemes to find the one slanted vertical histogram corresponding to the average slant of the word. Our current heuristic is to look for the greatest positive derivative in all of the slanted histograms computed. Once the average slant has been found, the image is corrected through a shear transformation (Fig. 15).

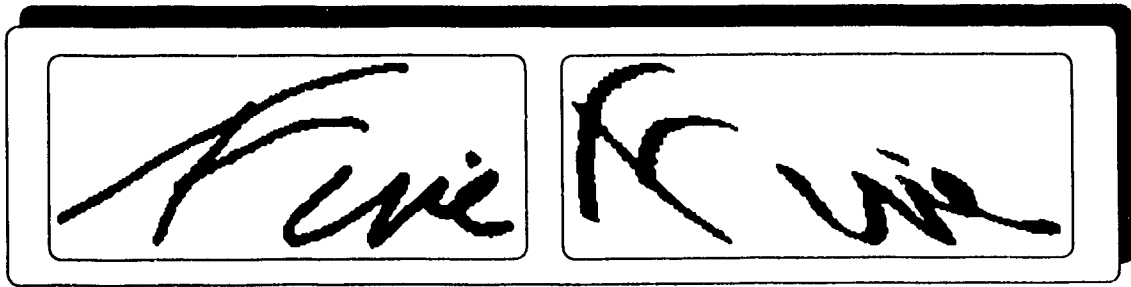


Figure 18: Example of a word for which our current slant heuristic fails

Experimental results

The simple method illustrated here performs extremely well on almost every single images in our database. However we could find a few exceptional cases where our current heuristic failed. These are words with extra fancy long strokes such as the one shown in Fig. 18.

If we were willing to treat these rare cases, then one could be willing to spend some time fine-tuning the current heuristic. One possible solution could be to consider the top n greatest positive derivatives for any given histogram. The value assigned to a given histogram could then be the average or median of these top n greatest positive derivatives as opposed to considering solely the greatest one. However, such images were so rare in our database that we did not feel worthwhile to spend some extra time fine-tuning our algorithm.

The algorithm has the potential to be efficient since all of the slanted histograms are computed through a single pass over the image. Its complexity is therefore $O(n)$ with n being the number of (foreground) pixels in the image. Full details of the algorithm are given in Appendix C.

4.1.4 Smoothing

Our smoothing procedures can be divided into two categories; e.g. those dealing with single pixels and those dealing with more than one pixel.

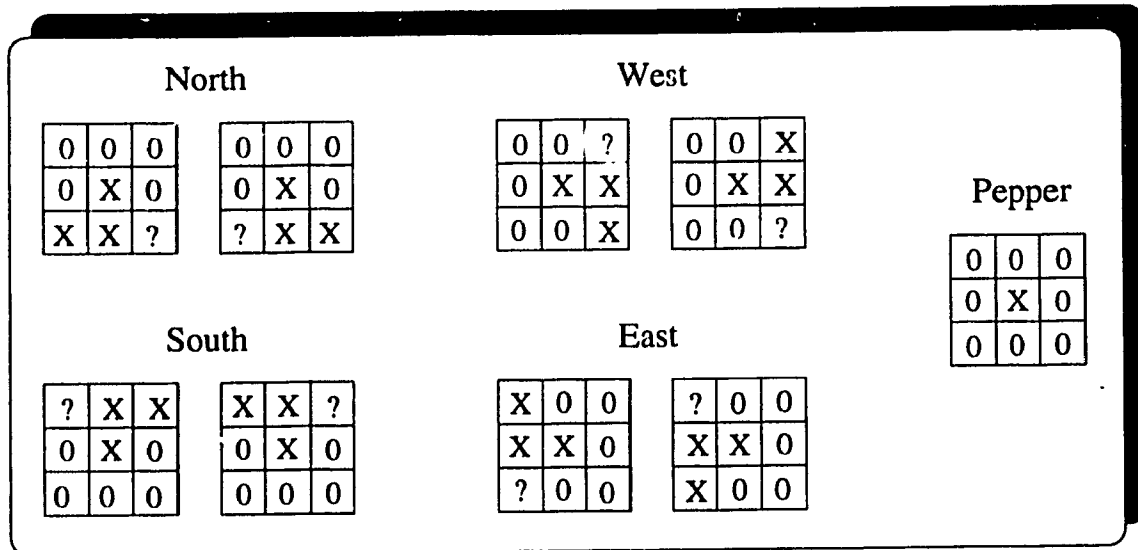


Figure 19: Situation for the removal of black pixel

Single pixels

Some masks are passed over the image to decide whether pixels should be removed or added. A black pixel is removed when it matches one of the situations described in Fig. 19. Similarly a white pixel is removed, that is a black pixel is added whenever there is a match with one of the masks depicted in Fig. 20. The north, south, east and west "remove situation" only differs in the rotation by 90° of the same basic 2 masks. In the masks, the letter '0' stands for a white or background pixel, 'X' stands for a black or foreground pixel and '?' can take either value black or white.

To apply the masks, a white border one pixel wide is first added to the image. Then starting from the top left corner of the image, we scan to the right and down. We look for transition pixels from background to foreground or vice versa. Indeed the masks need be applied only to transition pixels. Other pixels are never altered in a smoothing procedure. When a transition is found, we apply the smoothing masks to the black as well as the white pixel of the transition.

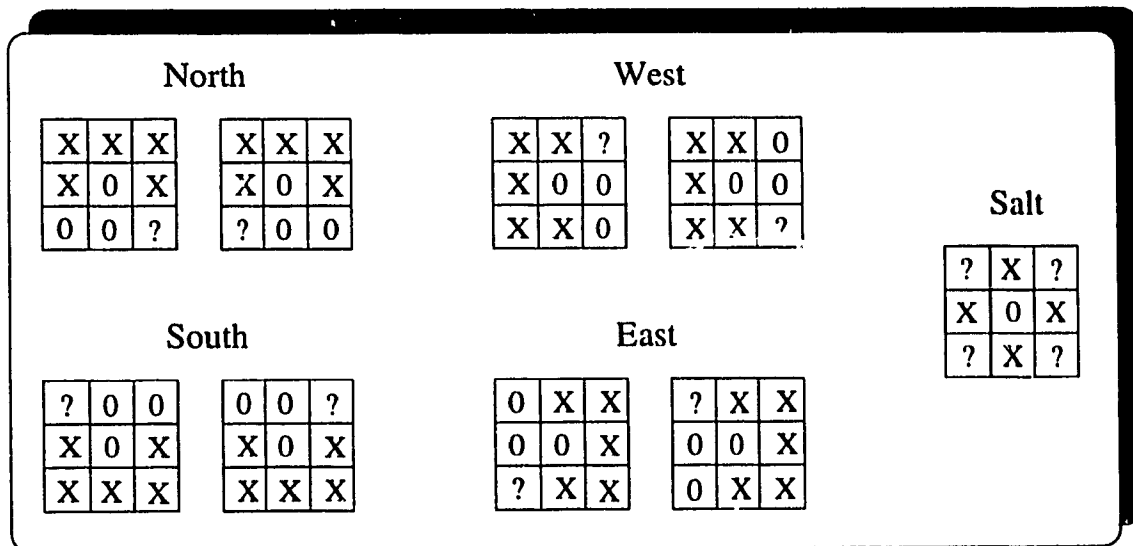


Figure 20: Situation for the removal of white pixel

Several pixels

In some cases, we might want to remove or add more than one isolated pixel. For example in Fig. 21(a), we wish to remove those consecutive pixels that are circled. It is important for us because we use the number of runlengths in the rows of the image for the computation of the reference lines as well as the estimate of a word "length". The proper search for the reference lines will dictate whether or not we are able to detect ascenders and descenders in a word image. The number of horizontal runlengths is also an important feature for our word classifier. So we defined a simple smoothing operation that process the chain codes of both outer and inner contours. The smoothed image is shown in Fig. 21(b).

The smoothing operation is described in Fig. 22 assuming that the segment shown is scanned from left to right. The processing is similar when the segment is scanned from right to left.

4.1.5 Noise removal

This module removes the small spots from the image that could not be removed by the smoothing procedure. The contour tracing algorithm [Str93] is run on the image

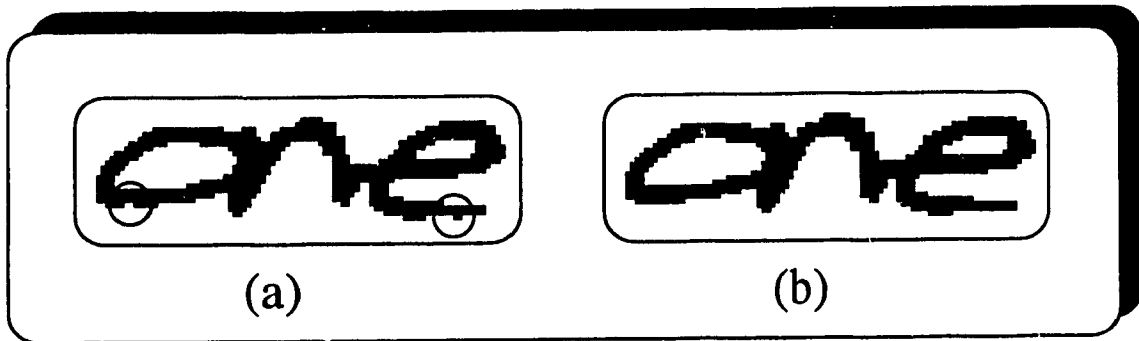


Figure 21: (a) Original image (b) Smoothed image

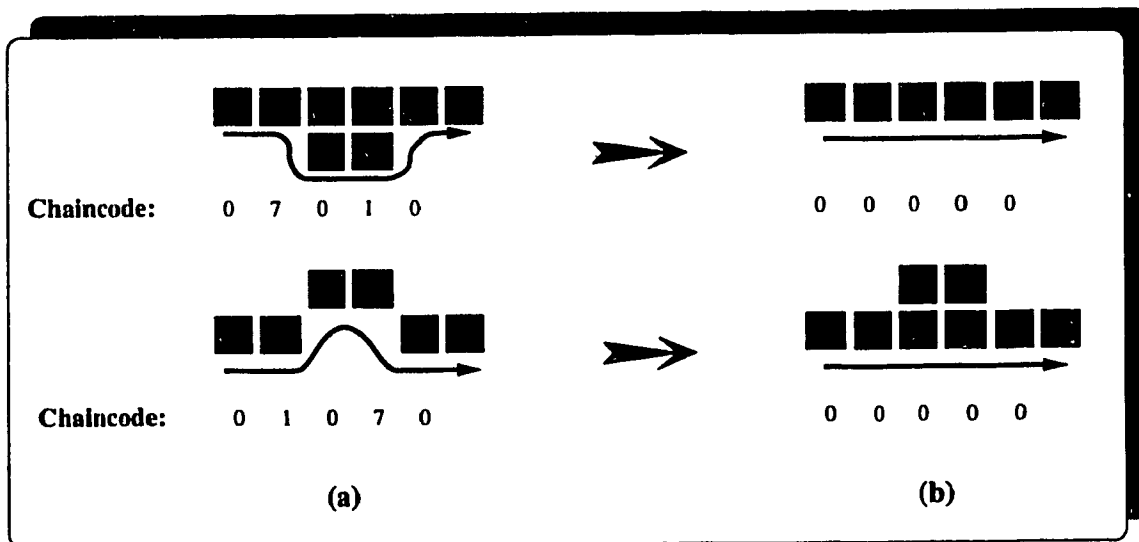


Figure 22: Contour: (a) original (b) smoothed

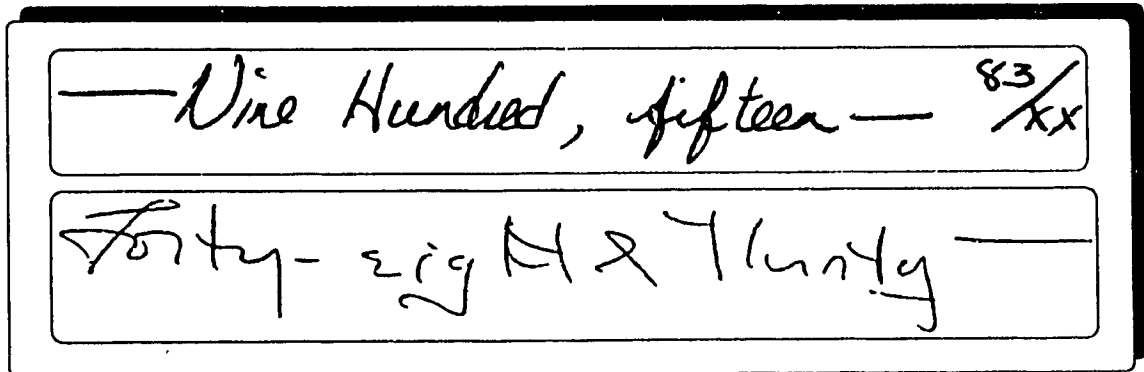


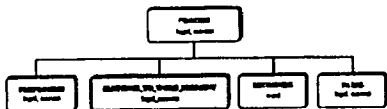
Figure 23: Examples of legal amounts to be segmented

to get the list of all the connected components. Those components whose mass is less than a given threshold are removed.

4.1.6 Average stroke thickness

The average stroke thickness is computed as defined in [Sch92]. First it is assumed that the stroke width is the same throughout the entire image. Second we assume that the set of all the components in the image can be modeled as a line with length l and width w . Then the number of black pixels P is equal to $l \times w$ and the circumference C of the shape is equal to $2(l + w)$. From the preprocessing stage, the values of P and C are known for all connected components. Solving these equations, the stroke width, w , can be readily determined.

4.2 Segmentation of the amount into words



The SENTENCE.TO.WORD.SEGMENT module (Fig. 10) is called upon to segment the sentence into words. From the legal amounts in Fig. 23 one can see that various modules need to be developed in order to recognize lines, commas, fractions, dashes and '&' sign.

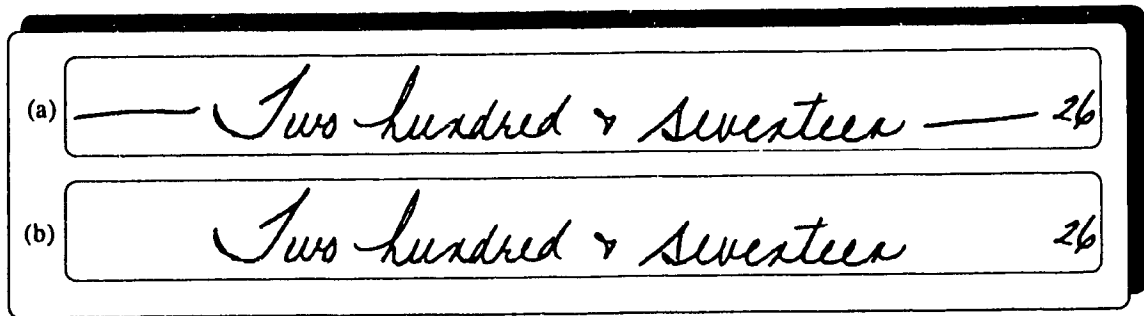


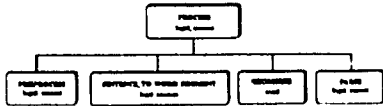
Figure 24: Line removal: (a) Original(b) Lines removed

The segmentation module produces a ranked list of paths based on the detection of lines, punctuations marks, spaces between connected components and the knowledge about the legal amount structure [BBD⁺93, SC94]. The knowledge about the lexicon (maximum, minimum length of words) and the possible existence of digits or fractions (e.g. $\frac{30}{100}$) at the end of the legal amount restrict the number of possible interpretations.

Due to the necessity of having a RECOGNIZE module to validate the various paths generated, priority has been given to develop a word recognizer. Various classifiers still need to be designed and trained to recognize and remove punctuation marks such as commas and dashes or signs such as '&' among other things. As an example of such modules, we designed and trained a Bayesian classifier based on simple features (i.e. aspect ratio and average vertical density) to recognize and remove the lines that people have the tendency to write at the beginning or end of the legal amount (Fig. 24).

Thanks to our simple, yet extremely powerful truthing procedure applied to our database, the ground work has been completed for the implementation of this module. For example, in order to design a classifier to remove commas, a database of all the commas from our data can be generated with all the necessary contextual information; and this can be done absolutely effortlessly. The proper implementation of the segmentation module is left as further work.

4.3 Word recognition



Given a list of paths produced by the segmentation sub-system, each of the paths is parsed and a new ranked list of paths is generated with their associated confidence value. The RECOGNIZE module is called upon to perform the word recognition. For a given input, it outputs a ranked list of classes.

Previous studies typically focused on the notion of characters in order to recognize words. Two approaches are generally taken. The first one involves segmenting the input word into individual characters or parts of characters [BS89] which are then sent to a character recognizer. The combination of these recognition results then produces a ranked list of possible words. The second approach involves scanning the whole word and extracting features to hypothesize the presence of certain character classes [FS90]. The use of Hidden Markov Models (HMM) [GL93, GBL93] also falls into this second scheme. While techniques based on the notion of characters have the ability to recognize word classes for which no samples are present in the training set, they have either to rely on the results of the difficult segmentation stage or tend to be sensitive to spelling mistakes, missing characters, poor handwriting and the like.

In the field of cheque processing, we possess a limited static lexicon and therefore alternate, less general, and possibly more robust approaches can be investigated. Our computational theory is based on the psychological model of reading of the fast reader (Chapter 2). The fast reader in a first step, bypasses the notion of characters to focus solely on words. If this proves to be insufficient, then the notion of characters is called upon to identify those letters located next to blank spaces (first and last characters of words). Eventually if these 2 previous steps prove to be insufficient to clearly identify the input word, then a more traditional word recognition approach based on the notion of characters would have to be applied (psychological model of the reading process for a slow reader).

The hierarchical diagram for the word recognizer is shown in Fig. 25. If we were to recall the model of reading of the fast reader as defined by Goodman, then our

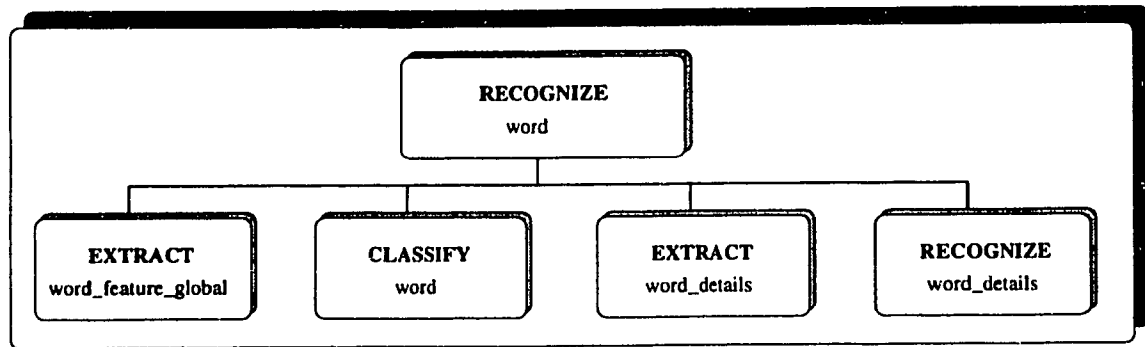
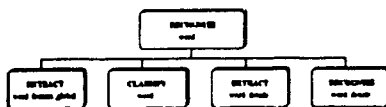


Figure 25: Hierarchical diagram of the word recognizer

extraction of global features match the sampling of a few graphical clues from the graphical input. Then we try to identify the word through the “classify” sub-module. If these few global features prove to be insufficient to identify clearly the input, then details are extracted. In our implementation, the details are going to be the first and last letter of words. These characters are then recognized with the “recognize” sub-module. The integration of results of the word recognition with the character recognition is performed in the “recognize” head module.

4.3.1 Global word features



At the moment we extract 7 types of global features: ascenders, descenders, loops, an estimate of the word length, as well as vertical, horizontal and diagonal strokes. These features are not so different from those used in other studies [FS90]. The main difference lies in the fact that these features do not present a means to hypothesize the presence of some character classes but rather are going to be used ‘as is’ to identify words.

Reference lines

The reference lines delimit the main body of the word (Fig. 26). In order to determine those lines, we compute the histogram of the number of runlength segments for each

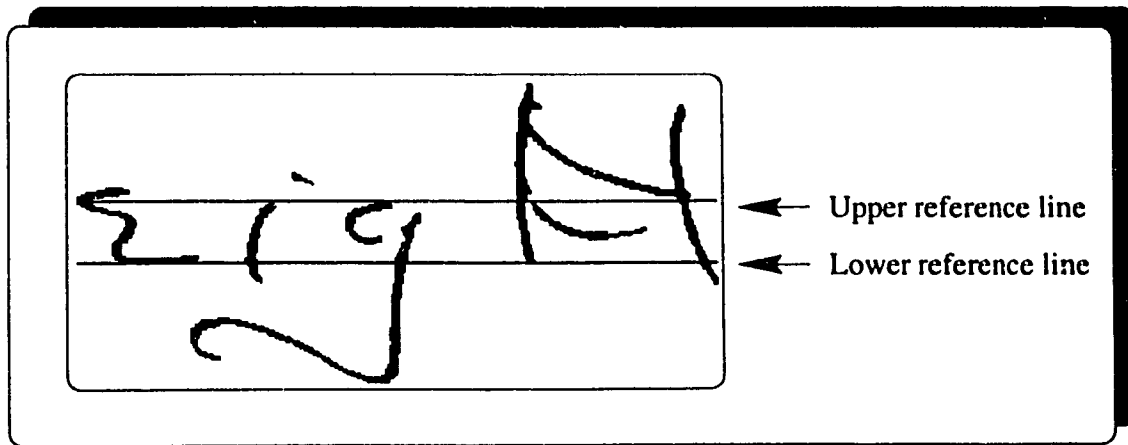


Figure 26: Lower and upper reference lines

row in the image. From the maximum value of this histogram, we locate the reference lines at the 50% cut off points.

The proper computation of these lines has a significant meaning to our approach. Indeed the location of the references lines will determine whether or not the input word has ascenders or descenders; which in turn will influence whether or not the word will be adequately recognized.

For that reason, one of the preprocessing steps is to smooth the contours of the input word. Additionally we integrate as much contextual information as possible by calculating the reference lines for a given input word as a combination of the lines computed on the entire legal amount and those computed on the word taken in isolation.

Ascenders, descenders

Empirically we noticed that the main body of a word is not uniform in height over the entire width of the word. As an example, one can consider the word shown in Fig. 27. The letter 'n' in that word is almost entirely below the main body while the letter 'e' is emerging above the top reference line. Also one can notice that the last letter of the word, the letter 'd' has a trailing tail that goes well below the lower reference line. Obviously we do not wish those characters to trigger the detection

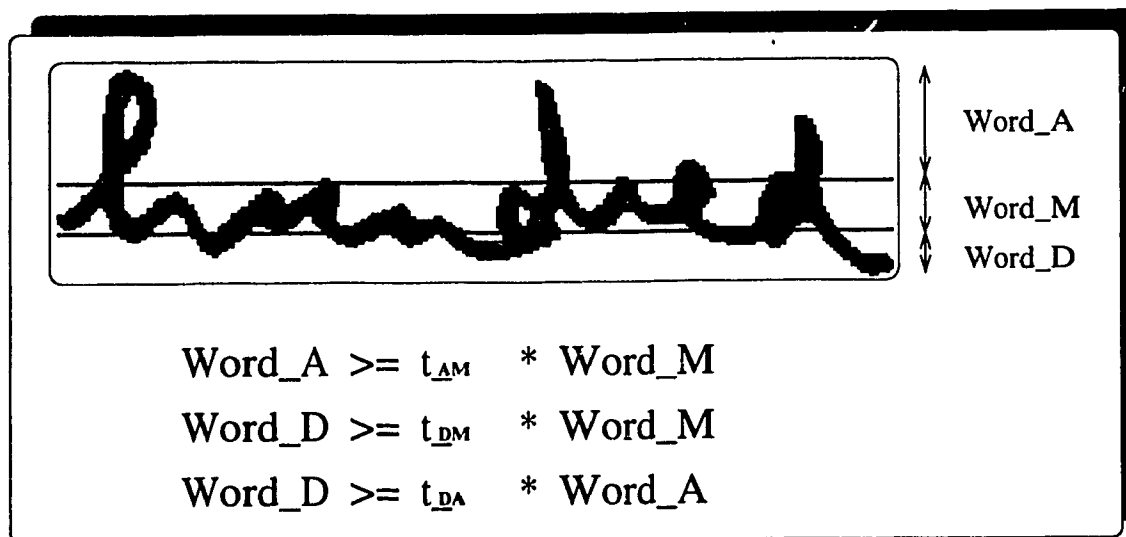


Figure 27: Ascender and descender bodies

of ascenders or descenders. Consequently we need to set some thresholds for the detection of ascender and descender bodies. These thresholds have been determined empirically and are expressed as a percentage of the main body height. The three conditions to be satisfied for the detection of ascender and descender bodies are shown in Fig. 27. The three variables t_{AM} , t_{DM} and t_{DA} respectively stand for threshold Ascender body to Main body, threshold Descender body to Main body and finally threshold Descender body to Ascender body.

If the ascender body height is smaller than the ascender threshold then the ascender body height is set to zero and we do not look for the existence of ascenders. The same procedure applies to the detection of descenders. An ascender (respectively descender) body is further subdivided into 3 equally sized buckets. We look for ascenders by following the upper outer contour of the word. An ascender is detected if the contour goes above the first bucket. The coordinates of the points where an ascender starts and finishes are stored. For coding into a feature vector, an ascender is said to occur at the midpoint between its starting and finishing points. The features extracted from the handwriting samples of Fig. 28 are shown in Fig. 29.

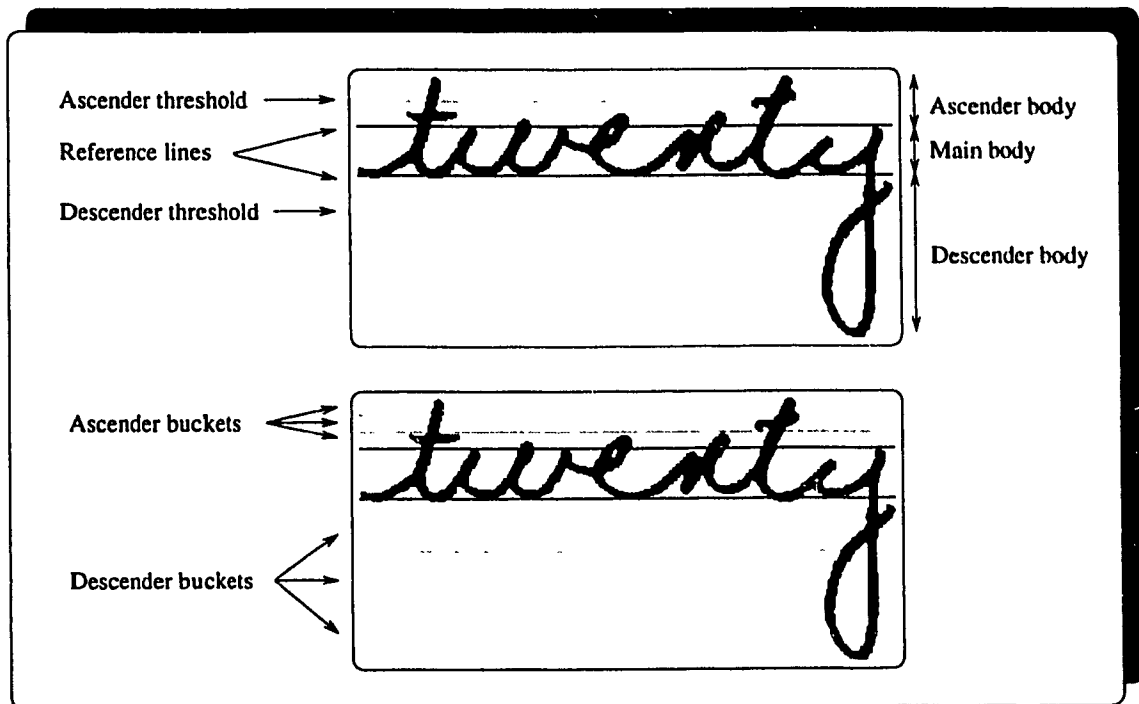


Figure 28: Ascender and descender thresholds and buckets

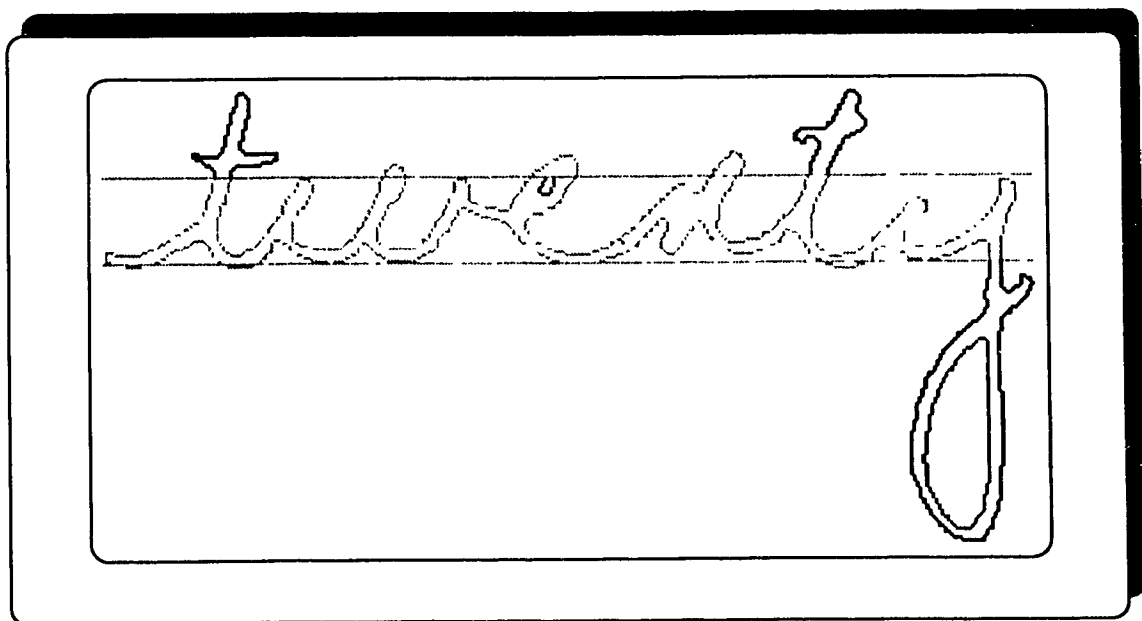


Figure 29: Ascender, descender and loop features

Loops

Loops are detected simply as being the inner contours (Fig. 29) obtained by running the contour tracing algorithm developed in our centre by Nick Strathy [Str93]. We consider solely those loops that are located within the main body of the word.

Word length

Our estimate of the word 'length' is the number of 'central threshold crossings', or in other words the number of runlengths of black pixels in the middle of the main body. For increased robustness, we take the combination of the run length counts at 3 different rows within the main body of the word.

Horizontal, vertical and diagonal strokes

Strokes are extracted using Mathematical Morphology (MM) operations. For an introduction to the basic operations of MM, the interested reader is referred to Appendix D.

The sizes of the structuring elements have been determined empirically and are expressed relative to the average stroke thickness of the image. For each of the 4 stroke classes (horizontal, vertical, south-east and south-west), we first open the image with an element whose length is a multiple (t_{str1}) of the average stroke thickness. We then close the resulting image with an element whose length is a fraction (t_{str2}) of the average stroke thickness. We perform this latter step in order to connect neighboring features. Finally we prune the resulting features by removing those connected components whose height (resp. width for horizontal strokes) is less than a fraction (t_{str3}) of the main body height (Fig. 30)

4.3.2 Feature vector

In our classification scheme, we differentiate between the *input feature vector* and the *class feature vector*. The input feature vector is computed when features are extracted from some input image. When we attempt to classify the input word, for each class in our lexicon we will convert this input feature vector to the class

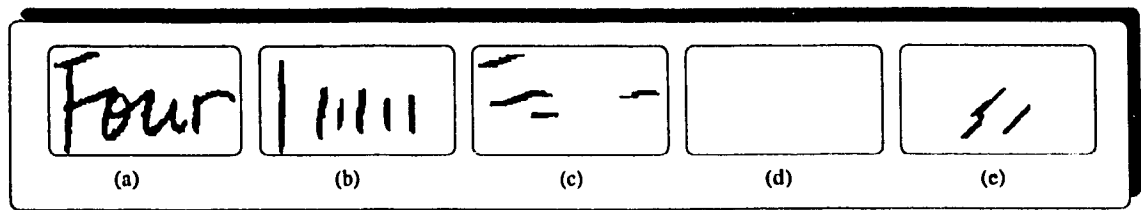


Figure 30: Stroke features: (a) original image (b) vertical (c) horizontal (d-e) diagonal

feature vector specific to the class being considered. The differences between these two vectors will be clarified hereafter. The word feature vector consists of the 11 following components:

- The relative position of the ascenders (array of 100 buckets)
- The relative position of the descenders (array of 100 buckets)
- The relative position of the loops (array of 100 buckets)
- The number of ascenders
- The number of descenders
- The number of loops
- The word length
- The relative position of the vertical strokes (array of 100 buckets)
- The relative position of the horizontal strokes (array of 100 buckets)
- The relative position of the south east strokes (array of 100 buckets)
- The relative position of the south west strokes (array of 100 buckets)

When an ascender is detected in an input image, its position relative to the input width is computed. Let us suppose that an ascender has been detected and it is located at 78% of the total word length, then we set the 78th bucket of the `ascender_position` array to '1'.

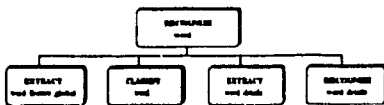
The only difference between the word feature vector and the class feature vector is the number of buckets used to express the position of the ascenders, descenders, loops and strokes. For example, let us suppose that an ascender is detected at the position 85% in some input word. When it is matched to the class 'and' which is a 3-letter word, the detected ascender will be converted into an ascender at the third position in the word. When the input is matched to the class 'hundred' which has 7

letters, then the ascender will be converted into an ascender at the 6th position.

The process of extracting features, producing the input feature vector and converting it to the class feature vector to match with the class “one” is illustrated in Fig. 31. In order to simplify the figures, we only consider the ascender, descender and loop features. Note that the class feature vector length is specific to the class being considered.

The stroke features are separated into 3 kinds according to their location. We compute a position stroke feature subvector for the ascender, main and descender bodies.

4.3.3 Probabilistic classifier



We investigated the possibility of using a probabilistic scheme similar to a Bayesian classifier for the classification of our data. For that scheme we used solely the ascenders, descenders, loops and word length features. Since our features might not be entirely independent from each other, we will refer in this section to *probabilistic scheme* and *probabilistic estimates* as opposed to *Bayesian scheme* and *Bayesian estimates*. However the computation of the estimates is exactly the same as the computation of Bayesian estimates.

Upon receiving an input image, its (word) feature vector is computed. This feature vector is then sent to the classifier. For each class in our lexicon, the feature vector is converted into a class feature vector which is sent to a module that computes the similarity measure between the input and the class under consideration. The basic algorithm for the classifier is as follows:

Given an input image:

1. Compute the word feature vector
2. For each class in the lexicon:
 - 2.1 Compute the class feature vector
 - 2.2 Compute the probability for the current class

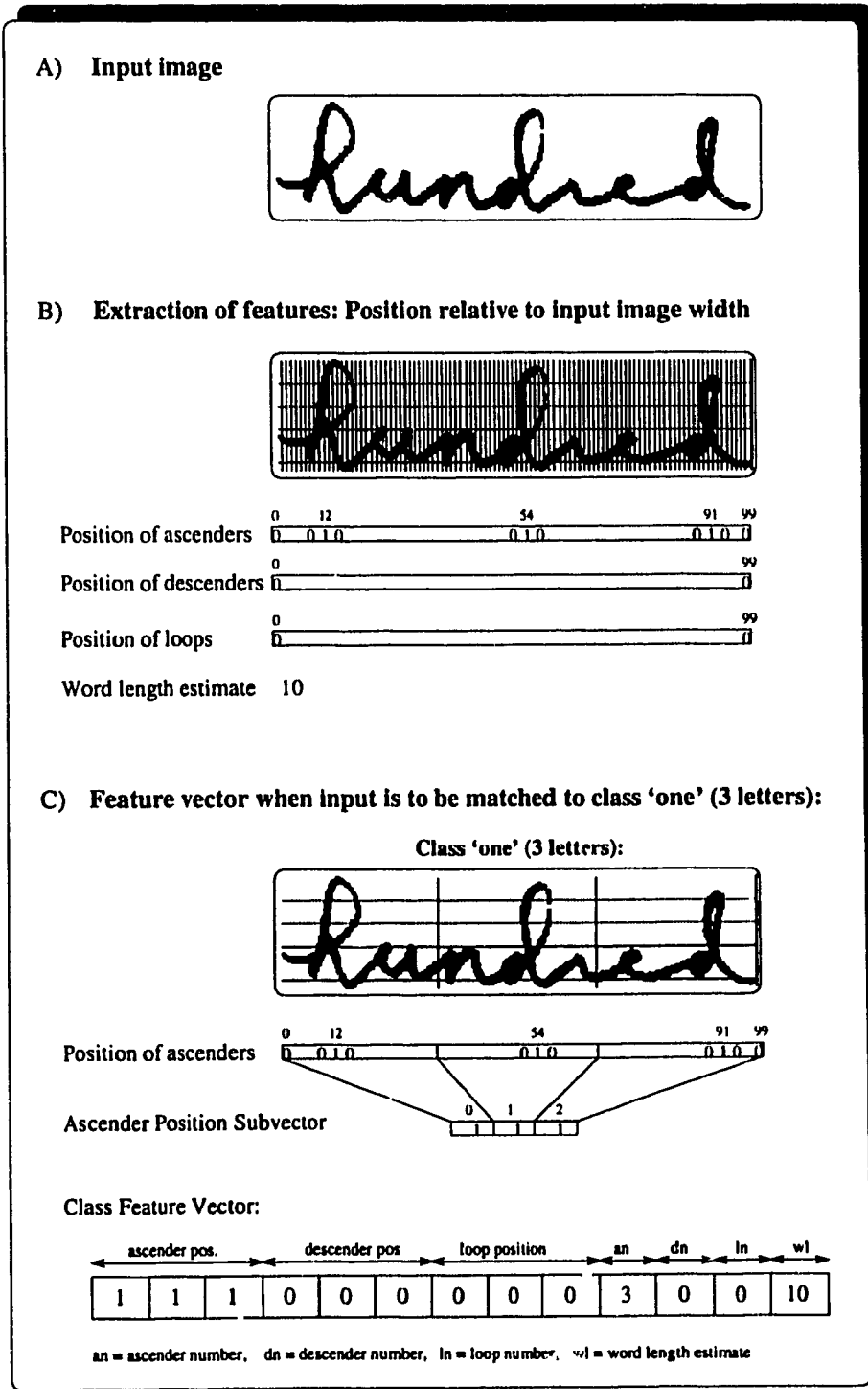


Figure 31: Creating the feature vector for a given class

3. Output the ranked probabilities

In section 4.3.2, we described how to compute the word feature vector and convert it to a specific class feature vector. We will now specify how we compute the probability of a given class using the class feature vector from some input image.

Using a training database, we computed for each class the probabilistic estimate of each feature. The matching between a feature vector and a class is done in two steps. First we compute the probability for each feature component, then these probabilities are combined assigning different weights to the various features.

The process of computing the class probability for a given input image is described in Fig. 32 D. To simplify the figure, we assume that we are working with a single feature (the position of the ascenders). In that figure we compute the probability of some input image (hundred) when matched to the class 'thousand'. The values given as the probabilistic estimates represent the probabilities that a feature be present at a given location. If the probabilistic estimate for a given position is greater than a certain threshold (say 50%), then we can assume that the feature ought to be present. Therefore to each vector of probabilistic estimates, we associate a vector of binary values which indicate whether or not a feature should be present at a given position. This procedure enables us to introduce different weights for the *match* or the *no match* of features. The probability is computed as indicated in Fig. 32 D. The probabilistic estimate p_i for a given position, represents the probability that a feature be present at that position. Therefore if a feature is present in the input, then we use p_i , else $1 - p_i$ in the formula mentioned in Fig. 32 D.

The probability for the position features for descenders and loops are computed in a fashion similar to the one described in Fig. 32 for the ascenders. The probability for the number of ascenders and descenders is simply the Bayesian estimate computed on the training database for that feature. The probability for the number of runlengths is the sum of the Bayesian estimates over a 'small' window. For example if the number of runlengths in the input image is 7, then the probability for the runlengths for a given class is the sum of the Bayesian estimates for that class for the values 6, 7 and 8. We allow this 'smoothing' for increased robustness since the standard deviation for the number of runlengths is much greater than the one for the number of ascenders

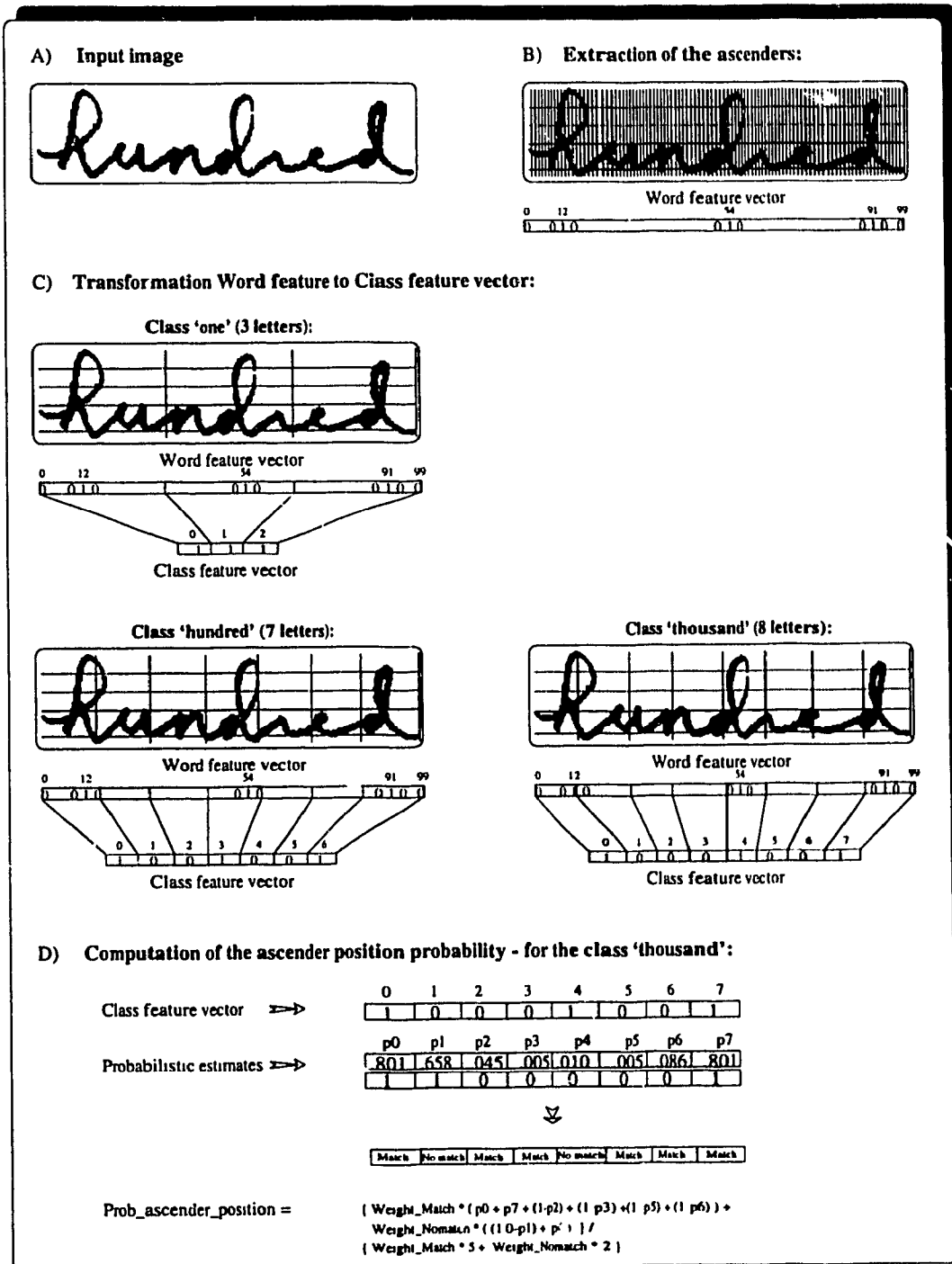


Figure 32: Probabilistic classifier: probability computation

or descenders.

Once the probabilities of each feature component have been computed, we combine them to produce the overall probability for the class under consideration. The combination is a weighted sum of the probabilities.

While this scheme produced satisfactory results, the clustering of the training data into subclasses had to be done manually. Unfortunately this is contradictory to our original wish to produce a fully trainable system. Therefore we investigated a different scheme, namely the nearest neighbour classifier.

4.3.4 Nearest neighbour classifier

A given input image is processed as follows. Features are extracted to create an input feature vector. For each class in our lexicon, the input feature vector is converted to a class feature vector. This latter vector is then compared to the vectors obtained from our training samples.

Distance between 2 feature vectors

We will illustrate here the computation of the distance between 2 vectors. The feature vector is made up of the eleven subvectors as illustrated in the previous section. In order to compute the distance between two feature vectors, we compute eleven sub-distances corresponding to the eleven sub-vectors. The eleven sub-distances will be expressed as:

- **dap**: distance ascender position
- **ddp**: distance descender position
- **dan**: distance ascender number
- **ddn**: distance descender number
- **dwl**: distance word length
- **dlp**: distance loop position
- **dvs**: distance vertical stroke
- **dhs**: distance horizontal stroke
- **dses**: distance south east stroke

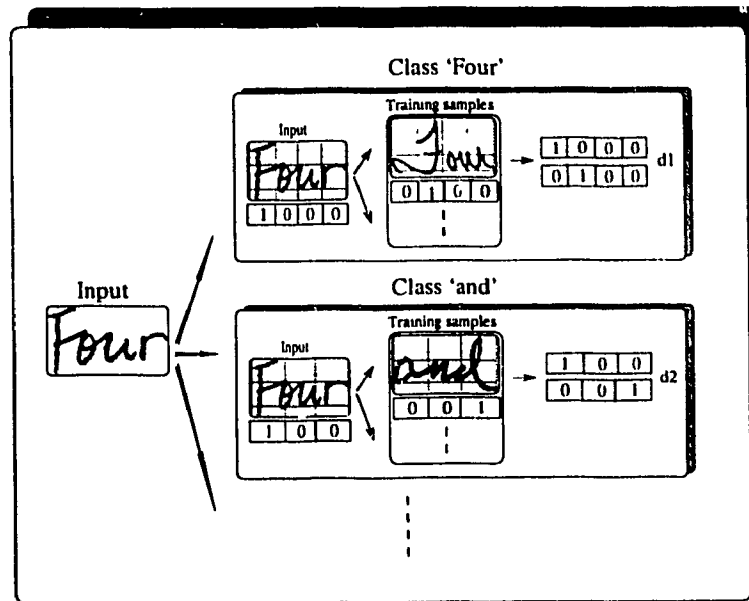


Figure 33: Nearest Neighbour Classifier

- dsws: distance south west stroke

The final distance measure is taken as a weighted sum of those sub-measures. An illustration of the matching of an input image to the stored feature vectors from the training samples is shown in Fig. 33. To simplify the figure we consider only the ascender features. From Fig. 33, intuitively we would like to have a measure that would 'tell' us that the input image ("Four") is closer to the "Four" training sample than to the "and". Indeed an ascender in the first position (as in the input image) is 'closer' to an image with an ascender in the second position (as in the sample "Four") than an image with an ascender in the last (right most) position (as in the sample "and").

A simple Hamming measure that computes the number of substitutions (Fig. 34a) is not suitable since it gives a similar distance (2) in both cases. Therefore we designed a minimum shift measure or *Guillevic measure* as illustrated in Fig. 34b. In this case, we find that the distance $d1$ to the sample "Four" is smaller than the distance $d2$ to the sample "and". This corresponds to the result that we would expect when looking at those samples in Fig. 33.

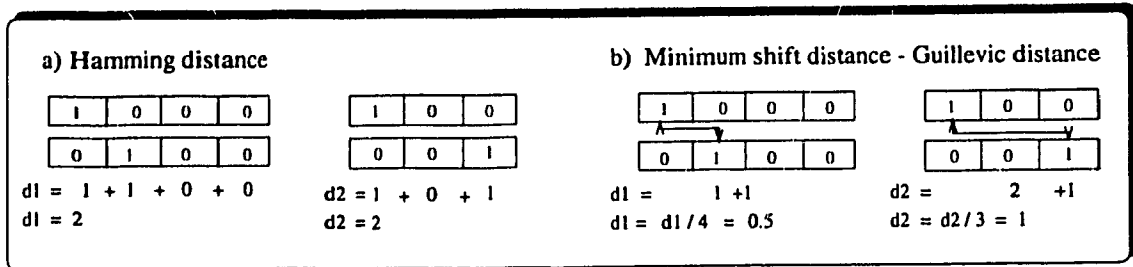


Figure 34: Vector distance computation

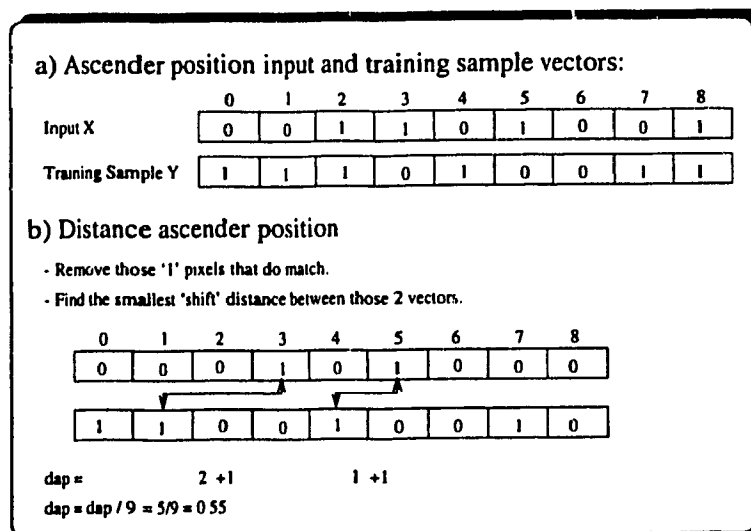


Figure 35: Example: Distance between 2 ascender position vectors

The measure is computed as the minimum sum of the difference between each possible pairing of features between the two subvectors. A similar notion was used in [SBSV94] to compute the distance between two digits represented as a set of arcs.

The heuristic we use is the distance of the shifting needed to have both vectors match. This heuristic incorporates the intrinsic notion of the 'position' features. The measure between 2 'position' vectors in a more general case is illustrated in Fig. 35. In that figure, we assume that we use only the ascender features and that the vectors represent the respective positions of the ascenders in the input image and a training sample. Note that we normalize the distance by dividing it by the vector's length (e.g. 9 in the example of Fig. 35).

We described in Fig. 35 the process involved in computing the distance between 2 position subvectors. The same procedure is used to compute the distances of the position subvectors for descenders, loops and strokes. For the other feature subvectors whose length is 1 (e.g. the word length or number of ascenders), the distance is computed with a simple subtraction. For example, if the word length of the input is estimated to be 10 and the word length of some training sample is 7, then the distance for the word length dwl is computed as $dwl = |10 - 7| = 3$.

The distance from one input vector to a training feature vector is then computed as a weighted average of the eleven sub-distances dap , ddp , dlp , dan , ddn , dln , dwl , dvs , dhs , $dses$, $dsws$:

$$d_{prototype} = (w1 * dap + w2 * ddp + w3 * dlp + w4 * dan + w5 * ddn + w6 * dln + w7 * dwl + w8 * dvs + w9 * dhs + w10 * dses + w11 * dsws) / (w1 + w2 + w3 + w4 + w5 + w6 + w7 + w8 + w9 + w10 + w11)$$

Optimization of the weights

We introduced 11 weights in our classification scheme, one for each sub-vector of the feature vector. We implemented a genetic algorithm [Gol89] to search for the optimal combination of these 11 weights. The fitness score associated with each combination of strings was chosen as the (top 3 choices) recognition rate computed on the entire training set. We allowed the weights to be in the range [1 – 255] (8 bits). The string of 11 weights consisted of 88 bits. The algorithm for the search is as follows:

1. Generate randomly a population of N strings.
2. Compute the fitness value for each string in the population.
3. Reproduction
4. Crossover
5. Mutation
6. Go back to 2.

The reproduction operator has been implemented as a biased roulette wheel where each current string in the population has a roulette wheel slot size proportional to its

fitness.

The simple crossover operation is applied on each pair of strings s_1 and s_2 . A random position p along the string is selected. A random length l beginning at position p in s_1 is selected. The substring of length l starting at position p in s_1 is interchanged with the substring of same length and position in s_2 . At this point we interchange weights rather than bits.

The mutation is the occasional random alteration of the value of a string position. For a given mutation, the random bit i of the random weight w in the random string s is flipped.

Modified k-nearest neighbour classifier

Given an input image, for each class in our lexicon, we compute the average of the k nearest samples belonging to that class. We rank the classes according to their respective average distance. A similar approach was used in [Ho92] (p.91) to reduce the effect of single outlying samples.

4.3.5 Confidence measure for the classifier

When considering the nearest neighbour scheme, the classifier outputs a distance value. This distance corresponds to the minimum distance of the feature vector to the feature vectors from our training database. It would be interesting to output instead a confidence value. Several approaches can be taken and we need to find the approach best suited to our problem.

Relative distance to the second choice

One approach suggested in [SBSV94] is to compute the confidence value as a relative distance of the first choice to the second choice. For example, say the first choice has a distance output from the classifier of 0.1 and the second choice has a distance of 0.2, then the confidence of the first choice would be $1 - \frac{0.1}{0.2} = 0.5$. Unfortunately that scheme does not produce satisfactory results for our particular system.

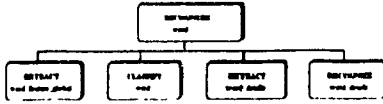
Confidence measure based on statistics

One method suggested by Jürgen Schürmann in his talk at CENPARMI is based on computing statistics from the training database. For each class in the lexicon, we compute one histogram. This histogram has as the x -axis the value taken by the distance (say in our case from 0.0 to 0.4). We can further assume that this interval is subdivided into 100 buckets. Now for each sample in our training database, we use the classifier to compute the distance from that sample to all of the classes in the lexicon. Then for each class in the lexicon, update the histogram for the given distance. If the input word identity is the same as the class whose histogram is updated, then we update the bin 'correct' for the given bucket. If the input word identity is different, then we update the bin 'others' for the given bucket.

After processing the entire training database, for each class, we analyze the histograms. For each value in the histogram, the confidence value is going to be the ratio of the 'correct' bin over the sum of 'correct' plus the 'others' bin. In other words, the confidence measure tells us for a given distance measure and a given class, the percentage of times that value corresponded to the right class. In our particular application, this scheme is used to identify those words for which further features need to be extracted to enhance their recognition results.

However this measure cannot be used to rank the solutions from our word classifier. Indeed for some word classes with few distinctive global features, their confidence value will never be higher than say 35%. This is due to the fact that word classes with few global features such as the set "One", "two", "Six", "Nine", "ten" have very similar feature vectors. As such, even though all the expected features might be present, training samples from other classes also possess exactly the same features and therefore a high confidence value can never be reached. That is the reason why a low confidence value is interpreted as the need to extract *word details*, while a high score means that we indeed recognized an input word belonging to a class which possesses significant and distinctive global features such as "hundred", "thousand", "eight", etc. . . .

4.3.6 Word details extraction



When an input word cannot be adequately recognized with just a few graphical clues and the contextual information, further details need to be extracted. The second step of our computational theory, namely the extraction of details, will help in recognizing those input words that are not recognized with a sufficient confidence value. Besides graphical clues, other important features for a fast reader [GS93] are the characters located next to blank spaces. Empirically we noticed that the first letter is often disconnected from the rest of the word (Fig. 36).

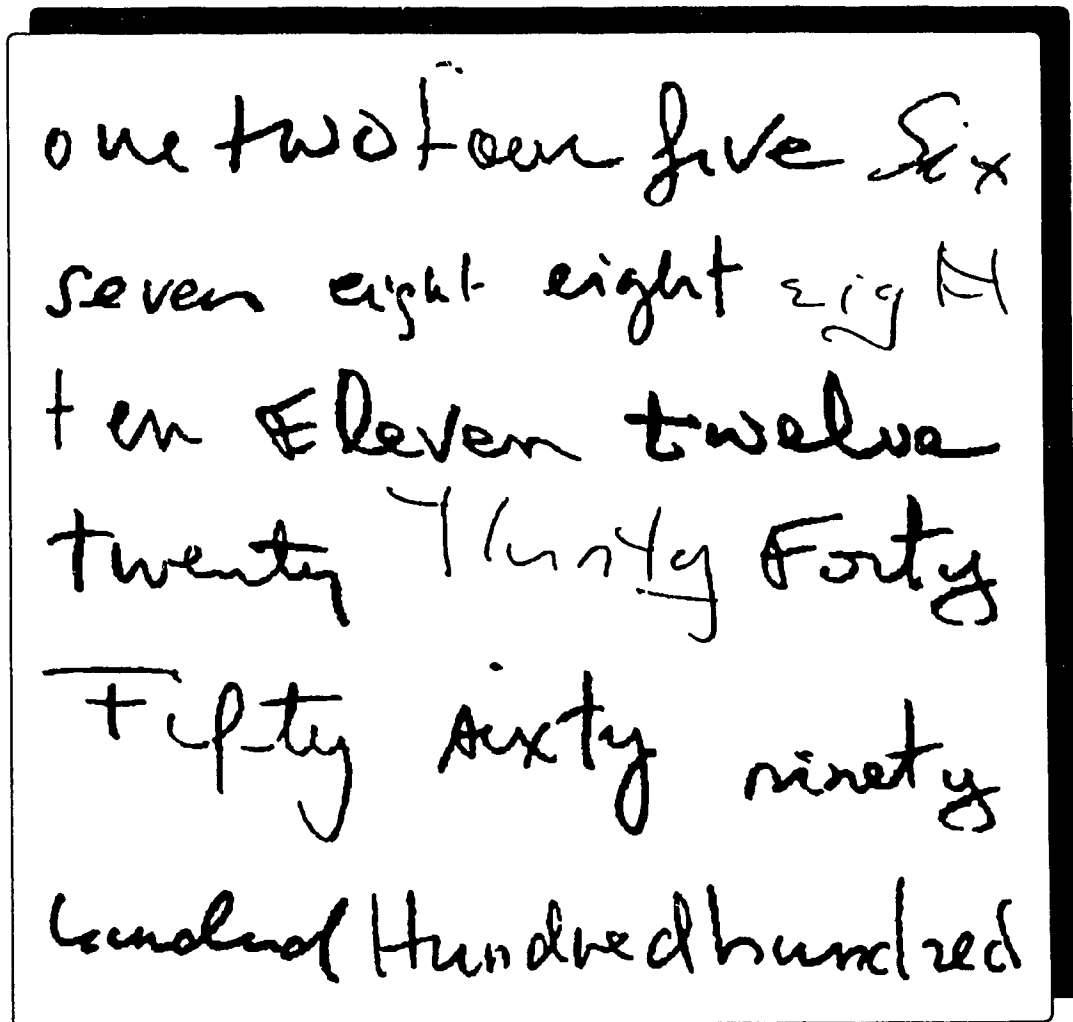
Segmentation-free character extraction

In a first step, we try to investigate whether the first and last characters of a given input word can be extracted without performing any segmentation. In other words, we wish to check whether the characters are disconnected from the rest of the word. A given input word is represented as a linked list of connected components (Fig. 37). Those components are ordered with respect to the position of their centroid along the y -axis. If a word contains only one single component (Fig. 37), this does imply that there is no isolated character in the word.

When trying to locate the first (resp. last) character, we consider the left- (resp. right) most components of the word. In our current implementation, we consider that a character might contain up to 2 connected components. This restriction simplifies the initial implementation as well as matches most of the cases but the ones corresponding to a poor binarization. We will describe here the process involved in locating the first character of a word. The same procedure applies for the last character.

We consider the first (left-most) component of the list. If the component is located outside of the main body of the word as in Fig. 38, then we check whether the next component is located within the main body. If this is not the case, then we exit reporting that no character has been extracted.

If on the other hand, the first component overlaps with the main body of the word



one two four five six
seven eight eight eight
ten eleven twelve
twenty thirty forty
fifty sixty ninety
hundred hundred hundred

Figure 36: Disconnected first and/or last characters of words

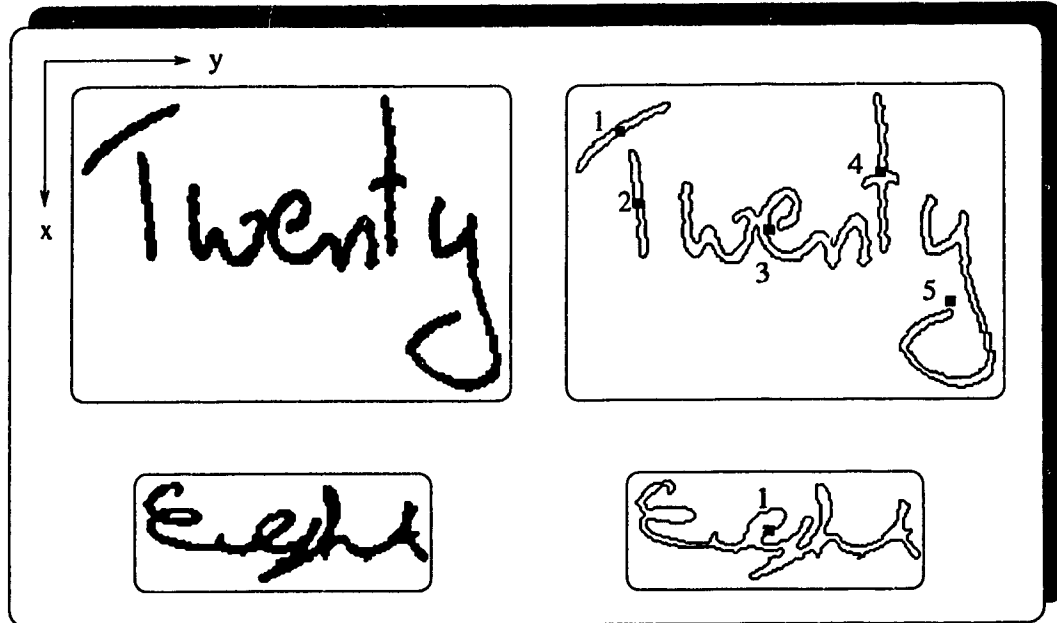


Figure 37: Word represented as an ordered list of connected components

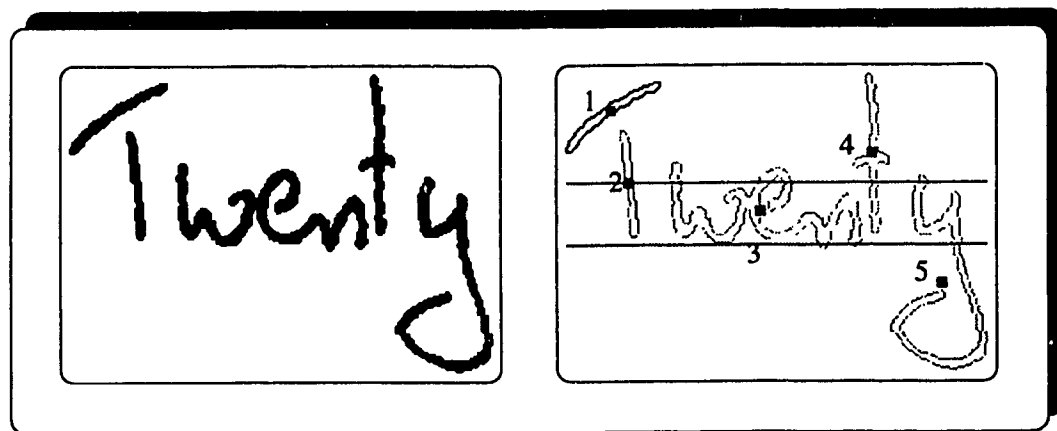


Figure 38: Locating the first character of a word

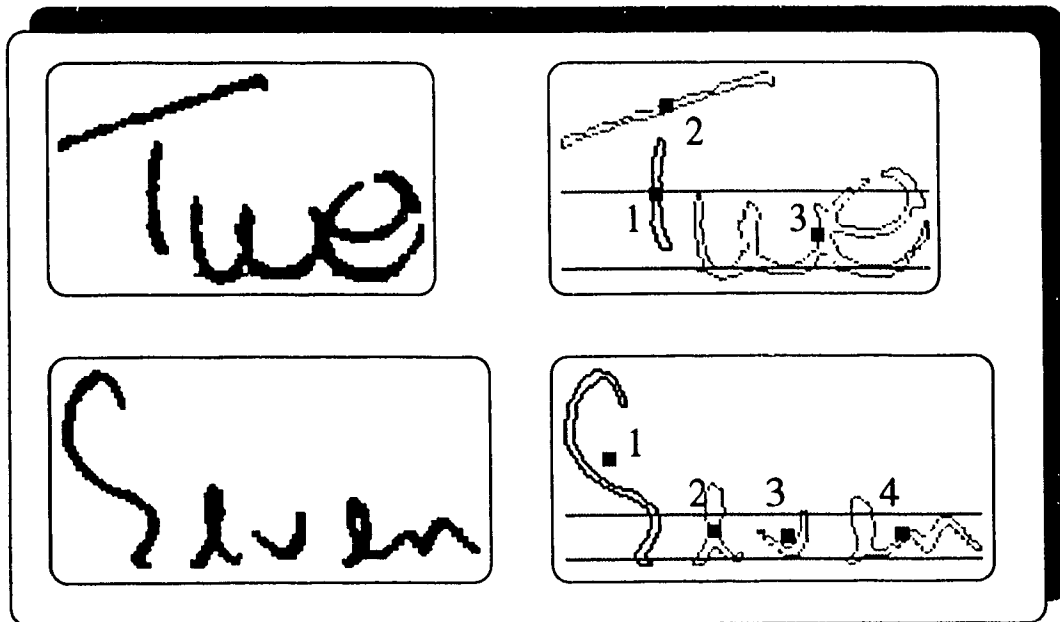


Figure 39: Character extraction: first component overlapping the main body

as in Fig. 39, then we check whether the next component is located above the main body. This latter case represents the possibility of having a disconnected 't' stroke.

Once the components making up the character have been extracted, we wish to check whether they can be reliably considered as a single character. Indeed the above described method will yield not only single characters but also groups of joined characters as depicted in Fig. 40. Therefore we impose some thresholds for the minimum and maximum relative widths of a character. The width of an extracted character is computed within the main body of the word as shown in Fig. 40. Indeed if we were to consider simply the width of the bounding box, then characters such as 'T' would have a great relative width to the word. Therefore letters 'T' might be interpreted as being groups of connected characters as opposed to single characters. The character width thresholds are expressed relatively to the estimated character width for a given input. From the word recognition results, the input word length in characters is estimated as the average length of the top 5 choices (Fig. 41). Once an estimated word length is computed, the estimated character width is expressed as the input word width divided by the estimated word length. An example of this process along

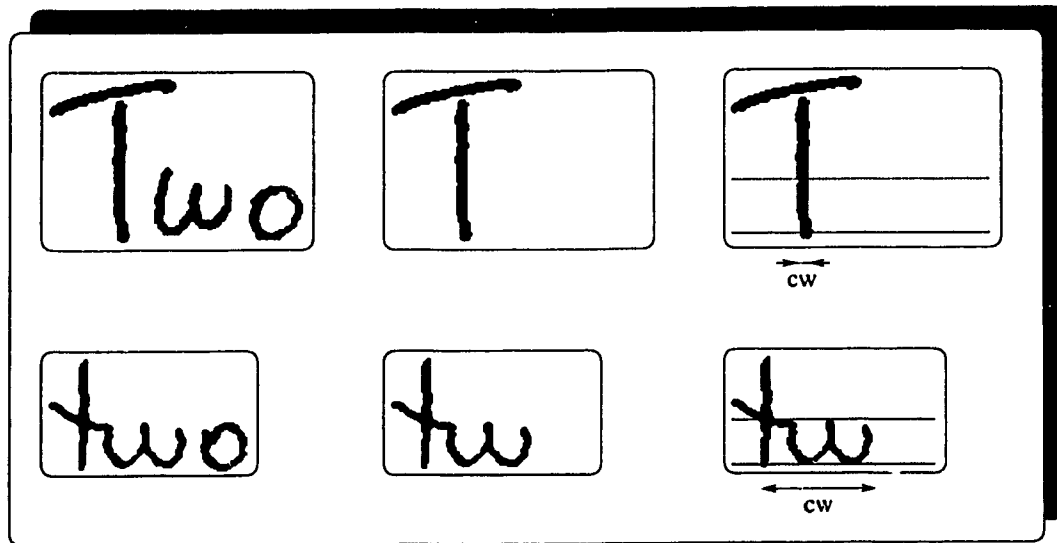


Figure 40: Character extraction: character width cw

with the selected thresholds is shown in Fig. 41. The thresholds t_{c1} , t_{c2} and t_{c3} have been set to 0.5, 2.0 and 0.4 respectively. If the extracted character is found to be too narrow or too large, then the procedure returns stating that no characters could be extracted. The same procedure that we just described for the extraction of the first character of words is applied to the last character.

Character extraction with segmentation

If the extraction modules for the first and last characters both return unsuccessfully, then a brute force segmentation approach is applied. We first compute an estimate of the number of letters in the input word as was described in Fig. 41. From this estimated number of letters, we get the estimated character width ecw for the given input image. For example, for an input word with an estimated number of 4 letters, the estimated width of a single character would be approximately 25% of the bitmap image. Therefore in the *brute force* approach, we would simply cut the first 25% of the bitmap and associate it with the first character. The parameter ecw is increased by a factor $f_c cw$ until the bitmap extracted contains at least one component overlapping with the main body of the word. Among others, this iteration takes care of input words with long horizontal 't' strokes. An example of the process involved in the

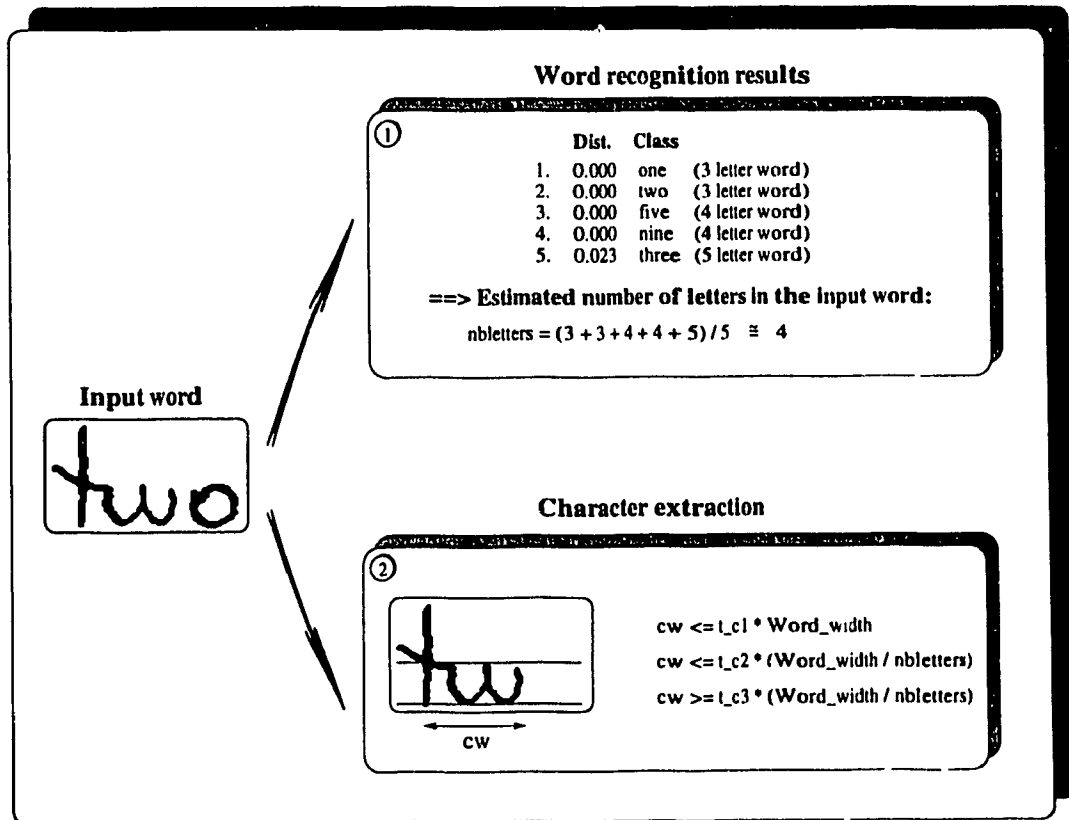


Figure 41: Character extraction: Thresholds for the character width

segmentation of the first and last characters for a given input word is described in Fig. 42.

The process described here for the segmentation of the first and last characters of words is naturally subject to improvement. A more thoroughly developed segmentation scheme should eventually be substituted. An approach similar to the technique used by Nick Strathy [Str93] for splitting touching digits could be used. Such a scheme makes use of significant contour points and heuristics to find the optimal segmentation path.

The complexity of a segmentation scheme, its design and training, though, should not be underestimated. Therefore the current approach is simple in order to allow for a faster prototyping and estimating the significance of the character extraction and recognition for the overall word recognition module.

4.3.7 Character recognition



While numerous approaches have been investigated for the recognition of Arabic numerals, few studies report results on the recognition of Roman characters. One difficulty associated with the Roman character set recognition is the greater size of the lexicon when compared to digits. indeed, instead of 10 classes, we have to deal with 26 to 52 classes whether or not we deal with both lower and upper case characters.

In recent years, one of the most popular approaches to recognize digits is the use of artificial neural networks. At the cost of a lengthy training time, high recognition rates and speed might be obtained. However, such networks do not seem to have been applied to problems with a lexicon size greater than 10.

We decided to opt for a nearest neighbour scheme to recognize the characters. Such a scheme will not only allow for fast prototyping but also allow for a recognizer to be designed even with few training samples.

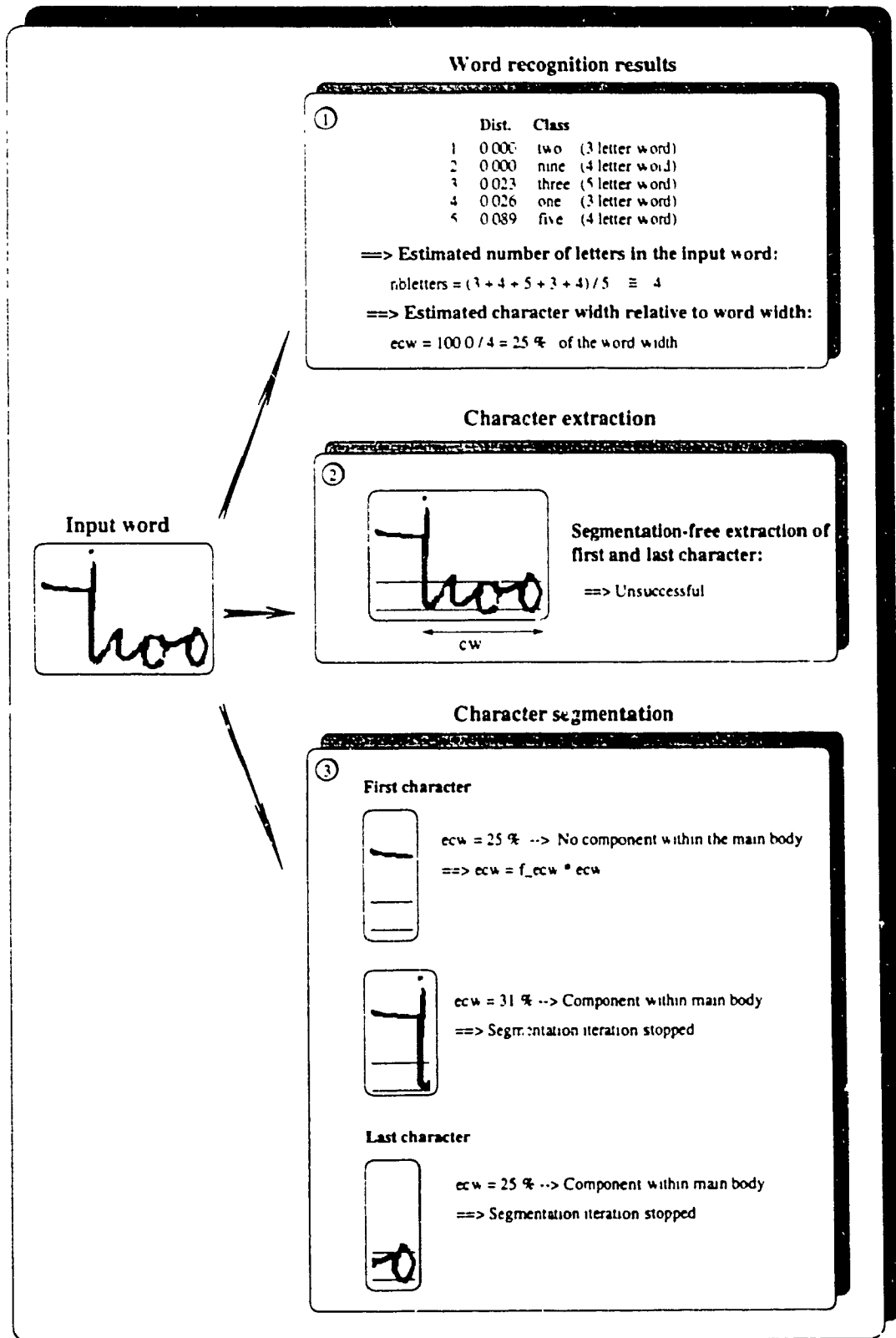


Figure 42: Character segmentation

Character position	Character set
First	a, d, e, f, h, n, o, s, t
Last	d, e, n, o, r, s, t, x, y

Table 9: Character sets for the first and last position of words

Problem analysis

We are not exactly faced with the recognition of Roman characters in general. Our problem domain provides us with a lot of constraints on the characters to be recognized. First of all, we are not to recognize every single character in the Roman alphabet. The characters we are to process are the ones corresponding to the first and last characters of words belonging to our lexicon. The sets of first and last characters we are to recognize are shown in Table. 9. Therefore the size of the character lexicon is restricted to 9, if we consider a single character case, as opposed to 26 (the size of the Roman alphabet) for a given character position.

We can further restrict the character set if we use the results of the word recognizer. The word recognizer provides us with a ranked list of the top 10 solutions for a given input word. From there on, we assume that the correct solution lies within the top 10 choices. The character recognition task is therefore translated into an attempt to shift up the correct choice as close to the top of the list as possible. Therefore the character recognition results are not intended to point to a word class not present in the top 10 list provided by the word recognizer. The process of using the word recognition results to narrow down the character set for a given word position is illustrated in Fig. 43.

When trying to recognize an input pattern, we not only know its position within the word (first or last position) but also we know its position relative to the word reference lines as well as information about the average slant of the legal amount and word to which the input belongs. An example of a character bitmap along with its associated information stored in the image message is shown in Fig. 44. The information about the position of the character relative to the reference lines will be useful in differentiating among character classes and possibly give a hint as to the

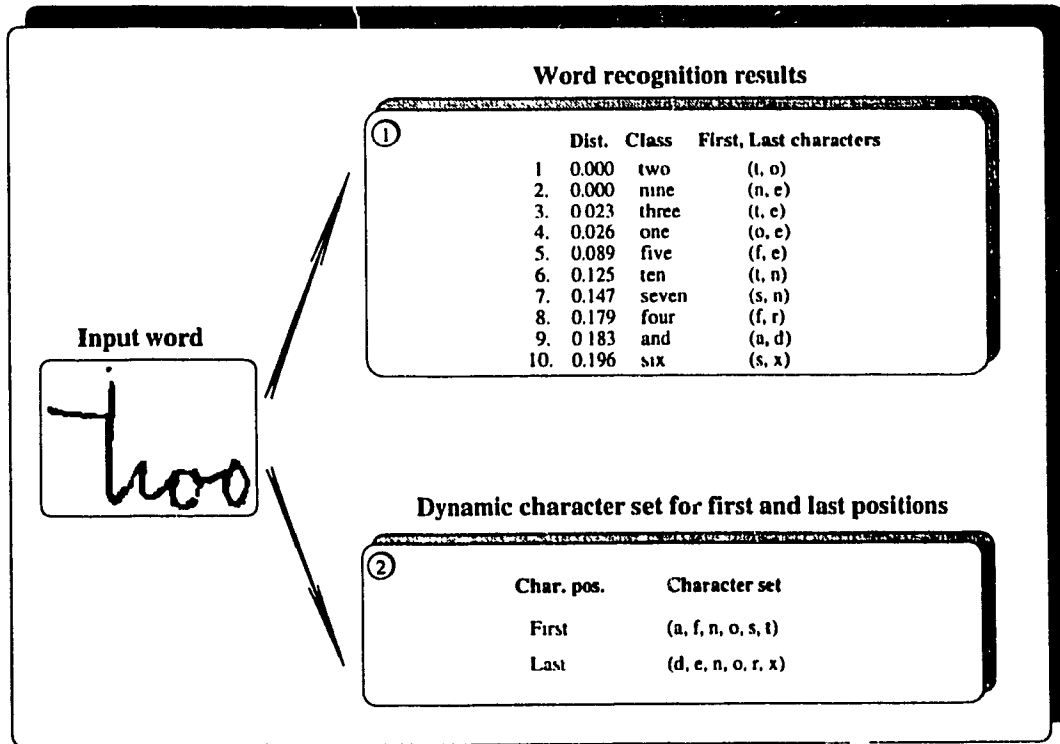


Figure 43: Dynamic character sets

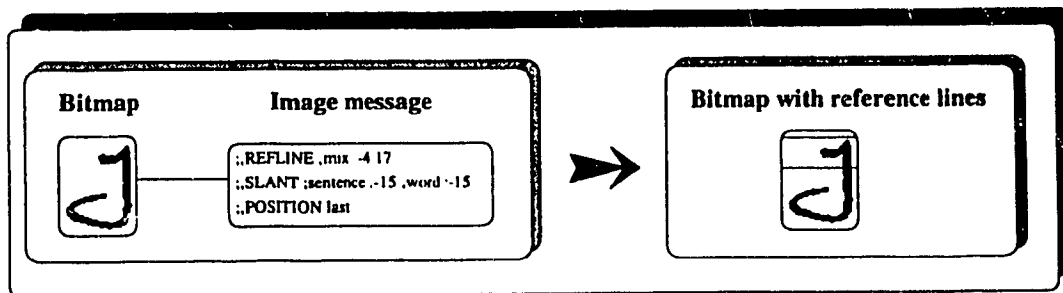


Figure 44: Information available for each character

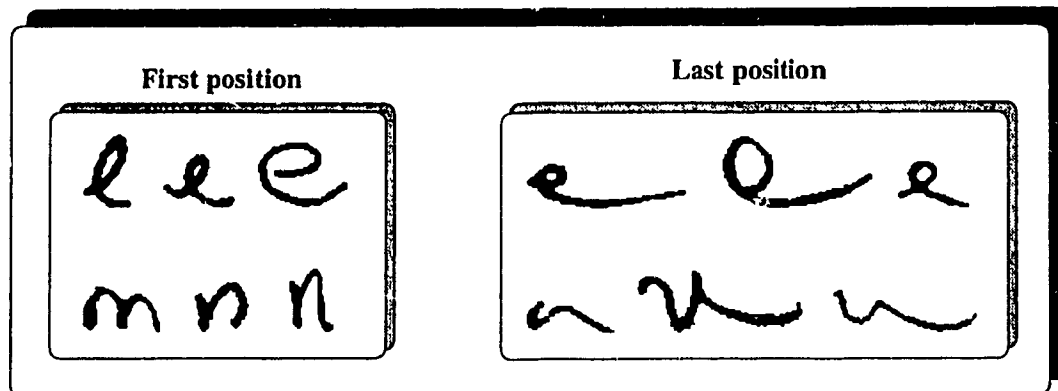


Figure 45: Character shape variations according to the position within a word

case of the character (lower or upper). Indeed if the character is located within the main body of the word then it corresponds most likely to a lower case character.

Cursive characters extracted from handwritten words might look somewhat different depending on their position within a word. Indeed some people have the tendency to exaggerate the writing of the last character of words by drawing some longer than necessary strokes. Some examples of the possible differences of character shapes according to the location are shown in Fig. 45. For that reason, we decided to have 2 separate classifiers, each trained respectively on first and last characters of words.

Features

We decided to opt for slightly more complicated features than simply the pixel values. We chose the pixel distance metric as defined in [SBSV94]. For mismatched pixels between the test and the training samples, it takes into account the distance to the nearest pixel of the same colour (e.g. foreground or background). We use an integral approximation to the Euclidean distance between pixels, truncated to a maximum distance of 4. The total difference between two bitmaps B_1 and B_2 is expressed as:

$$\sum (B_1 \oplus B_2)(D(B_1) + D(B_2)) \quad (1)$$

where $B_1 \oplus B_2$ is the sum of the pixel-wise exclusive or of the two bitmaps B_1 and B_2 , and $D(B_i)$ is the distance map representing distances to the nearest pixel of a

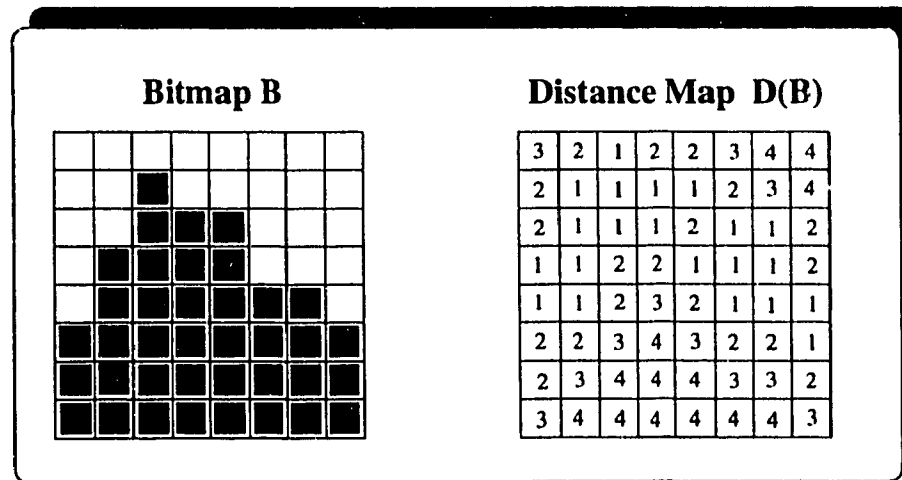


Figure 46: Example of the distance map computation

different colour. The computation of the distance map is illustrated with an example in Fig. 46.

Classifier

Character recognition algorithms are usually trained to recognize isolated characters, but no special training is performed to enhance the recognizer's ability to reject "garbage". Garbage here might consist of two or more touching characters or parts of characters. For our specific application, we cannot guarantee that the extracted characters are going to be perfectly isolated single characters. Therefore we need a recognizer with a good rejection ability. One such approach is to match the hypothesized character to a set of templates for each character class. With adequate features, it is possible to design a recognition system that will give high confidence value when features do match and very low score when there is a significant mismatch of features. Consequently we adopted a k-nearest neighbour scheme.

The character recognition system is trained on a set of isolated characters extracted from the training database. As mentioned previously, we train two separate recognizers respectively for the first and last characters of words. These recognition systems are trained solely on characters extracted respectively from the first and last character positions of the training words. This is made possible through the tagging

scheme applied to our database. Indeed the tagging of an input image enables us to identify those characters located at the beginning and end of words. In fact we train 4 classifiers, one for each combination (*position, case*) where *position* can take up the values *FIRST* and *LAST* and *case* the values *UPPER* and *LOWER*.

Since we are training specialized recognizers for a given character position and a given character case, the number of training samples is less than if we were to consider a general isolated character recognizer. This specialization puts an additional constraint on the need for an even larger training set of legal amounts.

The prototype characters are normalized into one of three different template sizes 24×16 , 16×16 or 16×24 , depending on the aspect ratio of the original character. The choice of three normalization sizes reflects the variety of the aspect ratios inherent to the Roman characters (e.g. l, a, m) as well as prevents spurious recognition results which can occur when the aspect ratio of a character is very distorted during normalization.

During a recognition cycle, the input character is normalized to the appropriate size and its feature map is computed. The input is then compared to all prototypes of the same size. The score that is assigned to the match between a prototype and the input is the distance described in Eqn. 1.

As mentioned earlier, a given input character is only compared to those classes that are part of the dynamic character set generated from the word recognition results as illustrated in Fig. 43. For each of those character classes, we compute the average distance of the k nearest prototypes belonging to that class. This distance is further normalized by the size of the input character (*height* \times *width*) in order to get homogeneous distances over the 3 different template sizes and somewhat within the range [0–1]. We then rank the classes according to their respective average distances. This corresponds to the modified k -nearest neighbour classifier that we also apply to the recognition of words (section 4.3.4).

Postprocessing

The information about the location of the reference lines relative to the position of the character is used for several purposes. First of all, it is used to determine the

First position									
Features	a	d	e	f	h	n	o	s	t
ascender	0	1	0	1	1	0	0	2	1
descender	0	0	0	2	0	0	0	2	0
Features	A	D	E	F	H	N	O	S	T
ascender	1	1	1	1	1	1	1	1	1
descender	0	0	0	2	0	0	0	0	0

Last position									
Features	d	e	n	o	r	s	t	x	y
ascender	1	0	0	0	0	0	1	0	0
descender	0	0	0	0	0	0	0	0	1
Features	D	E	N	O	R	S	T	X	Y
ascender	1	1	1	1	1	1	1	1	1
descender	0	0	0	0	0	0	0	0	2

Table 10: Character features: 0 = *NO*, 1 = *YES*, 2 = *MAYBE*

likelihood of being in the presence of a lower-case character. If the word possesses an ascender but the character does not, then it is assumed that the input character is in lower case. In other cases, no assumption is made and both recognizers are tried on the input. Additionally, it is used to differentiate among characters such as 't' and 'f' whose sole bitmaps can be very similar. If the first choice of the character classifier is the class 't' and the second choice 'f', then we check how deep the character reaches. If the character is found to have a descender, then the character results for the first and second choices are reversed.

We also designed a simple feature data structure that lists explicitly for a given character class, a given character case and a given character position, the presence (*YES*), absence (*NO*) or possibility (*MAYBE*) of ascenders and descenders (Table. 10). In other words, for a given character class, we wish to list the a-priori likelihood of having the character within the boundaries of the reference lines or spreading significantly over the top reference line or below the bottom reference line. This helps us in reducing the ambiguity among character classes that exists when

considering only the bitmap information. These values have been determined empirically and it is interesting to note that a given character in a given case (e.g. 's') might have different attributes according to its position within the word.

Character recognition output

For a given input character, both a lower and an upper case character classifiers are called. The output of the character recognition module is the result of the classifier with the lowest distance. For further processing, namely the integration of results with the word recognition module and matching with lexicon words, we are only interested in the identity of the characters and not their case. Indeed for us the words 'two' and 'Two', for example, belong to the same word class 'two'.

Further investigation

The training characters are the tagged isolated characters in our training databases. In place of those, one could produce the training characters by running the segmentation free character extraction procedure described in section 4.3.6 on the training images. Doing so, we could possibly get some training characters better suited to the testing data. As an illustration of this point, one can notice that the entities extracted from the first and last positions of words might not be single characters as we would expect (Fig. 47), but rather parts of characters or connected characters as in 'th' or 'ty'. Since the ultimate goal is to recognize words and not characters, we should not restrict ourselves to the recognition of pure characters, but rather aim at recognizing the entities located at the beginning and end of words that characterized a given class.

Additionally in the current approach the same training samples are used for the recognition of both the segmentation-free extracted characters as well as the segmented characters. In their place, one might wish to produce a training database possibly better-suited to the segmented characters by running the segmentation algorithm on the training words.



Figure 47: Non-characters extracted from the first and last positions of words

4.3.8 Integrating character results with word results

The word recognition module produces the primary result of our system. Character extraction and recognition is then used to supplement and possibly improve on the word recognition. For that reason, the integration of the character results should act as an agent to shift up and down the word possibilities. As a result we wish to shift up the right choice closer to the top position of the input word recognition list and shift down less possible word choices. Once the shifted solutions are close to the top choices, we perform some reorganization of the top choices so as to improve the recognition rate of having the correct solution among the top first or second choices while keeping the same recognition rate for the top (say 5) solutions.

Top character(s) selection

The first step we chose to perform is to implement some measure to reflect the confidence associated with the top choices of the character recognizer. Due to our limited and sparse training database for characters, we cannot reliably apply the confidence measure based on statistics described in section 4.3.5 to estimate the relative worth of the top first choice to the second choice. On the other hand, the relative distance to the second choice method can be applied regardless of the significance of the training database, and consequently we used that latter approach. If the distance of the second choice relative to the first choice is greater than the character recognition threshold crt , then we assumed a high confidence for the first choice and consider for further processing solely the first choice. On the other hand, if that condition is not met, then both the top first and second choices are considered in subsequent processing. An example of this top character selection is shown in Fig. 48.

While the confidence based on statistics is not adequate here, it can still be used later on in the processing. Indeed due to the limited number of training data, confidence values of zero can be obtained easily. This is obviously not reliable as it only reflects the lack of training samples. On the other hand, a high confidence value can be of some use.

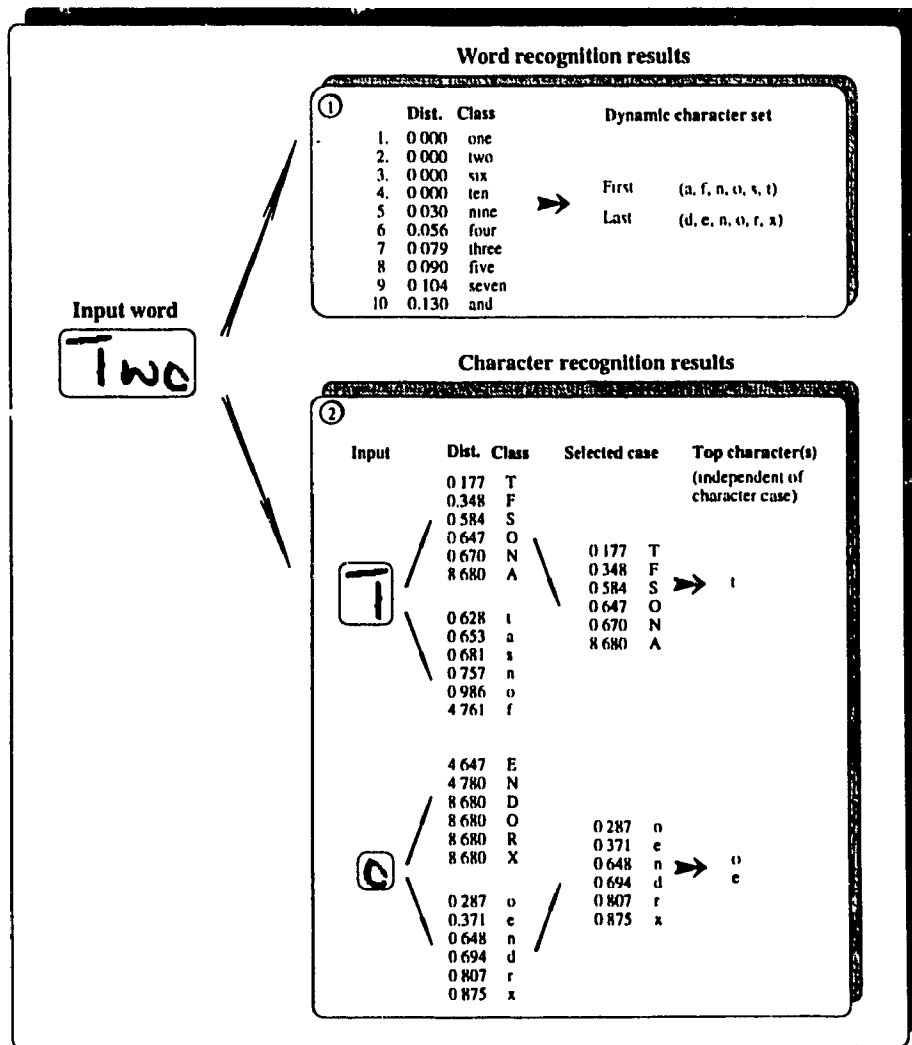


Figure 48: Selection of the top character(s) for subsequent processing

Shifting up and down

Upon selecting a maximum of 2 character choices for the first and last positions, we now wish to use this information to shift up and down the choices in the word recognition list. While we consider the top *NBTOPCHOICE* (10) choices of the word classifier, our goal is to maximize the recognition rate for the correct solution to be among the top *TOP_SOL* (5) choices. The first step we apply towards that goal is to make sure that words that do match any combination of the character recognition results are shifted up as close as possible to the top 5 choices. The algorithm for this processing step works as follows:

```

For( i = TOP_SOL; i < NBTOPCHOICE; i = i + 1 )
  If the word does match some character combination
  Then save that word and
  try to move it up the list as follows:
    1. Shift down the words from below TOP_SOL if
       they do not match the characters and
       they have a confidence value less than CONF_THRESHOLD
       Record the new empty location in the list.
    2. Try to look for a word in the TOP_SOL solutions
       that could be shifted down, that is shifted out of
       the TOP_SOL top choices.
       If such a word is found that:
       does not match the characters and
       does not have a confidence value greater than CONF_THRESHOLD
       Then shift down that word to the empty location down the list.
       Record the new empty location
    3. Place the saved word into the empty location.

```

It is much easier to view the process of shifting up and down with an example (Fig. 49).

Permutation of the top *TOP_PERMUTATION* solutions

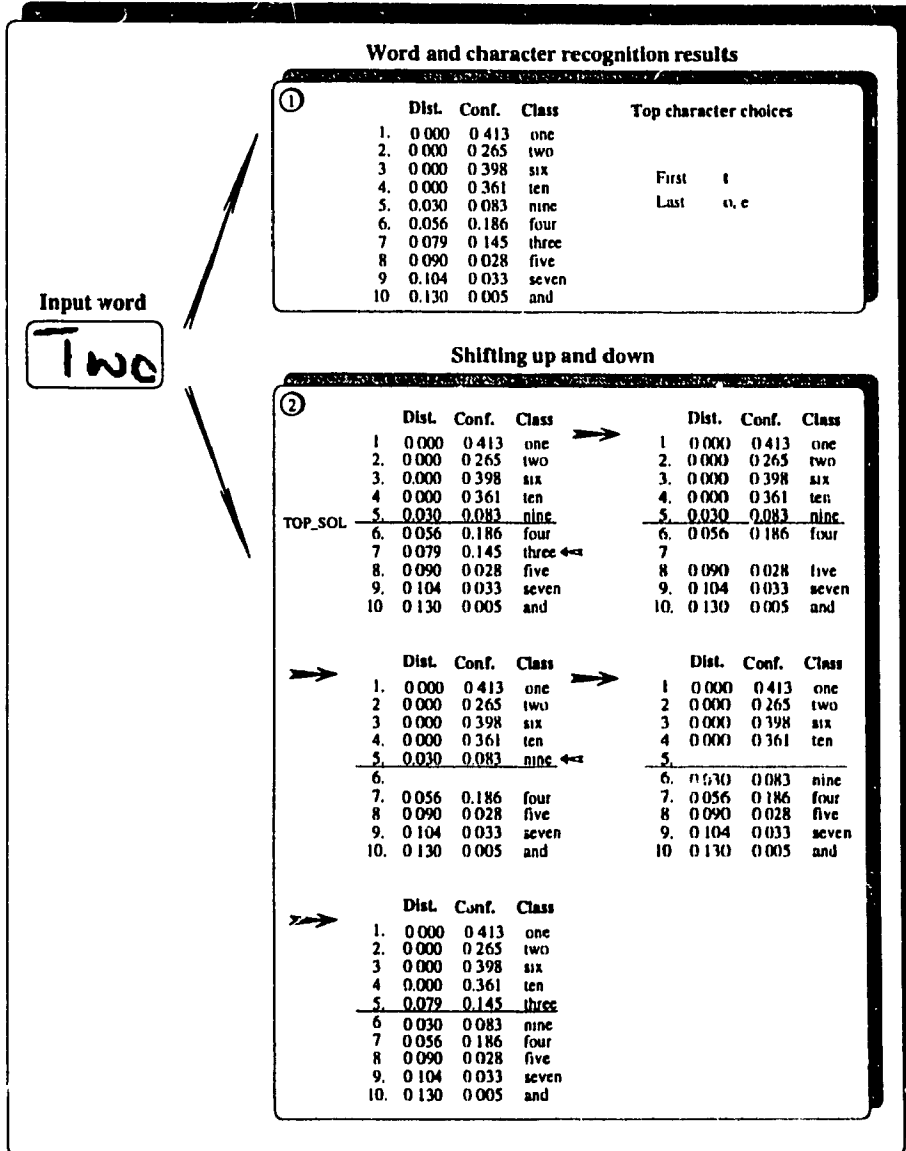


Figure 49: Shifting up and down of word solutions

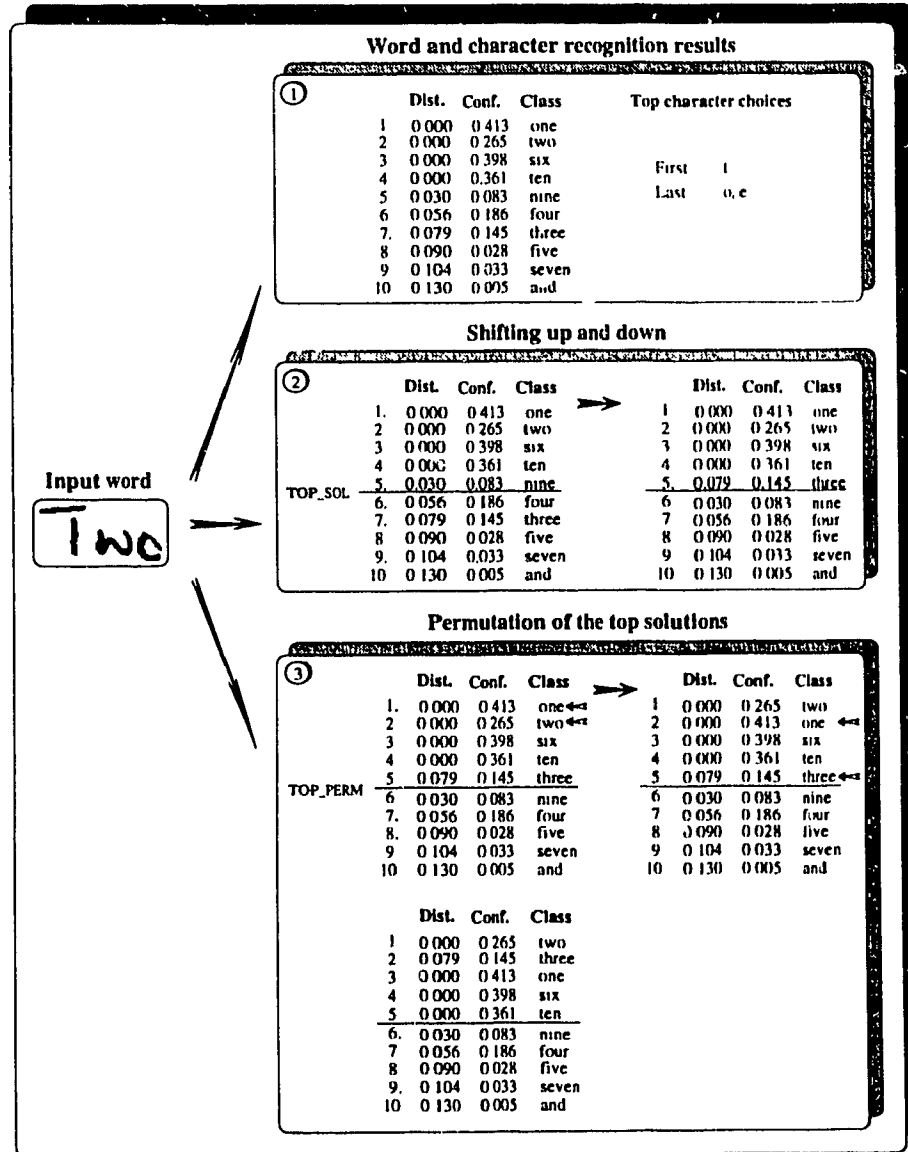


Figure 50: Shifting of the top *TOP_PERMUTATION* word solutions

After the shifting up and down of word solutions, the correct choice is now likely to be close to the top *TOP_SOL* solutions. We then choose to reorganize the top *TOP_PERMUTATION* (5) solutions by performing some permutations. Essentially we wish to move up those words that do match the character solutions as well as make certain that words that do match first choice characters appear before words that match second choice characters. The algorithm for the current implementation is presented below:

```
1. For( i = 1; i < TOP_PERMUTATION; i = i + 1 )
    If word_i does match some combination of the character results
    Then look for some word to perform permutation
    For( j = 0; j < i ; j = j + 1 )
        If word_j does not match any character combination and
        word_j's confidence is smaller than CONF_THRESHOLD
        Then shift down by one position those words from
        position j to position i-1 and
        move word_i up to the j position.
```

```
If no permutation has been performed,
Then relax the requirements necessary for permutations:
Concentrate on one character position at a time.
```

```
    If one of the character positions has a single choice
    Then move up first those words that match that character
    and then those words that match the other position choices
```

```
    Else
```

```
    If both character positions have 2 choices
```

```
    Then move up first those words that match the first position
    then move up those words that match the last position.
```

```
    Else
```

```
    If both character positions have a single choice,
```

```
    Then move up first words that do match the character
    with the highest confidence value (statistical) and then
```

those words that do match the other character position.

2. Make sure that words that do match top choice characters do appear before words that do match the second choice characters.

Once again it is easier for the reader to visualize the process at hand with the help of an example (Fig. 50).

4.4 Parsing module



Our computational theory for word recognition has been implemented so as to be fully trainable. This means that the system can be re-trained with very little effort on a different database and potentially on a different language. The restriction on the language would be that it also produces the ascenders and descenders found in the lower case of the Roman alphabet. The only module requiring extra coding would be the definition of the parser, since it is language dependent. Our present approach has been primarily tested on English cheques and therefore we shall discuss extensively the parser associated with that language. The parser for French amounts is also introduced as a natural extension of our work.

We recall that the segmentation module (section 4.2) produces a list of possible segmentations of the legal amount into individual words. In order to process a given path, the word recognition module described in the previous section is called upon to assign recognition results on each word. The parser then works on the recognition results to produce a ranked list of syntactically and semantically correct legal amounts.

4.4.1 English parser

A grammar has been designed to parse successfully the amounts found on cheques up to a value corresponding to 99,999\$. Our grammar will not be able to parse

S	→	(number teensminus.ten) S1 tys S2 ten S3
S1	→	hundred H thousand T E
S2	→	number S1 S1
S3	→	thousand T E
H	→	and H1 tys D (number teens) E E
H1	→	tys D (number teens) E EE
T	→	number T1 (teens e) E tys D and T2
T1	→	hundred H E
T2	→	number T1 (teens e) E tys D
D	→	(number e) E

Table 11: Grammar for the English legal amount

amounts of say 120,000\$. This limitation has been done intentionally to limit the complexity of the grammar as well as to increase the reliability of the system. The idea being that any amount greater than 100,000\$ should be handled manually due to the importance of such an amount. No error could be permitted. Therefore ‘big’ cheques are rejected from the recognition system so that it can be handled ‘safely’ by an operator.

The grammar allows for various formats such as ‘eleven hundred’, ‘one thousand and one hundred’, ‘one thousand one hundred dollars’, etc... The grammar is depicted in Table 11. Whereas the grammar is useful for the implementation, a diagram makes it easier for us to visualize the parsing as well as extend or simplify the grammar. The graph for the parsing of the English legal amount is represented with the transition network depicted in Fig. 51. The nodes represented in grey are referred to several times within the network. Their different background colours are intended to facilitate the reader’s search for those nodes.

The final state E (resp. EE) takes 2 forms depending on whether we wish to process only full dollar amounts; or whether we allow for the cent amounts to be parsed. Experimentally we noticed that sometimes some people do write the cent amounts in full letters as opposed to only writing them with digits at the end of the

E1	→	and F dollars F1 F1
F1	→	only F F
EE1	→	F

Table 12: English final states: solely full dollar amounts

E2	→	dollars G1 and G2 (only e) F
G1	→	and G2 (only e) F
G2	→	(number teens) G3 tys G4 F
G3	→	cents F F
G4	→	number G3 G3
EE2	→	G2

Table 13: English final states: dollar + cents amounts

sentence. Therefore there is a need to be able to parse such cases. "Two hundred dollars and fifteen cents" would be an example of a 'dollars + cents' amount fully written in alphabetical letters. We should note however that these cases are not as frequent as the writing of cent amounts in digits. Also we may want to try to recognize solely full dollar legal amounts and if this does not produce satisfactory results then switch to the second form. So we define 2 different grammars for the end state E (Table 12 & 13). E1 (resp. EE1) stands for the parsing of full dollar amounts, whereas E2 (resp. EE2) is able to parse both full dollar and dollar plus cent amounts.

In the grammar, identifiers beginning with a capital letter are non-terminal symbols. S represents the starting symbol. The keywords 'number', 'teens' and 'tys' represent the classes of terminal symbols described in Table 14.

The keyword 'teensminusten' represents as its name may imply, the set of terminal symbols 'teens' minus the symbol "ten". The strings "hundred", "thousand", "and", "dollars" and "only" are terminal symbols. The terminal symbol 'e' represents an

number	=	{“one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine” }
teens	=	{“ten”, “eleven”, “twelve”, “thirteen”, “fourteen”, “fifteen”, “sixteen”, “seventeen”, “eighteen”, “nineteen” }
tys	=	{“twenty”, “thirty”, “forty”, “fifty”, “sixty”, “seventy”, “eighty”, “ninety” }

Table 14: Classes of terminal symbols for the English grammar

empty input. The size of the lexicon is 32.

No left recursion, nor any non-determinism is present in the above grammar. Therefore it is acceptable for recursive descent parsing. We wrote a recursive descent parser using the above grammar.

The classes ‘number’, ‘teens’ and ‘tys’ have been implemented separately using a hash table (Table lookup algorithm from [KR88]). This implementation allows for fast matching of an input string to one of the classes.

At every stage of the parsing, the algorithm does print the expected classes for the next input word. For example after having recognized a ‘two’ at the beginning of the line, the parser tells us that it is expecting either an empty input, the word ‘dollars’, ‘hundred’ or ‘thousand’ as the next input. Any other string would result in an error message pointing out that the sentence does not form a correct amount. We implemented a look-ahead mechanism. These predictions can be used at the recognition level to limit the lexicon for a given input string.

4.4.2 French parser

Similarly as we did for English in the previous section, we designed a grammar that is able to parse the legal amounts written in French up to an amount of 99,999\$. Due to the nature of the French language, the grammar is more complicated than the one for the English language. The grammar is described in Table 15 and is represented in Fig. 52. At the moment, we limited the grammar to the parsing of full dollar amounts. We can see that when looking at the ‘simple’ end state E. Upon examining

a database of French legal amounts, we may have to refine the current grammar. The refinements may include allowing the parsing of the cent amounts if those are written in full letters. We may also need to add a few terminal symbols that may appear in the database of cheques.

Within the grammar, identifiers beginning with a capital letter are non-terminal symbols. *S* represents the starting symbol. The keywords *number*, *teens* and *tys* represent the classes of terminal symbols described in Table 16. The keywords ‘numbermoinsunquatre’, ‘teensmoinsdix’, ‘teensmoinsdixonze’, etc. . . stand for some subsets of the sets ‘number’, ‘teens’ or ‘tys’. For example, as we may guess from the name, ‘numbermoinsunquatre’ stands for the set ‘number’ minus the words “un” and “quatre”. The above grammar allow for various formats such as “Mille et un dollars”, “Mille un”, etc. . .

At the moment, the lexicon for the French amount consists of the sets *number*, *teens* and *tys* plus the words “cent”, “mille”, “et” and “dollars”. Therefore the lexicon size is 25, which is smaller than the lexicon size for the English amounts.

S	→ un E quatre S1 numbermoinsunquatre S2 dix D2 teensmoinsdix D1 (vingt trente quarante cinquante) T2 soixante T3 cent C1 mille M1
S1	→ vingt T1 cent C1 mille M1 E
S2	→ cent C1 mille M1 E
D1	→ mille M1 E
D2	→ (sept huit neuf) D1 D1
T1	→ dix D2 (number teensmoinsdix) D1 D1
T2	→ et U1 numbermoinsun D1 D1
T3	→ et U2 dix D2 (numbermoinsun teensmoinsdixonze) D1 D1
U1	→ un D1
U2	→ (un onze) D1
M1	→ (un e) E et C1 quatre C2 numbermoinsunquatre C3 dix D3 teensmoinsdix E (vingt trente quarante cinquante) T5 soixante T6
C1	→ un E quatre S3 numbermoinsunquatre E dix D3 teensmoinsdix E (vingt trente quarante cinquante) T5 soixante T6 E
C2	→ cent C1 S3
C3	→ cent C1 E
S3	→ vingt T4 E
D3	→ (sept huit neuf) E E
T4	→ dix D3 (number teensmoinsdix) E E
T5	→ et U3 numbermoinsun E E
T6	→ et U4 dix D3 (numbermoinsun teensmoinsdixonze) E E
U3	→ un E
U4	→ (un onze) E
E	→ dollars e

Table 15: Grammar for the French legal amount

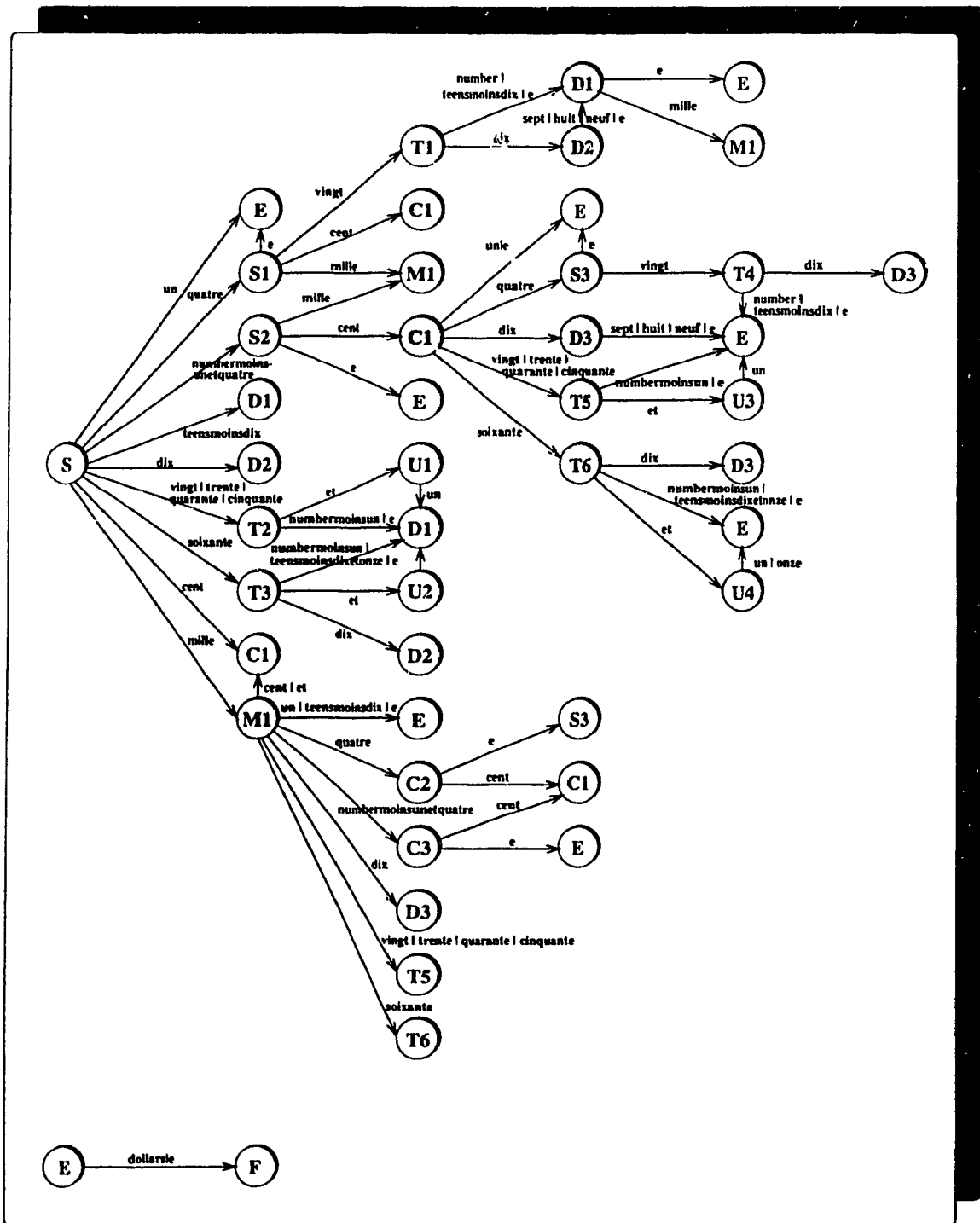


Figure 52: Parsing of the French legal amount

number	=	{“un”, “deux”, “trois”, “quatre”, “cinq”, “six”, “sept”, “huit”, “neuf” }
teens	=	{“dix”, “onze”, “douze”, “treize”, “quatorze”, “quinze”, “seize” }
tys	=	{“vingt”, “trente”, “quarante”, “cinquante”, “soixante” }

Table 16: Classes of terminal symbols for the French grammar

Chapter 5

Experimental Results

We describe in this chapter the results obtained after applying our computational theory to the processing of cheques from our database. We will first discuss the recognition of words using only our global classifier. The results on the recognition of isolated characters will then be described; and finally the results of combining both the word and character recognition results.

An example of the output produced by the legal amount recognition system is shown in Fig. 53. For each word in the legal amount, the word recognizer produces a ranked list of the 10 top possibilities (only the top 3 are displayed in Fig. 53). Then the parser produces a ranked list of possible legal amounts. Note that the values associated with each interpretation in Fig. 53 refer to a distance measure (e.g. the smaller the better).

5.1 Global features recognizer

The word recognizer has been trained on a set of 1,496 legal amounts (5,322 words) taken from our database of cheques described in Chapter 3. We tested the word recognizer on a different set of 712 legal amounts (2,515 words), some samples of which are shown in Appendix A. For clarification purposes, it should be noted that the people who wrote the cheques in the testing database are different from those who wrote the cheques in the training database.

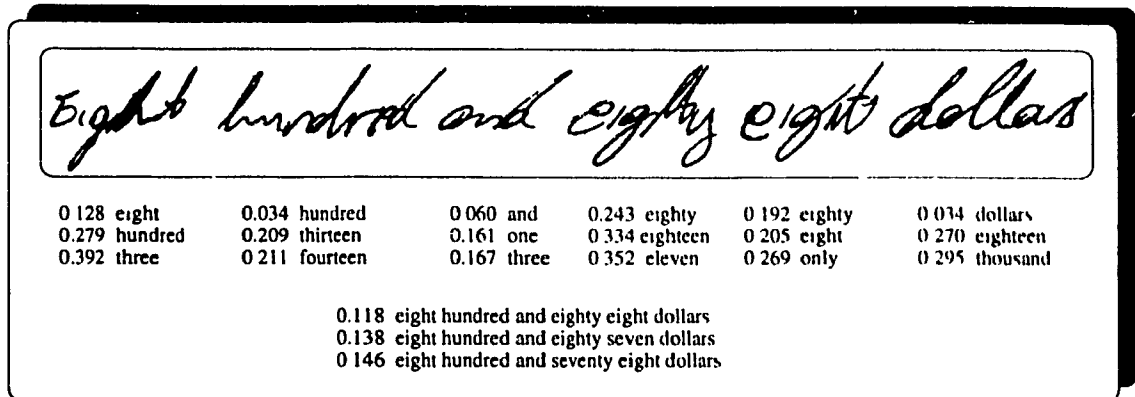


Figure 53: Example of legal amount recognition results

Features	$N =$	1	2	5	10
ADS		72.6	84.0	94.3	98.4
AD		63.9	79.1	92.7	98.0

Table 17: English word recognition results (% correct in top N choices)

We experimented with two versions of our classifier: (a) ADS as in “Ascender Descender Strokes” which considers the entire set of global features described in section 4.3.1, and (b) AD for “Ascender Descender” classifier which is a simplified version without the stroke features.

The size of our lexicon is 32 including the words ‘and’, ‘dollars’ and ‘only’. The results of our ‘global’ classifiers on those images are shown in Table 17 which lists the recognition rates for the correct result to be among the top N choices. Additionally we show the recognition results for each class of our lexicon (Table 18). Note that these results correspond to the words taken in isolation. The extent of the improvement resulting from the use of the parser will be discussed in section 5.4.

The relative importance of each feature for the overall recognition performance can be measured by the weights assigned by the genetic search (section 4.3.4) during training. The values of the weights are shown in Appendix E. We recall from section 4.3.4 that weights are in the range $[1 - 255]$. One can notice that greater weights are assigned to the position of features (w_{dap} , w_{ddp} , w_{dlp}) than to their numbers (w_{dan} ,

Features	ADS				AD					
Class	<i>N</i> =	1	2	5	10	<i>N</i> =	1	2	5	10
one		62.5	88.5	100.0	100.0		86.5	92.3	97.1	99.0
two		77.5	93.3	100.0	100.0		23.6	89.9	98.9	100.0
three		62.4	75.3	95.7	100.0		49.5	72.0	96.8	100.0
four		65.5	79.1	90.9	100.0		60.0	78.2	91.8	100.0
five		60.2	80.6	97.2	100.0		30.6	38.9	93.5	100.0
six		70.5	86.3	97.9	100.0		36.8	66.3	98.9	100.0
seven		52.6	75.3	91.8	100.0		54.6	74.2	91.8	100.0
eight		87.6	93.3	95.2	98.1		72.4	78.1	87.6	98.1
nine		63.0	80.4	93.5	100.0		43.5	60.9	91.3	98.9
ten		52.9	61.8	88.2	100.0		52.9	70.6	88.2	94.1
eleven		55.6	62.2	77.8	86.7		57.8	66.7	77.8	84.4
twelve		57.1	64.3	71.4	96.4		42.9	67.9	89.3	96.4
thirteen		58.8	73.5	94.1	100.0		35.3	52.9	73.5	88.2
fourteen		56.7	73.3	83.3	93.3		43.3	66.7	90.0	93.3
fifteen		59.5	71.4	83.3	92.9		40.5	54.8	71.4	88.1
sixteen		39.1	52.2	69.6	100.0		25.0	45.8	75.0	95.8
seventeen		45.5	54.5	77.3	86.4		45.5	50.0	72.7	81.8
eighteen		74.1	81.5	92.6	96.3		63.0	77.8	88.9	100.0
nineteen		36.7	60.0	90.0	93.3		48.3	69.0	93.1	96.6
twenty		70.0	77.5	90.0	92.5		45.0	72.5	90.0	92.5
thirty		73.3	84.4	86.7	91.1		55.6	68.9	84.4	95.6
forty		58.7	73.9	87.0	93.5		58.7	69.6	91.3	100.0
fifty		60.5	73.7	81.6	94.7		47.4	68.4	78.9	89.5
sixty		37.5	62.5	87.5	97.5		37.5	55.0	72.5	97.5
seventy		59.2	71.4	93.9	98.0		59.2	75.5	91.8	98.0
eighty		86.0	93.0	95.3	100.0		81.4	90.7	95.3	100.0
ninety		47.5	67.5	92.5	95.0		50.0	67.5	92.5	100.0
hundred		95.1	97.7	99.2	99.7		95.1	98.2	99.2	99.7
thousand		80.9	93.2	97.5	98.8		79.6	94.4	96.9	98.1
and		90.4	93.6	99.2	100.0		81.6	94.8	98.0	99.2
dollars		72.5	83.1	95.8	99.3		73.2	83.1	92.3	98.6
only		72.7	77.3	95.5	95.5		36.4	63.6	77.3	95.5
Average		72.6	84.0	94.3	98.4		63.9	79.1	92.7	98.0

Table 18: English word recognition results per class (% correct in top *N* choices)

w_{ddn} , w_{dln}). The greatest weight is assigned to the position of descenders. This reflects the fact that descenders appear less frequently than ascenders in the English language and consequently they carry more information. One can also notice that unexpectedly the weight assigned to the word length is almost the smallest one. This might reflect the fact that reliable estimation of the word length (e.g. the number of letters in a word) is hard due to the great variability in handwriting styles.

From the analysis of the results (Table 17 & 18), we note that the crude classifier performs better on those words that have distinctive global features such as ‘hundred’, ‘thousand’ or ‘and’. The distinction between some word classes can be problematic. For example, ‘Nine’ and ‘One’ both have similar features; an ascender at the beginning of the word and a loop at the end of word, the loop in the ‘O’ not being a robust feature. Since the AD classifier only makes use of mostly ascenders and descenders, it has difficulty differentiating among some pairs of word classes with few and similar features such as (‘One’, ‘two’), (‘One’, ‘Five’), (‘One’, ‘Six’), and (‘one’, ‘nine’).

In the results for the AD classifier (Table 18), it should be noted that for short words, due to the limited number of features in the input, it is often possible to get exactly the same distance of the input to several classes in the lexicon. For example, the input image of a word “nine” can often match equally the classes “one”, “two” and “nine”. Since the notion of a tie has not been implemented in our system, those classes, instead of appearing as tied in first place, would appear as first, second or third. In other words, in case of a tie, word classes are ordered in the same relative order as they appear in the data structure that defines our lexicon. Therefore in Table 18 for the AD classifier, recognition rates of 86.5 and 23.6 respectively for the classes “one” and “two” do not imply that samples of “one” are recognized more easily than those of “two”. In fact, it is only due to the fact that in case of a tie between “one” and “two”, “one” always appears first. In practice, for the AD classifier, both these classes are recognized with a low confidence value (section 4.3.5).

The use of stroke features in the ADS scheme allows to differentiate between the difficult cases discussed previously. Indeed one can easily foresee that the use of horizontal strokes should help significantly in differentiating among the classes ‘One’ and ‘two’. That is also what we realize experimentally. So the ADS scheme performs

Character sets	Character case	Lexicon size	$N =$	1	2	5
All	Lower	19		79.3	90.0	96.6
	Upper	15		78.9	90.6	97.5
First & Last	Lower	12		83.4	93.4	98.1
	Upper	10		81.2	92.9	99.1

Table 19: Character recognition results (% correct in top N choices)

better than the AD on those word classes with few non-distinctive global features. The AD and ADS schemes perform equally well on those word classes with distinctive global features. Those classes are recognized with high confidence values and therefore do not necessitate the extraction of additional features.

On the other hand, word classes with few global features (e.g. ‘one’, ‘two’, ‘five’, ...) are not recognized with high confidence values. Therefore, there is a need for some classes to extract another important feature for a ‘fast’ reader [Wat77], namely the characters located next to blank spaces. That is we should try to identify the first and last letters of the words. Empirically we notice that the first letter is often disconnected from the rest of the word.

5.2 Character recognition

The character classifiers were trained on a set of 5,615 characters (mixed lower and upper case) extracted from our training images. We then tested on a different set of 2,414 characters from our testing databases. A few character samples extracted from our testing databases are shown in Appendix A. Statistics on the number of samples per class can be found in section 3.7.

For our specific application, we are faced with a lexicon consisting of 19 different classes for the lower-case characters and 15 classes for the upper-case characters. The results for both lower and upper-case characters are reported in Table 19. Since we have to recognize only the first and last characters of words, we trained our classifiers with the first and last isolated characters of words from our training databases. In this

particular case, the lexicon sizes are reduced to 12 and 10 respectively for the lower and upper-case characters. The recognition results on those characters are shown in Table 19. It is to be noted that all of the results reported in Table 19 are solely based on the character bitmap information. None of the contextual information described in section 4.3.7 (e.g. the location of the reference lines) has been used here.

We should note that for these classifiers, the number of training patterns can be as few as 1 or 2 samples per class. The lack of data is especially noticeable when considering only the first and last characters of words. Still we believe that these classifiers can evaluate the effect of integrating the character results and the word recognition results.

5.3 Integrating character and word recognition

One should not underestimate the difficulty of integrating heterogeneous recognition sources, namely the character recognition results and the word recognition results. Numerous investigations have been made on the combination of homogeneous recognizers [Ho92] which might imply that the subject is non-trivial. The scheme we implemented for combining the character and the word recognition results is only a first step towards a more complete study on the art of combining 2 heterogeneous recognition sources. The present scheme uses only very slightly the contextual information described in section 4.3.7. As of today, we believe that we only investigated the feasibility of integrating those 2 recognition sources. The next step will be to fully use the complementarity of these 2 schemes in order to enhance the recognition performance.

When merging the character recognition results with the word results, using the scheme described in section 4.3.8, we obtain the new results shown in Table 20. One can notice that since the character results are used to reorganize the top 10 solutions output by the word classifier, the top 10 score remains unchanged. For the AD classifier, we notice a small improvement in the top 5 solutions, while the main enhancement resides in the boosting of the top 1 and 2 recognition rates. This is due to the fact that the AD scheme lacks power to differentiate between classes such

Classifier	Features	$N =$	1	2	5	10
Global + Character	ADS		73.5	84.6	94.8	98.4
Global only			72.6	84.0	94.3	98.4
Global + Character	AD		71.1	82.7	93.5	98.0
Global only			63.9	79.1	92.7	98.0

Table 20: English word recognition results when merged with character recognition

as ‘One’ and ‘two’. Upon integrating the recognition results on the first character with the AD scheme, a distinction can then be made between those 2 classes. The enhancement observed when using the ADS scheme is not as significant since the latter already has some capabilities of discerning some classes of first and last characters with the use of horizontal, vertical and diagonal strokes. While the enhancement in performance is not that significant for the ADS scheme, the integration of the character recognition results still serves the goal of increasing the confidence value of the top choice solutions when both word and character results overlap.

While we designed a scheme to integrate character with word results, little has been done to investigate how the distance or confidence value of a given word hypothesis should be boosted when its recognition is supported by the character results. This point becomes especially important when combining the word results to produce legal amount candidates.

5.4 Legal amount recognition

Upon recognizing each word, the results are sent to a language specific parser (section 4.4) in order to output a list of the semantically correct legal amounts. Since the parser puts constraints on the combination of words, the word level recognition results increase accordingly. We computed statistics on the new word recognition results after taking into account the context of a legal amount (Table 21). We also report the recognition rates of having the correct full legal amount in the top N choices (Table 22). In that respect, it is important to remember that the recognition

Parser	Features	$N =$	1	2	5	10
Yes	ADS		77.4	88.2	96.3	98.2
No	ADS		73.5	84.6	94.8	98.4

Table 21: English word recognition results when using the English parser

Features	$N =$	1	2	3	4	5	10	20
ADS		44.4	58.7	64.9	69.5	76.2	82.0	87.0

Table 22: English legal amount recognition results

of the legal amount is only a ring in the chain making up a cheque processing system. The legal amount recognition results are to be combined with courtesy amount results as well as the amount read on some utility bills. As such, the results displayed in Table 22 reflect only the power of the legal amount module.

5.5 Extra: Processing of French cheques

While we were working on designing and implementing the legal amount recognition system for the English language, a database of French cheques was being collected using the same procedure and tools described in chapter 3. The current database of French cheques consists of 1,861 cheques for an estimated number of 600 different writers. We separated our pool of cheques into a training database of 1,345 cheques (4,513 words) and a testing database of 516 cheques (1,622 words). Similarly as we did for the English database, the authors of the cheques in the training database are different from those of the cheques in the testing database. A few samples of the legal amounts found in our testing database are shown in appendix A.4.

Language	Features	$N =$	1	2	5	10
French	ADS		71.9	84.6	94.0	98.7
	AD		78.3	91.8	98.9	99.9
English	ADS		72.6	84.0	94.3	98.4
	AD		63.9	79.1	92.7	98.0

Table 23: Word recognition results (% correct in top N choices)

5.5.1 French word recognition results

The availability of this new database is an important step in order to test the design of our system. We recall that we designed the current system with the primary concern of creating a fully trainable module. One of the requirements being that the switch to a different database and/or language should not require any redesign nor any reprogramming. As such we simply substituted the English training database with a French one as well as substituted the English parser for the French one described in section 4.4.2. To confirm that the above requirement still holds, we did not perform any adjustments whatsoever of the parameters mentioned in Appendix E. The results obtained are reported in Table 23 along with the previously obtained results on English words (from Table 17) for easy comparison. We also show the recognition results obtained for each class in our lexicon (Table 24).

The first interesting point that we notice is that without any redesign nor any reprogramming whatsoever, the system was able to be reused for the processing of French cheques. May be even more surprising is the fact that without any modifications of the parameters mentioned in Appendix E, the results that we obtain for the French cheques seem satisfactory. One can notice that for 2 classes, namely “huit” and “dix”, the recognition rates are much lower than for other classes. Samples of “huit” are misclassified as “cent” due to similar global features and the samples of “dix” are misclassified as “six” and “deux”, once again due to similar global features. A large number of these misclassifications are taken care of by the French language parser when recognizing the words in the context of a legal amount.

Further analysis of the system’s behaviour comes from a comparison of the results

French Parser	YES				NO					
Class	<i>N</i> =	1	2	5	10	<i>N</i> =	1	2	5	10
un		86.7	93.3	96.7	100.0		86.7	93.3	100.0	100.0
deux		76.1	94.0	100.0	100.0		73.1	89.6	98.5	100.0
trois		81.3	96.0	98.7	100.0		76.0	96.0	100.0	100.0
quatre		92.4	99.0	100.0	100.0		95.2	97.1	100.0	100.0
cinq		89.6	96.1	100.0	100.0		92.2	94.8	100.0	100.0
six		72.6	90.4	100.0	100.0		64.4	83.6	100.0	100.0
sept		92.0	98.0	100.0	100.0		94.0	98.0	98.0	98.0
huit		69.9	91.8	98.6	100.0		27.4	78.1	100.0	100.0
neuf		76.6	95.3	96.9	100.0		71.9	92.2	98.4	100.0
dix		45.5	70.9	100.0	100.0		18.2	58.2	96.4	100.0
onze		84.4	91.1	95.6	100.0		79.5	88.6	95.5	100.0
douze		82.9	95.1	97.6	100.0		82.9	97.6	97.6	100.0
treize		59.5	83.8	100.0	100.0		73.0	89.2	100.0	100.0
quatorze		97.8	97.8	97.8	97.8		97.8	97.8	97.8	100.0
quinze		83.3	90.0	93.3	100.0		86.7	90.0	96.7	100.0
seize		51.4	68.6	97.1	100.0		51.4	74.3	97.1	100.0
vingt		92.6	98.1	98.1	100.0		85.2	94.4	98.1	100.0
trente		80.0	95.6	100.0	100.0		64.4	91.1	100.0	100.0
quarante		76.5	94.1	100.0	100.0		91.2	97.1	100.0	100.0
cinquante		95.0	100.0	100.0	100.0		95.0	100.0	100.0	100.0
soixante		84.5	97.2	98.6	100.0		84.5	94.4	98.6	100.0
cent		89.6	98.8	100.0	100.0		87.3	97.3	99.6	100.0
mille		89.4	98.5	100.0	100.0		80.3	93.9	98.5	100.0
et		95.7	95.7	100.0	100.0		95.7	95.7	100.0	100.0
dollars		96.9	98.5	100.0	100.0		89.2	92.3	96.9	100.0
Average		83.1	94.5	99.1	99.9		78.3	91.8	98.9	99.9

Table 24: French word recognition results per class with AD features

obtained respectively from the English and the French words (Table 23). The most obvious difference between the French and the English lexicon comes from the smaller size of the French lexicon. One can notice that while the results for the ADS classifier remained the same, the results for the AD classifier have significantly increased.

The results obtained for the French cheques seem to indicate that our feature extraction process is extremely robust versus different databases and languages. The results obtained for the AD classifier seem to indicate that our ascender and descender features and their associated parameters are independent of any database and/or language. On the other hand, the ADS classifier does not seem to have fully taken advantage of the smaller French lexicon; and that might imply that the stroke features are not as robust as their ascender and descender counterparts.

One additional reason for the significantly higher recognition results for French words when compared to English with the AD classifier (Table 23) is probably the fact that French words do possess more distinctive ascender and descender features among themselves. Whereas many English words have similar or no distinctive features, French words do possess comparatively many ascenders and descenders.

In order to obtain higher recognition results for the ADS scheme on the French database, one would need to optimize the weights of the classifier by running the genetic algorithms mentioned in section 4.3.4.

5.5.2 French legal amount recognition results

Similarly as we did for the English cheques, we computed statistics on the word recognition results when making use of the French parser (Table 25), as well as reporting results on the recognition of the correct full legal amounts among the top N choices (Table 26). Table 26 includes the results on English legal amounts for easy comparison.

Intuitively we expect the parser to increase significantly the recognition rates. Indeed due to the nature of the French language, French amounts are generally longer, in the number of words, when compared to their English equivalents. For example, the English amount “ninety” is expressed in French with the 3 words “quatre vingt dix”. Consequently, as can be seen from Figs. 51 & 52, the French parser has more

Parser	Features	$N =$	1	2	5	10
Yes	AD		83.1	94.5	99.1	99.9
No	AD		78.3	91.8	98.9	99.9

Table 25: French word recognition results when using the French parser

Language	Features	$N =$	1	2	3	4	5	10	20
French	AD		65.7	83.9	88.6	91.6	94.3	98.6	99.8
English	ADS		44.4	58.7	64.9	69.5	76.2	82.0	87.0

Table 26: Legal amount recognition results (% correct in top N choices)

constraints, e.g. is more complicated, than the English one. The longer legal amounts enable the parser to work more effectively; e.g. when an amount is made of a single word, the parser has absolutely no effect.

Experimentally, we notice that the French parser contributes significantly to enhancing the recognition rate at the word level of the top 1 and 2 choices when compared with the results obtained without the parser. Additionally, we notice that the combination of the smaller French lexicon with more distinctive French words and a more severe French parser leads to a real improvement when comparing the recognition of the French legal amounts (Table 26) with the English ones.

5.6 Comparison of results

We recall that the 3 major components that influence the complexity of a handwriting recognition problem are the size of the lexicon, the type of handwriting and the number of writers. The processing of handwritten bank cheques is a problem of a small static lexicon with unconstrained handwriting and unlimited number of writers. Therefore the only constraint is on the size of the lexicon and any attempts to tackle the bank cheque processing problem should be tested on databases that do respect to a certain extent the 2 major problem characteristics, i.e. unconstrained handwriting

	Language	Classifier	$N =$	1	2	5	10
Guillevic & Suen	English	Global features		73.5	84.6	94.8	98.4
Guillevic & Suen	French	Global features		78.3	91.8	98.9	99.9
Gilloux & Leroux	French	HMM		79	87	95	98

Table 27: Comparison of word recognition results (% correct in top N choices)

and an unlimited number of writers. Unfortunately, in the literature, very few studies on the processing of bank cheques do report results on databases which involve at least a few hundred writers. As mentioned in section 3.1, to the best of our knowledge most studies report results on small databases of only 5 to 25 writers. These restrictions are obviously contrary to the nature of the bank cheque processing problem. Therefore these studies fall into a class of problem dealing with a small lexicon, a small number of writers and a somewhat less “unconstrained” handwriting. The small size of the database coupled with the small number of writers involved make “true” unconstrained handwriting hard to achieve.

The lack of studies in the literature makes the comparison of results rather difficult. As far as we know, the only study reporting results at the word level on a significant database is done by a research team at the French Post Office [GL93]. They trained their HMM based recognizer on a set of 2,492 word samples written by unknown writers on postal cheques. The test set was made of another set of 2,492 words. Their results as well as ours are reported in Table 27. Comparing results is not easy since they refer to different databases, so we have to view the comparison on that basis. It seems that at present our results are comparable to theirs.

5.7 Novel word spelling

While working on the databases that we collected on the island of Montréal, we had to overcome some of our preconceptions about certain word spelling. Indeed we have been faced with word spellings that could neither be found in our English dictionary [Sin89] nor in our French dictionary [Dub86]. On the other hand, due the frequency of

Language	Original word	Novel spelling
English	fourteen	forteen
	forty	fourty
	ninety	ninty
French	dollar	dollard

Table 28: Novel word spelling

appearance of such novel spellings, they could simply not be ignored and were inserted into our training databases. The most common novel spellings that we encountered are enumerated in Table 28.

5.8 Demonstration's interface

In order to have “real-life” demonstrations, we designed a user interface (Fig. 54) incorporating all of the modules developed so far in our system. It incorporates the preprocessing operations, the basic segmentation module, the word recognition as well as the parser. Upon clicking on appropriate buttons, one can demonstrate the various features of the system as well as get the result from the recognition system. That interface is also used to recognize on the spot the handwritten legal amounts generated by the numerous visitors of our centre.

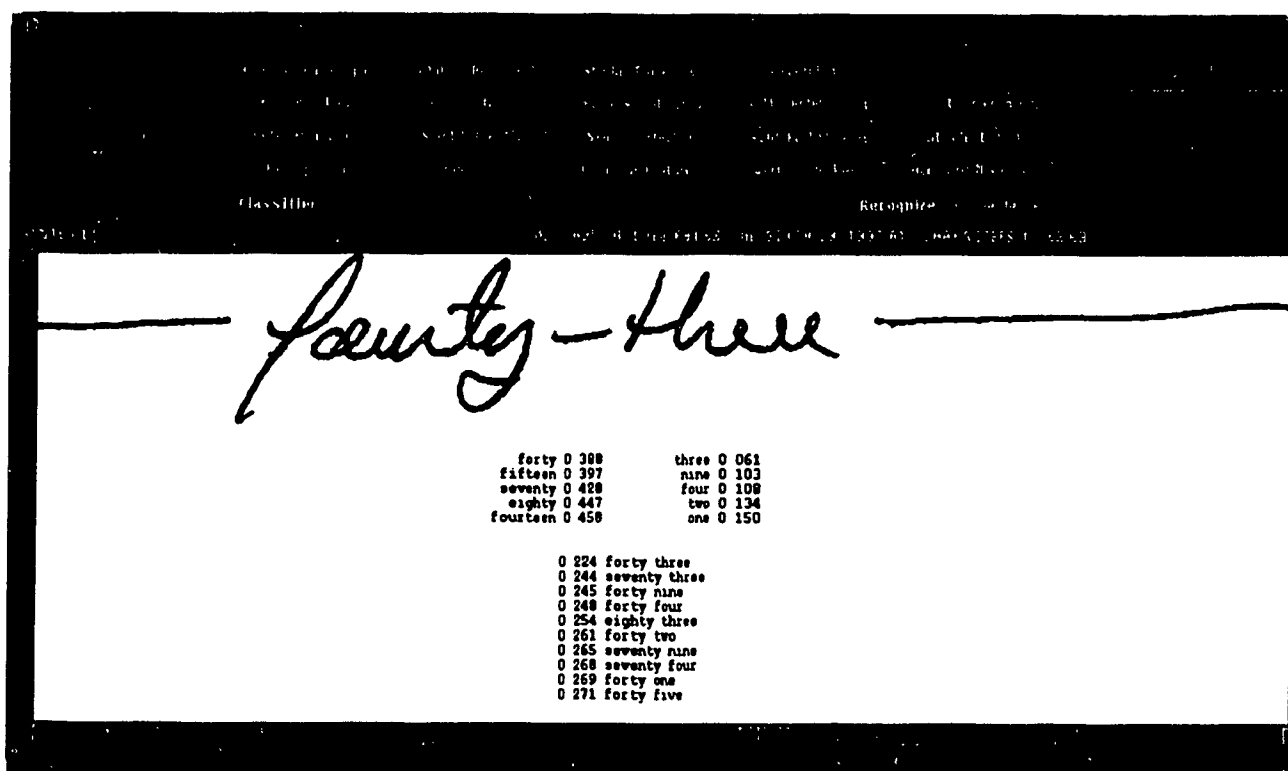


Figure 54: User interface for demonstration purposes

Chapter 6

Conclusion

6.1 Summary of contributions

The first major contribution of this work is the creation of a database of cheques that we later used for training and testing our system. This resulted in a *one of its kind* database in a university setting dealing with handwritten information from bank cheques. The *in-place* truthing procedure is also innovative and unique in the literature. Its simplicity, yet power, makes it a real asset for further generations of researchers working on our database. It can also be easily applied to the tagging stage of other databases.

This work introduced yet another simple, but no less efficient, algorithm dealing with the slant correction of handwritten words. Its power comes not only from its simplicity but also from its robustness versus the numerous handwriting types found in our database.

Last but not least, we designed a fully trainable word recognizer that should be able to handle any small static lexicon of words written with the Roman alphabet. So far tests have been made with the French and English languages. Our computational theory based on a psychological model of the reading process for a fast reader shines by its simplicity and still compares equally with other published schemes. It is to be noted that this work is the sole study in the literature reporting results on the recognition of handwritten English bank cheques tested on a significant database.

This work also compares equally with the sole other study reporting results on the processing of handwritten French bank cheques tested on a significant database.

It should also be noted that this work seems to represent the sole study implementing the psychological model of reading of the fast reader. In other words, our word recognizer in its first step appears to be the only one in the literature completely bypassing the notion of characters in order to recognize words. All other previous studies seem to have focused on the notion of characters in order to recognize words. They do so either by segmenting the words into characters or parts of characters or by extracting some global features to identify the presence of certain character classes. Therefore all previous word recognition studies seem to be implementing the psychological model of the reading process of a slow reader.

6.2 Strengths and weaknesses of the method

One goal of this study was the design of a word recognizer that would not require any major redesign nor any extensive retraining time when the switch would be made to a different training database and/or to different languages. Indeed being physically located in bilingual Canada, the ultimate goal of this study is to provide a system being able to process at least the 2 official languages, namely English and French. In this respect, some of the strengths of the current approach are:

- The automatic learning capability of the system. The switch to a different database or language does not require any redesign nor any reprogramming of the system. The current system can be used to learn and recognize any small static lexicon of words based on the Roman alphabet. So far it has been tested on French and English.
- Our procedure to extract global features such as ascenders and descenders has proven to be very robust versus the numerous handwriting types found in our database.
- The number of parameters (Appendix E) has been kept fairly small for a relatively complex system.

- The features as well as the design of the system are of such a general and robust nature that our system can be easily applied to different languages based on the Roman alphabet.

The principal weakness of the current approach is its difficulty in discerning word classes with few distinctive global features. In the current application domain in the English language, these are word classes such as ‘sixteen’, ‘seventeen’, ‘seventy’ and ‘ninety’. For those word classes, the global recognizer performs poorly. Therefore more emphasis and work has to be put into the extraction and recognition of *word details* as well as in the combination of the word and word-details recognition results.

6.3 Future work

This work has built a solid foundation for a more thorough investigation into the still sparsely studied recognition of handwriting applied to the processing of bank cheques. Future work will certainly concentrate on some of the following items:

- Improvement of the *word details* extraction. This might be realized by using a more thoroughly studied segmenter such as the one in [Str93].
- Creation of a *word details* training database using the *word details* extraction tools. It should be remembered that we do not aim at recognizing characters but rather distinctive entities located at the beginning and end of words (Fig. 47).
- Possibly extraction of word details in addition to the first and last characters of words.
- Thorough study into the combination of the heterogeneous recognition results, namely words and word details.
- Communication between and combination of the legal amount processing module, the courtesy amount module and the payment slip module.
- Eventually, as is done in the widely investigated field of digit recognition, the combination of several word classifiers should be studied. Due to their different

nature, the combination of our *word approach* computational theory with a more conventional character based approach should prove to be interesting.

References

- [BBD⁺93] C.J.C. Burges, J.I. Ben, J.S. Denker, Y. Lecun, and C.R. Nohl. Off line recognition of handwritten postal words using neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):689-704, 1993.
- [BG80] M. K. Brown and S. Ganapathy. Cursive script recognition. In *Proceedings of the International Conference on Cybernetics and Society*, pages 47-51, Boston, MA, Oct 1980.
- [BG83] M. K. Brown and S. Ganapathy. Preprocessing techniques for cursive script word recognition. *Pattern Recognition*, 16(5):447-458, 1983.
- [BHD91] M. Mohammad Beglou, M.J.J. Holt, and S. Datta. Slant independent letter segmentation for cursive script recognition. In *International Workshop on Frontiers of Handwriting Recognition*, pages 375-380, Bonas, France, 1991.
- [BM90] Dan Bloomberg and Petros Maragos. Generalized hit-miss operations. In *SPIE Image Algebra and Morphological Image Processing*, pages 116-128, 1990. vol. 1350.
- [BS89] Radmilo M. Bozinovic and Sargur N. Srihari. Off-line cursive script word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):68-83, Jan 1989.
- [Cas94] Sean Casey. Checks still america's favorite payment method. *Item Processing Report*, page 7, December 1994.

- [CGM93] T. Caesar, J. M. Gloger, and E. Mandler. Design of a system for off-line recognition of handwritten word images. In *Proc. European Conference dedicated to Postal Technologies*, pages 156–161, Nantes, France, June 1993. Service de Recherche Technique de la Poste.
- [CGM95] Torsten Caesar, Joachim M. Gloger, and Eberhard Mandler. Estimating the baseline for written material. In *Proc. International Conference on Document Analysis and Recognition*, pages 382–385, Montreal, Canada, August 1995. IEEE Computer Society Press.
- [CHHH⁺90] Peter B. Cullen, Tin Kam Ho, Jonathan J. Hull, Michal Prussak, and Sargur N. Srihari. Contextual analysis of machine printed addresses. In *Proc. USPS Advanced Technology Conference*, pages 779–793, 1990.
- [DH72] R. Duda and P. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1), January 1972.
- [Dub86] Jean Dubois, editor. *Dictionnaire de la Langue Fran caise, lexis*. Larousse, Paris, France, 1986.
- [EK75] Roger W. Ehrich and Kenneth J. Koehler. Experiments in the contextual recognition of cursive script. *IEEE Transactions on Computers*, c-24(2):182–194, February 1975.
- [Far79] Raoul F. H. Farag. Word-level recognition of cursive script. *IEEE Transactions on Computers*, c-28(2):172–175, February 1979.
- [Fav93] John T. Favata. *Recognition of Cursive, Discrete and Mixed Handwritten Words Using Character, Lexical and Spatial Constraints*. PhD thesis, State University of New York at Buffalo, 1993.
- [FS90] John T. Favata and Sargur N. Srihari. Recognition of handwritten words for address reading. In *Proc. USPS Advanced Technology Conference*, pages 191–205, 1990.

- [GBL93] Michel Gilloux, Jean-Michel Bertille, and Manuel Leroux. Recognition of handwritten words in a limited dynamic vocabulary. In *Proc. European Conference dedicated to Postal Technologies*, pages 148–155, Nantes, France, June 1993. Service de Recherche Technique de la Poste.
- [GL93] Michel Gilloux and Manuel Leroux. Recognition of cursive script amounts on postal cheques. In *Proc. European Conference dedicated to Postal Technologies*, pages 705–712, Nantes, France, June 1993. Service de Recherche Technique de la Poste.
- [Gol89] David E. Goldberg. *Genetics Algorithms in Search, Optimization and Machine Learning*. 1989.
- [Goo67] Kenneth S. Goodman. Reading: A psycholinguistic guessing game. In Frederick V. Gollasch, editor, *Language and Literacy: The selected writings of Kenneth S. Goodman*, pages 33–43. Routledge & Kegan Paul, 1967.
- [GS93] Didier Guillevic and Ching Y. Suen. Cursive script recognition: A fast reader scheme. In *Proc. International Conference on Document Analysis and Recognition*, pages 311–314, Tsukuba Science City, Japan, October 1993.
- [GS94] Didier Guillevic and Ching Y. Suen. Cursive script recognition: A sentence level recognition scheme. In *International Workshop on Frontiers of Handwriting Recognition*, Taipei, Taiwan, December 1994.
- [GS95] Didier Guillevic and Ching Y. Suen. Cursive script recognition applied to the processing of bank cheques. In *Proc. International Conference on Document Analysis and Recognition*, pages 11–15, Montreal, Canada, August 1995.
- [GW87] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing*, pages 125–130. Addison-Wesley, 1987.

- [HHS94] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1):66-75, January 1994.
- [Ho92] Tin Kam Ho. *A Theory of Multiple Classifier Systems and Its Application to Visual Word Recognition*. PhD thesis, State University of New York at Buffalo, May 1992.
- [HSZ87] Robert M. Haralick, Stanley R. Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(4):532-550, July 1987.
- [Hue08] Edmund Burke Huey. *The psychology and pedagogy of reading : with a review of the history of reading and writing and of methods, texts, and hygiene in reading*. Cambridge [Mass.] : M.I.T. Press, 1908. 1968 ed. contains new foreword and introduction.
- [Hul94] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550-554, May 1994.
- [Jai89] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.
- [KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1988.
- [KSN93] F. Kimura, M. Shridhar, and N. Narasimhamurthi. Lexicon directed segmentation - recognition procedure for unconstrained handwritten words. In *International Workshop on Frontiers of Handwriting Recognition*, pages 122-131, Buffalo, NY, 1993.
- [LF89] C.G. Leedham and P.D. Friday. Isolating individual handwritten characters. In *IEE Colloquium on 'Character Recognition and Applications' (Digest No.109)*, pages 4/1-7, London, UK, October 1989.

- [LL94] Seong-Whan Lee and Dong-June Lee. Slant estimation and correction for off-line hangul script using hough transform. In *International Conference on Computer Processing of Oriental Languages*, pages 353–358, Taejon, May 1994.
- [LSG⁺95] L. Lam, C. Y. Suen, D. Guillevic, N. W. Strathy, M. Cheriet, K. Liu, and J. N. Said. Automatic processing of information on cheques. In *IEEE Int. Conference on Systems, Man & Cybernetics*, Vancouver, Canada, October 1995.
- [Mas75] D.W. Massaro, editor. *Understanding Language : an Information-Processing Analysis of Speech Perception, Reading, and Psycholinguistics*. Academic Press, 1975.
- [MG92] Sriganesh Madhvanath and Venu Govindaraju. Using holistic features in handwritten word recognition. In *Proc. USPS Advanced Technology Conference*, pages 183–198, 1992.
- [Mor91] Jean-Vincent Moreau. A new system for automatic reading of postal checks. In *International Workshop on Frontiers of Handwriting Recognition*, pages 121–132, Bonas, France, 1991.
- [MSY92] Shunji Mori, Ching Y. Suen, and Kazuhiko Yamamoto. Historical review of ocr research. *Proceedings of the IEEE*, 80(7):1029–1058, July 1992.
- [NB65] U. Neisser and H.K. Beller. Searching through word lists. *British Journal of Psychology*, 56:349–358, 1965.
- [Pav82] Theo Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.
- [Pit29] Walter Boughton Pitkin. *The art of rapid reading; a book for people who want to read faster and more accurately*. New York : McGraw-Hill, 1929.
- [PM92] Theo Pavlidis and Shunji Mori, editors. *Proceedings of the IEEE*, volume 80. IEEE, July 1992.

- [Pre92] William H. Press. *Numerical recipes in C: the art of scientific computing*. Cambridge University Press, 1992.
- [Say73] Kenneth M. Sayre. Machine recognition of handwritten words: A project report. *Pattern Recognition*, 5:213-228, 1973.
- [SBM80] Ching Y. Suen, Marc Berthod, and Shunji Mori. Automatic recognition of handprinted characters - the state of the art. *Proceedings of the IEEE*, 68(4):469-487, April 1980.
- [SBSV94] Stephen J. Smith, Mario O. Bourgoïn, Karl Sims, and Harry L. Voorhees. Handwritten character classification using nearest neighbor in large databases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):915-919, September 1994.
- [SC94] Giovanni Seni and Edward Cohen. External word segmentation of off-line handwritten text lines. *Pattern Recognition*, 27(1):41-52, 1994.
- [Sch92] Jürgen Schürmann. Document analysis - from pixels to contents. *Proceedings of the IEEE*, 80(7):1101-1119, July 1992.
- [Ser82] Jean Paul Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1982.
- [She75] Wayne Shebilske. Reading eye movements from an information-processing point of view. In Dominic W. Massaro, editor, *Understanding Language: An Information-Processing Analysis of Speech Perception, Reading, and Psycholinguistics*, chapter 8, pages 291-311. Academic Press, 1975.
- [Sin89] John Sinclair, editor. *Collins Cobuild English Language Dictionary*. William Collins Sons & Co Ltd, London, England, 1989.
- [Smi69] F. Smith. The use of featural dependencies across letters in the visual identification of words. *Journal of Verbal Learning and Verbal Behavior*, (8):215-218, 1969.

- [Smi85a] Frank Smith, editor. *Reading*. Cambridge: Cambridge University Press, 2nd edition, 1985.
- [Smi85b] Frank Smith, editor. *Reading without nonsense*. New York: Teachers College Press, 2nd edition, 1985.
- [Smi88] Frank Smith, editor. *Understanding reading: a psycholinguistic analysis of reading and learning to read*. Hillsdale, N.J. : L. Erlbaum Associates, 4th edition, 1988.
- [Smi94] Frank Smith, editor. *Understanding reading*. Hillsdale, N.J. : L. Erlbaum, 5th edition, 1994.
- [SNL+92] Ching Y. Suen, Christine Nadal, Raymond Legault, Tuan A. Mai, and Louisa Lam. Computer recognition of unconstrained handwritten numerals. *Proceedings of the IEEE*, 80(7):1162–1180, July 1992.
- [Sri92] Sargur N. Srihari. High-performance reading machines. *Proceedings of the IEEE*, 80(7):1120–1132, July 1992.
- [Str93] Nicholas W. Strathy. A method for segmentation of touching handwritten numerals. Master's thesis, Concordia University, Montréal, Québec, Canada, September 1993.
- [Wat77] Gloria Sydna Waters. The word superiority effect in fluent and less fluent readers. Master's thesis, Concordia University, Montreal, Quebec, Canada, September 1977.
- [Zac84] G. Zachopoulos. Slant estimation and correction for off-line cursive script. Master's thesis, State University of New York at Buffalo, 1984.

Appendix A

Handwriting Samples

We show in this appendix some samples of handwriting taken from our English and French testing databases. It is to be noted that occasionally some images of legal amounts, words or characters had to be scaled down in order to be displayed in this appendix. Therefore the differences in size for a given legal amount, word or character are even greater than what is shown in the figures of this appendix.

A.1 English legal amounts

Some samples of legal amounts taken from our English testing database are shown in the following pages.

Nine hundred and fifty-one — .34

Nine hundred and nine, sixteen cents

— Fifty four — 53

— *Fifteen* — →

— Six Thousand and Eighty Four — 66

Forty - Seven —

Eighteen dollars only —

Seventy - three —

Seven thousands . Seven hundreds and sixty one — 00

— Nine — 00

— Four hundred eighty four —

seven thousand and Five Hundred and Twelve dollars only

Eight hundred nineteen —

one —

— one hundred seventeen — 35

- nine hundred and nineteen — 16
 Eight thousand and sixty-seven — 84

= six hundred and seven $\frac{00}{100}$

- seven hundred fifty eight — ~~12~~

———— Nineteen and 14
 Twenty eight Dollars & thirty six cents

Thirteen. Dollar ONLY —————

fifteen ————— .90

Eight - Thousand Twelve ————— 20

Nineteen ————— 00

ten ————— 67

Six hundred ninety two —

-Thirteen ————— 43

Eight hundred nineteen ————— 02

———— eleven dollars —————

One thousand Six Hundred & Fourteen
 five thousands two hundreds eighty one —
 only three hundred thirty four and eighteen cents.
 only Three thousand and eight dollars —

One and forty-nine cents

~~~~~ forty eight ~~~~~ 00

Six hundred & thirty four — 00/xx

~~~~~ fifteen ~~~~~ 00

Four hundred and twenty four dollars

Seventeen ~~~~~ 47

Fourteen ~~~~~ 68

Eight hundred and twenty-five 16

Two hundred sixty ~~~~~ 11

eight hundred and eleven ~~~~~ xx

Seventeen And ~~~~~ 55

Eight Hundred Eighty-Three and 93

Nine thousand seven hundred fifteen _____

- Six hundred nineteen _____ 39

three _____ 00

→ Fifteen dollars, _____ 00

→ Twenty Five _____

Four hundred and sixty-two

- Tree hundred - eleven _____ 88

Twelve dollars nineteen

one thousand seventy-six _____ 63

— four hundred and ten _____ 45

- Two hundred and eleven _____ 00

- Forty-three dollars _____ 19

two hundred and nineteen dollars _____

Fifty-two _____ 51

_____ eleven _____ 14

- Four hundred and eleven _____ 53

Two hundred and twenty four _____ 27

eight thousand four hundred and twelve _____ 33

Seven thousand three hundred

forty-one _____ 25

Six thousand five hundred thirty two ⁴⁵

Ten _____ 47

Four hundred and sixty-one dollars _____ 63

Two _____ 93

seven hundred and seventy-seven ⁸⁷//

Four hundred and sixty dollars _____ 01

Ninety-one dollars _____ 55

— Four _____

Two hundred & forty ²⁴

— nine hundred ninety Two and $\frac{05}{100}$ —

— seven thousand and ninety-four —

Twenty seven —————

five hundred and fifteen ————— 05

Four hundred & twenty ————— 00

three hundred and fourteen

— one hundred eighty nine —^{xx}

— Eleven ————— 52

— Ten ————— xx

sixty five ————— xx

Seventeen

One hundred seventeen —————^{xx}

— three thousand three hundred & fifty one — xx

— Six ————— 00

Five hundred nineteen ————— 06

sixty three dollars ————^{xx}

— one thousand nine hundred fifteen ————^{xx}

Three hundred and ²⁵

Only Eight Thousands four hundred and ten ————^{xx}

— three hundred thirty ————^{xx}

six hundred and nineteen ————⁰⁰

— eleven ————⁹⁸

— ninety-one thousand two hundred fifty ————⁹⁸

Six Thousand Four Hundred Seven ————^{xx}

Twenty five ————⁰⁰

— three hundred forty one ————⁶¹

thirty-five ————^{xx}

— Forty-four ————

— Four hundred and thirty dollars ————³⁷

Eight Thousand and Twenty seven Dollars ————

Eight hundred and thirteen ~ 44

Three hundred and sixty nine _____ xx

nine hundred eighty-four _____ |

Forty ~ _____ 95

_____ Forty Eight _____

eight hundred eleven. _____ | xx

Eight Hundred + Sixteen Dollars _____ $\frac{xx}{100}$

Six hundred and six _____ 56

— Nine thousand One Hundred Sixty Three —

Fourteen _____ 00

Two Dollars _____

seventy six ct _____ 02

— Seventy seven _____ 27

ten dollars and eighty three cents

seventeen dollars _____

- eight hundred thirty-four ———^{xx}

Seventeen dollars ———¹⁰

Three hundred nineteen
Thirty-five —————⁴²

Eight hundred thirty nine ⁵¹

————— two —————^{xy}

- Two hundred Eleven —————→ ⁰⁰

Eight Hundred Eighty two ⁸¹

Eleven —————^{xx}

————— eight thousand one hundred fifty ^{xx}

eight hundred twenty-nine dollars — ⁵⁴
_{xx}

Five hundred thirteen and cents eighty-seven only

Seven hundred and fifty two ^{xx}

Six hundred and eighteen ————⁰⁰

Four twenty-seven ————¹⁵

— five thousand fifty two —————*

Nine hundred eighteen ———**

- seven hundred and thirty three ———

fifty nine thousand fifteen $\frac{64}{100}$ ———

- two hundred sixty five —————⁵⁵

three thousand five hundred⁰⁰

Four Hundred and Twenty - Six ²⁵

Nine Thousand Nine Hundred And fifteen ———

one only —————

Seven hundred and thirty - one ———*

five hundred and twelve ———³⁰

Twenty thousand seven hundred ———¹³

Three thousand eight hundred and one ———

— four dollars only —————

Four hundred and fifteen dollars and forty - seven

Sixteen _____ 94
 — Seven Dollars _____ 50
 Sixty Thousand and Seventeen _____ 58
 Seventy three dollars _____ 57
 FIVE THOUSAND FIVE HUNDRED and FIFTY nine —
 Seven hundred and thirteen
 — fourteen _____ 70
 one hundred fourteen dollars only
 — nine _____
 Thirty three _____ 90
 Five hundred sixty-nine dollars & sixty-six cents
 — twenty six _____ 53
 — five — hundred — and — ninety — five — 00
 — Four Thousand and eleven dollars _____ 50
 — eight hundred twenty _____ 30

— Fifteen Dollars ———— 15

Nine Thousand Three Hundred & Eleven

— Two hundred and eighty ———— 41

nine hundred sixty two ———— 56

Seventy-Nine dollars ———— 38

One hundred seventy seven ———— 70

fifteen dollars ———— .00

Eighteen dollars

Five Thousand Eight Hundred ninety — 84

four thousand two hundred and seventeen $\frac{88}{24}$

— five thousand six hundred & eleven — 41

\$ Four hundred Eighty-eight ———— .51

Seven Hundred Seventy Two and "

twelve dollars ————

Four hundred sixteen —

fifty nine thousand six hundred fifteen — 48 |

Nine hundred and ninety three —

Forty-eight dollars only —

— Ninety five thousand and one hundred and four —

Two thousand seven hundred and eleven —
six hundred & eighteen — 51/100

Three hundred + ninety one — 54

One hundred and thirteen

— two hundred and fifteen — 22

seventy-four — 80

five thousand eight hundred twenty — 44

Fifteen

77

fourty one thousand one hundred fifteen — 31 |

three hundred and eleven — 18

Five Thousand One Hundred & Nineteen — 22

- Four Hundred Twelve _____ 90

_____ Two hundred and fifty two ^{xx}

_____ Eight Thousand and Forty Two _____ 80

Eight Thousand Five Hundred Eighty Seven 67

nine hundred & thirteen 49

eleven _____

Eighty-eight dollars - 51

three hundred and thirteen

nine hundred and eighty five - 01

eight hundred and ten _____ ^{xx}

Five hundred and fifteen dollar and 29

_____ three hundred and ten _____ ^{xx}

four thousand twenty-four dollars ^{xx}

Eight-Thousand Eight Hundred and Fourteen - 00

Fifteen _____ 38

eight hundred thirty eight — 86

Five hundred and sixteen — 05

~ Eleven ~ 84

One hundred dollars

~ Nine thousand + thirteen ~ 21

one hundred and eleven dollars —

Ten dollars and fifty-five cents

Two hundred sixteen —

forty nine — 00

~ Ninethousand seven hundred eleven — ⁶⁵ 65

five-hundred and eighteen *

One hundred and forty eight 60

Nineteen and thirty-three cents

Sixty-seven dollars — 55

A.2 English words

The words shown in this section are also taken from our English testing database. They have been slant corrected. The words are different from the samples of handwriting shown in the previous section (A.1).

| | | | | |
|----------|----------|----------|----------|----------|
| and | and | and | and | and |
| and | and | and | and | and |
| and | and | and | and | and |
| dollar | dollar | Dollars | dollars | Dollars |
| dollars | dollars | Dollars | dollars | dollars |
| dollars | dollars | dollars | dollars | dollars |
| eight | Eight | eight | Eight | eight |
| eight | Eight | Eight | eight | eight |
| Eight | Eight | eight | Eight | eight |
| Eighteen | eighteen | Eighteen | Eighteen | Eighteen |
| Eighteen | eighteen | eighteen | Eighteen | Eighteen |
| eighteen | Eighteen | eighteen | eighteen | eighteen |
| eighty | eighty | Eighty | Eighty | eighty |
| Eighty | eighty | Eighty | eighty | Eighty |
| eighty | Eighty | Eighty | eighty | eighty |

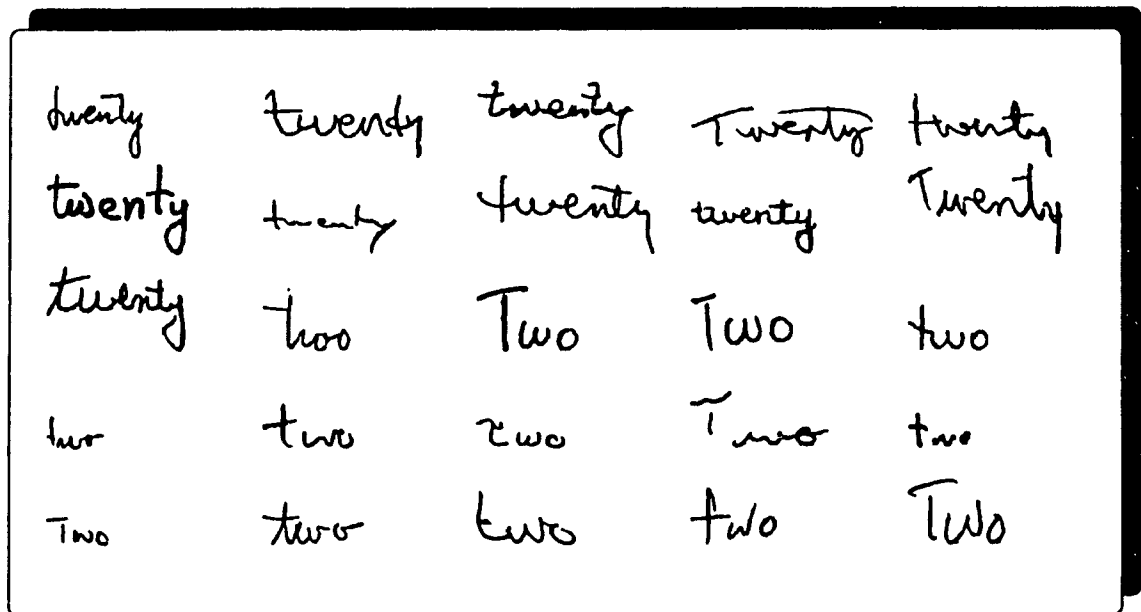
| | | | | |
|---------|---------|---------|---------|---------|
| eleven | eleven | Eleven | Eleven | eleven |
| Elava | Eleven | eleven | eleven | eleven |
| eleven | eleven | eleven | Eleven | eleven |
| Fifteen | Fifteen | fifteen | Fifteen | Fifteen |
| fifteen | fifteen | fifteen | fifteen | Fifteen |
| fifteen | fifteen | fifteen | fifteen | fifteen |
| Fifty | fifty | fifty | Fifty | fifty |
| Fifty | fifty | fifty | Fifty | fifty |
| Fifty | fifty | fifty | fifty | fifty |
| five | five | Five | Five | five |
| five | Five | five | five | Five |
| Five | Five | Five | five | Five |
| forty | forty | forty | forty | forty |
| forty | forty | forty | forty | forty |
| forty | forty | forty | forty | forty |
| forty | forty | forty | forty | forty |

| | | | | |
|----------|----------|----------|----------|----------|
| four | Four | Four | Four | Four |
| four | four | Four | four | four |
| Four | four | Four | four | four |
| fourteen | fourteen | fourteen | fourteen | fourteen |
| Fourteen | fourteen | fourteen | fourteen | fourteen |
| fourteen | fourteen | fourteen | fourteen | fourteen |
| hundred | hundred | hundred | hundred | Hundred |
| hundred | Hundred | hundred | Hundred | Hundred |
| Hundred | Hundred | hundred | hundred | hundred |
| nine | nine | Nine | nine | nine |
| nine | nine | nine | nine | Nine |
| nine | Nine | nine | nine | nine |
| nineteen | Nineteen | Nineteen | Nineteen | nineteen |
| Nineteen | Nineteen | nineteen | Nineteen | nineteen |
| nineteen | nineteen | nineteen | Nineteen | nineteen |

| | | | | |
|-----------|-----------|-----------|-----------|-----------|
| Ninety | ninety | ninety | Ninety | Ninety |
| Ninety | Ninety | ninety | Ninety | Ninety |
| ninety | ninety | Ninety | ninety | Ninety |
| One | One | One | one | One |
| one | one | One | One | one |
| one | one | one | one | one |
| only | only | Only | only | Only |
| only | only | only | only | only |
| only | only | only | seven | seven |
| seven | SEVEN | seven | seven | seven |
| seven | SEVEN | seven | SEVEN | SEVEN |
| seven | seven | SEVEN | seven | seventeen |
| Seventeen | seventeen | seventeen | seventeen | seventeen |
| seventeen | seventeen | seventeen | seventeen | seventeen |
| Seventeen | Seventy | Seventy | seventy | seventy |

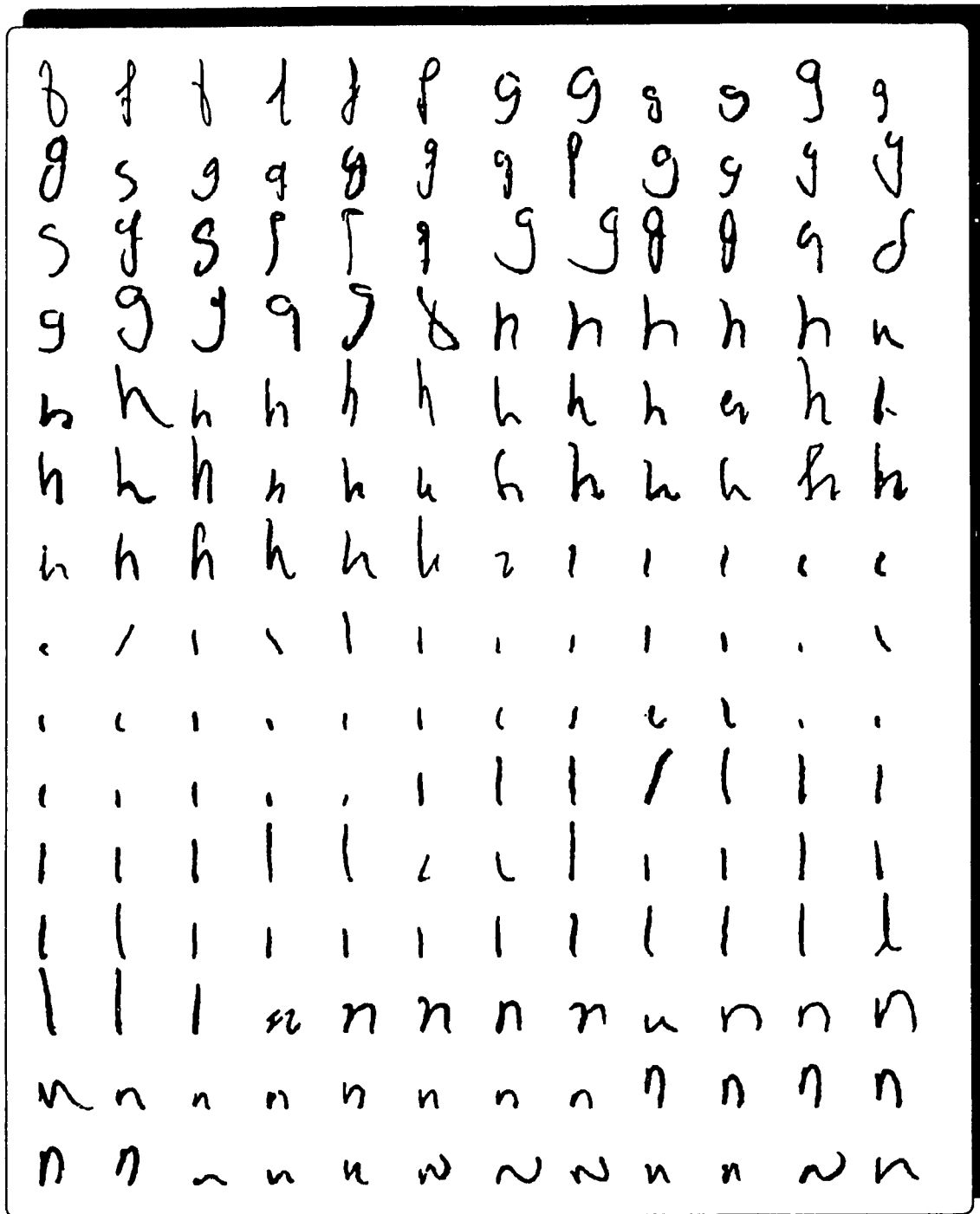
| | | | | |
|---------|----------|----------|----------|----------|
| seventy | Seventy | seventy | seventy | seventy |
| seventy | seventy | seventy | seventy | Seventy |
| Seventy | Six | Six | Six | ix |
| Six | Six | six | Six | six |
| six | six | Six | Six | Six |
| Six | sixteen | sixteen | sixteen | sexteen |
| sixteen | Sixteen | sixteen | sixteen | Sixteen |
| Sixteen | Sixteen | Sixteen | sixteen | Sixteen |
| sixteen | Sixty | sixty | sixty | Sixty |
| Sixty | Sixty | sixty | sixty | sixty |
| sixty | Sixty | Sixty | sixty | Sixty |
| sixty | ten | ten | Ten | ten |
| ten | ten | ten | Ten | Ten |
| ten | ten | ten | ten | ten |
| Ten | Thirteen | thirteen | Thirteen | thirteen |

| | | | | |
|----------|----------|----------|----------|----------|
| thirteen | thirteen | thirteen | thirteen | Thirteen |
| thirteen | thirteen | Thirteen | thirteen | thirteen |
| thirteen | Thirty | thirty | Thirty | Thirty |
| Thirty | thirty | thirty | thirty | thirty |
| thirty | thirty | thirty | thirty | thirty |
| Thirty | thousand | Thousand | thousand | thousand |
| thousand | thousand | thousand | thousand | thousand |
| thousand | thousand | Thousand | thousand | thousand |
| thousand | three | three | three | three |
| Three | three | Three | Three | three |
| three | three | three | three | Three |
| Three | Twelve | twelve | twelve | twelve |
| Twelve | twelve | Twelve | twelve | twelve |
| twelve | twelve | twelve | twelve | twelve |
| twelve | twenty | Twenty | twenty | twenty |
| twelve | twenty | Twenty | twenty | twenty |



A.3 Characters

A few characters extracted from our English testing database are shown in the following figures. These characters have also been slant corrected.



A.4 French legal amounts

A few samples of the legal amounts found in our French testing database are shown in the following pages.

| | | |
|---------------------------|----------|----|
| Douze | _____ | 21 |
| Cinquant sept | _____ 59 | |
| Cent-Quinze | _____ 39 | |
| Quinze Dollars | _____ 44 | |
| seize | _____ 03 | |
| six | _____ 31 | |
| Six cent seize | _____ | |
| Cent Douze | _____ 02 | |
| mill quatorze | _____ 40 | |
| trois | _____ 91 | |
| trois cent soixante seize | _____ | |
| cent Trente six | _____ 84 | |
| huit - huit | _____ 69 | |
| cent Quarante - huit | _____ 44 | |
| Cent dix douze | _____ 27 | |

Soixante-douze mille-sept-cent-onze — 46

dix mille quinze dollars — xx

— huit cent douze dollars — xx

———— Dix ————— xx

— six cent quatorze — 50

— Cinq mille quinze ————— .43

— Sept cent dix ————— 52

———— Treize ————— 58

— quatre-vingt-deux ————— 00

———— Treize ————— 24

Huit mille cinq cent vingt-quatre — 00

Quatre cent Treize ————— 77

— six mille huit cent douze — 87

———— Six cent quarante ————— 00

———— huit-cent quarante-deux — 64

Trois mille quatre-vingt-six ————— 49

Quarante cinq ————— 45

Six-vingt ————— 65

————— trente-huit ————— 00

————— cinquante et un ————— 73

neuf cent dix-huit ————— 28

Sept ————— 79

deux-cent-onze ————— 00

quatre cent cinquante trois — 00

Cinq cent quatorze ————— 61

-vingt-cinq seize ————— 24

Neuf Cent dix

six dollar ————— 83

Deux mille neuf cent quinze dollars ————— 51

Huit cent seize dollars ————— 23

- soixante neuf mille huit cents quinze - 57 |

- neuf cent cinq ————— .81

- sept cent dix ————— 88 |

————— quatorze ————— 29

————— douze ————— 84

Sept cent douze dollars ————— 44

Cinq cent quarante et une dollars *

Sept cent treize dollars ————— 35

Dix cent quatre vingt quatorze — 63

- treize dollars ————— 63

Huit mille trois cents quinze .97 |

Cinq cents trente trois

Deux

————— soixante quatorze — 53

————— trente-deux ————— XL

— sept cent sept —————

————— quarante ————— 62

Trois

Neuf cent quatre-vingts

— cinq cent quinze —————

————— Vingt-deux ————— 22

- quatre mille quarante cinq ————— 58

- Dix mille huit cent quinze ————— 54

— douze ————— 41

— Trois cent dix ————— 38

- Neuf mille sept cent vingt ————— 00

— Neuf mille-trois-cent-quarante-neuf ————— 44

————— Trois cent dix huit ————— 08

- quarante ————— x x

— trois-mille-cinq-cents-vingt-cinq ————— 73

trois ————— 16

Dix sept ————— 83

Cent Quarante Trois ————— 10

Soixante Trois ————— 26

sept cent dix-huit ————— 29

Trois Cent Soixante-cinq dollars |

Trente-huit dollars — ~~10~~ 99

Seize ————— 10

trois cent quatre vingt treize —

quarante quatre ————— 56

Deux mille Trente quatre ————— 52

soixante deux ————— 71

Neuf cent soixante Trois 82

neuf-mille-six-cent-dix-neuf —

Huit ————— 01

Onze _____ 10
 dix-neuf _____ 26
 huit-cent-onze _____
 sept-cent-quatorze _____ 88
 cinq-cent-onze
 dix-huit
 cinq-cent-cinquante _____ X
 Deux-Cents-Trente _____ 59
 Quatre-cent-dix _____ 36
 cent quatre-vingt-un _____ 34
 deux-cent-treize _____ 35
 dix-sept _____ 71
 huit-cent-seize _____ 88

Appendix B

Baseline Skew Correction

Several methods have been experimented in the literature in order to detect the baseline skew of a word. We will present in this appendix some of these methods as well as other approaches that we have tried.

B.1 Determination of the baseline skew

B.1.1 Principal axis decomposition

In order to determine the average skew of an image, we decided to experiment with the computation of its principal axes [GW87] which are the eigenvectors of the covariance matrix obtained by using the pixels within the image as random variables. The two eigenvectors of the covariance matrix point in the directions of maximal region spread, subject to the constraint that they be orthogonal. A measure of the degree of spread is given by the corresponding eigenvalues. Thus the principal spread direction of a region can be described by the largest eigenvalue and its corresponding eigenvector.

If we consider the coordinates of each pixel in the object to be two-dimensional random variables, then the mean is computed as follows:

$$m_x \cong \frac{1}{P} \sum_{i=1}^P x_i \quad (2)$$

and the covariance matrix:

$$C_x \cong \frac{1}{P} \left[\sum_{i=1}^P x_i x_i' \right] - m_x m_x' \quad (3)$$

where P is the number of pixels from the image that need to be rotated (e.g. foreground pixels), and x_i is the vector composed of the coordinates of the i^{th} pixel.

In order to find the eigenvalues and the corresponding eigenvectors for the covariance matrix, we need to solve the following two equations:

$$\det(\lambda I - C_x) = 0 \quad (4)$$

$$C_x X = \lambda X \quad (5)$$

where X is an eigenvector. Upon solving Eqn. (4), we obtain two values for λ . We sort these values so as to assign the greatest absolute value to λ_1 . Doing so, we will assign to the eigenvector e_1 , the principal axis for the image under consideration. We proceed with solving Eqn. 5. The result of these computations for a given input word is illustrated in Fig. 55. The algorithm for the computation of the eigenvalues and corresponding eigenvectors is as follows:

1. Compute the mean and covariance matrix as mentioned in Eqns. 2 and 3
2. Compute the eigenvalues using Eqn. 4.
Put the greatest absolute value of λ into λ_1 .
3. Compute the eigenvectors e_1, e_2 using Eqn. 5 corresponding respectively to the eigenvalues λ_1 and λ_2 .
4. From the coordinates of e_1 , output the value of the baseline skew corresponding to our coordinate system.

If we align the original coordinate system to the eigenvectors, we can normalize the baseline skew of input words so that it becomes aligned to some standard direction.

Experimentally, we notice some problems when using the eigenvectors to determine the baseline skew of an input image. Indeed since the eigenvectors are constrained to be orthogonal, they are not only sensitive to the baseline skew but also to the slant of the word (for a description of the word slant, the reader is referred to section 4.1.3). Therefore the baseline skew cannot be properly determined with

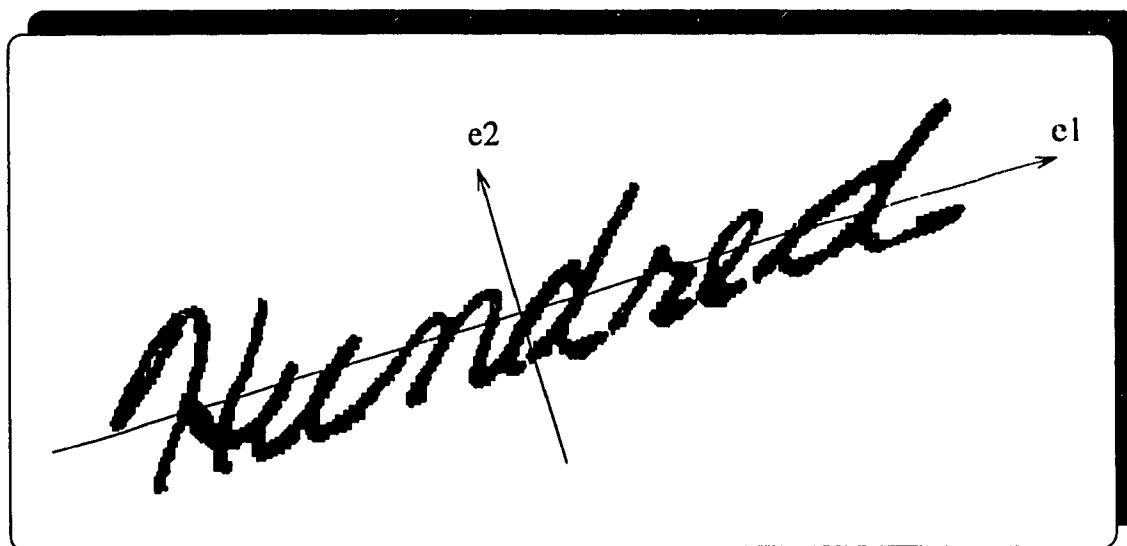


Figure 55: Eigenvectors e_1, e_2 : Principal axis decomposition

eigenvectors when the input word happens to be slanted. An example of this is shown in Fig. 56. Here the word has a true baseline skew of 20 degrees. The computing of the principal axes gives us a skew of only 16 degrees.

B.1.2 Least square method

One method commonly used for the detection of baseline skew is to fit a straight line through a set of data using a least mean square approach. The data can be a set of local minima corresponding more or less to the baseline of the text [BG83, CHH⁺90, CGM93] or the set of all foreground pixels from the input image [MG92].

The basic idea is to fit a reference line through a set of points so that the sum of the squares of the distances of each point to the line is minimized. The true distance of a point to a given line is the length of the normal vector joining the line to the point. For computation purposes, this distance can be approximated or evaluated more precisely. In the first case, the computations are easier. We will investigate both methods.

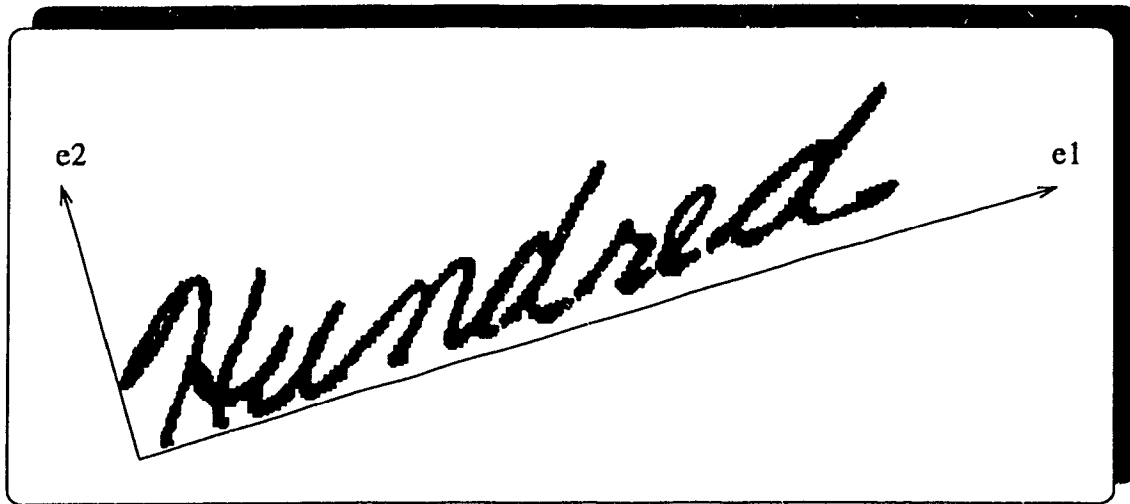


Figure 56: Eigenvectors: Not appropriate for baseline skew estimation

Approximate distances

Linear regression is used to fit a straight line to a set of pixels from the input image. The equation of the line is represented by:

$$y = ax + b \quad (6)$$

We compute the vertical distances between the line and the actual data points as shown in Fig. 57. In order to find the best line to fit our data, we wish to minimize the sum of these distances. This sum is called the error sum and is computed as follows:

$$\begin{aligned} E &= \sum_{k=1}^n d_k^2 \\ &= \sum_{k=1}^n (y_k - ax_k - b)^2 \end{aligned} \quad (7)$$

The minimum occurs when the derivative equals zero. Therefore we set to zero the partial derivatives of E with respect to the two unknown coefficients a and b . This gives us the two equations:

$$\frac{\partial E}{\partial a} = -2 \sum_{k=1}^n x_k (y_k - ax_k - b) = 0 \quad (8)$$

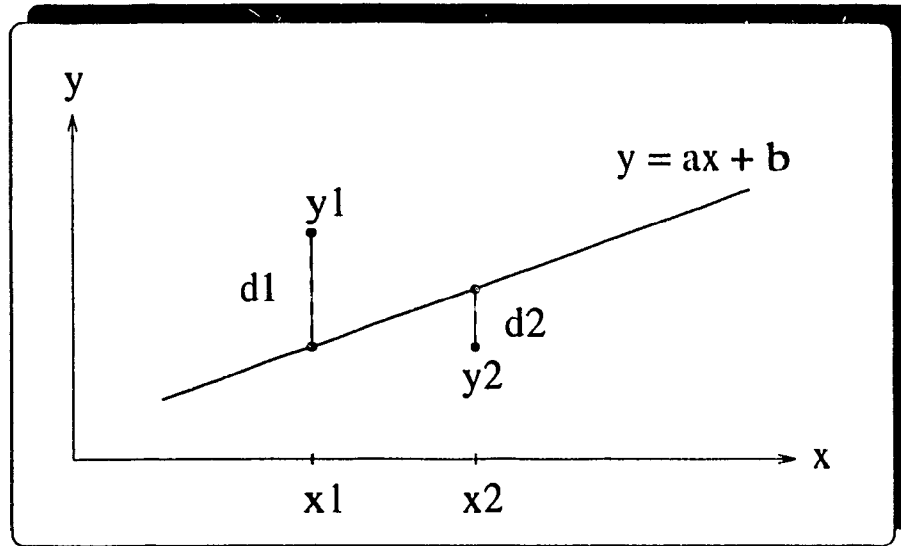


Figure 57: Linear regression

$$\frac{\partial E}{\partial b} = -2 \sum_{k=1}^n (y_k - ax_k - b) = 0 \quad (9)$$

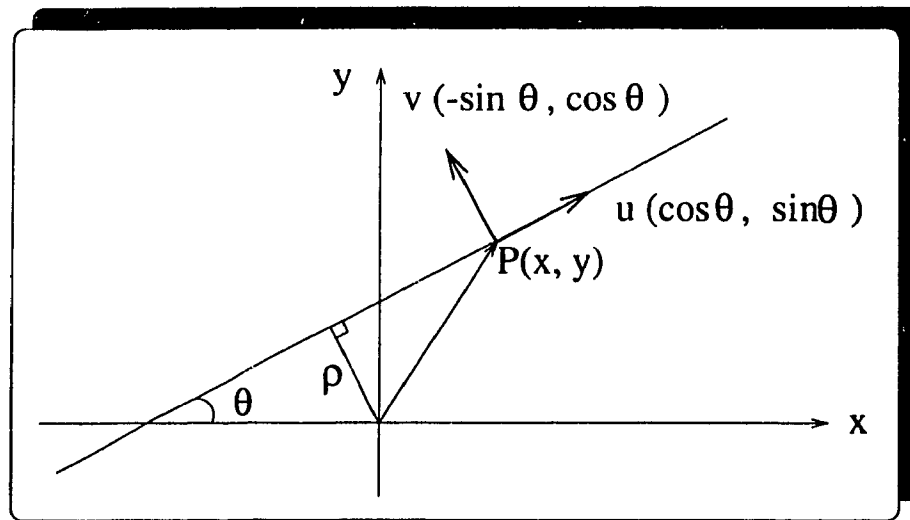
Solving these two equations for a and b gives

$$a = \frac{n \sum_{k=1}^n x_k y_k - \sum_{k=1}^n x_k \sum_{k=1}^n y_k}{n \sum_{k=1}^n x_k^2 - (\sum_{k=1}^n x_k)^2} \quad (10)$$

$$b = \frac{\sum_{k=1}^n x_k^2 \sum_{k=1}^n y_k - \sum_{k=1}^n x_k \sum_{k=1}^n x_k y_k}{n \sum_{k=1}^n x_k^2 - (\sum_{k=1}^n x_k)^2} \quad (11)$$

Once the slope a and the y -intercept b have been calculated, the equation of the straight line which best fits the data has been determined. The baseline skew can then be estimated as the angle between the best line fit and the y -axis and is expressed as $\theta = \arctan(a)$.

Experimentally, the linear regression method applied to the entire set of pixels from the input image and the eigenvectors method seem to give similar results for the approximation of the baseline skew. Better results could certainly be obtained if one were to find an effective way to separate the ascenders and descenders (section 4.3.1) from the main body of the word. However this involves further processing of the image that we do not see fit to discuss here.

Figure 58: Polar coordinate system: (θ, ρ)

Precise distances

The precise distance of a point to a line is understood here as the minimum distance of a point to a given line. This distance is obtained when we perform the orthogonal projection of the point on the line. The distance can easily be computed if we express the equation of the line with (θ, ρ) where θ corresponds to the angle of the line to the x axis and ρ is the distance of the line to the origin of the coordinate system (Fig. 58). The coordinates of every point on the line satisfy the following equation:

$$\vec{P} \cdot \vec{u} = \rho \quad (12)$$

As a result the equation of the line can be expressed as:

$$-x \sin \theta + y \cos \theta - \rho = 0 \quad (13)$$

This representation has the advantage of having well defined parameters for any types of lines. In the previous case where the equation of the line was represented by $y = ax + b$, problems could occur in determining the coefficients when the line is to become parallel to the y -axis. In this case, a becomes hard to evaluate since its value is heading towards the infinity.

The angular representation is also well suited for computing the distance of any given point to the line. Let us consider the first case of a point $P(x, y)$ which lies

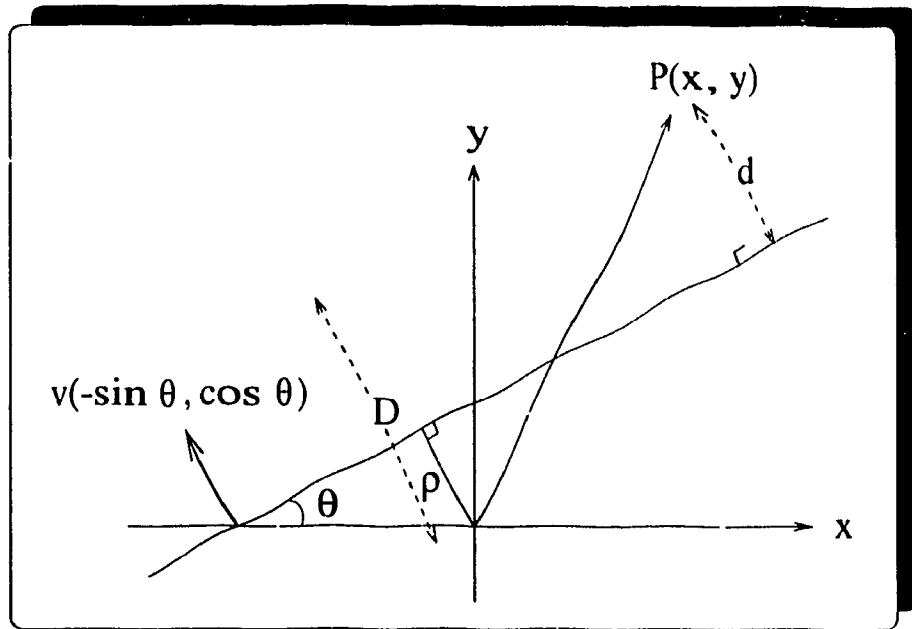


Figure 59: Polar coordinate system: positive distance of a point to a line

above the line as shown in Fig. 59. If we compute the scalar product of the two vectors \vec{P} and \vec{v} , we get the distance D shown in the figure. Since we are interested in the distance d from the point to the line, we need to subtract ρ from D :

$$\begin{aligned}\vec{P} \cdot \vec{v} &= D \\ \vec{P} \cdot \vec{v} - \rho &= D - \rho \\ \vec{P} \cdot \vec{v} - \rho &= d\end{aligned}$$

We note that d has a positive value.

If we consider the second case where the point is located below the line (Fig. 60), we get in a similar way the distance d' :

$$\begin{aligned}\vec{P} \cdot \vec{v} &= -D' \\ \vec{P} \cdot \vec{v} - \rho &= -D' - \rho = -(D' + \rho) \\ \vec{P} \cdot \vec{v} - \rho &= -d'\end{aligned}$$

So for any given point (x_i, y_i) , its distance to the line is computed as:

$$d_i = \vec{P}_i \cdot \vec{v} - \rho$$

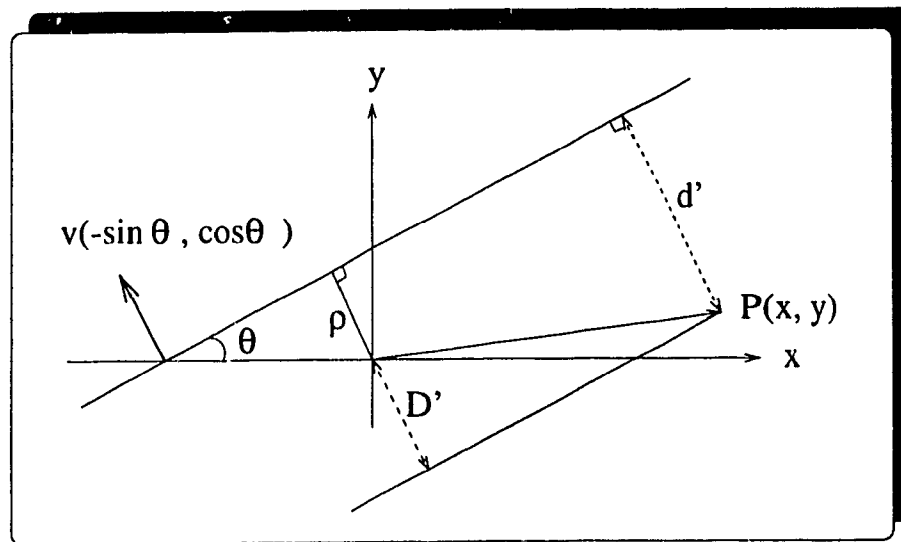


Figure 60: Polar coordinate system: negative distance of a point to a line

$$d_i = -x_i \sin \theta + y_i \cos \theta - \rho \quad (14)$$

with the value of d_i being:

$$d_i = \begin{cases} 0 & \text{if } P \in \text{line} \\ > 0 & \text{if } P \text{ is above the line} \\ < 0 & \text{if } P \text{ is below the line} \end{cases}$$

The notion of a point lying 'above' the line means that the point lies on the side of the half plane pointed to by the normal vector \vec{v} .

In order to get the least square fitting line, we have to find the values of θ and ρ that minimize the sum of the square of the distance of each point to the line. The sum of distances is expressed as follows:

$$E = \sum_{i=1}^N d_i^2 = \sum_{i=1}^N (-x_i \sin \theta + y_i \cos \theta - \rho)^2 \quad (15)$$

We want to find the values of θ and ρ that set the partial derivatives to zero. We have to solve the following two equations:

$$\begin{aligned} \frac{\partial E}{\partial \theta} &= 2 \sum_{i=1}^N (-x_i \cos \theta - y_i \sin \theta)(-x_i \sin \theta + y_i \cos \theta - \rho) = 0 \\ \frac{\partial E}{\partial \rho} &= 2 \sum_{i=1}^N (-1)(-x_i \sin \theta + y_i \cos \theta - \rho) = 0 \end{aligned} \quad (16)$$

From the second equation, we can express ρ as a function of θ . The set of equations (16) can then be simplified to:

$$A \sin \theta \cos \theta + 2B \sin^2 \theta - B = 0 \quad (17)$$

$$\rho = \frac{1}{N} \left(-\sin \theta \sum_{i=1}^N x_i + \cos \theta \sum_{i=1}^N y_i \right) \quad (18)$$

with the constants A and B defined as:

$$A = \sum_{i=1}^N (x_i^2 - y_i^2) - \frac{1}{N} \left(\sum_{i=1}^N x_i \right)^2 + \frac{1}{N} \left(\sum_{i=1}^N y_i \right)^2$$

$$B = \sum_{i=1}^N x_i y_i - \frac{1}{N} \sum_{i=1}^N x_i \sum_{i=1}^N y_i$$

In order to solve (17), we perform a substitution of variable:

$$\alpha = \sin \theta \Rightarrow \cos \theta = \sqrt{1 - \alpha^2} \quad (19)$$

Equation (17) becomes:

$$A\alpha\sqrt{1 - \alpha^2} + B(2\alpha^2 - 1) = 0 \quad (20)$$

By putting one term on each side of the equality sign and raising both terms to the power of two, we get the following:

$$C\alpha^4 - C\alpha^2 + B^2 = 0 \quad (21)$$

with the constant C defined as:

$$C = A^2 + 4B^2$$

We perform one more substitution of variables as follows:

$$\gamma = \alpha^2 \quad (22)$$

Equation (21) becomes:

$$C\gamma^2 - C\gamma + B^2 = 0 \quad (23)$$

The solution to the above equation is:

$$\gamma = \frac{1}{2} \left(1 \pm \frac{|A|}{\sqrt{C}} \right) \quad (24)$$

Since $C \geq A^2$, $\frac{|A|}{\sqrt{C}} \leq 1$, which means that both solutions for γ are positive. At this point we recall that $\gamma = \alpha^2$ and $\alpha = \sin \theta$. Since we will compute baseline skews approximately in the range of $[-20, +20]$, we are interested in the value of γ that corresponds to a small angle; e.g. in the range $[-20, +20]$. Therefore we choose the smaller of γ :

$$\gamma = \frac{1}{2} \left(1 - \frac{|A|}{\sqrt{C}} \right) \quad (25)$$

We know that α is equal to $\pm\sqrt{\gamma}$ and should satisfy the equation (20). This gives us a single solution for α . Lastly we get $\theta = \arcsin(\alpha)$:

$$\theta = \arcsin \left(\pm \sqrt{\frac{1}{2} \left(1 - \frac{|A|}{\sqrt{C}} \right)} \right) \quad (26)$$

Using moments, Jain [Jai89] gives a simpler formula for the computation of θ . For a shape represented by a region \mathcal{R} containing N pixels, the centre of mass is expressed as:

$$\bar{m} = \frac{1}{N} \sum_{(m,n) \in \mathcal{R}} m, \quad \bar{n} = \frac{1}{N} \sum_{(m,n) \in \mathcal{R}} n \quad (27)$$

The (p, q) order central moments become:

$$\mu_{p,q} = \sum_{(m,n) \in \mathcal{R}} (m - \bar{m})^p (n - \bar{n})^q \quad (28)$$

and the orientation, representing the angle of axis of the least moment of inertia, is given as:

$$\theta = \frac{1}{2} \arctan \left(\frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}} \right) \quad (29)$$

Experimentally, we did not notice any significant improvement when using the *precise* as opposed to the *approximate* distances of a point to a line.

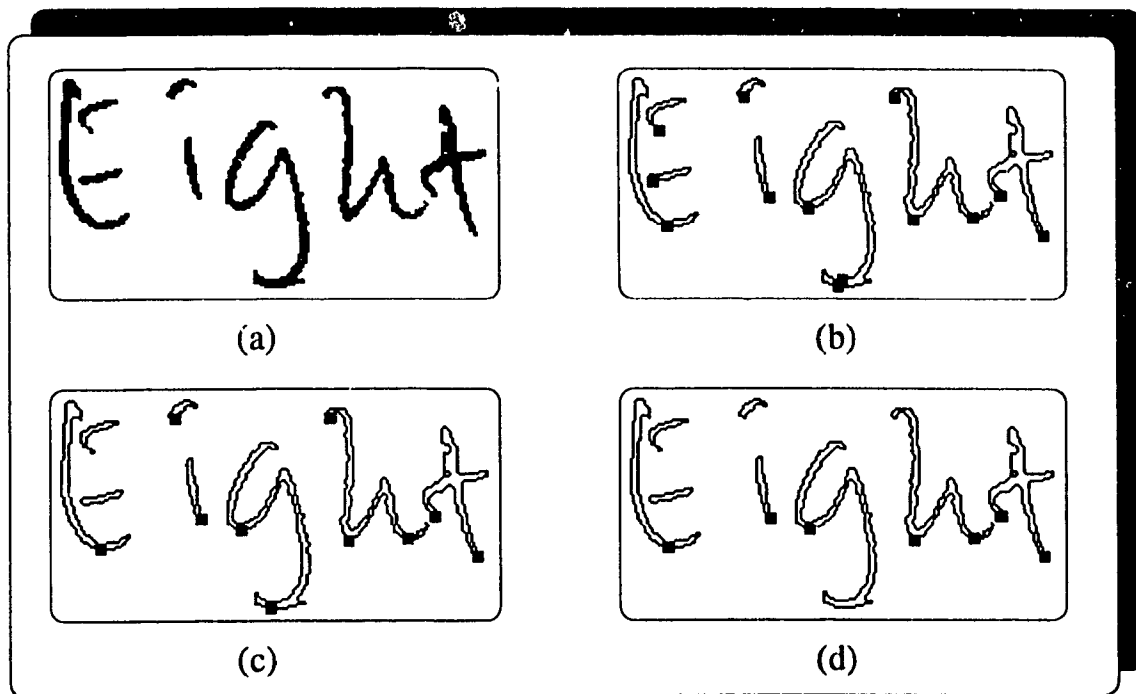


Figure 61: Local minimas: (a) original image (b) local minimas (c) pruned local minimas (d) outliers removed

B.1.3 Local minima

The above described least square fitting method is usually applied to a small set of points as opposed to all the pixels from an input image. Generally the local minima are chosen as the restricted set. An example of an input image and its associated local minimas is shown in Fig. 61 a,b. We recall that in our system an input image is represented as a linked list of connected components. For each of these components we compute the local minimas of the lower part of the contour. The resulting set of points (Fig. 61 b) need to be pruned to remove those points lying on retrograde strokes as well as those for which the component they belong to lie on top of another connected component (Fig. 61 c). In [CGM93], they refer to this pruning procedure as projecting the contour of all top level connected components to the lower margin. In this way, they determine the lowest contour parts.

In order to determine the baseline of the word, one still needs to remove those points that do not lie on the baseline, as for example the local minima of a descender.

Indeed orientation problems do occur when the local minima selected do not all lie on the true baseline of the word. This pruning is usually accomplished by fitting a line through the set of minima by the least square fit approach and removing the outlying points (Fig. 61 d). While this seems to work in some cases, it might have undesired effects on other images. In Fig. 62 c, we notice that the least square fitting line is positioned away from the points of interest so as to minimize the square distance to the outliers. As a result, when removing the points that are farther than one standard deviation (Fig. 62 d), we notice that we lost one of the desired minimas as well as kept one of the undesirable minima. For that reason, Press ([Pre92] p. 700) refers to the least square fitting as a non-robust technique, due to its undesired sensitivity to outlying points. Therefore a more robust approach should be investigated for the removal of outliers from the set of local minima. Such an approach can be minimizing the sum of the absolute distances

$$\begin{aligned} E &= \sum_{k=1}^n |d_k| \\ &= \sum_{k=1}^n |y_k - ax_k - b| \end{aligned} \quad (30)$$

as opposed to the square distances as in Eqn. 7.

Caesar & al. [CGM93, CGM95] use the least square measure but assign to each minimum a weight reflecting the curvature of the stroke where the point lies. A peaked minimum has a lower weight than a flat minimum. An iterative linear regression analysis is then applied to the set of minima. In each iteration, the furthest point is removed from the set. The procedure is repeated until the remaining least square distance is less than some acceptance threshold. This iterative weighted process seems to produce a reliable system for estimating the position of baselines.

B.2 Rotation of the original image

In order to normalize the baseline skew of our data, we need to rotate the images so that their baseline axis is aligned with the axis of the coordinate system. The processing involved is represented in Fig. 63. We note that the rotation is performed around

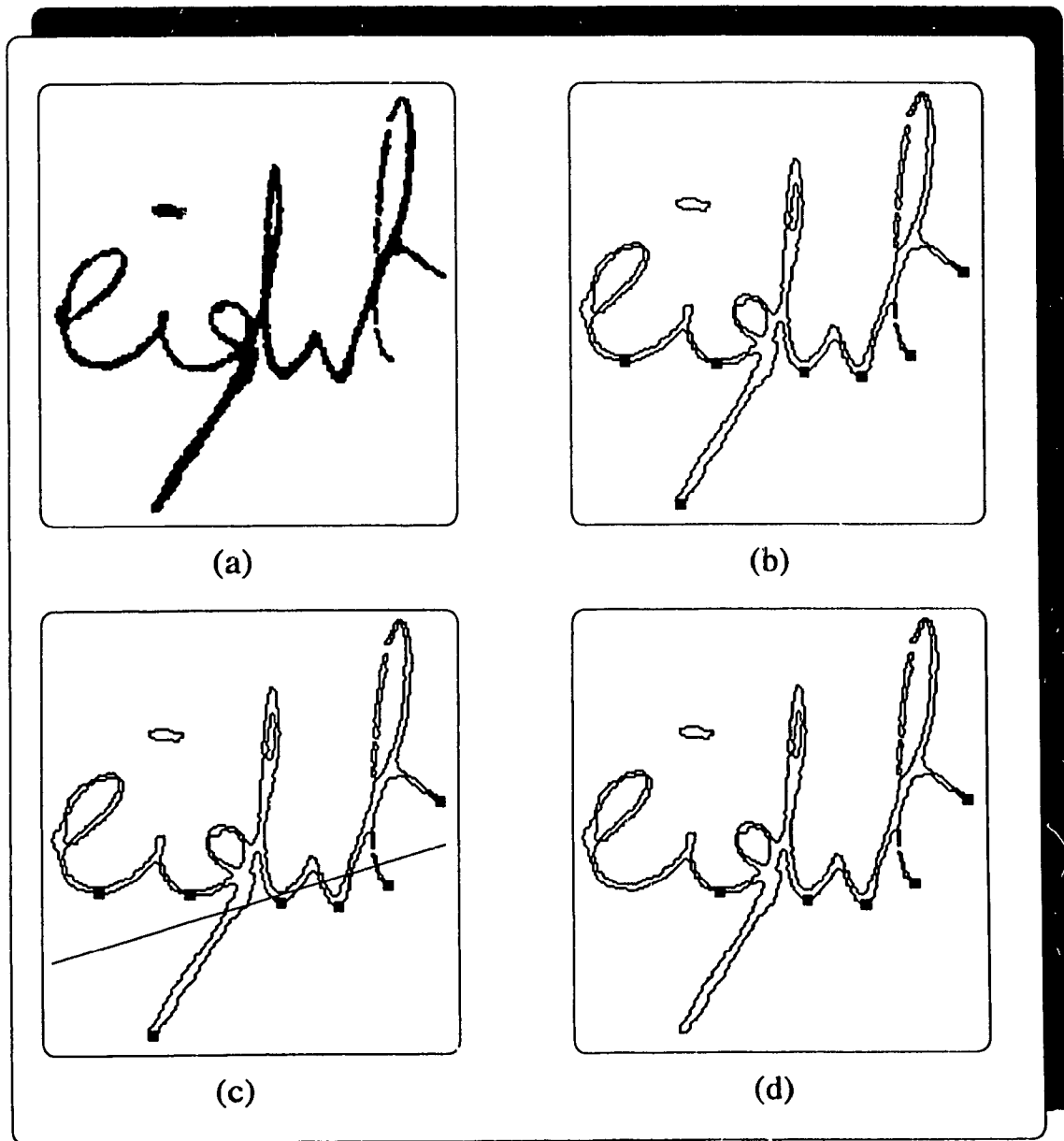


Figure 62: Local minimas: (a) original image (b) pruned set of local minimas (c) least square fitting line (d) resulting set of minimas

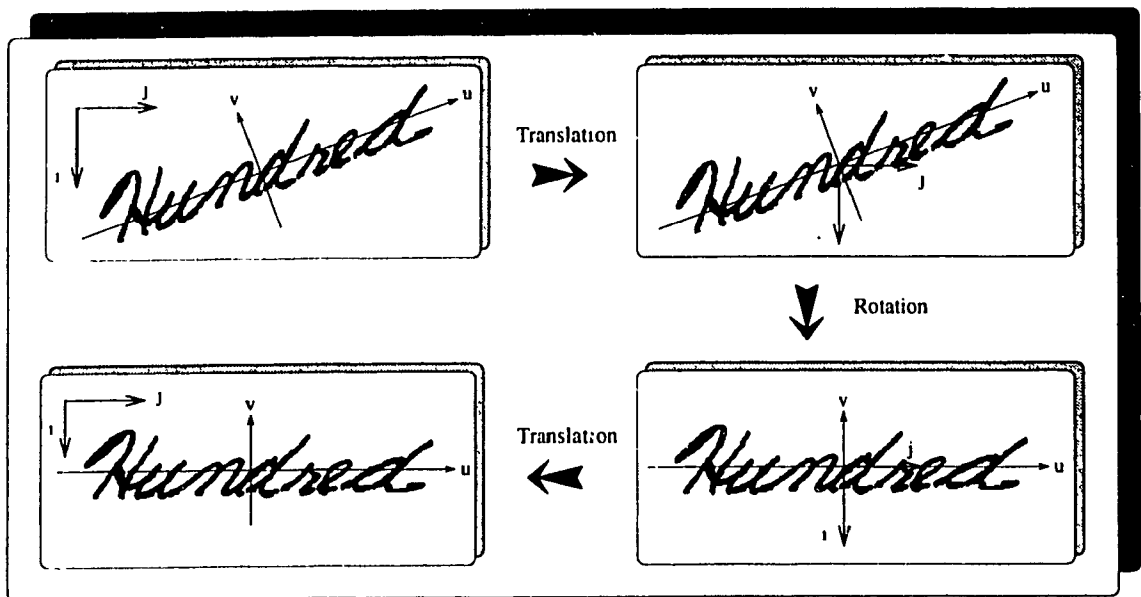


Figure 63: Baseline skew normalization

the mean of the image. The coordinates of this point are computed following Eqn. 2. The three operations of translation, rotation and final translation are combined into a single transformation matrix. The rotation algorithm we are using is as follows:

For every pixel of the final rotated image:

1. Compute the coordinates (i, j) of the corresponding pixel in the original image.

At this point, we have two options *A* or *B*:

- 2.A If the original pixel value is non-zero, set the rotated pixel.
- 2.B Compute the number of pixels among the neighbours of (i, j) which have non-zero values.

If the number of non-zeros is greater than a given threshold, then set the rotated pixel.

We may want to choose option *B* if the angle of the rotation is great (e.g. say greater than 50 degrees) and the image resolution is low. Option *B* does compensate for the distortion encountered in greater angle rotation. For our typical application, option *A* followed by a smoothing operation is sufficient.

Intuitively, it may seem more logical and faster to perform the rotation in a different way. That is to start from every non-zero pixel of the original image, compute the new coordinates in the rotated image, and set the corresponding pixel. Experimentally though, such an approach leads to output corrupted by a lot of white pixel noise. The results have a better graphic quality when rotation is performed as specified in the above described algorithm.

B.3 Summary

We described several techniques for the computation of the baseline skew of a given input image. None of techniques described seem to clearly outperform any of the others. More tests on actual skewed data would be needed to design and evaluate a truly reliable algorithm.

Appendix C

Slant Correction Algorithm

This appendix describes the computation of the vertical slanted histograms used in the slant correction algorithm as well as the shear transformation of an image by a given angle.

C.1 Computation of the slanted histograms

We compute histograms at various angles from the vertical axis by steps of '*delta_teta*' degrees. In a single scan of the image, all of the slanted histograms are computed. The algorithm used is as follows:

For each pixel of the image (*i, j*):

For each of the slant values (*k * delta_teta*):

1. Compute the new value *v* of the ordinate *j* in the slanted histogram.

Its value is computed as follows:

$$v = j - (\text{height} - i) * \tan(k * \text{delta_teta})$$

where *height* represents the height of the image.

2. Increment the count in the *v*th column of the *k*th histogram:

$$\text{histogram}_k[v] = \text{histogram}_k[v] + 1$$

The algorithm described here assumes that the origin of the image is at the top left corner with the *i* and *j* coordinates increasing downwards and to the left respectively.

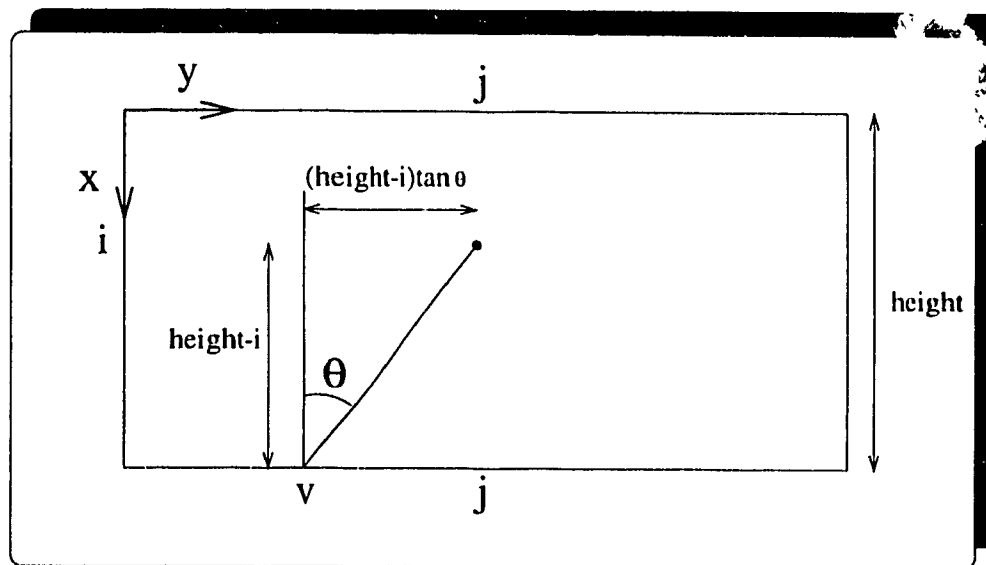


Figure 64: Computation of the slanted histograms

Fig. 64 gives a graphical representation of the computation of the new ordinate v from j . The formula is valid both for positive and negative angles; that is both for right- and left-slanted words.

In our application, we did fix the value of delta_theta to 5 degrees. It can be set to a lower value if processing time is not a matter. For both left and right directions, the number of histograms computed is ' nb_histograms ' (set to 15). As a result, we compute a total number of 29 histograms for angles varying from -70° to $+70^\circ$, by steps of 5° .

C.2 Shear transformation

Once the average slant of a word has been found, we wish to correct the image in order to obtain a 'slant-free' image. In order to do so, each pixel of the original image is transformed into a 'new' pixel with a formula similar to the one used for the computation of the slanted histograms. For each pixel (i, j) in the original image, we compute the new coordinates (u, v) of this pixel in the 'slant-free' image as follows:

$$u = i$$
$$v = j - (\text{height} - i) * \tan(\theta)$$

where θ is the average slant of the input word. Fig. 65 gives a graphical representation of the computation of the new coordinates (u, v) from (i, j) . Point A is transformed into point B . We note that points belonging to the bottom row of the image, as we would expect, are not modified by the transformation. It is therefore important to apply the shear transformation to images that have been previously skew corrected. It is also necessary to eliminate the possible blank rows (and columns) that are surrounding the image. This latest task is performed by an 'auto_crop' module. These operations are performed prior to the slant correction of the image.

We note that the general shape of an image with a small slant will not be changed significantly by the slant correction operation. On the other hand, the quality of the resulting image will be lessened for greater slant values. Indeed it is to be noted that while the y -coordinate is corrected, the x -coordinate remains the same. In other words, the height of the word is not changed while the width of the image will probably change. As a result, the greater the value of the slant, the greater the variation of the aspect ratio of the image. Experimentally, the observed degradation in the image quality is not significant for our processing. The worst case (most slanted handwriting) in our database is a sample of small handwriting (small height) with a slant of 60 degrees to the vertical. The image and its slant corrected version is shown in Fig. 66. One can notice that the elongated length of the letter 'h' for example is much shorter in the slant corrected version.

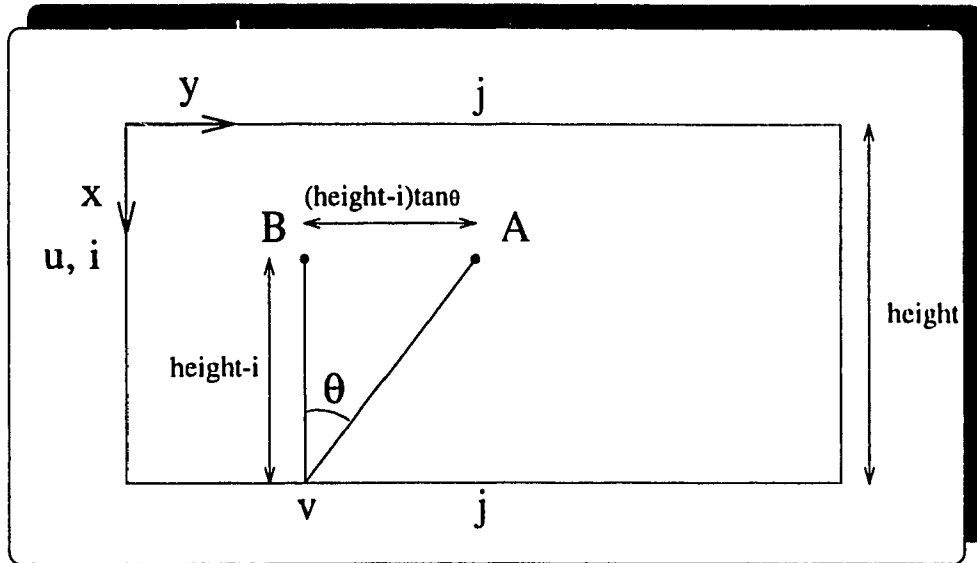


Figure 65: Shear transformation by a given angle θ

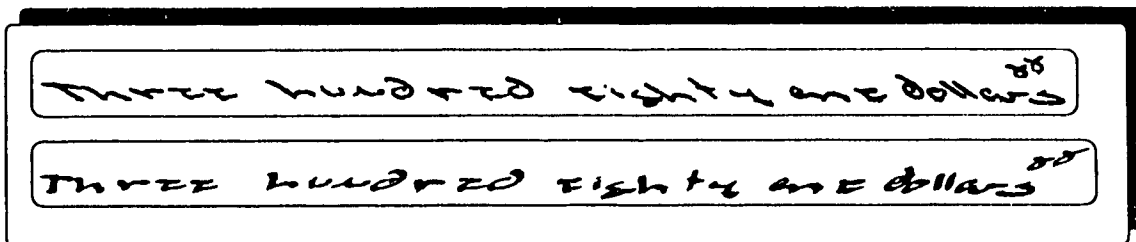


Figure 66: Shear transformation of an image initially slanted at 60 degrees

Appendix D

Mathematical Morphology

Mathematical morphology provides tools based on shape characteristics for the processing of digital images. A good introduction to the 2 basic operations of mathematical morphology (erosion and dilation) is presented in the paper by Haralick & al. [HSZ87]. The interested reader, though, should not overlook the book by Serra [Ser82]. Of special interest is the *hit-miss transform* (HMT) defined by Serra. Usual erosion and dilation is only a hit transform. We are restricted to “hits” and “don’t-care” in the structuring element. This severely limits the generality. The HMT, on the other hand, is an extremely general pattern matching operation. IIMT generalizes the function by looking for match simultaneously in the foreground and the background pixels. For some applications of the HMT, the interested reader is referred to [BM90]. In this latter paper, Bloomberg demonstrates among other things how to detect lines of only a given thickness.

I shall present in this appendix the 2 fundamental operations of MM (erosion and dilation), as well as the derived operations of opening and closing.

D.1 Dilation and erosion

Mathematical morphology is based on set theory. The set of all the black pixels in a black and white image constitutes a complete description of the binary image. Sets

in Euclidean 2-space denote foreground region in binary images. Sets in Euclidean 3-space may denote time varying binary images, or static graylevel images. Additional dimension space may denote additional image information like color.

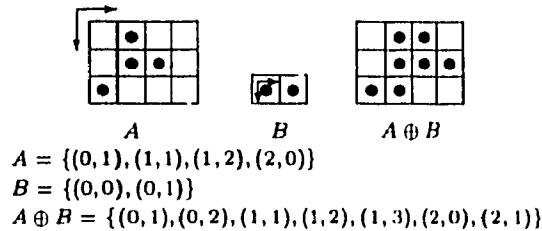
D.1.1 Dilation

Dilation combines two sets using vector addition of set elements.

Definition 1 Let A and B be subsets of E^N . The dilation of A by B is denoted by $A \oplus B$ and is defined by

$$\begin{aligned} A \oplus B &= \{c \in E^N \mid c = a + b, a \in A \text{ and } b \in B\} \\ &= \{c \in E^N \mid \exists a \in A, \exists b \in B, c = a + b\} \end{aligned}$$

Example:



The dilation is commutative and associative because addition is commutative and associative.

$$\begin{aligned} A \oplus B &= B \oplus A \\ A \oplus (B \oplus C) &= (A \oplus B) \oplus C \end{aligned}$$

In practice, the roles of the sets A and B are quite different. A is the image undergoing transformation and B is referred to as the structuring element.

The dilation operation can also be considered in terms of image translations:

Proposition 2 A a subset of E^N . $x \in E^N$.

The translation of A by x , denoted $(A)_x$ is defined as:

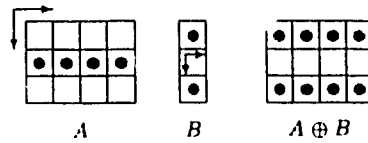
$$(A)_x = \{c \in E^N \mid c = a + x \text{ for some } a \in A\}$$

The dilation of A by B can be computed as the union of translations of A by the elements of B :

$$A \oplus B = \bigcup_{b \in B} (A)_b$$

When the origin belongs to the structuring element, the dilation is said to be extensive. This means that the dilated image contains the original. On the other hand, if the origin does not belong to the structuring element, then it may happen that the dilation of A by B has nothing in common with A .

Example:



D.1.2 Erosion

The erosion combines two sets using the vector subtraction of set elements. For A and B , sets in the Euclidean N -space, the erosion of A by B is defined as follows:

Definition 3 The erosion of A by B is denoted by $A \ominus B$ and is defined by:

$$\begin{aligned}
 A \ominus B &= \{x \in E^N \mid x + b \in A \text{ for every } b \in B\} \\
 &= \{x \in E^N \mid \forall b \in B, x + b \in A\} \\
 &= \{x \in E^N \mid \forall b \in B, \exists a \in A, x = a - b\}
 \end{aligned}$$

The erosion of an image A by a structuring element B can also be expressed in terms of intersection of image translation:

Proposition 4 The erosion of an image A by a structuring element B is the intersection of all translation of A by the points $-b$ ($b \in B$):

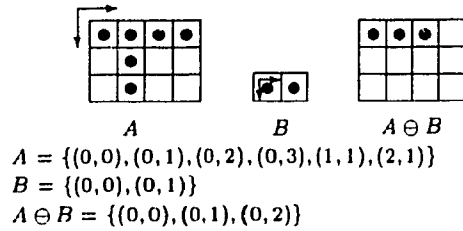
$$A \ominus B = \bigcap_{b \in B} (A)_{-b}$$

Still another way to express the erosion follows:

Proposition 5 *The erosion of an image A by a structuring element B is the set of all elements x of E^N for which B translated to x is contained in A .*

$$A \ominus B = \{x \in E^N \mid (B)_x \subseteq A\}$$

Example:



The erosion is non-commutative.

D.2 Opening and closing

In practice, the dilation and erosion operations are usually applied in pairs. An image is either first eroded then dilated or first dilated and then eroded. As a result, the image details smaller than the structuring elements have been suppressed.

The opening operation is defined as the combination of the erosion followed by the dilation. Closing is referred to as the compound of the dilation followed by the erosion. The definitions of these two 'new' operations follow:

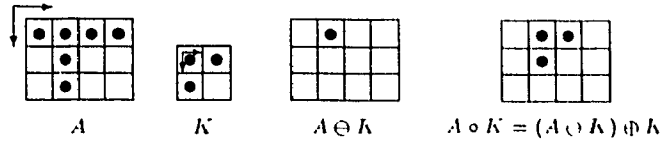
Definition 6 *The opening of an image A by a structuring element K , denoted by $A \circ K$ is defined as:*

$$A \circ K = (A \ominus K) \oplus K$$

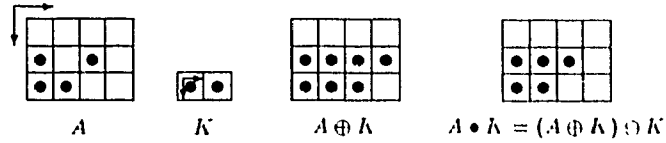
Definition 7 *The closing of an image A by a structuring element K , denoted by $A \bullet K$ is defined as:*

$$A \bullet K = (A \oplus K) \ominus K$$

Example:



Example:



D.3 Gray scale morphology

In the previous examples, mathematical morphology has been expressed in terms of set theory. A pixel in the transformed image was either selected or non-selected. Hence we presented the binary morphology. The operations described earlier can be extended to handle gray scale images by using a min or max operation. The gray-scale dilation can be computed according to the following proposition:

Proposition 8 *Let $f : F \rightarrow E$ and $k : K \rightarrow E$. Then $f \oplus k : F \oplus K \rightarrow E$ can be computed by:*

$$(f \oplus k)(x) = \max_{\substack{z \in K \\ x-z \in F}} \{f(x-z) + k(z)\}$$

In a similar way, we can express the gray-scale erosion:

Proposition 9 *Let $f : F \rightarrow E$ and $k : K \rightarrow E$. Then $f \ominus k : F \ominus K \rightarrow E$ can be computed by:*

$$(f \ominus k)(x) = \min_{\substack{z \in K \\ x+z \in F}} \{f(x+z) - k(z)\}$$

Gray scale opening and closing are defined in a similar way as with binary images.

Definition 10 *Let $f : F \rightarrow E$ and $k : K \rightarrow E$. The gray scale opening of f by the structuring element k , denoted $f \circ k$ is defined by:*

$$f \circ k = (f \ominus k) \oplus k$$

Definition 11 Let $f : F \rightarrow E$ and $k : K \rightarrow E$. The gray scale closing of f by the structuring element k , denoted $f \bullet k$ is defined by:

$$f \bullet k = (f \oplus k) \ominus k$$

D.4 Implementation

The basic algorithm for both the binary and gray scale dilation operation is as follows:

```

For each pixel X of the final image
{
  max=0
  for each non-zero pixel Z of the mask
    if X-Z is a non-zero pixel of the image
      then max = MAX( max, f(X-Z) + k(Z) )
  final image: value of pixel X: max
}

```

$f(X-Z)$: value of the pixel at coordinates $(X-Z)$ in the original image

$k(Z)$: value of the pixel at coordinates Z in the mask

We note that:

$$\begin{aligned}
 & (Z \text{ non-zero pixel of the mask } K) \wedge (X - Z \text{ non-zero pixel of the image } F) \\
 & \Rightarrow Z + (X - Z) = X \text{ non-zero pixel of the resulting dilated image}
 \end{aligned}$$

by definition of the dilation of 2 images.

Using a slightly different algorithm, we notice that the algorithm performs better in terms of time efficiency. Therefore, we used the following algorithm for the dilation:

```

For each non-zero pixel X of the original image
{
  max=0

```

```
for each non-zero pixel Z of the mask
  max = MAX( max, f(X) + k(Z) )
final image: value of pixel X+Z: max
}
```

f(X): value of the pixel at coordinates X in the original image

k(Z): value of the pixel at coordinates Z in the mask

Appendix E

Parameters

| Section | Parameters | Description | Value |
|---------|---|--------------------------------------|-------|
| 4.3.1 | t_{AM} | Ascender, descender detection | 0.5 |
| 4.3.1 | t_{DM} | Ascender, descender detection | 0.6 |
| 4.3.1 | t_{DA} | Ascender, descender detection | 0.6 |
| 4.3.1 | t_{str1} | Extraction of strokes | 2.5 |
| 4.3.1 | t_{str2} | Extraction of strokes | 0.5 |
| 4.3.1 | t_{str3} | Extraction of strokes | 0.5 |
| 4.3.4 | K | knn classifier | 3 |
| 4.3.4 | Nearest neighbour classifier weights adjusted by Genetic algorithms | | |
| 4.3.6 | t_{c1} | Extraction of character | 0.5 |
| 4.3.6 | t_{c2} | Extraction of character | 2.0 |
| 4.3.6 | t_{c3} | Extraction of character | 0.4 |
| 4.3.6 | f_{ecw} | Segmentation of character | 1.25 |
| 4.3.8 | crt | Character confidence | 1.5 |
| 4.3.8 | $CONF_THRESHOLD$ | Word confidence threshold | 0.35 |
| 4.3.8 | TOP_SOL | Integration word - character results | 5 |
| 4.3.8 | $TOP_PERMUTATION$ | Integration word - character results | 5 |

| AD scheme: Weights assigned to each feature sub-distance | | | |
|--|------------|---|-------|
| Section | Parameters | Description | Value |
| 4.3.4 | w_{dap} | weight for distance position of ascenders | 194 |
| | w_{dan} | weight for distance number of ascenders | 111 |
| | w_{ddp} | weight for distance positions of descenders | 240 |
| | w_{ddn} | weight for distance number of descenders | 83 |
| | w_{dlp} | weight for distance position of loops | 187 |
| | w_{dln} | weight for distance number of loops | 148 |
| | w_{dwl} | weight for distance word length | 96 |

| ADS scheme: Weights assigned to each feature sub-distance | | | |
|---|--|---|-------|
| Section | Parameters | Description | Value |
| 4.3.4 | w_{dap} | weight for distance position of ascenders | 248 |
| | w_{dan} | weight for distance number of ascenders | 92 |
| | w_{ddp} | weight for distance positions of descenders | 122 |
| | w_{ddn} | weight for distance number of descenders | 238 |
| | w_{dlp} | weight for distance position of loops | 219 |
| | w_{dln} | weight for distance number of loops | 108 |
| | w_{dwl} | weight for distance word length | 177 |
| | w_{dvs} | weight for distance vertical strokes | 156 |
| | w_{dhs} | weight for distance horizontal strokes | 177 |
| | w_{dses} | weight for distance south-east strokes | 123 |
| w_{dsws} | weight for distance south-west strokes | 95 | |