Diploma Thesis

Repair in ~~Off-Line~~ On-Line Handwriting Recognition

Wolfgang Hürst

Interactive Systems Laboratories
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213, U.S.A.

Supervisors:

Prof. Dr. Alex Waibel
and Dr. Jie Yang

March 1997

Hiermit erkläre ich, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Pittsburgh, 14.3.1997

Wolfgang Hürst

Acknowledgements

I would like to thank all the people who helped me and gave me support during this work. A special thanks goes to my two advisors Jie Yang and Alex Waibel but also to Bernhard Suhm, Stefan Manke, and Ralph Gross.

Also I would like to thank Jie Yang, Bernhard Suhm, Ralph Gross, and Matthias Denecke who had the time and patience to donate some data for me that I needed to evaluate my work.

Another thank you goes to the people who had numerous discussions with me (there are too many to remark everyone here) and who helped me a lot with good ideas and inspiring comments.

Abstract

Multimodal user interfaces offer great advantages in the area of human computer interaction. Typical examples are speech- and handwriting recognition, gaze tracking, gesture recognizers, etc. There are two important issues in the design process of such interfaces: recognition accuracy and user acceptance. Only if both of these two aspects are realized in a reasonable and satisfying way the interface will be useful for humans in different applications. One way to increase user satisfaction and acceptance is to integrate some repair and correction handling features in a common multimodal user interface. This report covers the topic of repair in automatic on-line handwriting recognition. On-line means that the recognition algorithms are not only applied on a handwritten bitmap but also on dynamic writing information. Based on the thesis that errors and repair appear and will always appear in automatic handwriting recognition some heuristics are proposed here to handle the arising problems. The proclaimed thesis will be proven by some empirical studies that also serve as a basis for a classification of errors and repair types that usually appear in human handwriting. Since errors always happen you have to deal with them somehow to fulfill the request of a user for a high usability and satisfaction. Algorithms and heuristics for both, repair in a handwritten input signal and in a printed text, i.e., corrections of a recognition result, will be introduced. Evaluations of the proposed heuristics will be done and some ideas for a better interface design in automatic handwriting recognition will be discussed.

Contents

Preface

Drei Viertel meiner ganzen literarischen Tätigkeit
ist überhaupt das Korrigieren und Feilen gewesen.
Theodor Fontane, dt. Dichter

Three quarters of my literary work
has actually been correcting and polishing.
Theodor Fontane, german writer

Multimodal user interfaces, like automatic handwriting recognizers, speech and gesture recognizers, etc., offer great advantages in the area of human-computer interaction ([WVDM95, VHY+95, WD94]). In contrast to the "usual" interaction devices, keyboard and mouse, for example, they can allow more flexibility for the users and be more "natural" to use by supporting typical human communication modalities (e.g., speech or handwriting). Interfaces easier to understand and easier to use can higher user satisfaction and increase the acceptance of the interface. Combined use of different modalities also provides the chance to improve recognition performances by gaining from different and opposite information sources, like, for example, acoustic speech recognition and lipreading (see [VW93]). Several kinds of interaction modalities, e.g., a pen or a speech based interface, limit the amount and size of needed hardware and therefore allow the construction of smaller, mobile computers, usable under conditions and situations where todays use of computers is uncomfortable or impossible.

In the design process of such new multimodal input devices there are two important aspects: first the recognition accuracy and second the user acceptance. Since communication between two partners only makes sense, if they can understand each other and put a meaning into the different communication skills, it is obvious that a high recognition accuracy should be the first and most important goal in the design of a new interface for human-computer interaction. Without the ability to recognize each others sayings, movings, writings, etc., communication does not work. Therefore most of the effort in research in this area so far has concentrated in achieving better recognition performances. But the better the recognition rates get the more important the other aspect becomes: user acceptance and satisfaction. Of course, an interface with low recognition accuracy would not achieve a high acceptance by a user, but even with high recognition rates, user satisfaction is not guaranteed. Other aspects are also very important, like an easy to learn and understand, or an easy to handle interface, or the ability to recover from input errors.

In [FHM95] an empirical study can be found that deals with the dependency of user satisfaction and recognition performance of pen interfaces. Another study, that was done in [Sch94], verifies the human recognition accuracy of cursive handwriting recognition. Since the recognition rate is below 100%, the author draws the

conclusion that the asymptote for automatic cursive script recognition accuracy is below the 100% goal.

But even if a "prefect" recognition could be possible, there still will be errors performed by the user, like misspellings or mistakes resulting from a wrong handling of the interface. Therefore even with a recognition accuracy of 100% you still would face the problem of misinterpretations.

For this reason a way must be found to deal with misunderstandings, misinterpretations, and recognition errors, to increase the user acceptance of a multimodal interface. Research in speech recognition has demonstrated, that even with an unreliable baseline spoken language interpretation technology, it is possible to significantly shorten the time to interact with a system via spoken language by developing strategies which effectively deal with the problem of interpretation errors (see [SMW96]). This is a good example, how additional techniques can help to overcome the problem of wrong recognition or misinterpretation and how to improve user acceptance and satisfaction even without 100% recognition accuracy. Therefore we should not wait for a "prefect" recognition system before we start to design new useful devices for human-computer interaction, but begin to concentrate on developing usable and useful interfaces, that can handle corrections and repair and are able to recover from errors.

The thesis, that a usable human-computer interface should offer some repair and error handling features, is attributed by the fact, that also the "usual" communication interfaces (keyboard and mouse) offer the possibility to do some corrections. So even if you can achieve 100% recognition accuracy, what keyboards and mouses usually do, the user demands some tools or ways to correct and repair his input[1].

Different input styles, like speech, handwriting, or movements of body parts (hand waving, head shaking, etc.), all have their typical applications. There are situations for every one of them where you can think of it as the most suitable input possibility under some special circumstances. Therefore no "best" input style for *every* application exists. This is one reason why today a lot of effort is put into a great variety of different multimodal human-computer interaction approaches. This report deals with automatic handwriting recognition and addresses the problem of repair and corrections in an on-line pen-based user interface. Automatic handwriting recognition can be defined as a computer program's analysis of digitized handwriting for the purpose of converting it into computer-readable ASCII text ([Bli97]). It is divided into Optical Character Recognition (OCR) and on-line handwriting recognition (see [WMO92], section II). OCR is performed after the writing is complete. Only an image of the handwriting, i.e., a bit pattern, is used as input signal for the recognition algorithms. On-line recognition, by contrast, recognizes each character as it is being written. Dynamic writing information is used in the recognition process. Therefore additional hardware, like a touchscreen or a graphic tablet, is needed in this case. The problem of repair in automatic handwriting recognition deals with corrections that were made by the user of a pen based interface himself in his input or in the result indicated after the recognition. These corrections might happen as a result of a recognition error, but also can occur implicit in the writing process, e.g., because the user wants to improve the shape of some parts of his handwriting or to correct a misspelling, independent from the recognition result. Some empirical studies about this will be shown in this report.

The handling of repair in automatic handwriting recognition is very important for different reasons. First of all, like in every interface for human-computer interaction, errors will always happen: on one hand errors in the recognition process,

[1] If you do not agree with this statement, try selling keyboards without a backspace key. I bet, you won't get rich!

on the other hand errors by the users. So you need something to deal with this situations, like some tools to perform different kinds of repair. But even if repair is not supported or the user is not asked to do any, it always happens in the pen input of human users. I will prove this later by some empirical studies. Therefore repair handling algorithms also can help to improve recognition performances. They can make an interface easier and more natural to use and offer some flexibility to the user, too. This can lead to a higher user acceptance, which is an important issue in the design of pen based input devices.

This report is divided in two parts. *Part I* addresses the problem of errors and repair in human handwriting. It takes care of the questions what kinds of errors usually happen when a human person uses a pen based input device and how users usually try to correct these errors. These studies are taken under special consideration of errors and repair in on-line based handwriting recognition. *Part II* contains the discussion and evaluation of some algorithms and heuristics that allow a user to perform repair in a handwriting recognizer. Heuristics for corrections in the handwritten input signal and in the recognition result, i.e., in printed ASCII text, will be introduced. Each part closes with a final discussion of the respective topics.

Part I

Error Repair in Handwriting Recognition - Classifications and Empirical Studies

Chapter 1

Introduction

Handwriting recognition provides an important channel for human-computer interaction. Handwriting is one typical form of human-to-human communication, used by many people every day. Therefore it is an obvious approach to use it in human-computer interaction, too. It offers great advantages in a wide area of applications, like, for example, form filling, notebook and calendar management, etc. The need for a keyboard and the need to learn how to type would be eliminated. This would open fast computer access to a lot of people who can write, but not type. Also smaller, mobile computers could be built with a pen based user interface.

different tasks and techniques

For this reason a lot of effort has been spend over the past years in designing reliable recognition algorithms for human handwriting input (see [MSY92] and [WMO92]). These algorithms can be classified and should be distinguished according to different tasks and different assumptions about the input devices, the input signal, the recognition algorithm, etc.

First different domains for handwriting are possible: letters in the handwritten text (e.g., [MFW96], [MFW95b]), gestures in text editing (e.g., [MCvMK95]), symbols like notes in music or mathematical expressions (e.g., [Win96]), etc. Also different languages could be recognized, like English or Chinese letters (e.g., [CKJ90]). The handwriting can be boxed, what means that every letter is in a special area of the input, spaced, which means that the letters are well separated, or the different characters can overlap, like it usually happens in cursive handwriting.

The most important classification related to the input signal is to distinguish between Optical Character Recognition (OCR) ([MSY92]) and On-Line ([WMO92]) Handwriting Recognition. In OCR only the bitmap of the pen input can be used to perform the recognition algorithm, that is, the handwritten input signal is only defined through a set values corresponding to the x- and y-coordinates. In On-Line Recognition on the other hand additional information about the time of downwriting of the different points of coordinates is available and can be used in the recognition process.

Another classification is, if the recognition algorithm works writer dependent, writer independent or writer adaptive (for example, see [SHTA93]), which means, that some parameters of the recognition procedure are set and adapted during or after the first few words are written by a new writer.

Different units can be modeled by a system, like strokes, characters or words. Segmentation can be done implicit in the recognition process or in a separate step prior to it. A variety of techniques, for example, neural net based approaches (e.g., [MFW96] and [MFW95b]), statistical methods, like Hidden Markov Models, (e.g., [BBNN, KA93, AKLP93] or [NBS+95]), pattern matching, and so on, can be done to perform the recognition.

Also it can be distinguished between character and word based recognition. Character based recognition tries to recognize every single letter that was written independently form the preceding and following letters. If the recognition is word based, the whole written word is recognized as a single unit. Therefore the result has to be from a fixed set, the dictionary. In character based recognition no fixed size exists for the recognizable vocabulary. Every word resulting from a random character order can be recognized. Also some mixtures exist, e.g., character based recognition in a first step and after that a second parse that tries to correct wrong recognized letters by using a fixed dictionary of different words.

An overview about different techniques and the state of the art in automatic handwriting recognition can be found in [ea95], Chapter 2.

the NPen^{++} recognizer

This work is limited to on-line handwriting recognition of single words with letters from the English alphabet. It uses the **NPen^{++} recognizer**, an on-line based handwriting recognition engine to recognize single words from a fixed dictionary, written writer independent in any writing style (printed, cursive, or a mixture of both). The NPen^{++}system was developed by Stefan Manke, at the University of Karlsruhe, Germany. An illustration of the modules of the whole system can be found in Figure 1.1.

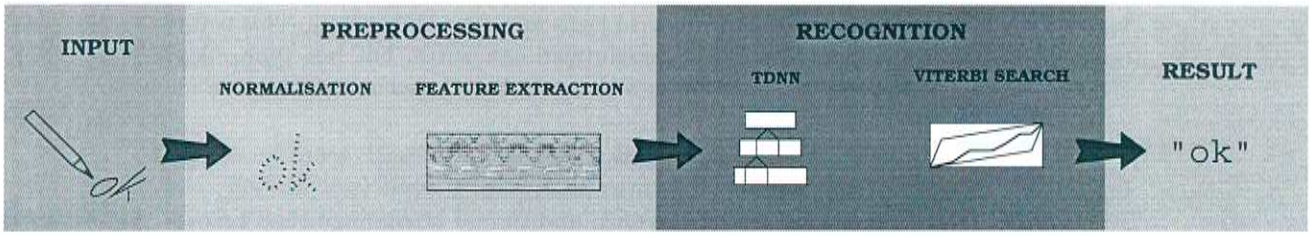

Figure 1.1: The NPen^{++}handwriting recognition system.

It is divided in two submodules. The first one performs a preprocessing on the input data. In the normalization step undesired variabilities from the original input signal are removed. After that a feature extraction is done along the normalized pen trajectory to get the features most usable for the second module: the recognition process. The output units of a Time-Delay Neural Network (TDNN, see [WHHS90]) model three states (begin, middle, and end) for each letter of the alphabet. Then a Viterbi search ([Rab90], pages 274 and 283) on each word of the recognition vocabulary is done and a score for each word from that dictionary is calculated. The word with the highest score is seen as the recognized result. Segmentation is done implicitly in the search, no explicit segmentation step is required. The single modules will be described in more detail in later chapters, if necessary. Otherwise it is referred to [BM93, BMW93, MB94, MFW94, MFW95a, MFW95b] or [MFW96].

contents part I

In the following chapters a discussion and several data studies will be done. *Chapter 2* addresses the question what kinds of errors usually happen in human handwriting. A classification in different error types and a data study according to this classification will be done. Also the special problems that occur, if on-line based algorithms are used for the recognition process, will be discussed. *Chapter 3* takes care of the question what kinds of repair actions are performed by human users to correct errors in their own handwritten input. A database will be examined according to what kind of repairs happen, what features are typical for the different corrections, etc. A classification of some classes for different repair types will be proposed, that is used in the second part for the development and discussions of repair handling heuristics.

Chapter 2

Errors in On-Line Handwriting Recognition

In nearly every case repair is the result of an error. An error in an on-line handwriting recognition system can be produced by both, the recognition algorithm and the user. Therefore a correction by the human can be caused by a wrong recognized word, but also, for example, by an error performed by the user himself, e.g., a misspelling. So if one is interested in handling different kinds of repair and in designing different repair tools and algorithms, it could help to have a look at what types of errors usually occur in automatic handwriting recognition, because these errors often result in a repair.

In nearly every case repair is the result of an error. An error in an on-line handwriting recognition system can be produced by both, the recognition algorithm and the user. Therefore a correction by the human can be caused, for example, by a wrong recognized word, but also by an error made by himself, e.g., the misspelling of a word. So if one is interested in handling different kinds of repair and in designing different repair tools and algorithms, it could help to have a look at what types of errors usually occur in automatic handwriting recognition, because these errors often result in a repair.

2.1 Typical Errors in Handwriting Recognition

There are some typical kinds of errors that usually or often appear in a pen based interface for human-computer interaction. A list with **common errors in automatic handwriting recognition** of textual input can be found in [Sch94]. These errors are

error classification

- *device-generated errors*; e.g., random noise, unresponsiveness of switches, etc.
- *badly spelled words*; e.g., one letter is missing
- *input legible by humans but not by the algorithm*; e.g., fused characters if the algorithm was designed to recognize only well separated letters
- *badly formed shapes*; e.g., the letter "n" is written in a way that it looks like a "u"
- *unknown words*; e.g., if the recognizable vocabulary is limited to some words from a specific dictionary and the user writes one that is not included in this vocabulary

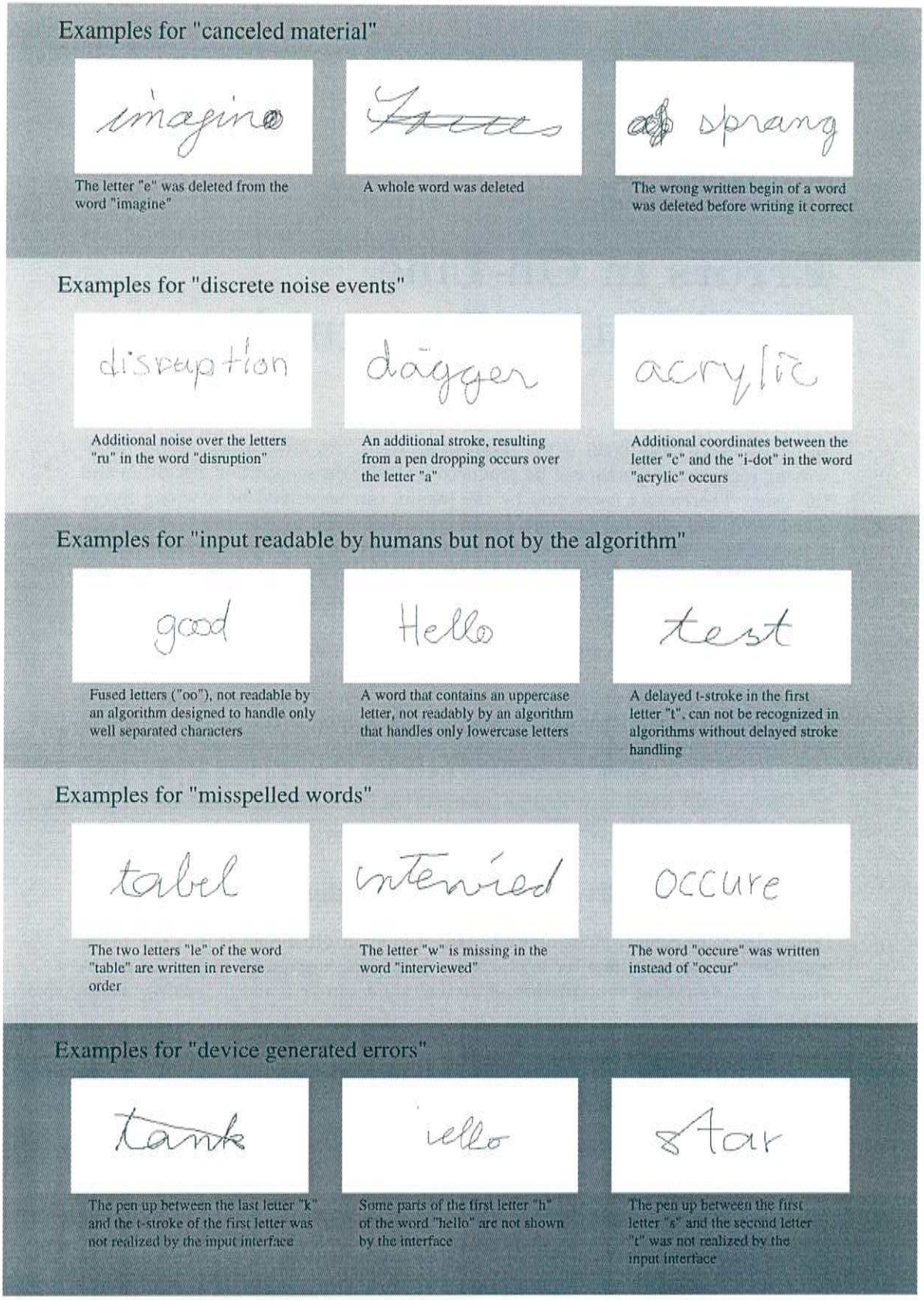

Figure 2.1: Examples for typical errors in human handwriting of textual input.

- *discrete noise events*; e.g., dots or lines caused by dropping the pen on the writing surface
- *canceled material*; e.g., a writer starts writing a word, than stops and starts scribbling over it in an unpredictable way

Some examples for this different error classes can be found in Figure 2.1. Note that this is not an exhaustive list and that other types of errors might exist.

database analysis

According to this classification an **analysis of a database** concerning the kinds of errors happening in human handwriting was done. This database consists of 3466 words and 3410 text segments of which each contains about eight words. The data has been collected on a graphic tablet connected to a computer. There was no feedback from any recognizer. The user was just asked to write down some words that were shown on the screen. He was not requested to do any kind of repair or corrections. After the data collection the words and the text segments were labeled "GOOD", if they were usable as training data for an on-line handwriting recognizer. Otherwise they were labeled with "CROSSOUT", "SPELLING" or "JUNK". "CROSSOUT" labeled data contains deletions or overwritings from the user, data labeled with "SPELLING" contains wrong spelled words or words with letters with bad shapes. The data that were labeled "JUNK" has been corrupted otherwise. It turned out that about 13 % of the words and about 23 % of the text segments were not usable for training and therefore labeled "JUNK", "SPELLING" or "CROSSOUT". See Figure 2.2 for the distribution of the labels in the database.

This "bad" labeled data was analyzed according to the list of errors in automatic handwriting recognition introduced above. Since point three (input legible by humans, but not by the algorithm) and five (unknown words) from the list of error types depend not only on the data but also on the realization of the recognition algorithm, they have been ignored in this study. The result can be found in Figure 2.3.

Note, that the values are only approximately, because the classification above is not straight and sometimes it is not possible to decide to which error class a handwritten word belongs. For example, an additional stroke can result from a badly formed shape of a letter by the user or from dropping the pen on the writing surface. In this case a correct classification would only be possible, if one participates and immediately evaluates the data collection process. A correct classification based on some previously collected data is impossible in such a situation. For this reason the analysis of the errors occurring in this database of human handwriting should not be seen too straight. But it gives a strong impression, that errors occur very often in human handwriting.

Also this classification in error types is not unambiguous. For the grouping of different error styles and types several possibilities can be thought of. **Another study about occurring errors** in human handwriting can be found in [Kas95], where some handwritten data was collected and analyzed, if it is usable for the training of an on-line based handwriting recognizer. The data consisted of letters and digits. Samples that were not usable to train the parameters of a recognizer have been analyzed and classified. The result can be found in Table 2.1.

You can see that about 5% of the digits and about 25% of the letters contain wrong, bad, or additional data, that makes them not usable for the training of a recognizer. Different kinds of errors happened, i.e., pen skips, case errors, and misspellings. They are a good motivation to offer the possibility to a writer to do some repair in a pen based user interface.

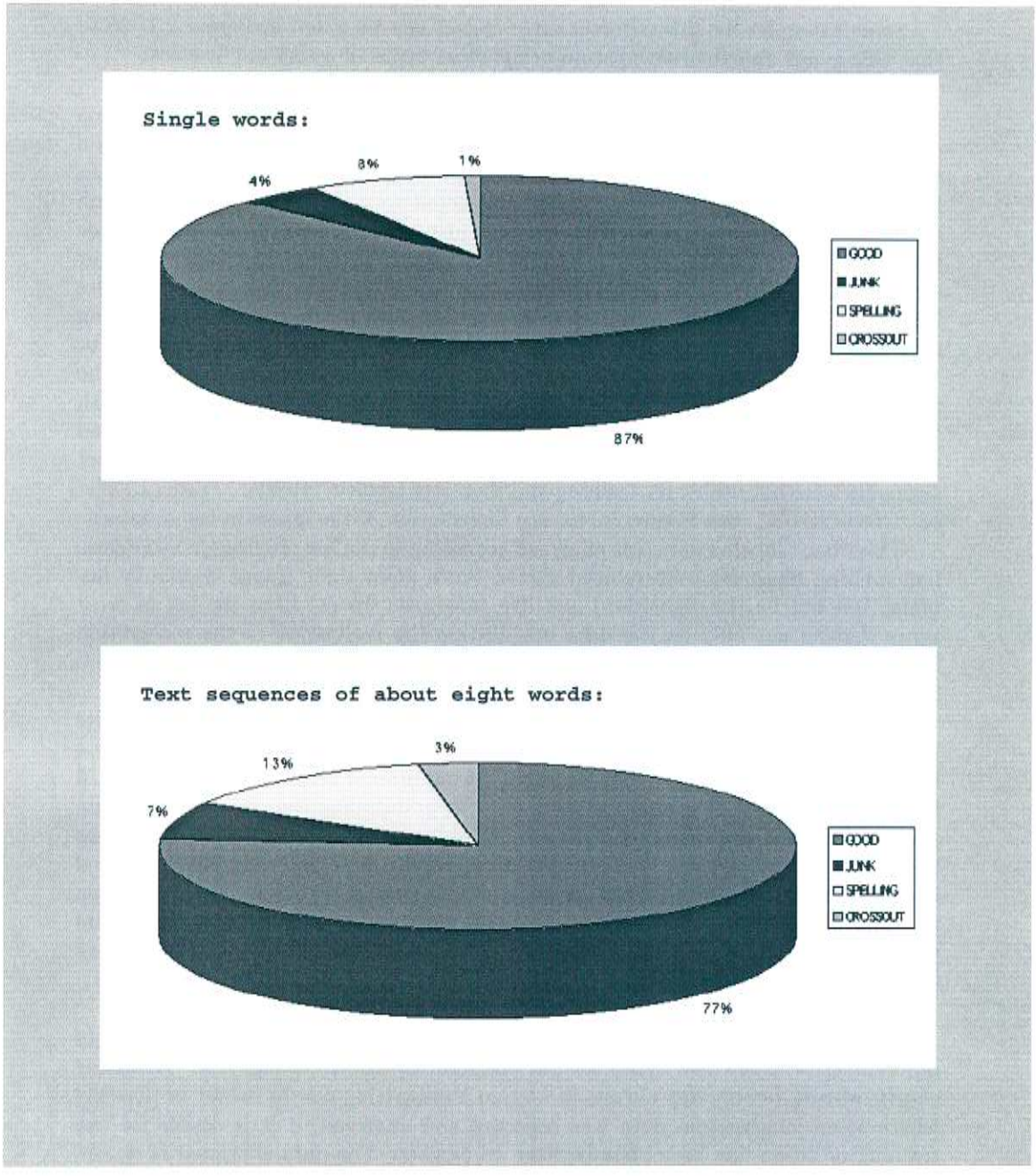

Figure 2.2: Quality of the data in the whole database.

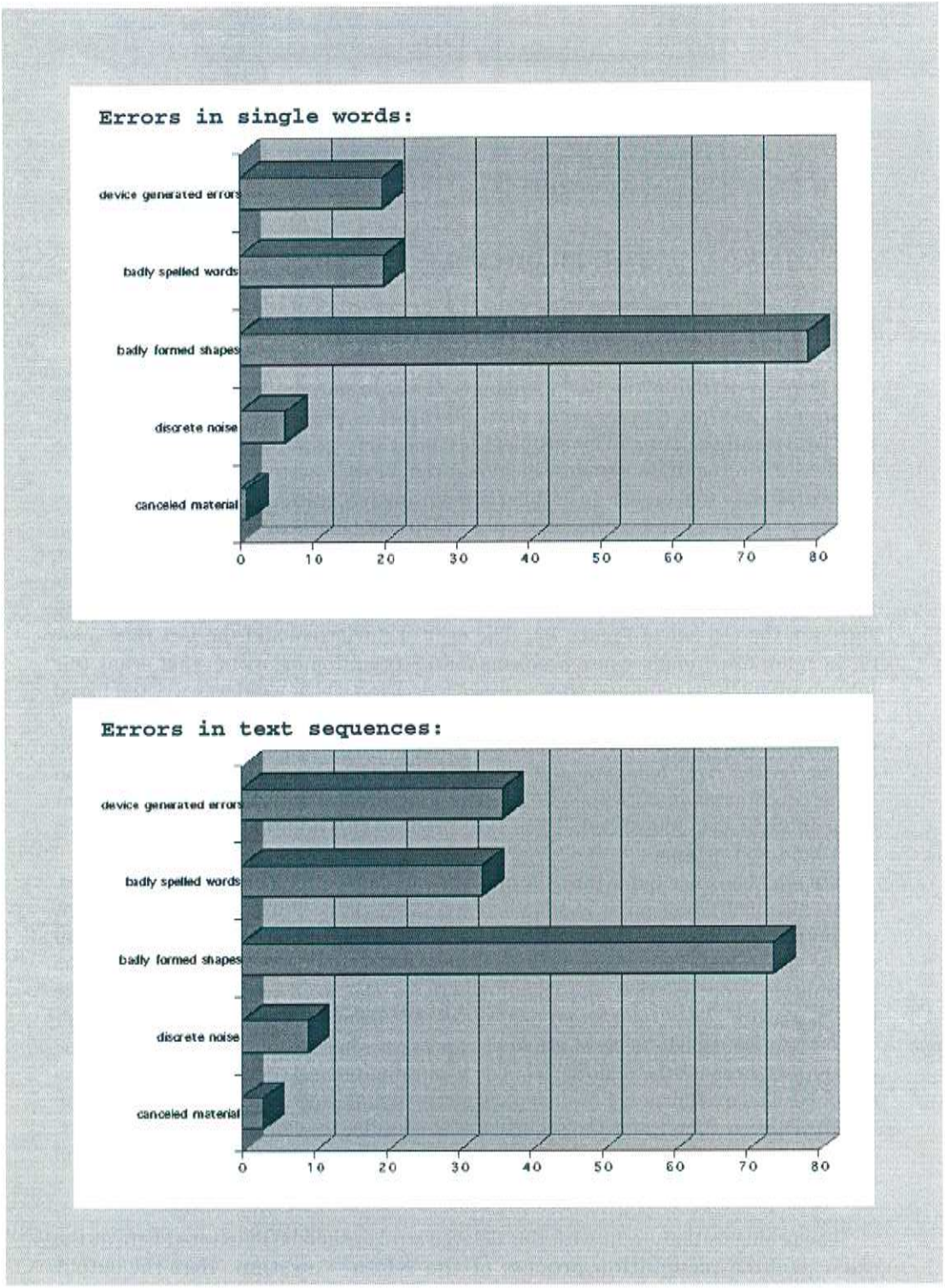

Figure 2.3: Errors in the words and text sequences from the database labeled "bad".

Transcription property	All symbols	Letters only	Digits only
Count	12402	8427	3975
Differs from prompt	3.7%	4.7%	1.4%
Contains ligatures	9.8%	13.9%	1.1%
Contains trash	2.2%	2.7%	1.1%
Contains pen skip	1.8%	2.1%	1.2%
Contains case error		1.2%	

Table 2.1: Data analysis from [Kas95].

2.2 The Delayed Stroke Problem

One error, reported in the list of common errors in automatic handwriting recognition from the last chapter, was, that the **input is readable by a human but not by the algorithm**. This problem becomes very important in a lot of on-line handwriting recognition systems, including the NPen^{++}system.

Like already told earlier, in on-line handwriting recognition not only the x- and y-coordinates of the handwriting exist, like in Optical Character Recognition (OCR), but also temporal information about the time of downwriting of these coordinates. This additional information can be used in the recognition process and usually leads to better results than with "pure" OCR. As an illustration see the handwriting shown in the top left of Figure 2.4. The drift of the letters and the fact that some of them are fused might cause problems for a recognition algorithm that scans this bitmap from left to right and tries to make a decision which word was written based on the letters of this word. But if you look at the top right side, where the same coordinates are shown in a 3-dimensional diagram with the additional information of time on the right axis, you will see, that with this information a segmentation of the coordinates in corresponding letters is well possible[1]. Some problems that occur in OCR can be avoided, if you take profit of this additional information in a right kind and manner.

But on the other hand this information can cause problems. There are cases, where also additional noise is added to the input data. An example is shown at the bottom of the same Figure. If you look at the "pure" bitmap, nothing special will be recognized. But if you look at the x,y,t-diagram, that shows also the time information, you see that the user has written the word and at the end made two "t-strokes", one "i-dot", and one stroke that completes the shape of the letter "x". These so called "delayed strokes" can cause some problems in many on-line recognizers, because the relation between x-coordinates and time t of downwriting is no longer linear. But many on-line recognizers assume such a relation. The broken match between time t and the sequence of x-coordinates can be seen even clearer, if you look only at this two values, i.e., set the y-value to zero. Such a x,t-diagram is shown in the lower part of Figure 2.4.

"delayed stroke problem"

The problems that occur with this delayed strokes shall be illustrated here at the example of the **recognition process of the NPen^{++} system**. Here the output units of a Time-Delay Neural Network are aligned to the models of each word in a dictionary by the Viterbi algorithm. The output units model each letter of the recognition alphabet through a sequence of three states for each letter. Every word

[1] Note that the time t here refers to the number of an (x,y) pair in the time ordered sequence of coordinates. It is not proportional to the time intervals between two coordinates writings.

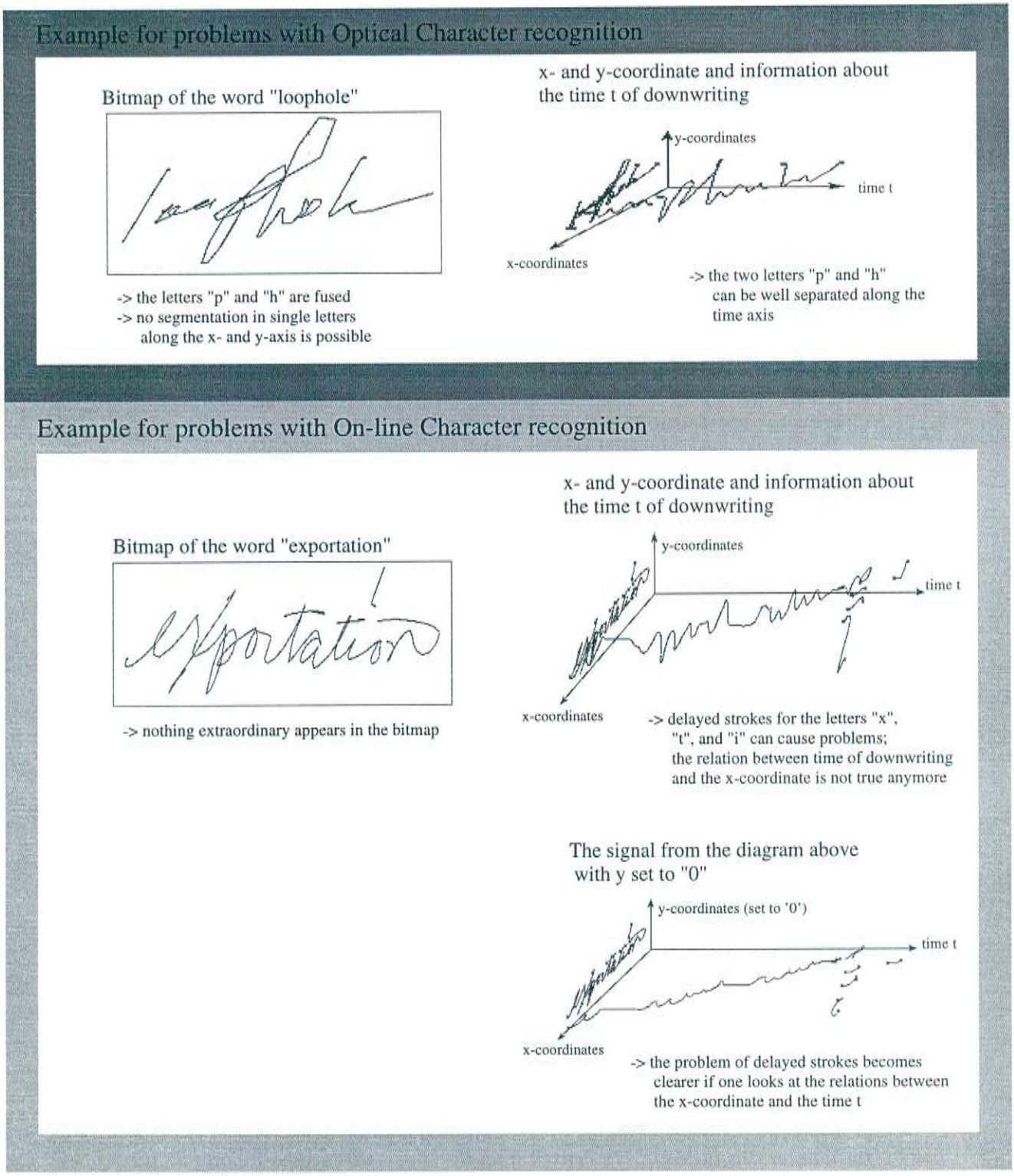

Figure 2.4: Examples for problems with Optical Character (top) and On-Line handwriting recognition (bottom).

that can be recognized is then modeled through a sequence of these letter models. Based on the word models the Viterbi search is done on the output of the Neural Network to make the final decision to obtain the recognition result. The whole recognition process of the system is illustrated in Figure 2.5.

The mapping from the TDNN output to the word models assumes, that all the strokes belonging to one letter are written in about the same time region. But this in not always true in human handwriting, like seen in the example above. The most typical examples are delayed t-strokes and i- or j-dots. As a consequence of this you can have a bitmap from the input coordinates that looks like a "perfect" written word but the recognizer is not able to match the time sequence correctly to the word model, because the sequence is sorted according to the time of downwriting and not according to the space along the axis of x-coordinates.

Another problem in an on-line based recognizer could be the order of coordinates within one stroke. For the recognition process it should be no problem, if a writer draws a stroke, e.g., from the bottom left to the top right or from the top right to the bottom left, if it has about the same shape in the final bitmap. In the NPen^{++}recognizer it turned out that the Neural Network is able to generalize different stroke orders so they do not cause big trouble here.

But the so called **"delayed stroke problem"** proposes big problems, especially in the case of repair and corrections, like it will be further discussed in the next chapters. Different approaches have been tried to solve this problem. One possible remedy, that is also used in the NPen^{++}system, is the introduction of a so called "hat-feature". With a heuristic the system tries to detect all t-strokes and dots and removes them from the sequence of coordinates. At every remaining coordinate that lies in the same area as the removed strokes and dots the hat-feature is set to "1" otherwise it its set to "0". This is the reason, why the dot over the "i" in the example of the normalized handwriting at the bottom left of Figure 2.5 is invisible, because the delayed t-strokes and i- or j-dots are removed in the normalization step of the preprocessing. Therefore the letter "t", for example, in an input signal for the recognizer would look more like the letter "l", because the t-stroke is missing. But because of the hat-feature a correct classification is still possible. I trained the NPen^{++}recognizer with a dictionary that included 51866 words first with the use of the heuristics to remove delayed t-strokes and i-dots and the adding of the hat-feature to the input feature sequence, and second without any handling of the delayed strokes. The recognition accuracy on the test set increased by about 2% with the use of this special handling for delayed strokes. Critical with this approach is that not only t-strokes and i- or j-dots can occur as delayed strokes. Completions of some parts of a character or insertion of a forgotten letter between two previously written ones are typical phenomena often regarded in human handwriting. The recognizer usually generalizes over very small delayed strokes, like they occur in the case of completions of some parts of a letter. But if the delayed stroke is longer, like in the case of repair, e.g., when a missing letter is inserted, it is obvious that problems arise and the recognition can fail.

Another way, different to the hat-feature, to overcome this problem is to resort the strokes in a preprocessing step before the recognition is done. This approach is performed, for example, in [Kas95]. Here every stroke is reordered in the sequence of handwritten strokes according to the horizontal position of his center. The motivation for this heuristic was derived from the three strokes used to form a "tt" sequence with a common cross stroke. But it is also kind of critical, since it can happen that one stroke covers more than one letter and therefore the parts of the handwriting belonging to one single letter still can be separated by some coordinates belonging to some different letters. It seems like an insertion of a delayed stroke in the middle of another one is sometimes needed to fulfill the requirements of the

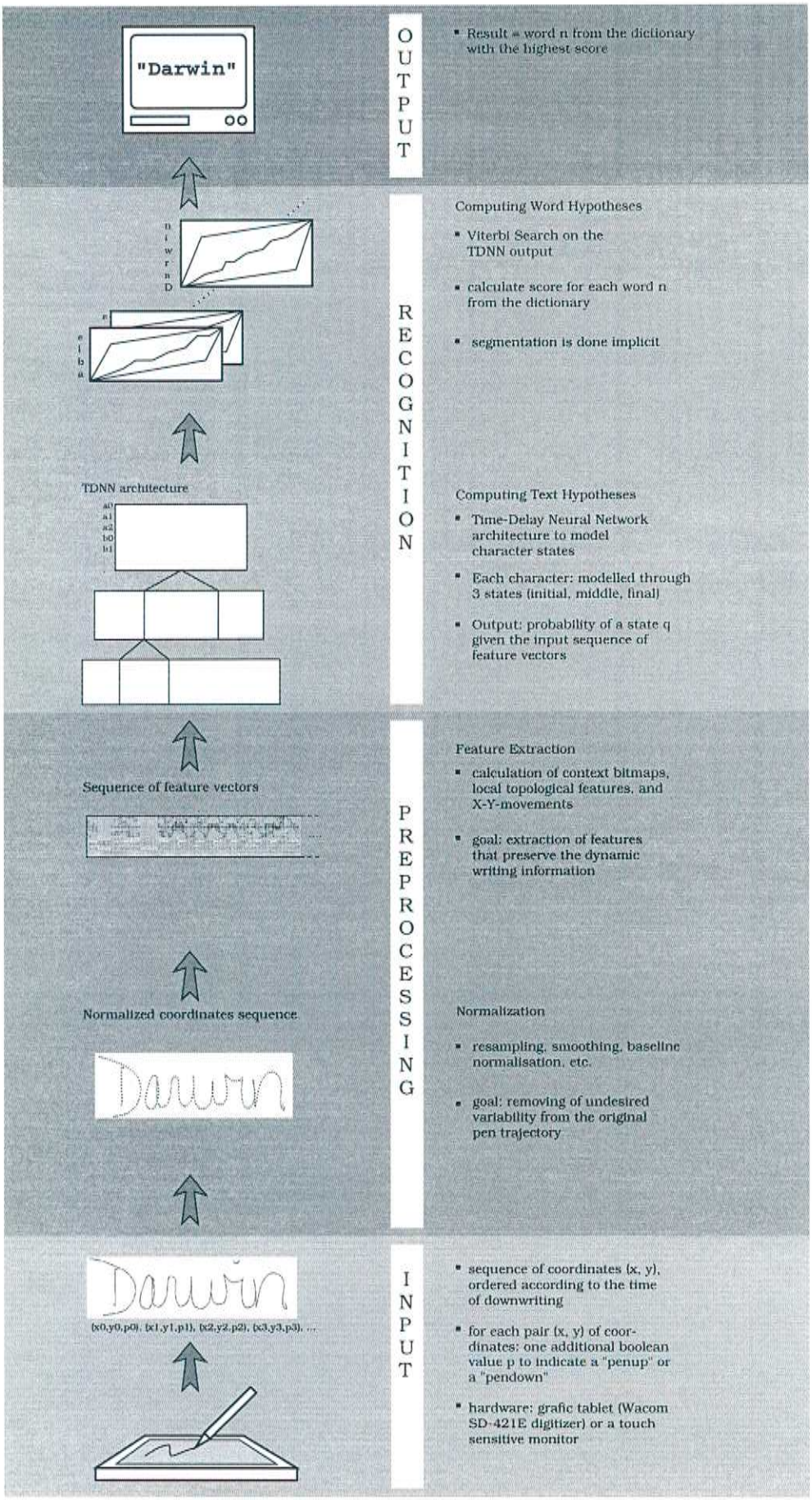

Figure 2.5: The recognition process of the NPen^{++}on-line handwriting recognition system.

recognition algorithm. I will propose such a heuristic in context with the problem of repair handling in part II of this report.

Chapter 3

Repair in On-Line Handwriting Recognition

In the last chapter some typical errors that usually occur in automatic handwriting recognition of pen based textual input have been classified and analyzed with a database. In this section we will try to find out which kinds of repair can happen as a result of such errors and how a classification can be done.

Like in the case of errors, there are also different possibilities to sort repair types in some general groups with similar features and characteristics. To get an idea what kinds of repairs can happen in human handwriting and how they can be classified as a basis for the development of some repair handling algorithms a **data study** with the database already introduced in chapter 2 was done here. For this purpose it was analyzed, if there is any kind of repair in it. The data labeled "JUNK", "SPELLING" or "CROSSOUT", i.e., the "bad" data, not usable for the training of a recognizer, has been investigated. From the 3466 single words about 13% and from the 3410 eight words long text sequences about 23% had this labels, like already reported in chapter 2 (compare Figure 2.2). These data has been visually checked with the purpose to get an idea what kinds of repair usually happen in human handwriting.

By analyzing the different correction styles in the database I tried to find possible characteristics usable for a classification of different repair types. One possibility could be to group repairs according to different shapes or gestures, done by the user to correct something. Another approach is to classify repair in relation to the error list discussed in chapter 2. The different groups for repair could be based on what kind of error they try to correct. But since here we are mainly interested in the design and implementation of repair handling algorithms and heuristics, I propose the following **classification of repair types**, that is oriented on the kind of reaction a repair handling tool should perform on the repaired input signal: *repair classification*

- *deletion*:
  the user scribbles over something he has written before with the intention to delete his previous writing; see the top of Figure 3.1 as an example;
- *completion and insertion*:
  the user goes back in x-direction and adds some strokes to some previously written things; see the middle of Figure 3.1 as an example;
- *overwriting*:
  the user writes something over some parts of the word he has written before; see the bottom of Figure 3.1 as an example;

	overwritten strokes	repair strokes
Deletion	deletion	deletion
Overwriting	deletion	insertion
Completion and Insertion	– –	insertion

Table 3.1: Repair handling actions for the different repair types.

Figure 3.1: Examples for the different repair classes.

In practice also a combination of these three repair types might happen. This classification is not a complete and detailed. Especially the borders between the overwriting and the completion case are very fuzzy. But from the viewpoint of repair handling these classifications make sense. Every repair class that I propose demands for another kind of reaction by the repair handling algorithm. Table 3.1 shows how different reaction can be assigned to this repair types. A deletion requires, that the strokes that are intended to be scratched out, i.e., the repaired strokes, and the strokes that perform the deletion, i.e., the repair strokes, have to be removed from the input signal before sending them to the recognizer. The overwriting case calls for a deletion of the strokes that should be overwritten, and also for an insertion of the strokes that overwrite them. Just an insertion, without the need for deleting anything, has to be done in the completion case.

database analysis

An **analysis of the database** has been performed according to this classification. The results how often which repair class appears in the data can be found in Figure 3.2. Once again is it important to keep in mind, that the user was not asked

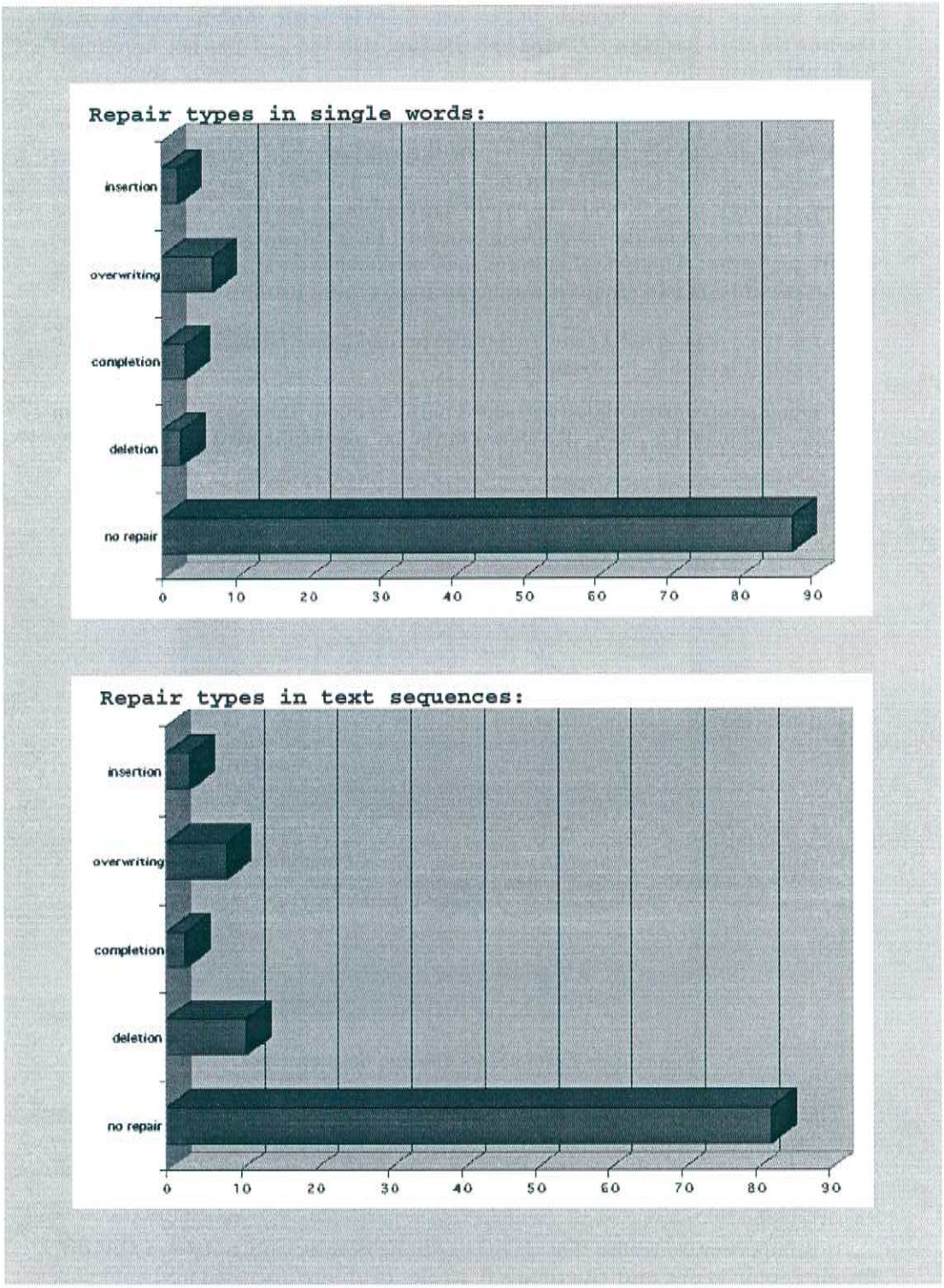

Figure 3.2: Occurrences of the different repair types in the database.

to do any kind of repair during the process of data collection and that he did not have any feedback from the recognizer. Therefore the number of corrections found in the database is not very high, but in fact, there is repair. And in relation with the way the data has been collected and the fact, that the user was not introduced to do any repair, this confirms the proclaim that human users demand some repair and error handling mechanisms in automatic handwriting recognition.

A closer look to the samples found for the different repair classes shows some interesting aspects of the different types of corrections. First of all, *completions* did not happen very often. Even in the case of written word sequences their number was very low. *Deletions* on the other hand happened more often, especially in the case of text sequences. The typical gestures that were mainly used to delete something can be classified in two groups according to some typical features and properties:

- relatively long strokes compared to the normal handwriting, like shown in the top of Figure 3.3, for example, and
- many strokes (sometimes very short ones) in about the same area, placed in an unpredictable order, like shown in the bottom of Figure 3.3, for example.

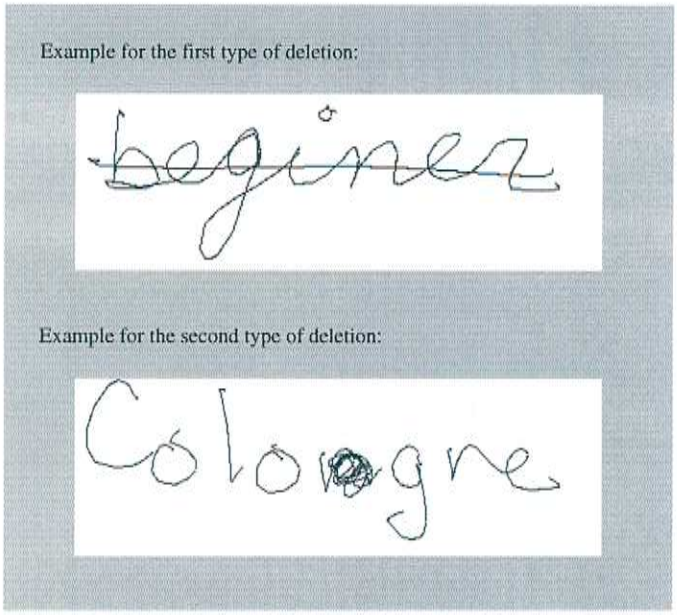

Figure 3.3: Examples for the two deletion types.

These gestures are usually used to delete (a) the whole word, or (b) one single letter. Other deletions, like deleting two letters or only a small part of one letter, did not happen very often in the database. *Overwriting* was also done mainly on letter level, but no "usual" correction scheme was found for this case of repair.

repair and delayed stroke problem

It is important to realize that there is a strong **connection between the different repair types and the delayed stroke problem** discussed in chapter 2.2. Repair often occurs as delayed strokes and requires technically the same handling as, e.g., delayed t-strokes and i- or j-dots. In fact, a delayed t-stroke, for example, can be seen as the completion of an incomplete letter "t" by adding a (delayed) t-stroke. For this reason a unified framework will be proposed here to handle delayed strokes and repair sequences in the same manner: all the delayed sequences

are regarded as a special kind of repair and handled in the same global framework. The cases of delayed t-strokes or i-dots are therefore only a special case of repair, i.e., a completion of a letter that has a bad or incomplete shape. An illustrative example for this can be found in Figure 3.4. On the left side the bitmap of two words written by two different users are shown. The right side shows the x- and y-coordinates in a diagram together with the time t. This value indicates the position in the sequence of pairs of (x,y) coordinates ordered according to the time of downwriting. The pen-up sequences in the bitmaps, indicated through a dotted line, show, that in the lower case the user has made a delayed t-stroke after writing the whole rest of the word "counterflow". In the upper bitmap the user has not only set a delayed i-dot, he also made a repair, i.e., a completion of the letter "x", because a part of it was missing to fit into the scheme of the "usual" shape of such a letter. But if you compare the two examples, the repair and the delayed t-stroke case, you will see, that they fit into the same template of repair or writing actions by a user. On one hand, an incomplete letter "x" has been repaired by adding an additional stroke to it to improve his shape. On the other hand, the letter "t" in the lower bitmap would look like the letter "l", if the t-stroke would not have been added. Therefore the writing of this delayed t-stroke can also be interpreted as a special kind of repair, i.e., the completion of a letter that has a bad shape and is hard to recognize by each, human and automatic recognition algorithm. Note, that in this example a repair only occurs, if the recognition is based on the time signal. If it would be performed only with the bitmap as an input, no repair handling should be necessary.

Figure 3.4: Example for the connection between the delayed-stroke problem and repair. The bitmap of the two written words is shown at the left. The right side indicates the x,y,t-diagram, with t corresponding to the order of downwriting.

errors and repair

To get an idea about the **connection between repairs and the errors** proposed by [Sch94] (see chapter 2) an interesting question is, which kind of repair usually follows after which error type. Table 3.2 tries an assignment of the different kinds of errors to the proposed repair classes. For each error the repair types are listed that happened the most in the database. Some errors caused "typical" repair

Canceled material	Deletion
Discrete noise events	Deletion
Input not legible by algorithm	Deletion, Completion, or Overwriting
badly spelled words	Deletion, Insertion, or Overwriting
unknown words	-
device-generated errors	Deletion, Completion, or Overwriting

Table 3.2: Error types and mainly resulting repair types.

actions by a user. For example, what else should you do with an additional letter, that you inserted wrongly into a word, than deleting it? On the other hand there are errors that do not have a forecast for what kind of repair will probably happen. For example, a letter that has a bad shape can be repaired in all three ways: you can delete it and write it again, you can try to overwrite it, or you can try to "improve" his shape by adding some additional strokes without overwriting any of the existing ones.

In this context it is also important to realize, that there are errors that are seen as such by a user, but also errors that a writer does not realize. For example, a misspelled word represents an error, but the user maybe does not know, that he wrote the word wrong. Therefore no repair reaction, or at least not the one that would lead to a correct recognition result, can be expected by the user. This is one reason, why there are errors from which you can expect an interface to recover, but there are also some that make a recovery from a wrong recognition very hard, if not impossible. For example, you can recover from the error that a letter is missing by providing insertion in your recognition tools.

One should also keep in mind that additional errors can occur in case of a repair. For example, an overwriting that is misclassified by the repair handling algorithms as a deletion can make the recognition result even worse, because it deletes too much of the input signal. Therefore it is important to take care in the design of such algorithms, that they do not only resolve existing errors but also prevent the arise of new ones. Part II will deal with the problem of the design and implementation of such repair handling mechanisms.

Chapter 4

Conclusion

In the previous chapters the **conculsion** was drawn, that **one must offer some error and repair handling features in an interface used for human-computer interaction** to fulfill the users request and satisfy his desire for a high usability of this interface. This conclusion was based on the two theses that

- first: it is unrealistic and probably impossible, that automatic handwriting recognizers will ever achieve 100% recognition accuracy (see [Sch94])

and

- second: humans make and will always make errors and mistakes.

The probably best argument for the second thesis is, that even a keyboard, the "usual" input device, offers the possibility to do some repair (e.g., through a backspace key) although usually a recognition accuracy of 100% is achieved. An **empirical data study** of some human handwriting data collected from different writers also verified this thesis. The analysis of the database indicated, that errors and especially repairs happen, even if the users have not been asked to do corrections. The handwritten input sequences have been analyzed according to what kind of errors occurred in the data not usable for the training of a standard on-line handwriting recognizer. These words has been checked for any kind of repair performed by the different writers. The most surprising result was, that, even if they have not been asked to do any kind of corrections and they did not have any feedback from the recognizer, some repair gestures and corrections can be found in the database. From the whole data set, containing 3466 single words and 3410 text sequences, 13% of the single words and 23% of the sequences of about eight words did contain errors and were not usable for the training. From these "bad" data, 12.6% of the words and 17.8% of the text sequences contained corrections and repair gestures.

It is important to keep in mind, that all the data used for any statistical analysis in this chapter has been collected without the request for repair and without the support of some repair handling features. These statistics therefore can only be a first orientation and can help in the design of repair handling algorithms. But every kind of repair tool offered to a user will affect his behavior in using the interface and therefore change the statistics about occurring errors and repairs. On the other hand, their appearance is a strong argument for the thesis that human users demand for a possibility to repair and to correct in their handwritten input.

The data study gave some ideas and statistics what kinds or errors and repair usually happen in human handwriting of textual input. Typical errors were found in

the database that correspond to a list given by L. Schomaker in [Sch94] for common errors in automatic handwriting recognition of textual input. These **error classes** are:

- canceled material,
- discrete noise events,
- input legible by humans but not by the algorithm,
- badly spelled words,
- unknown words,
- device-generated errors.

For the occurring corrections in the user inputs that were found in the database the following **classification of different repair types** was proposed:

- Deletion,
- Completion and Insertion,
- Overwriting,
- Mixtures of all cases.

These classes are based on the different kinds of reactions a repair handling algorithm should perform to deal with these corrections. In the case of a deletion the parts of the words which are crossed out and the ones that performed the repair should be removed from the original input signal before a recognition starts. If an overwrite happened, the overwritten strokes have to be removed from the handwriting signal, too. But the strokes that overwrite them should be inserted at the right position into the remaining signal. In the completion case only an insertion of the repair strokes has to be done. No part of the word must be deleted.

The questions how these different repair handling actions can be realized, what kind of correction tools can be offered to a user, how they can be implemented, etc. are the topic of the next part of this report.

Part II

Repair Handling - Heuristics and Algorithms

Chapter 5

Introduction

In the first part of this report empirical studies have been done to prove, that repair and corrections by people in human handwriting always happen. Therefore you have to deal with this task somehow, if you want to design a pen based human-computer interface that has a high acceptance by the users. Interfaces that allow no repair handling, no corrections from the user side, and no recovery from errors will be useless as new input devices for human-computer interaction.

To deal with the problem of errors occurring in human handwriting different approaches exist. In [Sch94] L. Schomaker proposes some **solutions to handle or to avoid** the **errors** he showed in the list introduced in chapter 2.1. This remedies are the following:

- *constrain the writer:*
  to obtain cleaner pen data, the writer's attitude may be constrained in some acceptable way, like single guidelines, pen-driven word segmentation methods, gesture-driven or time-out driven word segmentation, etc.

- *give the writer more control:*
  new visual widgets and methods for handling the input of material for which no acceptable recognition performance can be reached, e.g., make a toolbar of symbols like .,;:

- *provide more powerful mechanisms for error handling:*
  for example, offer a large undo stack to the user

- *clarify to the user what is going on:*
  allow input validation by the user, e.g., graphically echo symbols of gestures or entered characters before executing them

- *give up the paper metaphor:*
  the paper-mimicking approach without on-line guidance leads to unrealistic user expectations and consequently low real-life recognition rates

These suggestions to deal with the problem of errors in automatic handwriting recognition are all very restrictive to the user. Therefore they are critical regarding the goal of high user acceptance and satisfaction. For example, it is a usual technique to have user-driven segmentation methods when using a keyboard, i.e., pushing the space bar or the enter key, but it could be hard for a human to write a continuous text with entering a gesture after every single written word, because he is not used to do so. The problem is, that a pen is a tool that people use nearly every day

under special circumstances. If they change, it can be very hard for a human to use this tool now in a very different way.

Buttons, additional toolbars, etc., might be too complicated to handle, if they cover the whole spectrum of repair. To offer buttons for every thinkable repair a lot of (maybe too much) buttons, some with very different, some with very similar functionalities are needed. On the other hand, if there are only a few buttons, there might be some functionality missing and if they are to general, they might be to unflexible to use. For example, imagine just one "clear" button that clears the whole screen in a pen based interface. That should turn out to be to inflexible, if you just want to delete/clear one letter.

On the other hand an interface that offers too much freedom to the user, e.g., in allowing him very unrestricted repair possibilities, would probably result in a bad recognition performance. The more corrections are allowed in the input, like crossing outs, overwritings, etc., the more likely is a misinterpretation and a recognition error. That will lower the acceptance for the interface by the user.

It seems like the "right way" for the interface design lies somehow in between in giving the user enough freedom in using the interface and in restricting him to improve the recognition performance.

It can be seen from the solutions proposed by L. Schomaker for error handling in automatic handwriting recognition, that approaches to deal with errors and corrections do not only include different repair detection and handling methods, but also some approaches in the area of interface design. A special kind of interface can not only support repair and correction handling features, but also influence the user in a way that errors do not happen or can easy be controlled and corrected.

As an example I like to introduce the so called **run-on recognition** , that has been implemented in the NPen^{++}system (see [Gro97]). This modification of the "regular" recognition process fulfills in some sense the last two solutions proposed by L. Schomaker, i.e., to clarify to the user what is going on and to give up the paper metaphor. Usually the processing and the recognition of a word that is written into a pen based interface starts after the user has finished the writing of the whole word. In run-on recognition the preprocessing and recognition process starts automatically after the user has written a set large enough to perform a first preprocessing or recognition step. To process the features and to do the calculation of the hypotheses for the first letters in the word it is not necessary, that the user has already finished writing. The main motivation for the use of this run-on recognition is to lower the time a user has to wait till the recognition result will be indicated. But since the first parts of the word that already have been processed by the recognizer are shown in a different color and the result calculated so far is already indicated, the user gets a kind of a feeling, that the recognition is done based on the time sequence of the input. On a piece of paper he just sees the bitmap and does not get any feeling for the time sequence resulting, when he writes something down. But with the coloring of already processed parts of the word and the early indication of the first letters of the result the paper methaphor is given up. The probability that a user goes back and writes into an area that has already been processed will be lowered. This can reduce the risk of misclassifications as a cause of delayed strokes. Note that on the other hand there are cases where the probability for a repair to occur can be higher: if the early indicated result of the sequence processed so far is wrong, a repair in this part of the input should be more likely.

error handling vs. avoiding

Summarized there are in general two ways to deal with errors in human handwriting recognition. I will refer to this **two important concepts** in the following as

- the **concept of error handling**, that is trying to recover from occurring errors and mistakes, e.g., by offering some repair tools and correction possibilities to the user,

and

- the **concept of error avoidance**, that is trying to reduce or avoid the occurrence of errors and therefore improve the recognition accuracy and limit misclassifications by designing the interface, the recognizer, the repair handling tool, etc., in a corresponding way that supports the goal of error avoidance.

To design robust human-computer interfaces in automatic handwriting recognition one can and should take advantage of both concepts. They should be used to reduce errors and allow recovery from occurring ones in a way that is easy to learn and understand for a user and with which he feels comfortable and satisfied.

In the case of error handling you have to differentiate between errors and corrections in the handwritten human input and errors that occur also or only in the recognition result. Repair handling tools can be offered to correct your own handwriting, but there usually also is a need to allow repair of the indicated recognition result, i.e., to do corrections in printed ASCII-text. Both of them have their needs and are useful under the corresponding circumstances. I will refer to each of the advantages in the following chapters. In *chapter 6* the task of repair in a handwritten input signal is investigated. *Chapter 7* deals with the problem of repair in text strings, i.e., corrections of the recognition result that is indicated to the user. Different approaches will be discussed and several heuristics have been implemented and evaluated.

contents part II

Chapter 6

Repair of Handwritten Input

Before offering repair handling and correction tools in human handwriting you should face the question, why it is necessary or useful to offer such possibilities at all. Of course, it is important to have repair handling algorithms, like already discussed in part I. But why correction of the input signal? Since the main purpose of handwriting here is the automatic recognition, why not allowing repair only in the recognized result that is indicated by the recognizer?

First of all users might feel more comfortable, if they can correct their own handwriting, instead of a printed result indicated by the interface. Especially in cases and applications, where the recognition result appears on a different position on the screen, this might be uncomfortable for the user, because he has to move his pen from the area in which he usually writes to another one to perform his corrections. And in fact, there are cases, where it is not even possible to do corrections at the position of the indicated result, e.g., when a graphic tablet is used as input device and the result is shown on a computer screen[1]. Also there are situations possible in which it seems more "natural" to correct the input signal. Imagine you have written the word "tested" and after a correct recognition you realize that you wanted to write "test" instead. Of course, you can just cross out the letters "ed" in the recognition result, but some users might feel more comfortable, if they can correct the part that is wrong. And in this case that is your handwriting and not the recognition result.

The thesis, that some users prefer to correct their own handwriting instead of a printed text with the recognition result, is also supported by the data study in part I that showed, that repair of the handwritten input signal happens, even though the users are not asked to do any. If they correct their own handwriting without being forced to, it is more than likely, that they will also do and prefer to do it, if repair and corrections are allowed. The fact that this kind of repair occurs (and can occur without being noticed by the user) is another strong argument, why a repair handling tool that operates on the input signal should be part of a comfortable, useful, and user satisfying interface.

It should be noted that the problem of detecting and handling repair and corrections of the input signal in on-line handwriting recognition is a very hard one. The main difficulties result from the fact that humans write and especially correct based on the bitmap they see. An on-line recognizer on the other hand usually bases his recognition decision on the time ordered sequence of coordinates. The user has a different view of the input data than the recognizer has. Therefore the advantage

[1] This situation can be compared to the one, where you type on a keyboard and instead of using the backspace key to delete a letter, you always have to point at the position on the screen where this letter is indicated. It is obvious that this is not what can be seen as a user friendly interface.

of additional available information that is used in the recognition process (compare chapter 2.2) turns out to be a disadvantage in the case of corrections and repair.

Another problem is that repair is a very varied area. There are some typical, usually used kinds of repair but no fixed rules or predefined shapes for repair gestures exist. Technically nearly every input is "allowed". Humans are able to generalize and use common knowledge, so no need for fixed rules in the repair case is necessary. But, on the other hand, there are some typical gestures or at least some typical features usually used by writers to repair. One example was reported in the data analysis in part I for the case of deletions. The challenge in the design of repair handling tools is to profit from these "common" properties to propose an interface to the user that fulfills his demand for good repair and error handling features.

In the design of repair handling algorithms and heuristics some typical steps have to be taken and different problems have to be solved. One possible approach is the following:

- first: detect a repair and separate the corresponding coordinates from the "normal" handwriting
- second: classify which type of repair class the coordinates belong to
- last: handle the specific type of detected repair

This stepwise approach is similar to some techniques used in the task of repair in speech recognition (see [BKZ94] and [BDS92], for example). But it should be noted, that the problem of repair of a spoken signal is different to the one discussed here, since repair in speech recognition is usually done on word level. Also the detection and handling of corrections does not have to be done in separate steps, like proposed here. It can also be integrated in a general manner into the preprocessing and recognition algorithms so that no singular detection, classification, and handling modules are necessary. The decision of where to apply your repair handling heuristics, in separate modules or integrated into the recognition process, depends strongly on the techniques and algorithms used for preprocessing and recognition.

Here I decided not to change the recognition engine but to apply some repair detection and handling heuristics in an additional preprocessing step. On one hand I did not want to change the structure of the NPen^{++}recognizer, since the performance of that system on "clean", not repaired data is very high. The integration of repair handling features directly into the NPen^{++}system can not be done with some small changes of the system. Bigger modifications on the principles according to which the system is working would be required. But these changes would risk a higher decrease in the recognition performance of the system on "clean" data, which is not acceptable.

Another reason to integrate the repair heuristics into some additional modules is that humans usually do not have any problems in the detection of a repair, like, for example, a deletion, even though it is a kind of correction they have never seen before. That is because repair usually is extremely different from "normal" handwriting. It does not fit into the "usual" scheme of textual handwriting. But not only repair types that can be recognized by humans are well separated from the "normal" handwriting. Also the ones that only should be classified as a repair in the context of a specific recognition, like delayed strokes in an on-line recognizer, often have typical features that let them appear different from the textual input without any corrections. Therefore an additional detection and classification step prior to the repair handling seems to be a very promising approach.

6.1 Repair Detection and Classification

6.1.1 Introductory Remarks

To perform repair handling on the input signal of human handwriting **a level** has to be fixed **on which the repair and correction handling should be done**. For example, many single word recognizers just offer one "clear" button as repair possibility, which is technically the highest level one can choose for repair handling. But this "all-or-nothing" approach is very unflexible. For example, if you just misspelled one letter, you have to rewrite the whole word, instead of correcting only this single character.

It should make more sense to allow corrections on a level that is oriented on a semantic interpretation of the handwriting, for example,

- word(s),
- letter(s),
- parts of a letter.

But there are some problems with this approach, too. First repair is not always done on letter level, even though the users intention is just to correct only one single letter. In the database, described in chapter 2, examples can be found, where a user, who wanted to correct a letter by overwriting it, not only overwrote this single letter but also the begin of the following one. The second problem is, that it is very hard, if not impossible, to find a usable segmentation into letters within the handwriting. In fact, correct segmentation of a handwritten word into its letters is a problem as hard as the recognition problem itself (keep in mind that we are trying to find a segmentation on a wrong or badly written word or on one that contains corrections).

To avoid these problems other approaches should be thought of. One possibility is to define different layers of units from the handwriting signal not according to the semantic meaning but based on information gained from the time and the area in which the coordinates were written. This leads to one possible **hierarchy of levels of a handwritten word**: *levels of one word*

- **everything**, i.e., the whole word;
- **strokes**, i.e., all the coordinates that were written between a pen-down and a pen-up;
- **up-down strokes**, i.e., all the coordinates that were written between two neighboring local extrema of a stroke;
- **parts of up-down strokes**, for example, an up-down stroke can be divided into the parts that lie between the base-, center-, descender-, and ascenderline that are calculated in the preprocessing step (see [MFW95b] or [MFW94] for more information about the preprocessing in the NPen^{++}system);
- the single **coordinates**, which is the lowest possible level;

An example for this different layers can be found in Figure 6.1.

In this report the detection of repair is done on the level of strokes. Everything that is written within one stroke is regarded either as "normal" handwriting or as a repair. There are some cases where this level is too rough for a classification, but if a misclassification happens, the repair handling algorithms will take care of it.

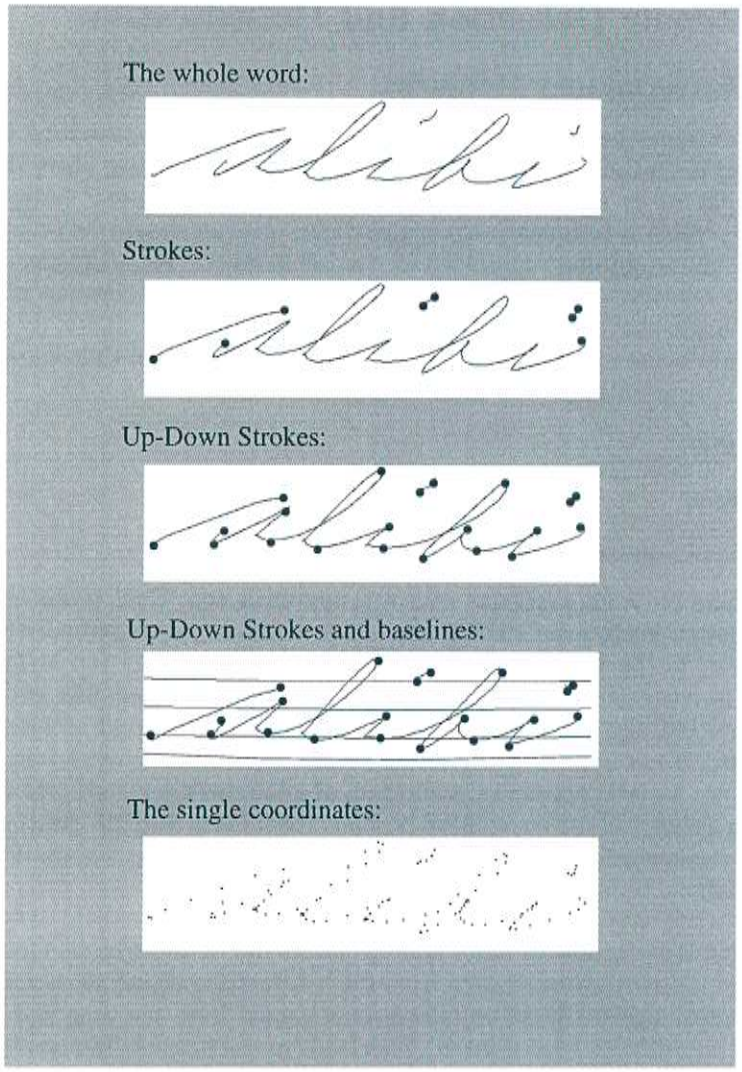

Figure 6.1: Example for different segmentations of a handwritten word.

The strokes level is to strong in the case of handling the detected repair. For example, a whole word can be written in one stroke. But now to remove a single letter from the word a unit smaller than all the coordinates between two pen lifts is required. Otherwise the whole word would be deleted instead of the single letter. It turns out that the level of up-down strokes is a reasonable choice to handle repair and corrections. There are, of course, cases in which this approach might fail, but they are rare. Therefore they are accepted here in favor of a high speed which turns out to play an important role in the repair handling heuristics proposed in this thesis (see the following chapters). Also, like in the case of a misclassification of strokes as repair, I will take care of such critical cases in the repair handling algorithms. Note that also a reasonable segmentation of a handwritten word into its letters can be done not in any but in a lot of cases on the up-down stroke level. And usually one up-down stroke does belong to one word, or at least to two words in the case, when it contains the segmentation border between these two letters. Therefore they propose a good basis to apply repair handling algorithms and heuristics.

To detect and classify typical repair gestures and actions you need to have a look at "normal" human handwriting, on how repair and corrections differ from this writing, and how they can be separated from it. These "typical" differences can be used in a detection and separation process of corrections and "normal" handwriting.

Hollerbach's model of handwriting

One well known model for human handwriting is **the oscillatory motion model of handwriting** introduced by Hollerbach (see [Hol81] or [ST94]). In this model cursive handwriting is described by two independent orthogonally operating oscillatory motions, one in the horizontal direction, and one in the vertical direction. An independent, relatively slow, constant linear rightward drift along the line of writing is added to the horizontal oscillatory component, providing the model with a constant rightward progression. A schematic representation of this idea can be found in Figure 6.2.

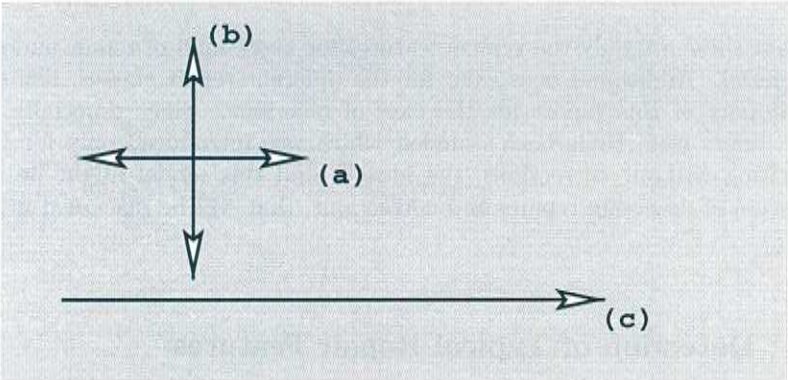

Figure 6.2: Schematic representation of Hollerbach's movement components (from [LMT96]). (a) horizontal oscillatory component, (b) vertical oscillatory component, (c) constant rightward progression.

The system can be described by the following differential equations:

$$M_x\ddot{x} = K_{1,x}(x_1 - x) - K_{2,x}(x - x_2) \tag{6.1}$$

$$M_y\ddot{y} = K_{1,y}(y_1 - y) - K_{2,y}(y - y_2), \tag{6.2}$$

where $K_{1,x}$, $K_{2,x}$, $K_{1,y}$, $K_{2,y}$ are the constants and x_1, x_2, y_1, y_2 are the equilibrum positions of the spring muscle system modeling the human muscular motor system, responsible for the production of human handwriting. Solving this equations set with the initial condition that the system has a constant velocity (drift) in x-direction, yields the following parametric form:

$$x(t) = A\cos(\omega_x(t - t_0) + \phi_x) + C(t - t_0) \tag{6.3}$$

$$y(t) = B\cos(\omega_y(t - t_0) + \phi_y). \tag{6.4}$$

The angular velocities ω_x and ω_y are determined by the ratios between the spring constants and masses. A, B, C, ϕ_x, ϕ_y, and t_0 are the integration parameters determined by the initial conditions. This set describes the two independent oscillatory motions, superimposed on the linear constant drift C along the line of writing, which generate cycloids. Different cycloidial trajectories can be achieved by changing the spring constants and zero settings at the appropriate time. The relationship between the horizontal amplitude modulation $A_x(t)$, the horizontal drift C, and the phase lag, $\phi(t) = \phi_x(t) - \phi_y(t)$, controls the letter corner shape.

The horizontal drift velocity C can be estimated by the following equation:

$$\hat{C} = \frac{1}{N} \sum_{i=1}^{N} V_x(n), \tag{6.5}$$

where N is the number of digitized points.

typical repair features

Typical repair gestures and actions found in the data analysis in chapter 3 usually violate this model of the human cursive handwriting process, which was introduced for "clean" data without any corrections. The features most typical for the occurrence of repair are

- going back in time or staying long into the same area of writing, i.e., violating the constant drift along the line of writing in Hollerbach's motion model, and
- the writing into an area where the writer has already written something before.

Note that these are only the typical features for *every* kind of repair and therefore very general. Additional ones exist for the different repair classes, like shown in the first part of this report for the case of deletions. Since especially the first feature differs from Hollerbach's model, which was introduced only for "normal" handwriting without corrections, the idea behind this model might be useful in the process of detecting repairs and corrections, that will be discussed in the next section.

6.1.2 Detection of Typical Repair Features

To classify a repair and to separate it from the "normal" handwriting one has to face the question, what is typical for a correction and what features are useful to find such categories. In the last chapter it was already reported that the two most typical and obvious characteristics of a repair are that the writer goes back in x-direction and writes something in an area in which he already has written before.

A first approach to detect such a "going-back" in space could be to define a **fixed threshold** Ξ that is moved with the pen while writing on the surface. If a new pen-down happens left of the actual position of the threshold Ξ, a repair starts. An illustrative example can be found in Figure 6.3. The problem with this approach is, that with different writing styles different values for the thresholds Ξ are required. If the letters written by a user are very high but narrow the threshold Ξ should be small, if the letters are very wide regarding the x-direction it should be much bigger. Note that not only different writers but also a single writer sometimes changes his writing style even within one word. For example, a well known phenomenon is, that the shapes of single letters are often getting wider but lower towards the end of a word (see [LMT96]).

To avoid these problems I did some experiments with **Hollerbach's oscillatory motion model of handwriting** introduced in the last chapter. The idea is to take a change in the velocity of the pen movements that is responsible for the drift to the right in the writing direction (see equation 6.5) as an indication for a repair. A strong decrease in this velocity should be an indication for a going back with the pen in x-direction and therefore be a good indication for the occurrence of some corrections. To do a check, if a possible correction occurred, the relative change in velocity should be regarded. One problem with this approach is, that the velocity changes depend on different letters, e.g., if a user writes two letters "ee" the velocity

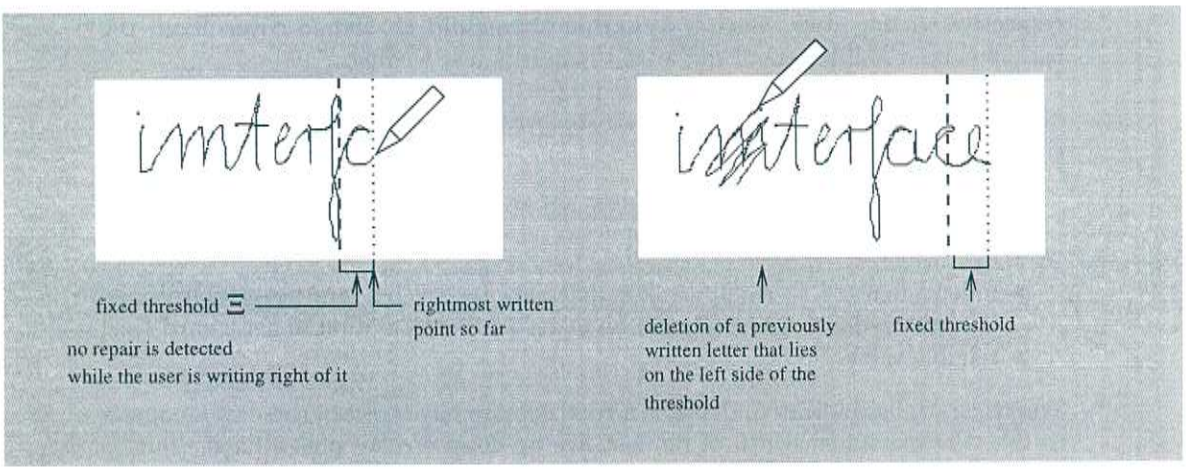

Figure 6.3: Example for the fixed threshold method for repair detection.

will be smaller than when he writes the letter sequence "ll", even if they take about the same space in x-direction. This is, because the calculation of this velocity in the model is intended to be calculated on the whole word, not step by step with every new up-down stroke, like it is done here. To avoid this problems I changed the calculation of the velocity in a way that I did not base it on the number of pixels written but on the number of up-down strokes. Some examples of written words and the corresponding velocities calculated in this way can be found in Figure 6.4. With this change the velocity criterion became more stable, but the problem of changing writing styles still remained. It also took very long, till the velocity became stable enough to be used for a reliable classification of repair strokes. This is not acceptable, especially in single word recognition.

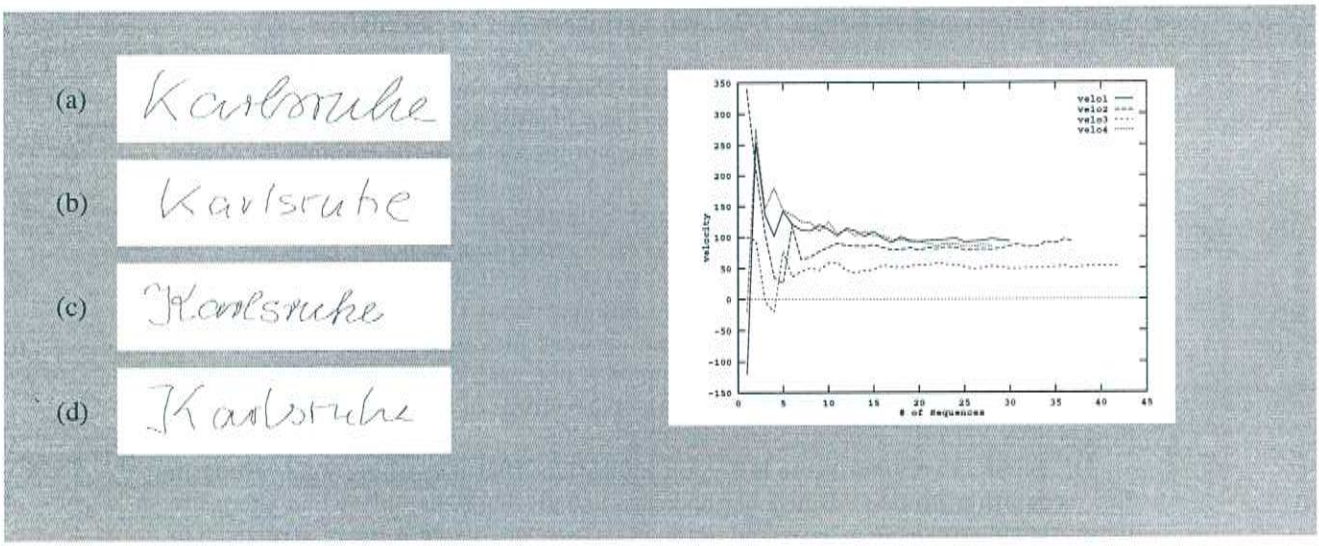

Figure 6.4: Example for the use of Hollerbach's model for repair detection.

Therefore another approach was tried that is a kind of combination of the use of a fixed threshold Ξ and Hollerbach's oscillation model. Like in the first approach, a threshold Ω is used, but this time it is calculated and adapted dynamically to the

dynamic threshold approach

respective writing style. Such a **dynamic threshold** Ω should cover about the last one and a half letters of the word the user is just writing.

The idea is to set the threshold Ω proportional to the width (= extension in x-direction) of the last few up-down strokes written by the user. The value of this threshold Ω should be

- adapted to the actual writing style, so not to much up-down strokes should be taken for its calculation, and
- independent of to small strokes that can appear but are not "typical", so enough up-down strokes should be taken for its calculation to get a statistical reliable value.

Empirical studies indicated, that it is a good compromise between these two requests to take the maximum width of the last five up-down strokes plus a fixed value as an adapted dynamic threshold Ω.

Another modification that made this criterion more stable was not to take the maximum x-value written so far as index for the threshold check but the begin of the last completely written up-down stroke.

One problem with this approach is that repair in the last letter usually is not recognized. Completion and insertion can not happen in this case but, of course, deletion and overwriting are two types of repair that also occur at the end of a word. To solve this problem an additional heuristic was added to the dynamic threshold criterion. I will refer to this in the later chapters, where I describe the algorithms and heuristics for the corresponding repair types.

Another problem is the fact of misclassification. Sometimes a new stroke is classified as a repair, even if it belongs to the "normal" handwriting. But you should keep in mind, that this detection heuristics are only a first check and that misclassifications can be handled by "clever" repair heuristics and algorithms. For example, a stroke classified wrongly as a repair can be reinserted in the "normal" handwriting sequence just like a delayed t-stroke would be inserted.

An additional approach I tried was to use the **score** and the output of the Time-Delay Neural Network to detect irregularities, like deletions or other corrections. The idea was, that an unusual input that was not presented in the training database, like, for example, the crossout of a letter in a word, should result in a low score of the recognizer and therefore be a good indicator for a repair. But it turned out, that the score calculated by the NPen^{++}recognizer is not a good basis for such a decision. It is the nature of neural networks that data not presented to the net in the training phase and completely different from the training data cause a nonpredictable output when shown to the network in a test phase. Therefore the score calculated in the recognition process was not usable for repair detection. A high score does not say anything about the correctness of the corresponding result in this case.

Of course, a very low score is a good indication, that something went wrong in the recognition process. But a big decrease usually happens only when there are many repair strokes, e.g., a whole letter is crossed out using a lot of strokes. On the other hand, if, for example, only one single short stroke is used to do a deletion, the decrease in the score usually is not that significant, like it should to be useful for a good repair indication. Like already mentioned in the discussions about the delayed stroke problem in section 2.2, the recognizer generalizes over short delayed strokes, i.e., they are ignored. Small strokes do not influence the score calculation significantly. Also the features calculated before the recognition algorithms are applied might not look very different in this case, too. To understand this we have

to take a closer look at the preprocessing step of the NPen^{++}recognition engine. Figure 6.5 illustrates the first part of the preprocessing, the normalization step. Here the goal is to remove undesired variability in the pen input. For this normalization it does not matter, if the handwritten input signal contains repair or not[2]. For a description of the different preprocessing steps in the input normalization see [MFW95b] or [MFW94]. The feature extraction, which calculates the values that are used in the recognition process, is illustrated in Figure 6.6. If only a short stroke is used as a repair, most of the features do not differ very much from the ones calculated on the "regular" handwriting. The absolute y value, that is, the distances from the base- and the centerlines, are about the same. This is also true with the hat feature, already introduced in section 2.2 and the pen feature, that indicates, if a pen is lift from the writing surface or not. The local features only give information about writing direction and curvatures. Since this is an information coming form a very narrow area of the handwriting it does not differ much for repair strokes. The context bitmap which is a downsampled version of a bitmap that moves with the pen is global in time, since it contains all the values of the coordinates written in a special area, no matter when they were written. But it is local in space. If there are only a few repair stokes (e.g., no "wild" crossouts), the context bitmaps of these strokes do not look very different, because it is only a downsampled version. Therefore the calculated features of some repair strokes that are presented to the recognition algorithms do not differ that much from the ones of the "regular" handwriting. As a consequence of this the score calculated in the recognition process can still be high, even if a repair appears, that looks very different from the rest of the handwriting in the bitmap of the whole word.

Note that there is no information about the absolute x value used in the recognition process. The only way for the recognizer to find something out about the x position is by realizing that there is a relatively long pen up sequence before something new, e.g., a repair, is written. I will use this feature as an indication for a repair in the heuristics that will be proposed in the following chapters, too.

I also thought about the calculation and use of the so called **entropy** from the Neural Network output (see [Pom92]) to detect a repair. This entropy is calculated by

$$E = -\sum_i \frac{hyp_i}{\sum_j hyp_j} \log \frac{hyp_i}{\sum_j hyp_j}, \tag{6.6}$$

with hyp_i is the activation of the i-th output neuron of the Neural Network. The value E for the entropy is minimal, if exactly one output neuron has the value "1" and all the others are "0". It is maximal, if all the neurons have the same activations. Therefore the bigger the entropy value E is, the more unreliable is the output of the corresponding network. But this measure is also not usable as a dependable repair indication. The reason why is, that only a high entropy is a sure sign for an unreliable output, but a small entropy does not necessary mean, that the output decision is correct. Data extremely different from the one presented to the network in the training process can cause an output of the network that is wrong but nevertheless has a very small entropy for the same reasons as discussed above in the calculation of the score. Therefore the entropy is not a good measure to detect repair and corrections, too.

[2]Note that this is not completely true: if the repair strokes differ in their size form the "regular" handwriting, the calculation of the baselines will change.

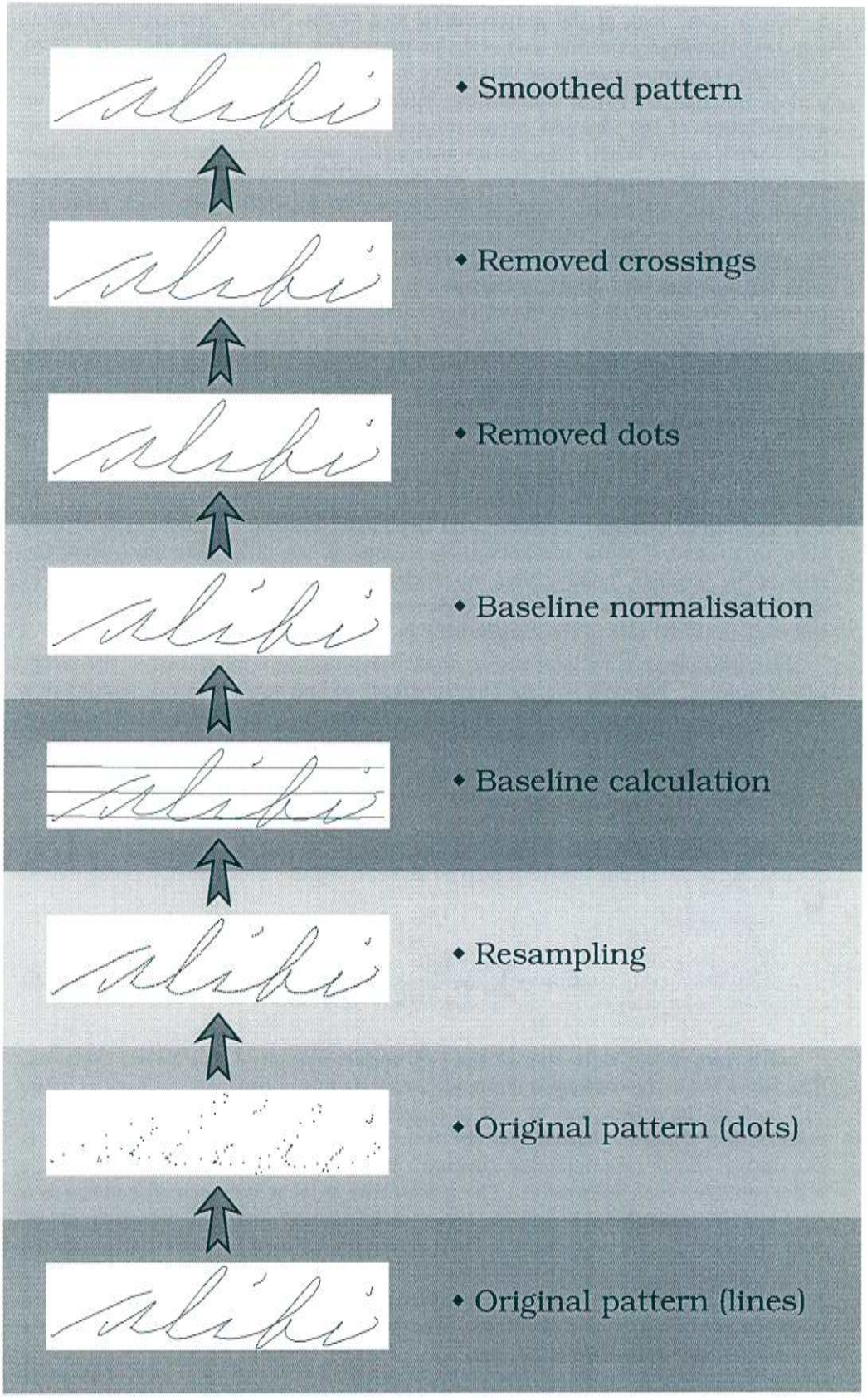

Figure 6.5: The normalization steps in the NPen^{++} preprocessing module.

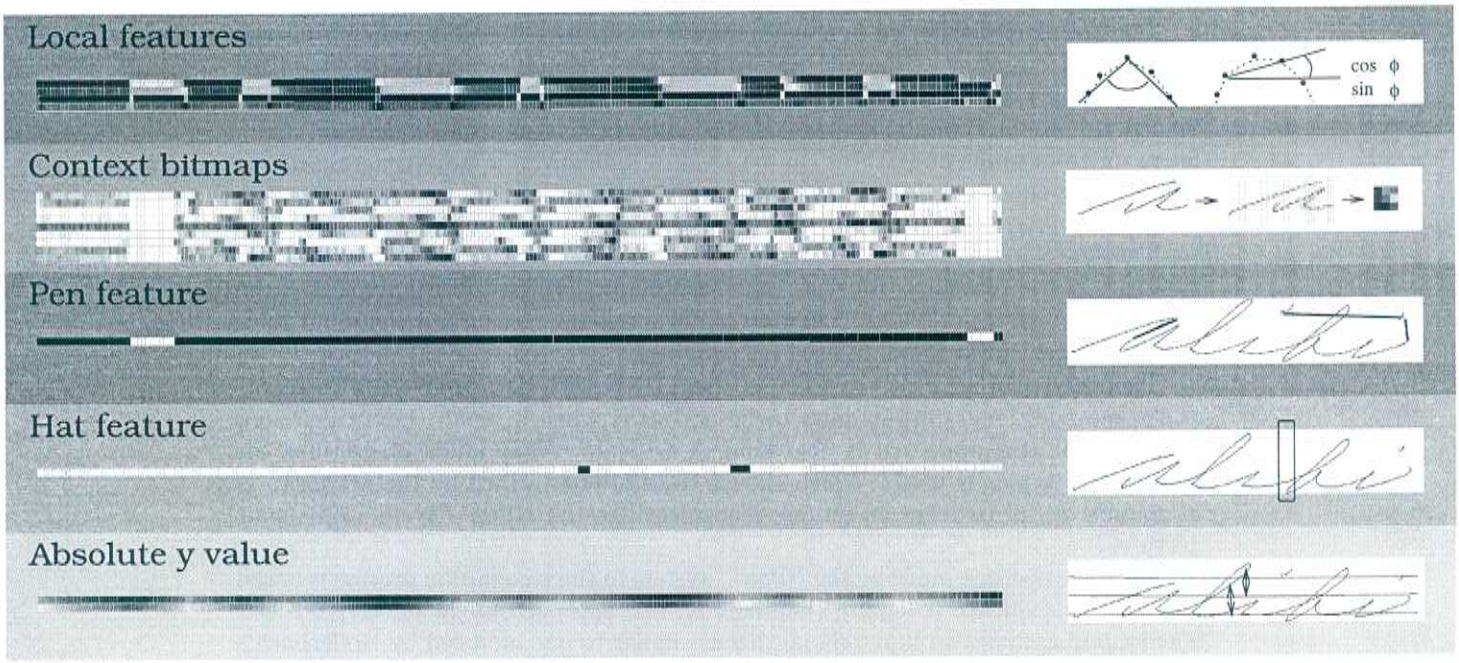

Figure 6.6: The feature extraction steps in the NPen^{++} preprocessing module.

6.2 Handling of Each Single Repair Type

In chapter 3 a classification for different kinds of corrections was introduced. These repair types are **deletion**, **completion/insertion**, and **overwriting**. It was mainly based on the kind of reaction the respective sort of correction should cause in some repair handling algorithms. While insertion and completion require just to find the best points where to insert, in overwriting also the strokes or the parts of a stroke that was overwritten by the correction gestures and strokes need to be deleted. In the deletion case the repair strokes have to be removed, too, instead of being inserted into the "normal" handwriting.

In the following chapters different kinds of repair and correction handling algorithms and heuristics will be discussed. The first section takes care about the case of deletion. Overwriting and completion/insertion handling heuristics are introduced in the sections 6.2.2 and 6.2.3.

6.2.1 Deletion

Typical approaches to offer the possibility to delete some parts from a written sequence by the user are additional buttons or some special gestures.

For example one solution could be to offer a kind of a "rubber"-button that changes the functionality of the pen from writing to deleting and vice versa. One problem with these approaches that use additional buttons is, that they are usually not very flexible and intuitive to use.

Gestures would be a more "natural" approach, because the user can stay with his pen within the area where he is usually writing instead of having to move his pen across the screen to a fixed place where the deletion and other buttons are placed. But I agree with L. Schomaker who wrote in [Sch94]:

"... gestures are nice, but they require highly motivated users who must store and recall them in/from human memory ...".

The approach I followed in this thesis was to arrange a general schema for the detection of deletions. The goal was to offer some common deletion mechanisms to the user instead of limiting him to a set of special shaped gestures or symbols. Since there are no common known "official" rules or characters for a deletion and a lot of different kinds of gestures can be used to indicate it, a more general schema should be applied in a deletion detection and handling heuristic.

The first idea to detect deletion gestures in a handwritten input was to use the score of the recognizer output. But like already discussed in chapter 6.1.2, the output of a neural net based recognizer is not a reliable measure, if the input data is very different form the data used for training.

Therefore another idea was to train the TDNN with data that contains repair and some additional output units to be activated when a repair, i.e., a deletion, occurs. But this approach is also kind of critical. What kinds of gestures for deletion are typically used? Which should therefore be trained by the network? Is it possible at all to train an on-line recognizer like the NPen^{++}system on repair gestures?

Since the decision made by the NPen^{++}system in the recognition process is based on the time sequence (compare chapter 2.2), this approach might be very critical. The system assumes an input signal that is similar to one modeled by Hollerbach's oscillatory motion model of handwriting. Even if small deviations are compensated by the system, it nevertheless supposes some special orders of downwriting within a word and sometimes even within a letter. But there are no fixed rules or typical stroke orders in which a repair gesture is usually made. Several different kinds can be thought of. It seems quite impossible to find a training database that covers enough different repair and deletion gestures with different stroke orders, even if you limit it to some typical, often used gestures. The ways people write this gestures down differ to much, that one can hope to find a database that would be needed for such a training.

For a human reader it is sufficient, if the deletion gestures differ enough from the "normal" handwriting to do a correct classification in the reading process. This is the reason, why I did not try to handle typical deletion gestures, but based **a classification on typical features, characteristics, and differences.** It was already noticed, that going back in x-direction and writing in an area in which was already written before is a good indication for any kind of repair. By analyzing the database I found two typical repair gestures with some special characteristics that are mainly used for a deletion (see chapter 3). Note that these are not gestures that have a special kind of shape. Especially in the second case the strokes can be ordered in a very wild kind and manner. But they are gestures that can be grouped because of some typical features they have in common. The two classes are

- relatively long strokes compared to the normal handwriting, and
- many (sometimes very short) strokes in about the same area, placed in an unpredictable order.

deletion heuristics

Easy **heuristics** can be applied **to classify these two types of deletion gestures.** First a detection is done with the dynamic threshold approach introduced in chapter 6.1.2. If a repair is detected, it is checked, if it is a deletion. For this, the length and height of a repair stroke is taken, i.e., the extensions in x- and y-direction. These two values are compared with the maximum extensions in x- and y-direction of the up-down strokes from the "normal" handwriting. If the repair strokes are ϕ times bigger than the maximum extension in x-direction respectively in y-direction,

with ϕ being an empirically fixed threshold, than a deletion is classified. The deleted up-down strokes, i.e., the ones that have been overwritten from the repair stroke, and the repair stroke itself will be removed from the input sequence of coordinates. The x-values of the coordinates left from the repair have to be repositioned by adding the width of the deletion area. An exemplary illustration of this approach can be found in Figure 6.7.

Another heuristic is used for the detection of the second class. The typical feature here is, that the number of repair strokes is much higher than the one of (up-down) strokes from the normal handwriting, if you compare the deletion strokes and the deleted strokes in the same area. Therefore the total length of all the up-down strokes lying in the space behind the dynamic threshold Ω introduced in chapter 6.1.2 is calculated. It is compared to the total length of all the up-down strokes from the "normal" handwriting that were overwritten by the repair strokes. If the length of the deletion strokes exceeds the length of the overwritten up-down strokes, a deletion is detected. The corresponding coordinates are removed from the input sequence and a repositioning of the remaining ones is done. An example for this heuristic process is illustrated in Figure 6.8.

For deletions at the end, i.e., deletions of the last letter, the dynamic threshold approach does not work. Therefore an additional criterion was used to cover this case: if the number of up-down strokes lying on the *right* side of the dynamic threshold Ω was higher than in the "usual" handwriting, a repair was detected, too. Note that this criterion fits with the case that the constant drift velocity to the right from Hollerbach's motion model (compare equation 6.5) is decreasing.

The second heuristic is kind of critical. To compare the two lengths and to make a decision a threshold φ is needed that can cause problems, especially if not only deletion, but also overwriting is allowed as a repair action. If this threshold φ is set too low, the risk is very high, that an overwriting is wrongly classified as a deletion. This would cause the repair handling algorithm not only to delete the overwritten strokes but also the repair strokes that instead should be inserted into the sequence. On the other hand, if the threshold φ is set to high, the user probably will not make enough strokes to be classified as a deletion. There is no "perfect" setting for the threshold φ. Examples can be found for every possible value, where it turns out to be too small or too high. Section 6.3 will describe this problem in more detail and propose a possible solution. Also an analysis and evaluation of the heuristics will be done there.

To get a feeling of how the threshold φ should be set to achieve the best results I did some **experiments** with data from the database introduced in part I. I took the deletion cases form the word samples and the deletions that were easy to separate from the text sequences (since the recognizer only works with single words). So I got 73 samples that contained this repair type. A list with all the written words can be found in Appendix A, Table A.1. I applied the proposed heuristics to this samples and checked them visually, if the performed corrections were handled properly with different threshold settings. The results for the threshold settings for φ from 1.2 to 2.0 can be found in Figure 6.9. The values at the x axis indicate how many of the deletions were classified and handled properly with the corresponding value for φ shown at the y axis. Like expected, the lower the threshold φ is set, the better the repair heuristics work. Lowering φ under 1.2 was done with the wrong classified samples and improved correct repair detection and handling at about another 12%. But then also 12% of the data was classified wrong, i.e., a deletion was detected in the part of the word, where no such correction has been done (for

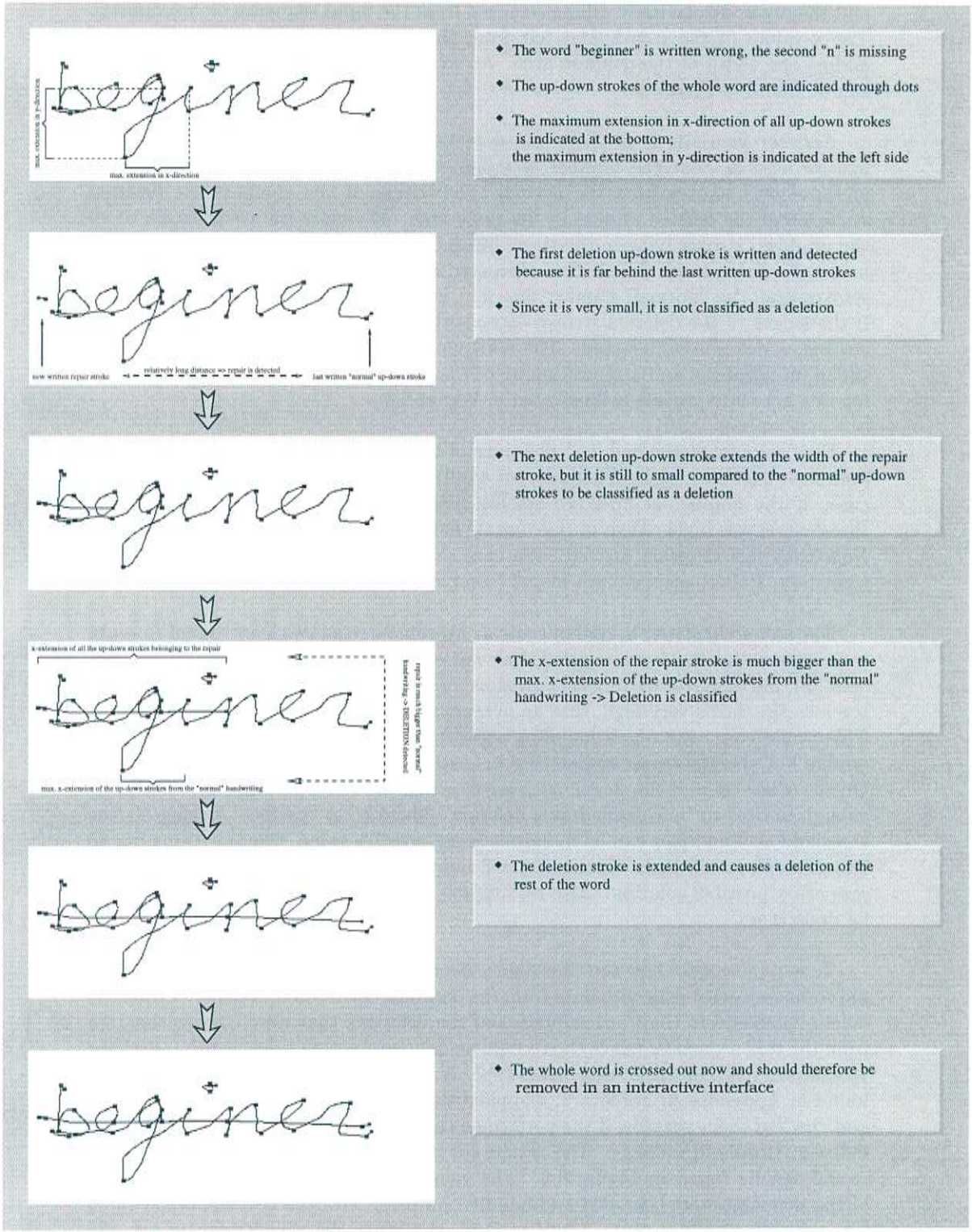

Figure 6.7: Example for the first heuristic to detect deletions.

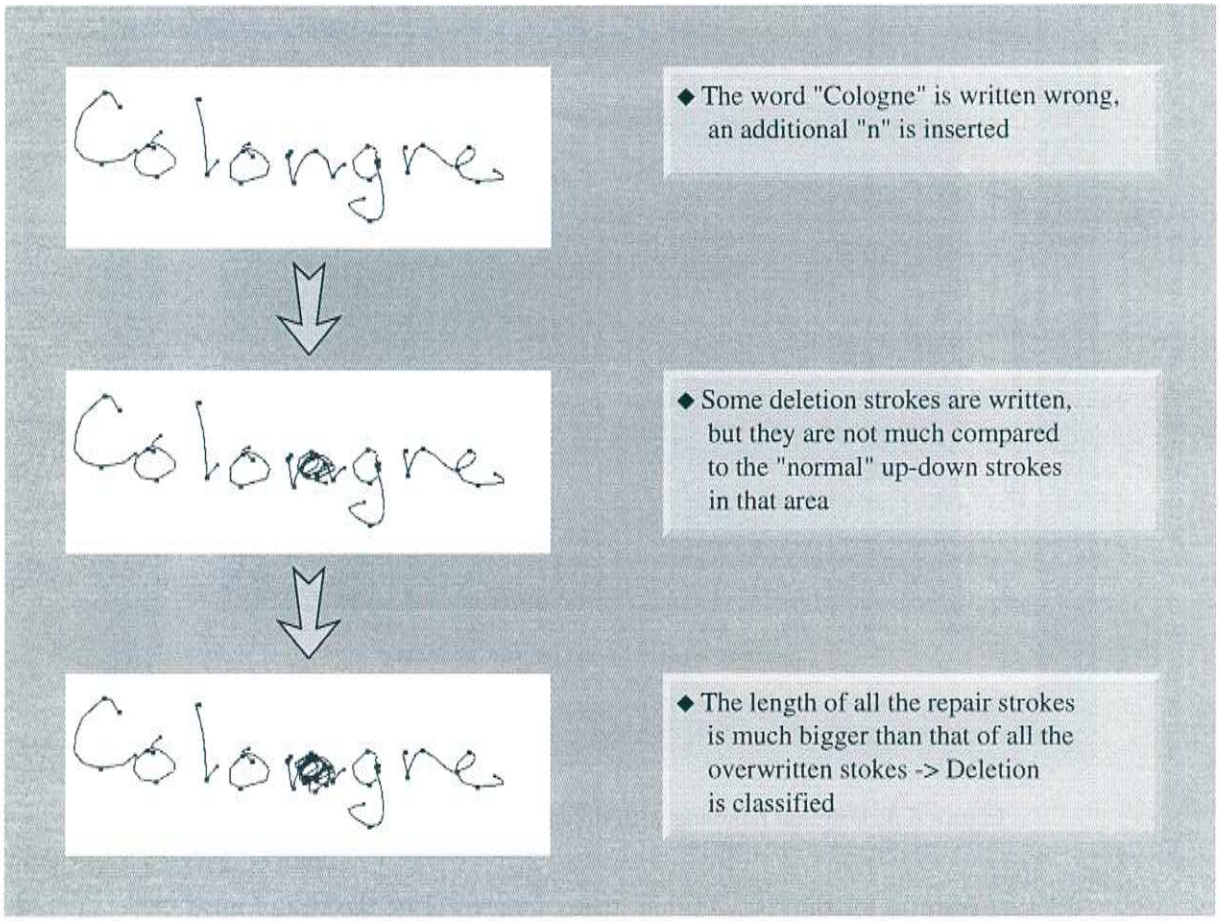

Figure 6.8: Example for the second heuristic to detect deletions.

example, a relatively long t-stroke can be interpreted as a deletion). Therefore a higher reduction of the threshold φ should not be done. It should be noted also, that the repair gestures that were handled with the heuristics here were the ones that occurred most in the database, but in fact there are a few others that are not covered by them. Therefore 100% repair handling in this case can not be done with the used data.

These kinds of heuristics for detecting and handling these special deletion types have some **advantages** over other approaches. First they cover the deletion gestures that appeared the most in our database. They are not restricted to a special shape and offer therefore more freedom to a writer. He is not limited to use some special symbols that he might be unfamiliar with or unused to. Also you should keep in mind, that in on-line handwriting recognition the user usually is sitting in front of the screen or the input device and can see his handwriting and the system reactions immediately. You can profit now from this kind of interactivity by indicating a deletion to a user, after it is detected. If the deleted strokes disappear after scribbling over them, this is kind of similar to the situation where you have written something on a paper with a pencil and delete some parts of it with a rubber. Therefore the approach seems very natural to use. This kind of interactive repair indication will be discussed in more detail in a later chapter, because it also can help to overcome the problem of finding a correct setting of the threshold φ for the second heuristic mentioned above. If you use the detection of some typical

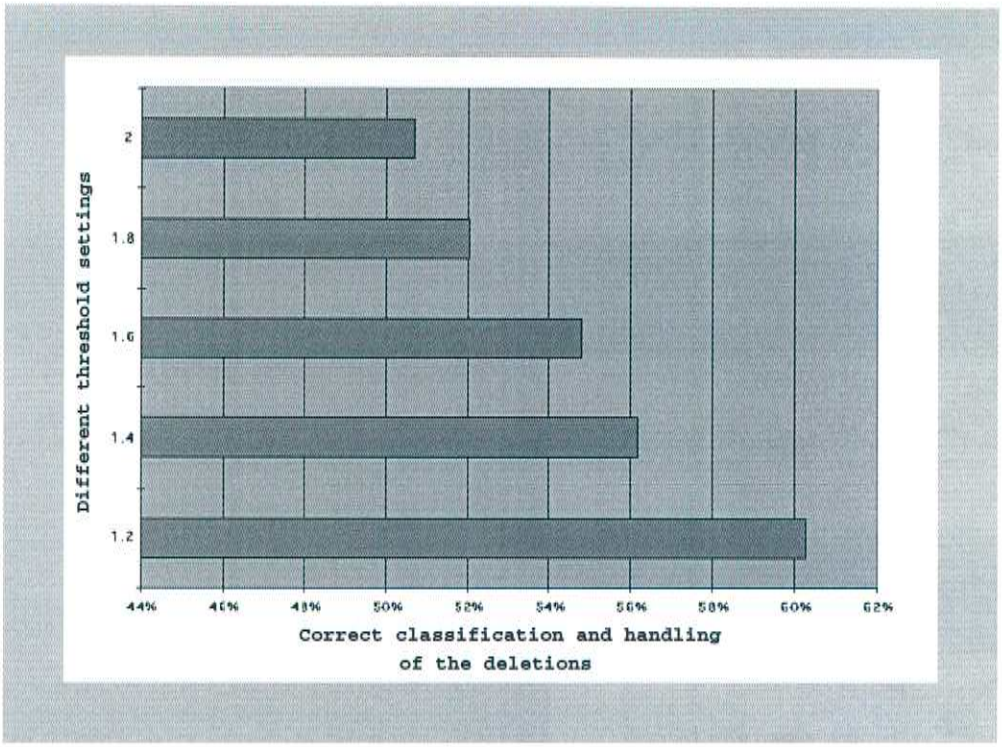

Figure 6.9: Deletion handling with different settings for the threshold φ.

deletion gestures for the classification, there always will be the risk of misclassification. What should be done in such a case? Writing the same gesture again does not seem to be a very promising approach. But if a deletion is not classified in the heuristic above, because the threshold φ is set to high, the user can just continue (or do again) his "crossing out's", and the deletion will be detected.

6.2.2 Overwriting

Another class often found in the database from section 2.1 is **overwriting**. The main occurences of an overwrite are on letter level. It sometimes happened that a user was also overwriting small parts of the preceding or following one so the handwriting that corrects the error fits better into the original signal. In most of the cases the users tried to correct a letter by overwriting it, if it was (a) wrong or (b) had a bad shape that made it look like a different one or not like a known letter at all. Sometimes also more or less than one letter was overwritten. But this two cases appeared very rare, the repair on letter level was by far the one that occurred the most.

What has to be done in the case of overwriting? What kinds of reactions should a repair tool perform to handle an input signal of human handwriting that contains this type of correction? After the detection and correct classification of an overwrite the following steps have to be done:

- first: delete the strokes (or the parts of a stroke) that are supposed to be overwritten, and

- second: insert the strokes that performed the overwrite into the input sequence of coordinates.

In other words: delete the repaired strokes and insert the repair strokes. In the following I will represent different heuristics to handle an overwrite and analyze them with some samples that occurred in the database from section 2.1.

Like discussed in section 6.1.1, repair handling will be done here on the level of up-down strokes. Only whole up-down strokes that are overwritten will be removed from the input signal. So what is a good indication for the situation, that an up-down stroke is overwritten? How can the coordinates that are supposed to "disappear" from the handwritten sequence be found? Since they are overwritten by some repair strokes, a good idea is to compare the **bounding boxes of the up-down strokes** from the "normal" handwriting with the ones of the repair strokes. A bounding box is the smallest square that covers the whole up-down stroke and has horizontal and vertical sides (see the example on the left side of Figure 6.10). If two up-down strokes overlap, i.e., if one overwrites the other, their bounding boxes will have a high overlap, too. An example can be found in the middle of Figure 6.10. Note that this is not true the other way: two bounding boxes can overlap completely, but the coordinates of the corresponding strokes do not touch each other. One such case is illustrated in the right part of Figure 6.10.

One important question that arises with this approach is, what should be considered as a "high" overlap? If an up-down stroke is overwritten only partly by another one, their bounding boxes will also not overlap completely. So when should an up-down stroke, whose bounding box is covered only in parts by the ones of some repair coordinates, be removed? Different ways between the two extremes to delete as much or as few as possible can be thought of. Therefore I took 53 examples of overwrites that I found in the database from part I and tested different heuristics with them. A complete list with the data is shown in Appendix A, Table A.2.

overwriting heuristics

Every heuristic that I will propose here deletes an overwritten stroke, when a "reasonable" overwrite occurs. The only difference is the interpretation of "reasonable". Several methods for checking overwrites and overlaps of bounding boxes are possible. The first heuristics have different meanings for what an overlap is. You can talk of an overlap from one bounding box by another one, if it is completely covered. Two examples are shown in the left of Figure 6.11. In the following I will refer to this view of overlapping as **"total overlap"**. This criterion for an overlap might be a very straight rule for repair handling, because, even if the overlap is not complete, there still might be a very large area of both bounding boxes overlapping. Of course, you can handle this problem by "thresholding" the bounding box, like I will discuss it below. But I will also introduce another criterion here that I refer to as the **"middle criterion"** in the following. Here a bounding box is seen to be overwritten, if the middle of this box lies in the area that is overlapped by the other bounding box. An illustrative example of this rule can be found in Figure 6.11 on the right side.

Like already mentioned, another modification is the introduction of a **threshold β to the bounding box borders** . If only the "pure" bounding boxes are compared, some up-down strokes that should be seen as overwritten might not be covered by the bounding boxes of the repair strokes. Therefore it is a good idea to augment the bounding box borders through adding a fixed threshold β. Two approaches were tried here: first a **symmetric threshold**, i.e., the same threshold value is used for all four bounding box sides, second an **asymmetric threshold**. In this case the left border of the bounding box was set to zero. This was done,

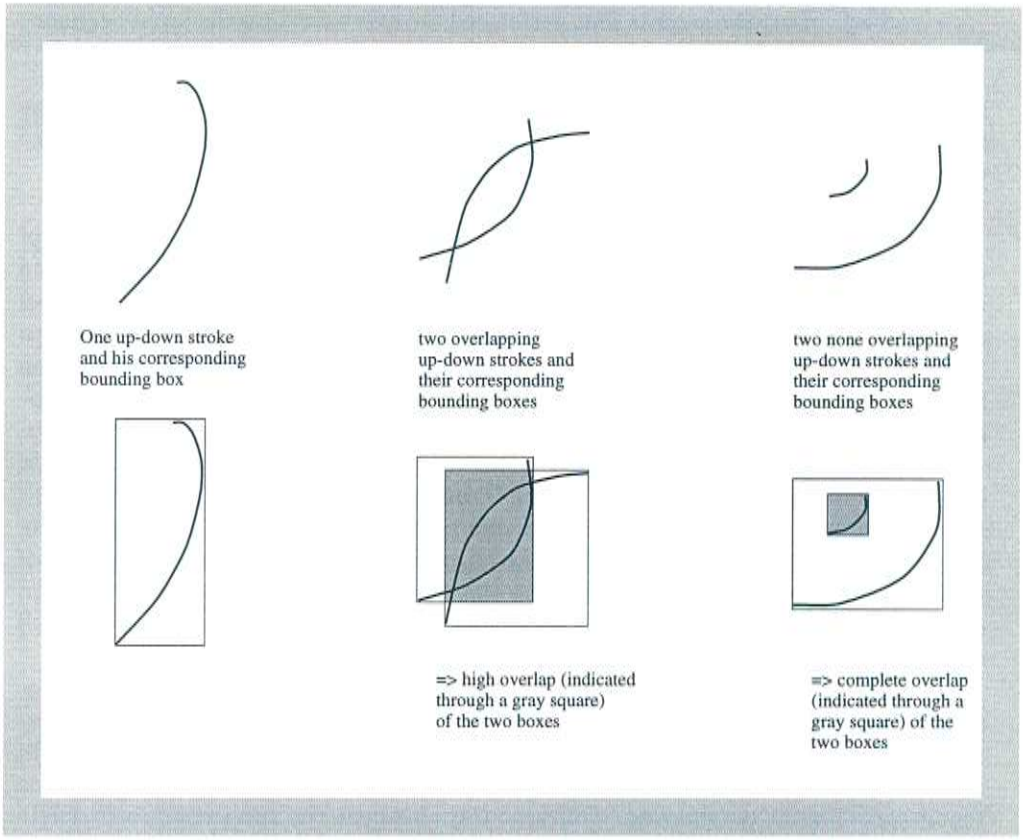

Figure 6.10: Examples for different up-down strokes and their corresponding bounding boxes.

because many humans write with a drift to the right, which results in the risk of a wrong overwriting handling, if the value for the threshold on the left side is set to high. Examples for a symmetric and an asymmetric threshold and different overlap handling can be found in Figure 6.12.

Like already discussed earlier, there are cases, where two up-down strokes that do not touch each other can have overlapping bounding boxes. For this reason I did another modification that I will call **"half bounding boxes"**. Here the bounding boxes of each up-down stroke are divided in the middle of the stroke. The resulting two boxes are thresholded and used now as basis for an overwriting check. An example can be found in the top of Figure 6.13. Overlap only takes place, if both "half boxes" of the overwritten up-down stroke lie in the thresholded "half bounding boxes" of the up-down stroke that is supposed to be an overwrite. Here both criteria, the "total overlap-criterion" and the "middle-criterion" from above can be used. This is illustrated in the middle and the bottom of Figure 6.13.

The approach with the "half bounding boxes" is very extreme in a sense that only up-down strokes that are written very close to the overwriting strokes are removed. The other extreme would be to remove all the up-down strokes that lie in a wide area around the overwriting strokes. This case is illustrated in Figure 6.14. In this example the letter "h" is overwritten by the letter "l". If an overwriting check with the bounding boxes will be applied with the methods described above, one up-down stroke that should obviously be removed will stay in the repaired

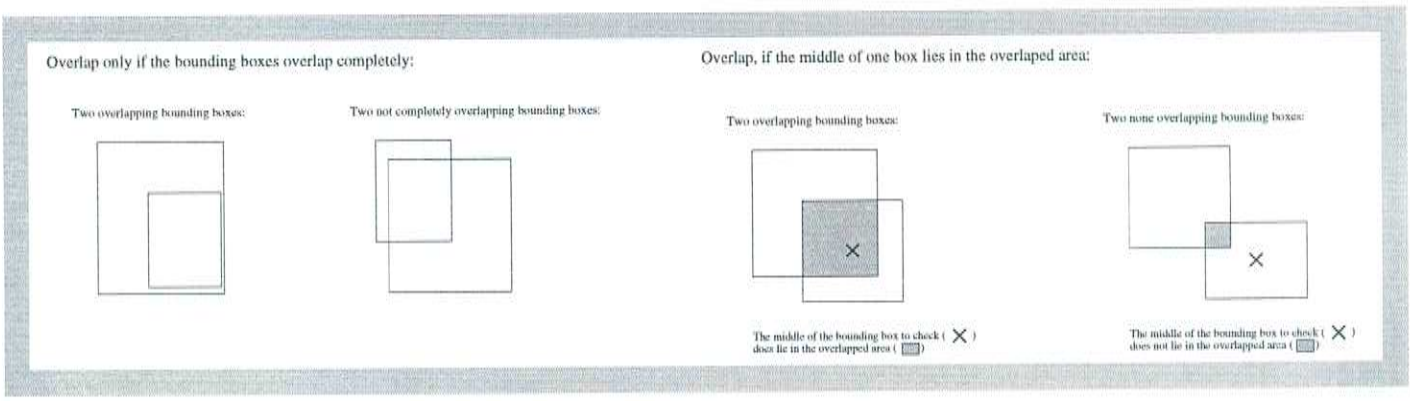

Figure 6.11: Examples for two different definitions of what is seen as an overlap: a "total overlap" (shown left) and an overlap according to the "middle criterion" (shown right).

sequence. Therefore another way to handle overwriting is to **delete every up-down stroke** that lies in about the same area according to the x-coordinates like the repair strokes, if the bounding box of at least one up-down stroke of the original sequence is covered by the bounding boxes of the repair strokes.

By combining some of the heuristics introduced above I got **six different methods to handle an overwriting**.

- **Version 1:**
  compare symmetric thresholded "half bounding boxes" with the "middle-criterion"
- **Version 2:**
  compare symmetric thresholded "half bounding boxes" with the "total overlap-criterion"
- **Version 3:**
  compare symmetric thresholded bounding boxes with the "total overlap-criterion"
- **Version 4:**
  compare asymmetric thresholded bounding boxes with the "middle-criterion"
- **Version 5:**
  compare symmetric thresholded bounding boxes with the "middle-criterion"
- **Version 6:**
  delete all the up-down strokes in the overwriting area, if at least one up-down stroke is overwritten according to the criterion in Version 4

Note, that other combinations of the different methods can be made, but these are the ones that seemed to make the most sense to me. This list is in general ordered in a way that increasing numbers indicate a higher probability for the deletion of an overwritten stroke. This means that Version 6 should delete the most, while Version 1 should delete the fewest coordinates from the original written word compared to all the other heuristics.

To evaluate the different versions I took the samples with overwrites that I mentioned earlier from the database and applied different tests on them.

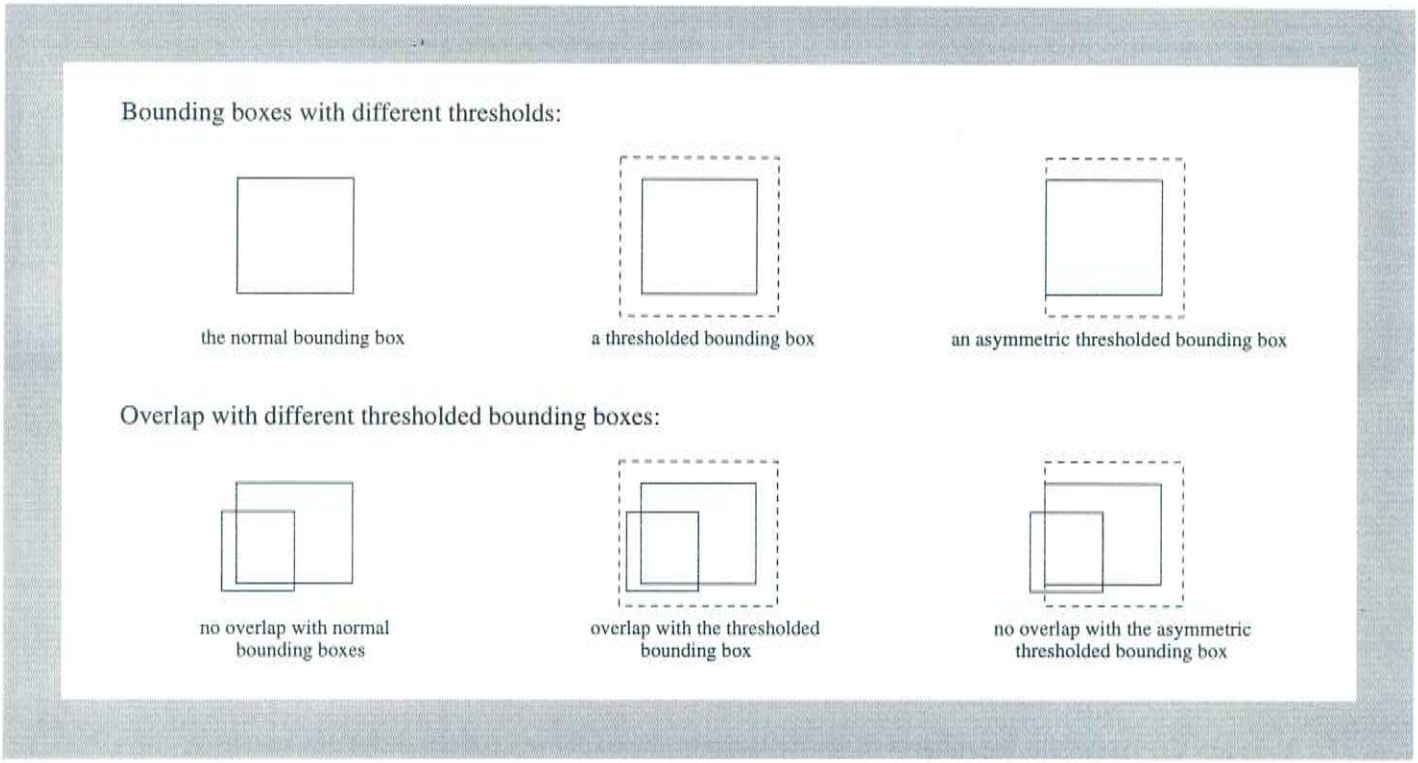

Figure 6.12: Examples for a symmetric and an asymmetric threshold added to a bounding box.

First I checked the overwriting handling of the different versions visually. The results can be found in Figure 6.15. On the y axis the version that was taken to handle the overwrite is indicated. v1, v2, ..., v6 refer to Version 1, Version 2, ..., Version 6, respectively. The values at the x axis indicate correct repair handling. Since I classified an overwriting as handled correct, only if it was 100% what you as a human would expect, the numbers of correct results are not every high. But it should be noted, that this does not have to result in a low recognition accuracy, too. If one up-down stroke is deleted to much or one that should be deleted is not, the recognition result can still be correct. I will show some word accuracy results later, after I have introduced some other heuristics needed to handle the repair type overwriting. It turns out, that the methods designed to delete the most parts of the overwritten word do the best repair handling. The tests with the "half bounding boxes" failed most of the time.

After the deletion of the overwritten up-down strokes from the input signal with the heuristics described above, the repair strokes, i.e., the strokes that do the overwrite, have to be inserted into the remaining sequence of coordinates from the input signal. This should not be a big problem, if Version 6 from above is used for the overwriting check. If every up-down stroke in the overwritten area is deleted, the repair strokes only have to be inserted between the remaining parts to the left and the right. But insertion is of course a big problem, if some up-down strokes of the original signal stay in the overwriting area, even after the deletion of the overwritten ones. For example, how should the remaining up-down stroke from the letter "h" in Figure 6.14 be inserted into the repaired input signal?

insertion heuristics

Therefore I will also introduce and test some heuristics to handle an insertion of

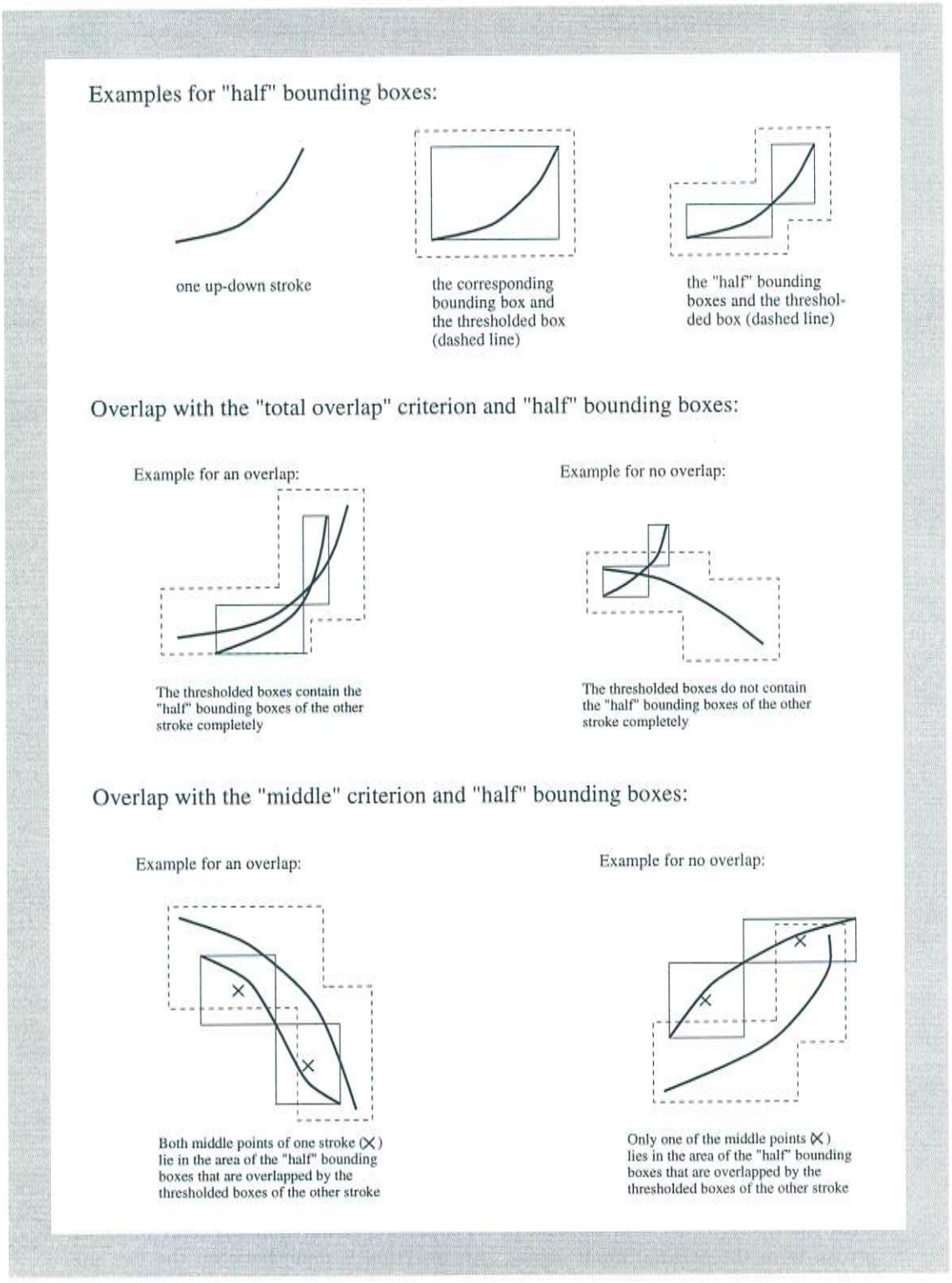

Figure 6.13: Examples for thresholded "half bounding boxes" and their use for overlap checking.

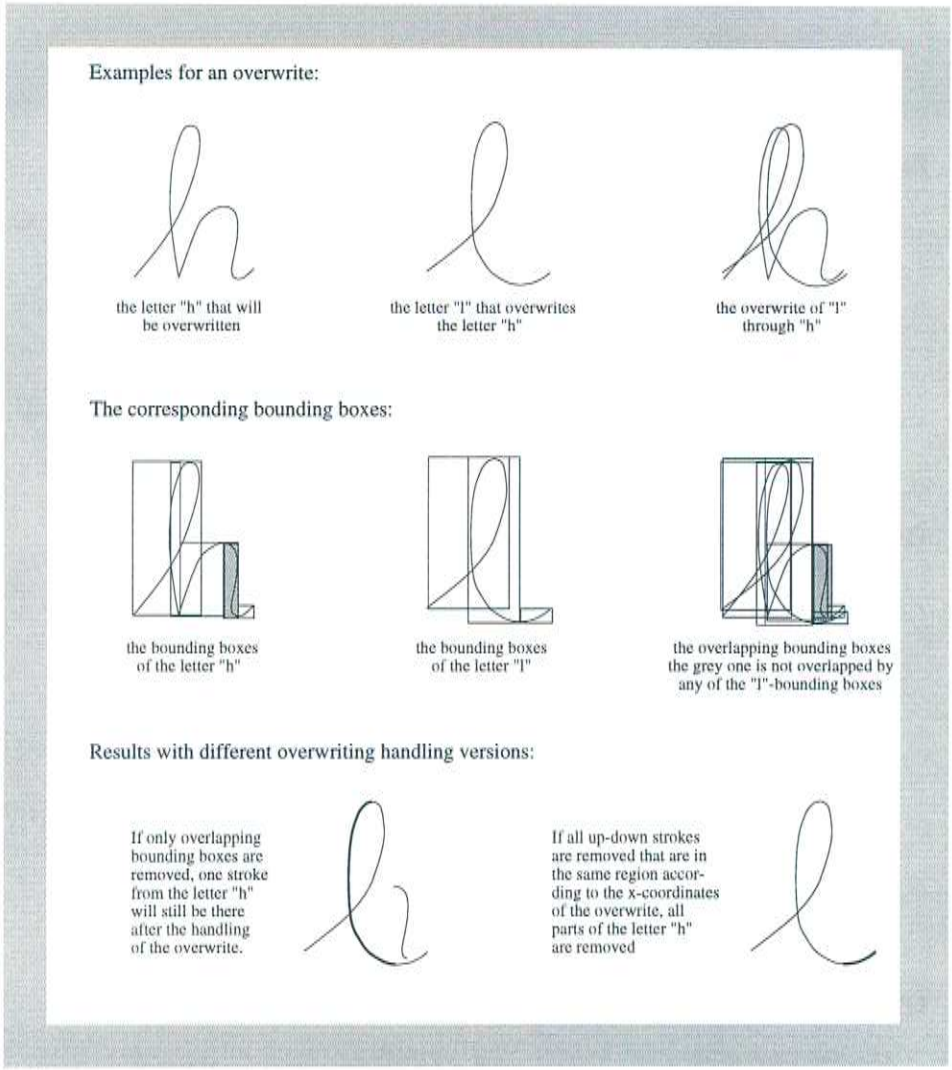

Figure 6.14: Example for the approach where all up-down strokes in the overwritten area are deleted.

a stroke or some up-down strokes into a sequence of coordinates from the original input signal. Examples for all the heuristics I will use can be found in Figure 6.16.

The first one is illustrated at the top of this Figure under "Version 1 & 2". The correct insertion position is calculated as follow: for every pair of successing up-down strokes a measure is calculated. This value should be lower than another one, if an insertion of the stroke at this position seems more reasonable. In this version the distance in direction of the x-coordinates between the end of the previous up-down stroke and the begin of the stroke to insert is calculated. The same is done with the distance between the end of the insertion stroke and the next up-down stroke from the original input signal. An insertion is done between the two up-down strokes for which the sum of this two distances is a minimum over all possible insertion borders in the input signal.

The second heuristic, shown in Figure 6.16 under "Version 3 & 4", does the same like the first one. The only difference is, that not only the distances in direction

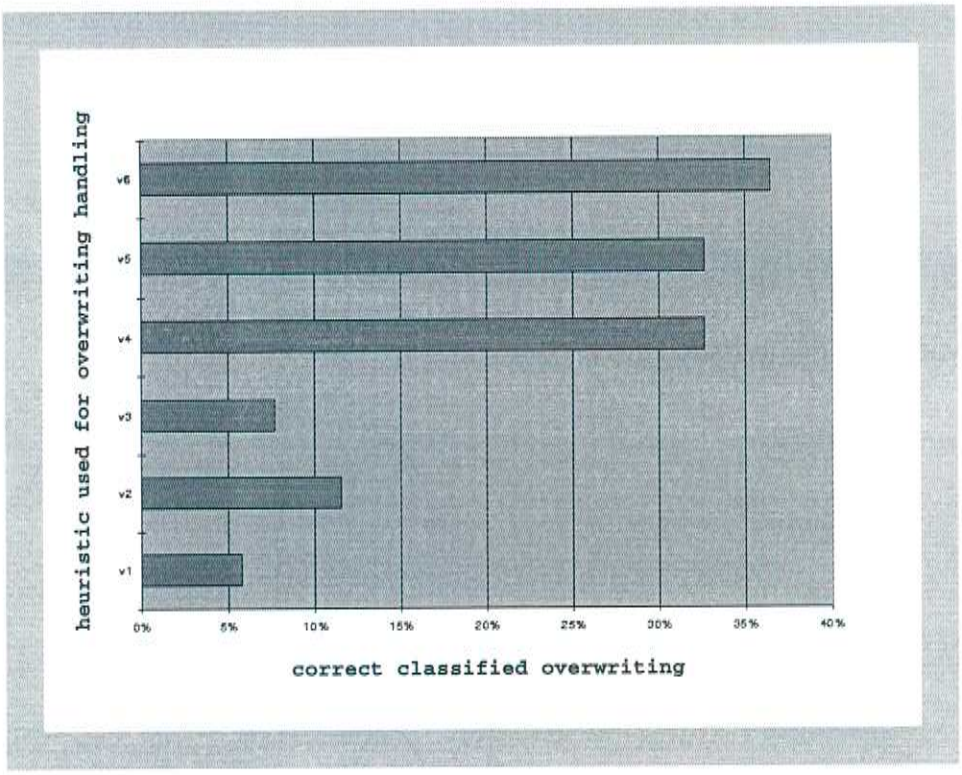

Figure 6.15: Evaluation of the different versions used to handle an overwrite.

of the x-, but also of the y-coordinates are calculated. The sum of all these four distances has to be minimized to find the best insertion position.

An example for the next case is shown in Figure 6.16 under "Version 5 & 6". Here the bounding boxes of the repair stroke and every up-down stroke are compared. Insertion is done after the up-down stroke with whichs bounding box the highest overlap exists. For this purpose the size of the area that overlaps is calculated. If no stroke overlaps at all, the first heuristic, described above, is used.

The next method, shown in Figure 6.16 under "Version 7 & 8", is similar to the first one. But here the distances between the middle x-values of every up-down stroke and the stroke to insert are calculated. The insertion position is the one between the two up-down strokes with a minimum value for the sum of these distances.

The last heuristic is illustrated in Figure 6.16, too, under "Version 9 & 10". Here the distance between the end of a previous up-down stroke and the begin of the stroke to insert and also the one between the end of this stroke and the following up-down stroke in the input sequence is calculated. The minimum of these two values is taken and compared to the respective distance values of the other possible insertion positions. Insertion is done at the position with the lowest value.

Another modification can be done with all the heuristics from above. You can insert a complete repair stroke at once or you can insert every up-down stroke of it step by step with one of the heuristics from above. This is similar to allow "breaking" of a repair stroke or not. An example for this is illustrated in Figure 6.17. Here it seems more reasonable to allow up-down strokes from the normal

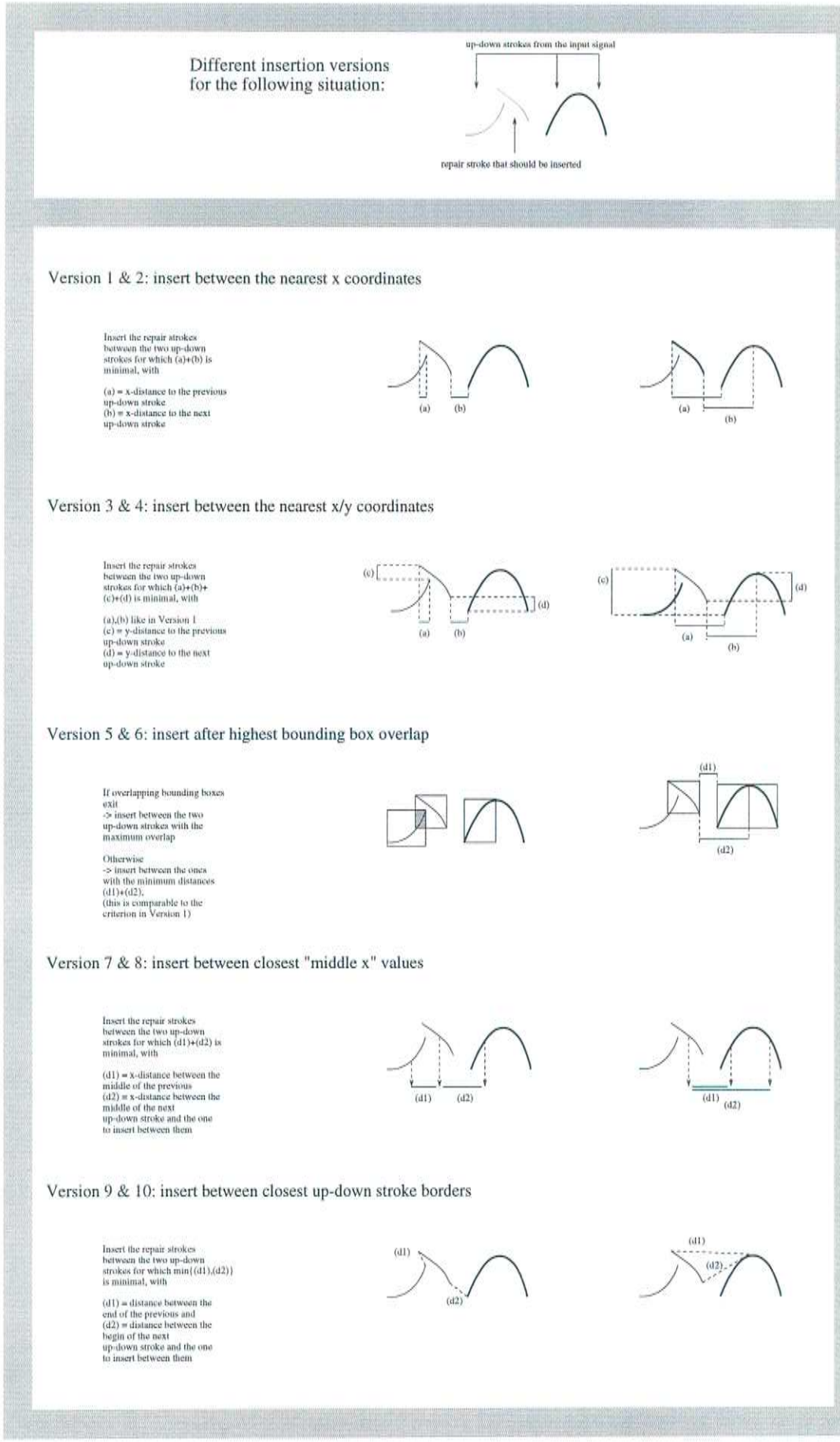

Figure 6.16: Examples for the different heuristics used to perform an insertion.

handwriting to be inserted within a repair stroke. But there are also cases, where it would be better not to allow such a "breaking" of one stroke.

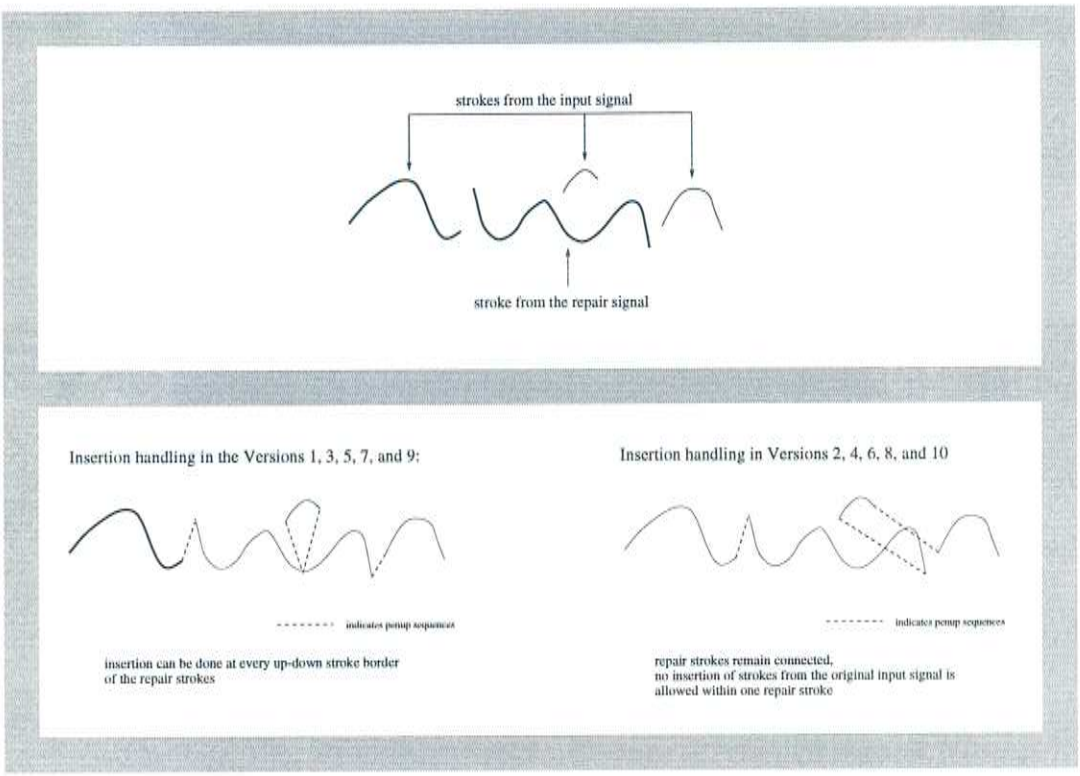

Figure 6.17: Example with allowed and not allowed "breaking" of repair strokes.

Therefore in total there are ten heuristics that I evaluated with the data already used to test the different cases of overwriting handling. The results can be found in Figure 6.18. On the y axis the different insertion versions can be found. v1, v3, v5, v7, and v9 refers to the versions described in Figure 6.16 with the same index. Here every up-down stroke of the whole stroke to insert is handled separately. The versions v2, v4, ..., v10 correspond to the preceding versions with the only difference, that here the whole repair stroke is inserted at once. The x axis values indicate how often an insertion is handled correctly. The cases where the whole stroke is inserted at once usually received better results than their corresponding counterparts. The best results were achieved with the versions number 2, 4, and 8.

After applying the complete repair handling to this data, i.e., after deleting the overwritten strokes and inserting the overwriting ones, I checked them with the NPen^{++}recognizer. The word accuracy, that is the relative number of correct recognized words, was calculated on the test set with the use of every bounding box check and every insertion heuristic discussed above.

The results can be found in Figure 6.19. In the upper diagram the word accuracy is indicated for every combination of an overwrite handling heuristic with an insertion heuristic. In the lower left of this Figure, a "landscape" is drawn by connecting the values of neighboring versions. Note, that in principle it makes no sense to connect the word accuracy values of two different heuristics, since there is no heuristic "between" two existing ones and therefore no interpolation should be done. But if you look at this diagram from the top, you get a "map", that is shown

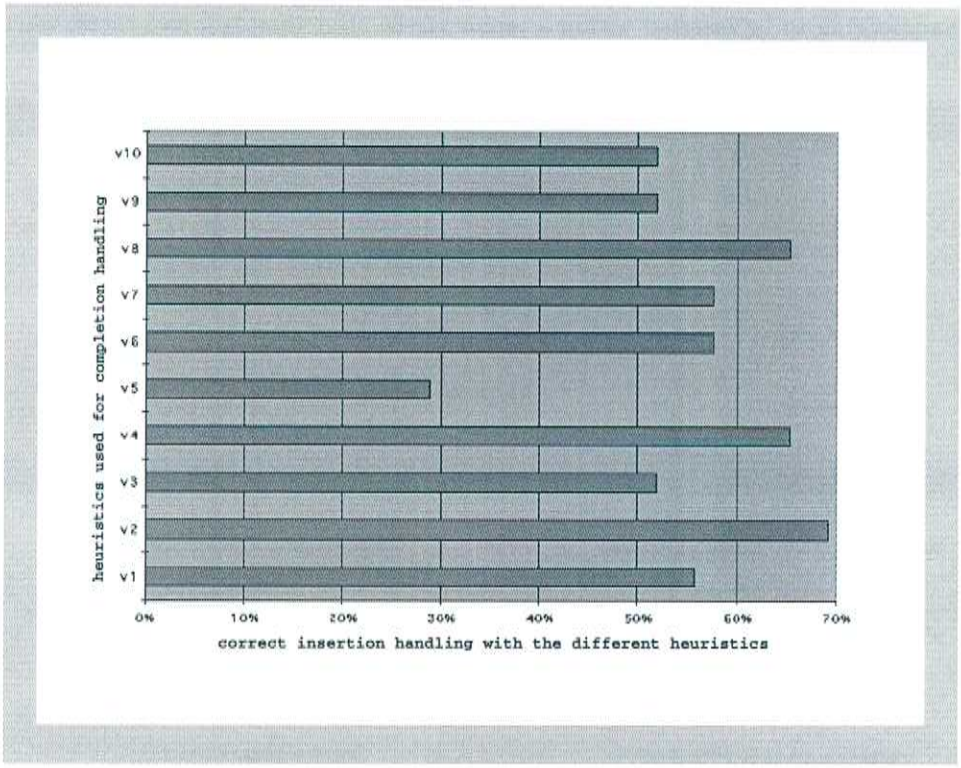

Figure 6.18: Evaluation of the different heuristics used to handle an insertion.

on the lower right of this figure. Since there are only discrete values in the original diagram that correspond to the different heuristics, only the grid points indicate "real" values. All the others in between are and cannot be based on a recognition. But I included these two diagrams here, since you can see the heuristics that achieve the best recognition results much clearer and easier this way. It turns out, that the approaches that had the best repair handling are also the ones that received the best recognition results.

It should be noted, that the goal of these tests was to evaluate the different heuristics according to which parameters are the best and which ones are most suitable to use. The purpose was not to achieve the best possible recognition results. Nevertheless, the best result with the use of the overwriting handling heuristics received here is 31.4%, which is an improvement over the "regular" recognition without any repair handling, that achieved a word accuracy of 25%.

6.2.3 Completion

The last repair type of the classification I proposed in part I is **completion**. It appeared, that this case happened very seldom in the database. Nevertheless it did appear. Usually it occurred in a sense of "closing" a letter, e.g., an "a" that looked like a "u" was "closed" by adding an additional stroke at the top.

Once you have handled the overwriting case, it should be possible to deal with the repair type completion in a similar way. The reason for this is, that both cases are identical after you removed the overwritten up-down strokes, if an overwrite was done. In completion usually no strokes or parts of them should be necessary to delete, only an insertion has to be done.

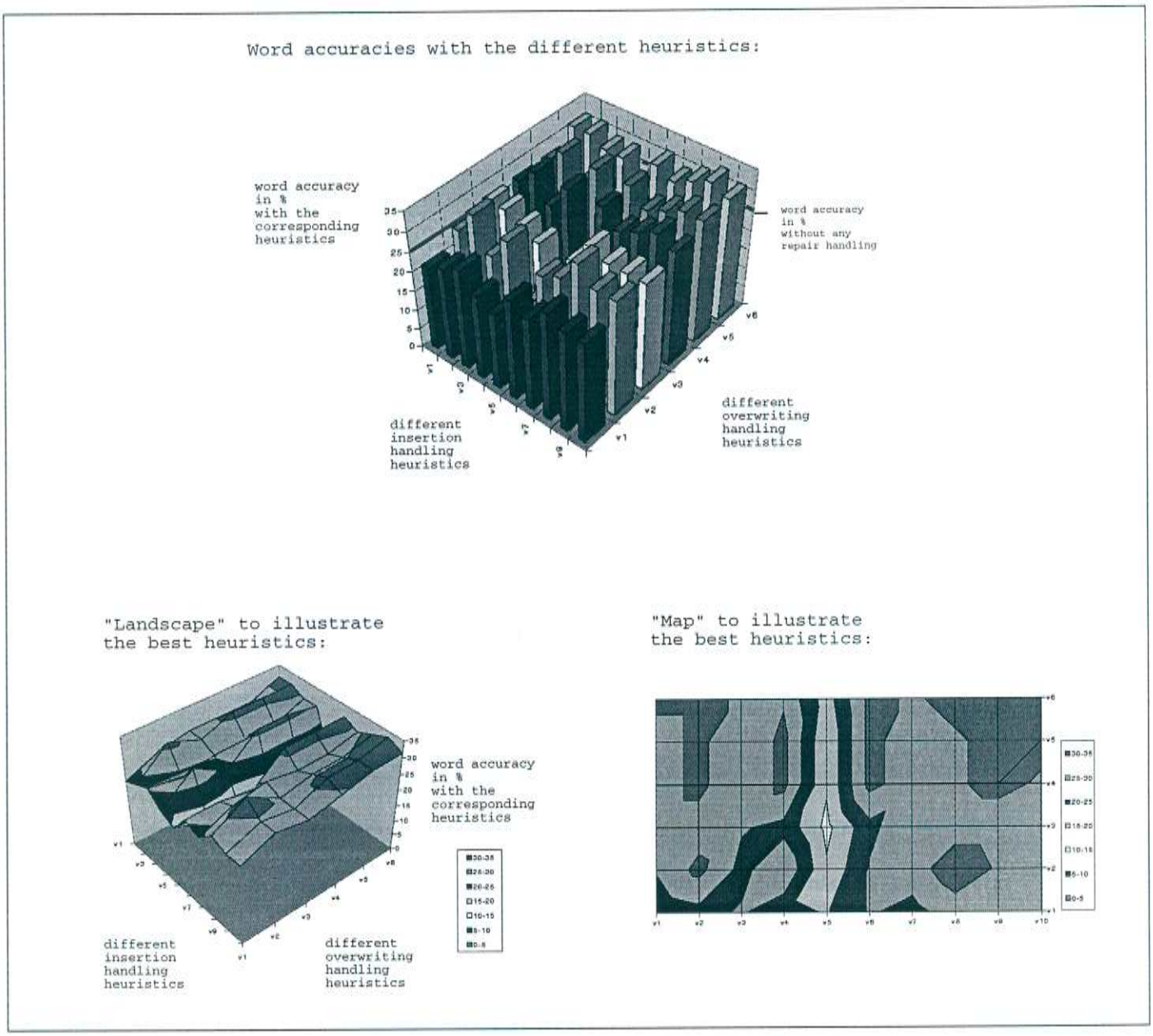

Figure 6.19: Word accuracy for the overwriting examples with different overwriting and insertion handling heuristics.

Therefore I tested the insertion heuristics introduce in the last section on 28 samples I found in the database. The list with the words tested here can be found in Appendix A, Table A.3. The results of the visual check, if repair handling was done in a reasonable way, can be found in Figure 6.20. On the y axis the used insertion versions are indicated through v1, v2, ..., v10 like in Figure 6.18. The x axis value indicates the number of samples handled correctly. The best results are achieved with version v2 and v4. These two cases were also under the best ones in the case of overwriting. But even with them the number of correct handled input sequences is not very high. One reason for this is, that in some cases a "perfect" insertion can not be done, because the additional stroke does not fit exactly in the previously written word.

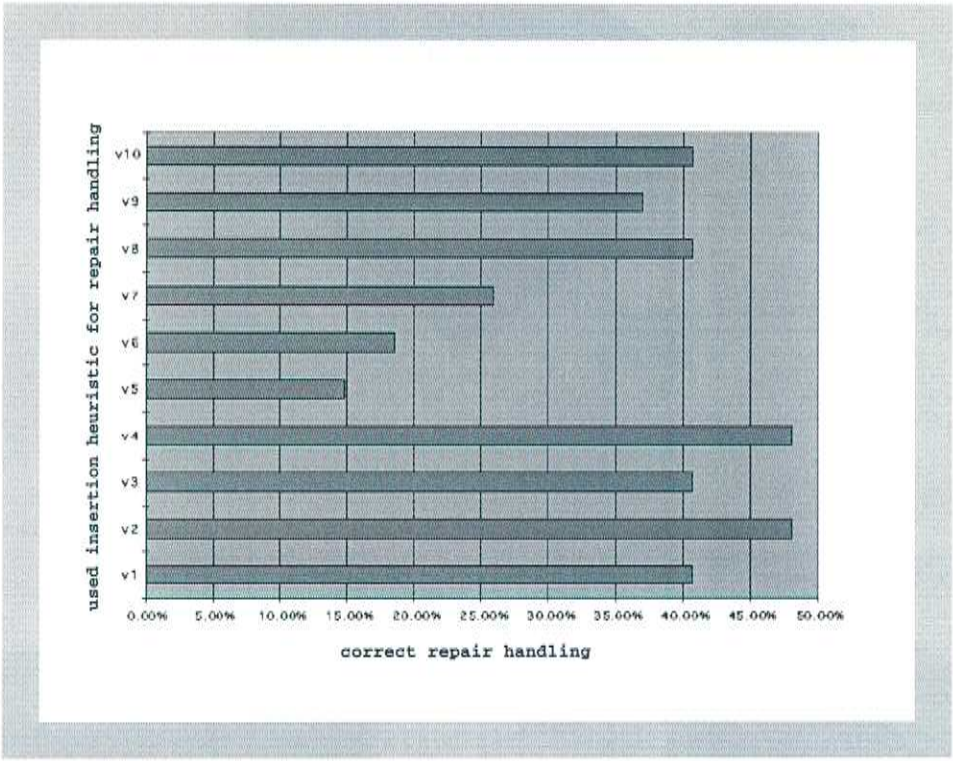

Figure 6.20: Repair handling with different insertion heuristics.

On the other hand, since completions are usually only very short strokes, maybe it is not even necessary to insert them in a "perfect" way. Even with a "nearly" correct insertion the correct recognition result might be achieved. I tested the sequences with all combinations of overwriting and completion heuristics introduced in the last section. I also did the overwriting check before insertion handling, even if it should not be necessary in the case of completion. But if you do this additional step, there is no need for a separation between this two cases in the implementation of a system that contains both of them. I will discuss the aspects of intergrating the proposed repair handling algorithms of all repair types together in one tool in the next section. Regarding the word accuracy results for the completion type, those are shown in Figure 6.21, it turns out, that the versions that did the best repair handling in the overwriting case also achieved the best results with the completion examples. The diagrams at the bottom of this Figure have to be interpreted like the corresponding ones described for Figure 6.19.

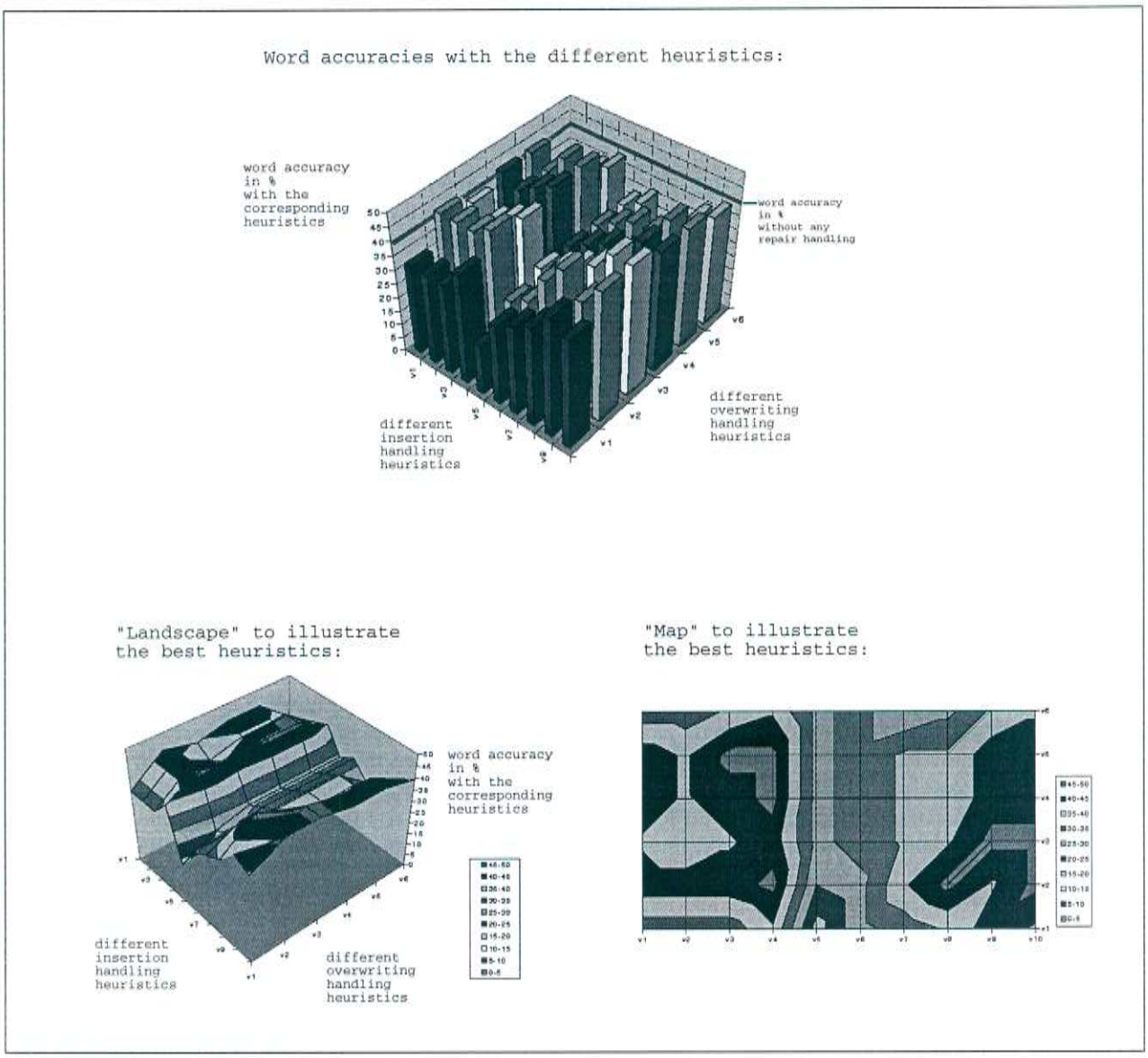

Figure 6.21: Word accuracy for the completion examples with different overwriting and insertion handling heuristics.

6.3 Handling of All Repair Types

In the last section every repair class was checked separately. Heuristics have been introduced that handle each correction type for its own. The topic of this chapter is how these different repair handling heuristics can be combined in a single tool and applied all together to any kind of input.

6.3.1 Coordination of the Heuristics for Different Repair Types

The combination and synchronization between the proposed heuristics in the last section for handling the different repair types is very obvious and easy. A schema of **the whole process of applying all the methods for every single repair type in one tool** is shown in Figure 6.22. The diagram starts at the top left corner of the figure. First a deletion check is done. The repair data is found through the dynamic threshold method, introduced in chapter 6.1.2, or by the heuristic that detects a repair at the end of a word. If a deletion is classified, i.e., if the length of the up-down strokes that are classified as repair is much bigger than the length of the up-down strokes that are overwritten by them or their size is much larger, the repair is handled by deleting both, the repair and the repaired strokes. After that the x-values of the coordinates left from the position of the "scratch out" will be repositioned. Nothing else has to be done. If no deletion is classified, an overwriting is checked. This means that the bounding boxes of the repair strokes are compared with the bounding boxes of the up-down strokes in about the same area. If an overwrite according to one of the proposed heuristics exists, the overwritten up-down strokes are removed. After this removing of parts of some strokes the overwriting case reduces to a completion case, i.e., all that has to be done now is to insert the repair strokes in the right way into the remaining input signal. That is the same situation that happens in a completion case, i.e., when no overwritten strokes are detected. After applying one of the heuristics introduced in the last section the repair process is finished. The repair signal can be send to the recognizer now.

An **example for the case of a deletion** is shown in Figure 6.23. After the classification of the repair strokes and the up-down strokes from the "normal" handwriting in this area, a deletion check is done. Since the length of the repair strokes is much higher than the length of the up-down strokes that are overwritten, the result of this check is positive. Therefore no further tests have to be done. The repair strokes and the deleted parts of the handwriting are removed from the original input signal. After a repositioning of the x-coordinates of the strokes that are on the left side of the deletion area a recognition can be done.

The case of an overwriting is illustrated in Figure 6.24. In this example the letter "a" of the word "man" is overwritten by the letter "e" to change the word from "man" into "men". Since the length of both up-down stroke pairs, the overwritten and the repair strokes, have about the same length and size, no deletion is detected. Therefore an overwriting check is done. Because the bounding boxes of the letter "e" overlap the ones of the letter "a" completely, this character is deleted, i.e. the referring up-down strokes are removed from the input signal. After that the repair strokes, i.e., the strokes that represent the letter "e", are inserted with one of the insertion heuristics into the gap in the remaining input signal. Now a "regular" recognition can be done on the repaired sequence of coordinates.

An example for the last repair type, **completion**, can be found in Figure 6.25. Here only the left part of the letter "v" in the word "lover" was written and the user corrected it by adding two additional strokes to complete this letter. Both

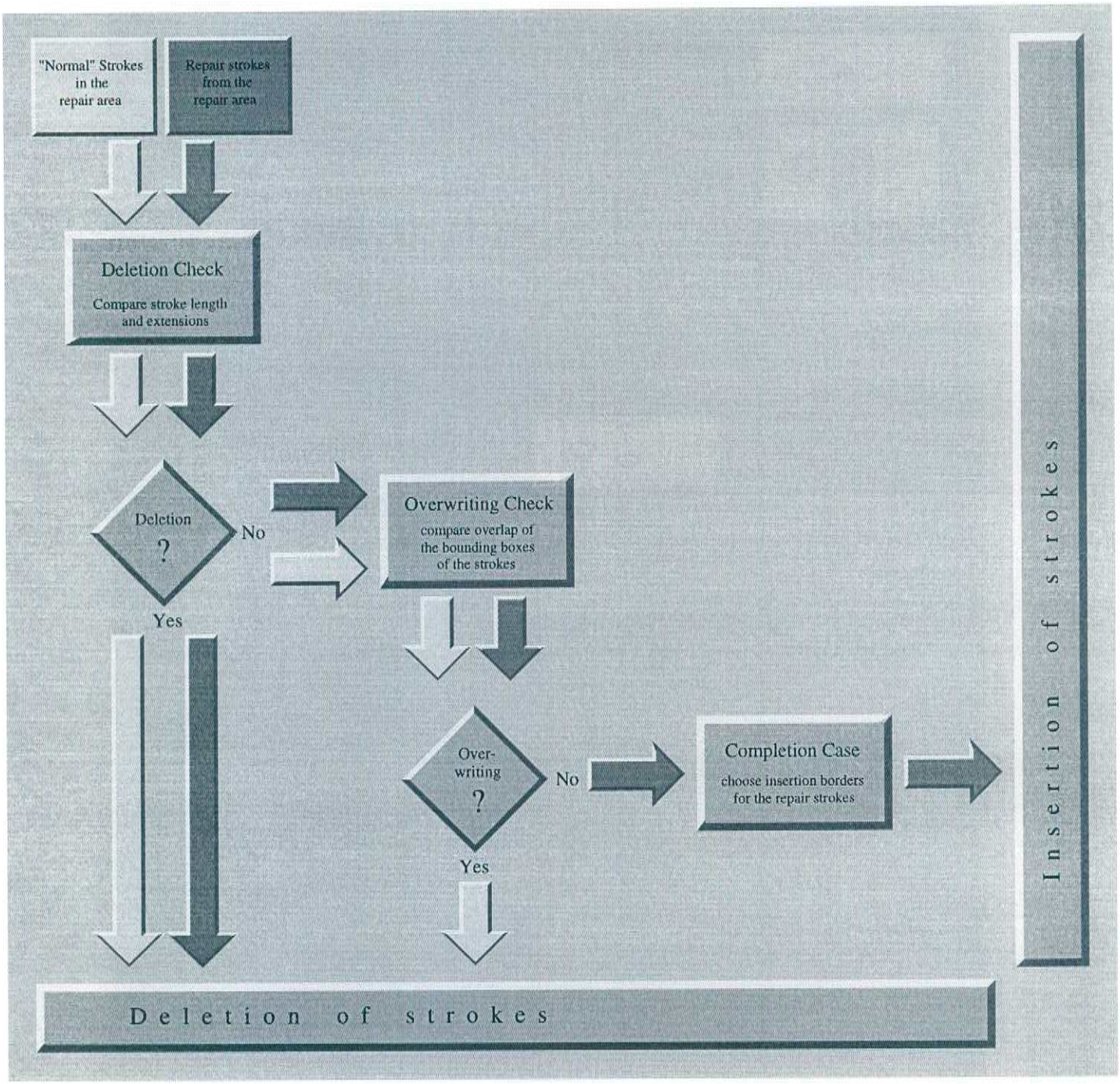

Figure 6.22: Illustration how the different repair handling mechanisms work together.

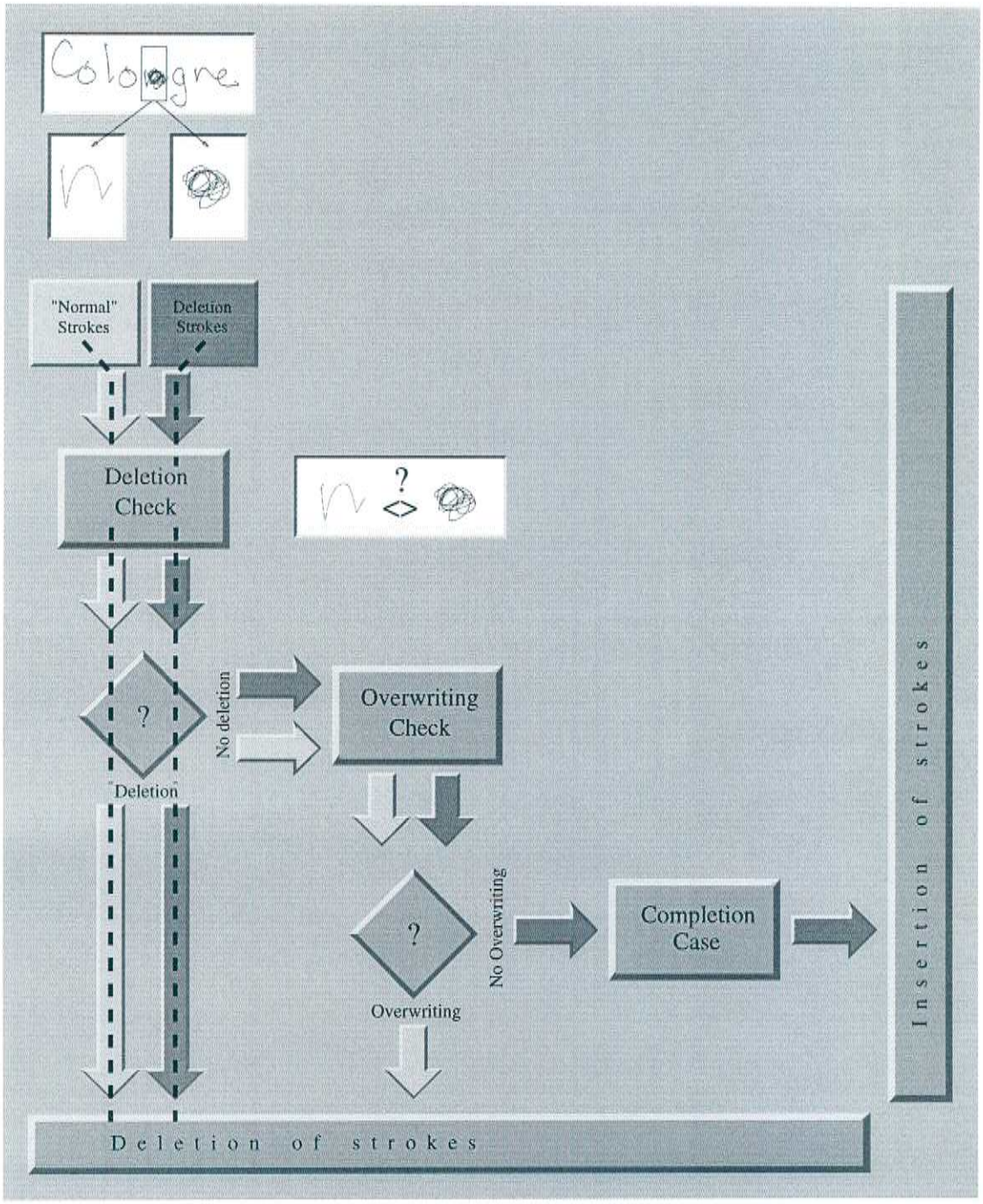

Figure 6.23: Example for the deletion case.

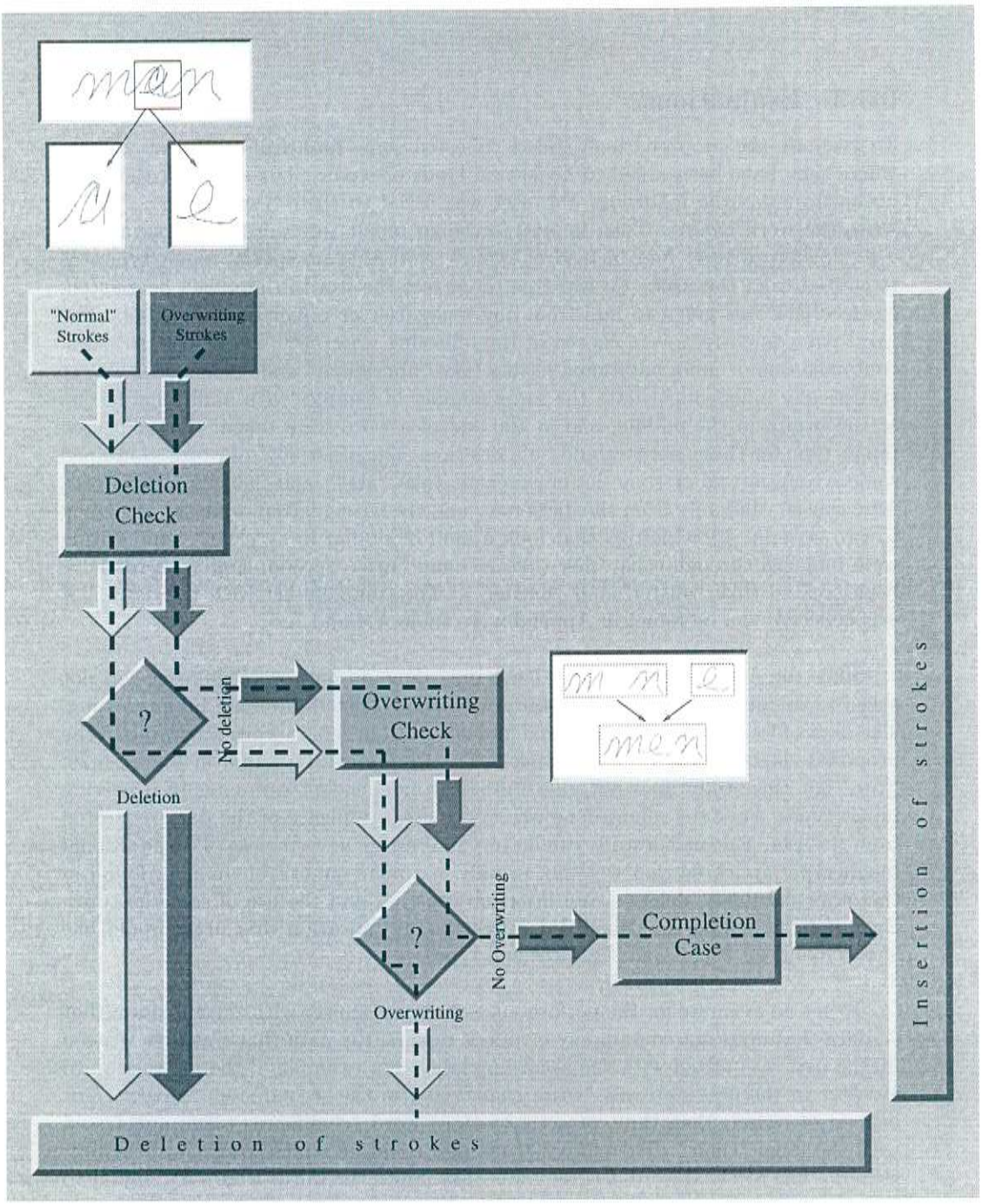

Figure 6.24: Example for the overwriting case.

parts of the repair have about the same size, therefore the deletion check is negative. Also no bounding boxes of the repair strokes have an overlap that is high enough to delete one of the overwritten up-down strokes. Therefore a completion has to be done. After inserting the repair strokes into the original signal the repair handling is finished.

6.3.2 Evaluations

evaluation

To evaluate the proposed heuristics I collected some data from different writers, where they have been asked to do several kinds of repair. The setup for the **data collection** was the following: the user was asked to write a word, e.g.: "please write the word *chair*." Then he was forced to repair this word in a special way, e.g.: "please correct: *hair* instead of *chair*." Note that no special repair type was requested from the user. He was free to correct the word in any way he wanted. No feedback was given to him from any recognizer or repair handling tool. The word pairs to correct were chosen in the following way: 50%, i.e., 200 word pairs, contained one or more additional letters (like "aboriginal" and "original"). To get statistically independent data the same number of samples with additional letters at the begin, in the middle, and at the end of a word were taken. The expected repair type for these words would be a deletion. The other 50% of words contained some wrong letters or some letter reversals (like "abel" and "able"). The words were chosen randomly from the dictionary and the wrong letters were set according to two criteria. First, letters that look a kind of similar in a "clean" handwriting were replaced through each other (like "n" and "m"). Second, similar words have been created (like "mister" and "master"). The complete list with the data that was collected can be found in Appendix A, Table A.4 and A.5.

Since the data analysis in part I was only done on data were the user was not asked to do any repair, a similar study with the collected data can be interesting. The data of four writers was analyzed and classified in repair types according to the proposed classification in the first part. The results can be found in Figure 6.26. Note that this study does not give statistical independent estimates of occurring repair types, since the repair done was under great influence of the data requested from the user. But such a study can be very interesting nevertheless. Two additional repair types appeared that were not or only in a none remarkable number found in the other database: deletion and insertion together and the use of some gestures. Even if their number was small this might be an indication that users would like and demand for such repair gestures and types.

To get an estimate for the performance of the recognizer with "clean" data, that is, data that does not contain any errors or repairs, the data that was first written by the user was tested with the according label. For example, if the instruction to a writer in the first step was "write *chair*", and in the second step "correct: *hair* instead of *chair*", then the (correctly written) word "chair" was taken and tested with the label "chair". Therefore 200 samples could be tested for each of the four writers. The received word accuracy was 88%. Since the words were very similar to the repaired ones, this value is a good benchmark for the evaluation of the repair tools. It should be noted that this recognition is very hard since all the words in the test set have counterparts in the dictionary that are very similar (i.e., the ones that were asked for from the user in the second data collection step).

Also the repaired data, i.e., the data that was collected in the correction step of the data collection, was tested with the recognizer. Since the NPen^{++}system does

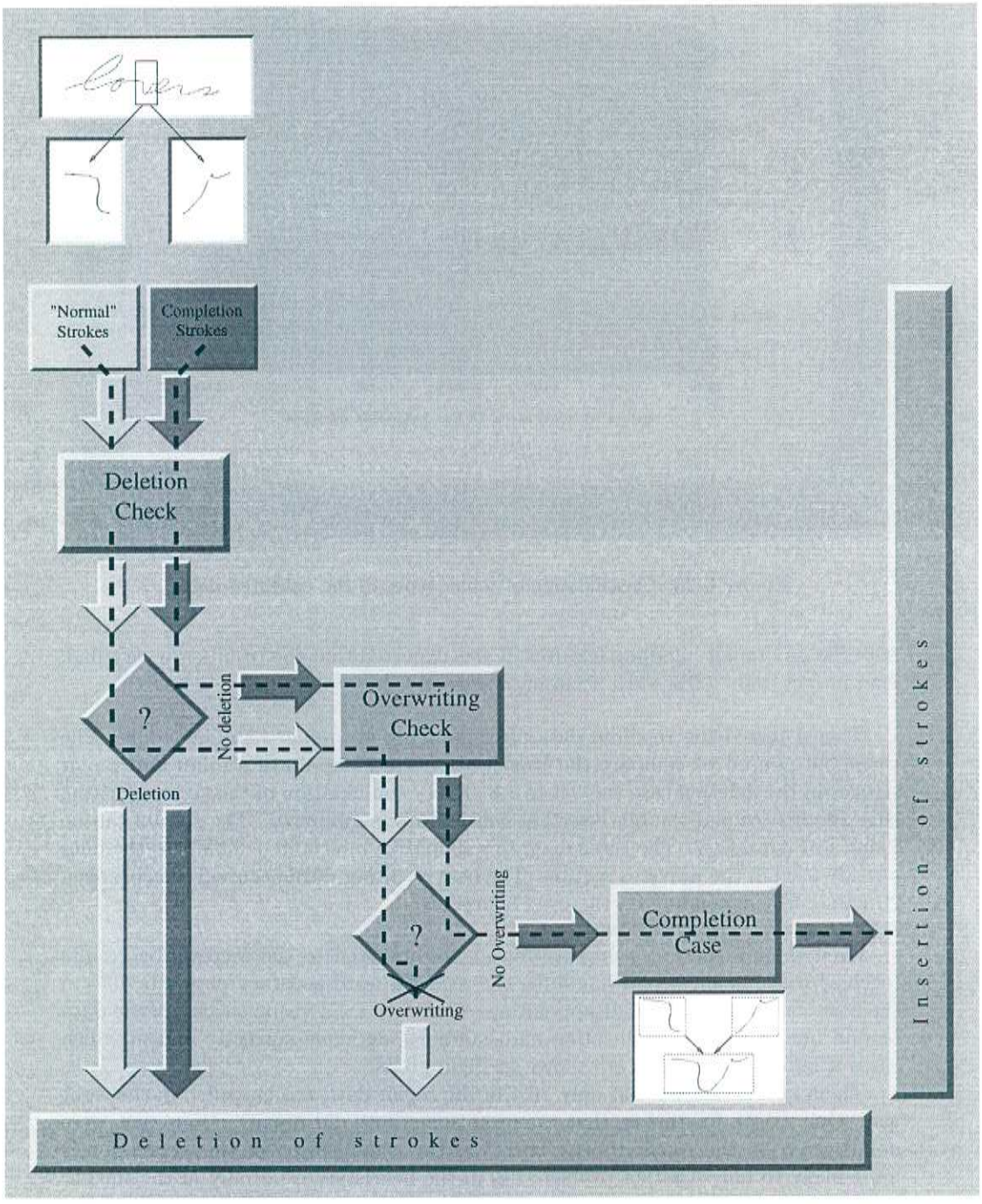

Figure 6.25: Example for the completion case.

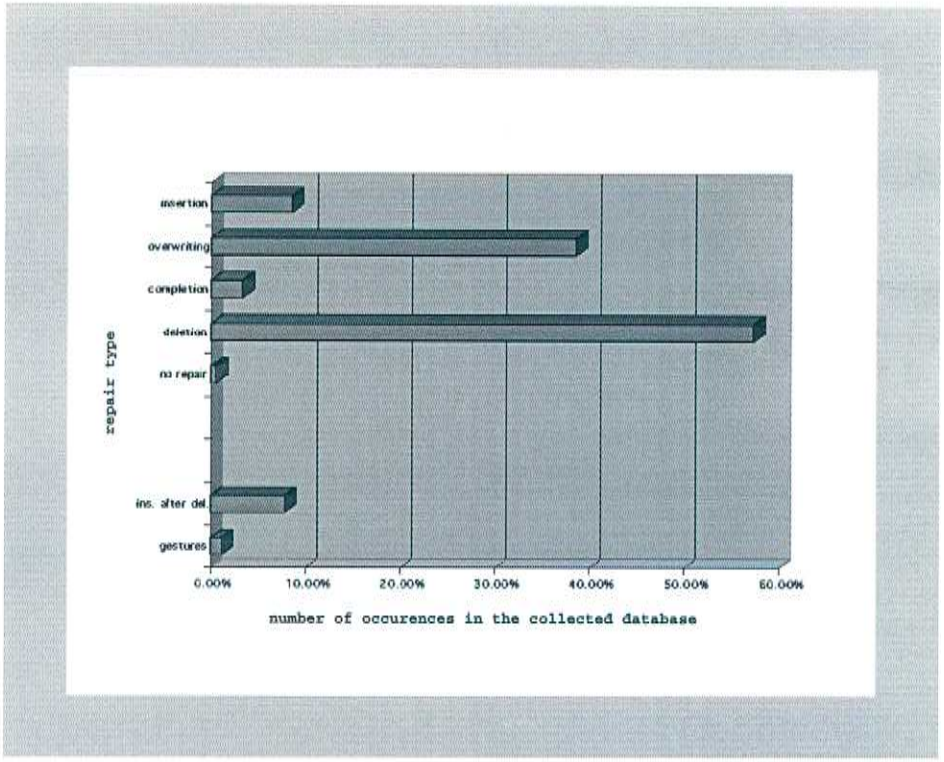

Figure 6.26: Classification of repair types in the collected data.

not offer any repair handling features, it was expected that the results are very bad. And in fact, only 7.9% word accuracy was received.

Then I tested the repaired data after applying my repair handling tools. The threshold φ used to compare the length of "normal" up-down strokes and repair strokes in the deletion case was set to 1.8. Every combination of the six overwriting check heuristics and the ten insertion heuristics was evaluated. The results can be found in Figure 6.27. The three diagrams should be interpreted similar to the ones already used in the previous section. The best resulting word accuracy was between 20% and 40% depending on the used heuristics.

discussion

Even though there is a big improvement compared to the recognition results received without any repair handling, the received word accuracy of nearly 40% in the best case seems kind of disappointing. Therefore I did visual checks on the data to find out, if repair classification and handling was done correctly and in which cases it failed. The results are shown in Figure 6.28.

The analysis showed that only 26% of the repair data was classified in the right way. One reason for this is, that the users sometimes did repair actions that were not covered by the repair tools. Therefore the classification of this correction is most likely to fail. Another problem lies in the heuristics, especially in the threshold φ used for a deletion check. If this threshold is set to high, the probability that an overwrite is classified instead of a deletion is much higher. 74% of the wrong classified data were overwrites instead of a deletion, which would be correct. Overwrites were only misclassified as deletions in 26% of the cases. I tried to optimize the threshold φ responsible for these misclassifications by hand and improved the rate of correct repair handling nearly 100%, from 26% to 49%. This is a strong evidence, that this is the most critical value in the proposed heuristics. In repair

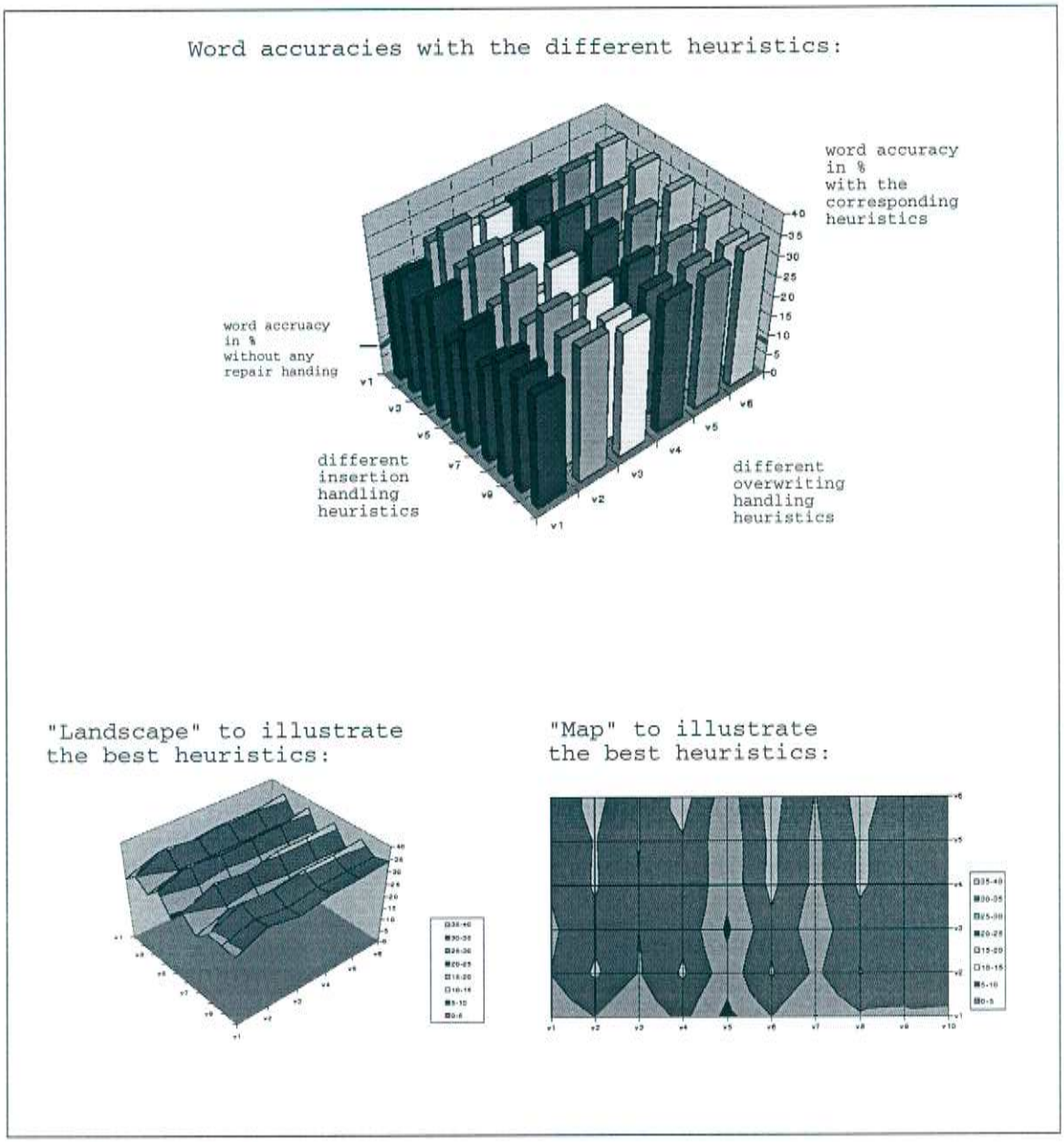

Figure 6.27: Recognition results with repaired data and the use of the repair tools before the standard recognition.

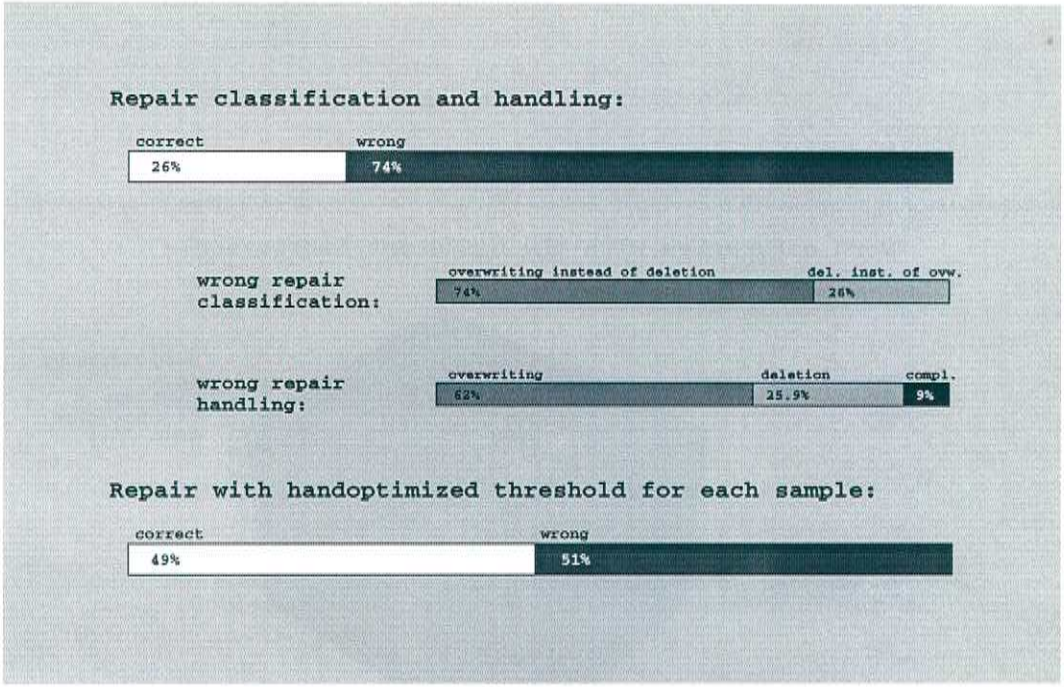

Figure 6.28: Analysis of the repair handling tools.

handling the most errors occurred in the case of overwriting. The main reason for this is, that the users did their repair very often in a "fuzzy" kind and manner. That is, they did usually not overwrite or delete the parts exactly that they wanted to be deleted. They placed their repair a little bit to much to the left, a little bit to much to the right, made the deletion strokes not long enough, etc. This is of course a problem with the repair heuristics used here, but it would be a problem in any kind of repair handling algorithms. For example, if a user deletes the second part of a word, but does not cross out the last letter. How should a repair tool know, if he wants this letter to be deleted or not?

Summarized there are the following **problems with this approach**. The main difficulties are:

- wrong classification because of a false setting of the threshold φ:
  if the threshold φ for detecting a deletion is set to high, an overwriting might be classified instead and vice versa;
- wrong handling because of a "fuzzy" repair by the user:
  some repairs done by the users were very rough and vague; these corrections might be still readable by humans, but pose problems for any kind of repair handling algorithms (not only for the ones tested here);
- not all cases are/can be covered by the heuristics:
  pairs of examples exist, where contradictory handling actions are required; this is also a problem that appears in any kind of repair algorithm, not only the ones used here; the most reasonable way to deal with it seems to try to cover as much as possible and use some generalizations in the recognition process when a misclassified or wrong handled repair happened; note that this is done in the approach taken here: even if a few coordinates are deleted that should not be, the recognition result can still be correct, because the

TDNN has the ability to generalize and therefore handle also some input that is partly "fuzzy";

Some other, minor reasons are also responsible for the low recognition accuracy:

- repair types were used that were not covered by the algorithms:
  some deletion gestures were used, that were not covered by the heuristics; also "double strokes" happened; the users wrote one part of a word more than once in this case, to indicate, that it is a repair and to separate it from the wrong handwriting, that was intended to be corrected; this might be good to recognize for humans, but is very hard to handle by an automatic algorithm;
- wrong data:
  even after a repair happened in a particular part of the word, some still contained an error at another position; others contained strokes that were just not readable, not even by a human; therefore a correct recognition with this words is unrealistic, even if a perfect repair handling was done;
- highly confusable dictionary:
  the dictionary and the test set of data used here are highly confusable, since both, the word written in the first phase and the one written or corrected in the second step are in the dictionary; it is more likely, for example, to make an error, if the dictionary and the test set contain a lot of word pairs like "cam" and "can", as if it contains only words that are more different from each other.

So on one hand the heuristics proposed for repair handling in the previous chapters sound very reasonable and intuitive in most of the cases. But on the other side they failed, if applied on a "real" data set containing repairs and corrections. For this reason I will propose an extension to this approach in the next chapter, that should solve at least some of the problems listed above.

6.3.3 Interactive Approach

Like seen from the analysis at the end of the last section, there are some critical thresholds in the heuristics and some typical user actions that cause problems with the proposed repair handling algorithms. Here I would like to recall the two important concepts to deal with errors in human handwriting I introduced in chapter 5: error handling and error avoidance. The algorithms and heuristics described so far all deal with the first concept: to discover an error and to allow recovery from it by offering the possibility to the user to do some repair. It turned out that there are big problems with the proposed and probably with all, or at least many, other approaches, too. Users sometimes do not "behave" like it is expected from the input interface and the repair handling tools, for example, by making a repair that is not covered by the algorithms or that is "fuzzy" in a way that there is no "correct" repair but one that can be interpreted in different ways. Therefore the user maybe corrects a previous error but also creates a new one: an error resulting from a wrong, uncovered, or fuzzy repair. So why not use some methods referring to the concept of error avoidance to reduce this risk of additional appearing errors?

The approach I will present here deals with the fact, that in on-line handwriting recognition the writer usually sits in front of his input device, e.g., a touchscreen, and has the possibility to get feedback from the recognizer or the repair handling tools. The idea now is to do the repair handling immediately and to indicate it

to the user as soon as possible. This is illustrated in Figure 6.29 and 6.30. Figure 6.29 shows the repair and recognition process like it was used so far. After the handwritten input from a user, the corrections are handled by the repair tools. The repaired input signal goes to the recognizer, which indicates the result to the writer. The user does not get any feedback from the repair handling tools. He does not know, if his repair actions were handled correctly or not. The only information he gets is the final result. In the interactive approach, illustrated in Figure 6.30, not only feedback from the recognizer but also from the repair tools exists. The corrected input signal, that is used to apply the recognition algorithms, is also indicated to the writer.

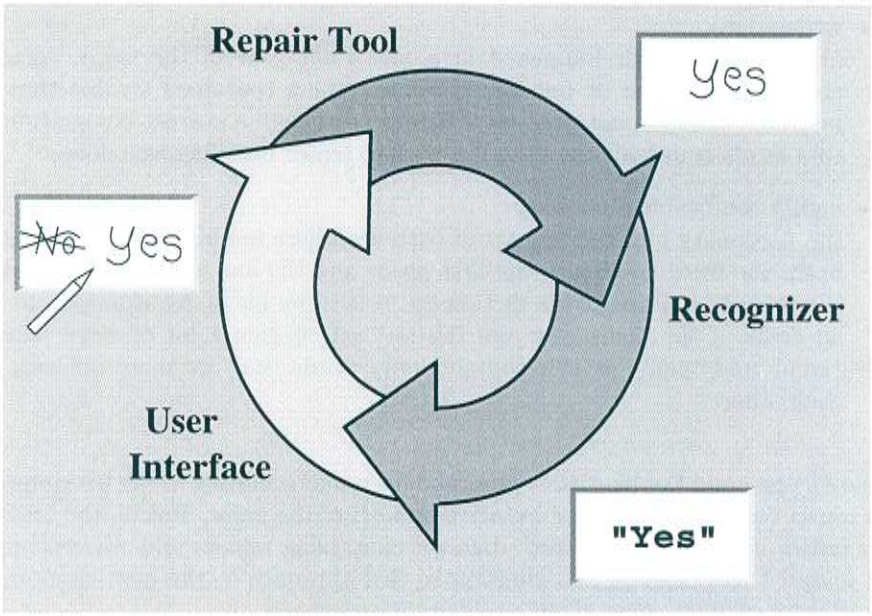

Figure 6.29: Illustration of the repair process without user interaction.

This kind of **interactive repair approach** has several advantages that might help to reduce the problems described in the last section.

First of all, there is the problem that it is very hard if not impossible to set the threshold φ responsible for the result of the deletion check correctly. If it is to low, you risk that an overwrite is misclassified as a deletion. If it is set too high, a deletion might not be discovered but handled as an overwrite. The idea is now to set this threshold φ high enough, so that most of the overwriting cases are handled correctly. The problems with deletions misclassified as an overwrite can now be handled through interactivity. If a deletion is shown immediately to the user after its detection, he knows, that the repair tool has interpreted his corrections in the right way. But he also realizes, if the algorithms did not handle his input proper and now can react to this situation. Especially the approach I introduced in the previous chapter, where the length of the strokes are compared, turns out to be very helpful in this situation. Imagine that a user has scribbled over some parts of a word with the intention to delete them. Since the threshold φ to detect a deletion is set very high now, his corrections might be interpreted as an overwrite first. The user recognizes this, because only the overwritten strokes disappear from the screen, but not his repair gestures. What would he do now? I think, that the most intuitive way to react to such a situation is to try to cross out the repair gestures, too. And if he does this, the length of the repair strokes will raise and therefore a deletion

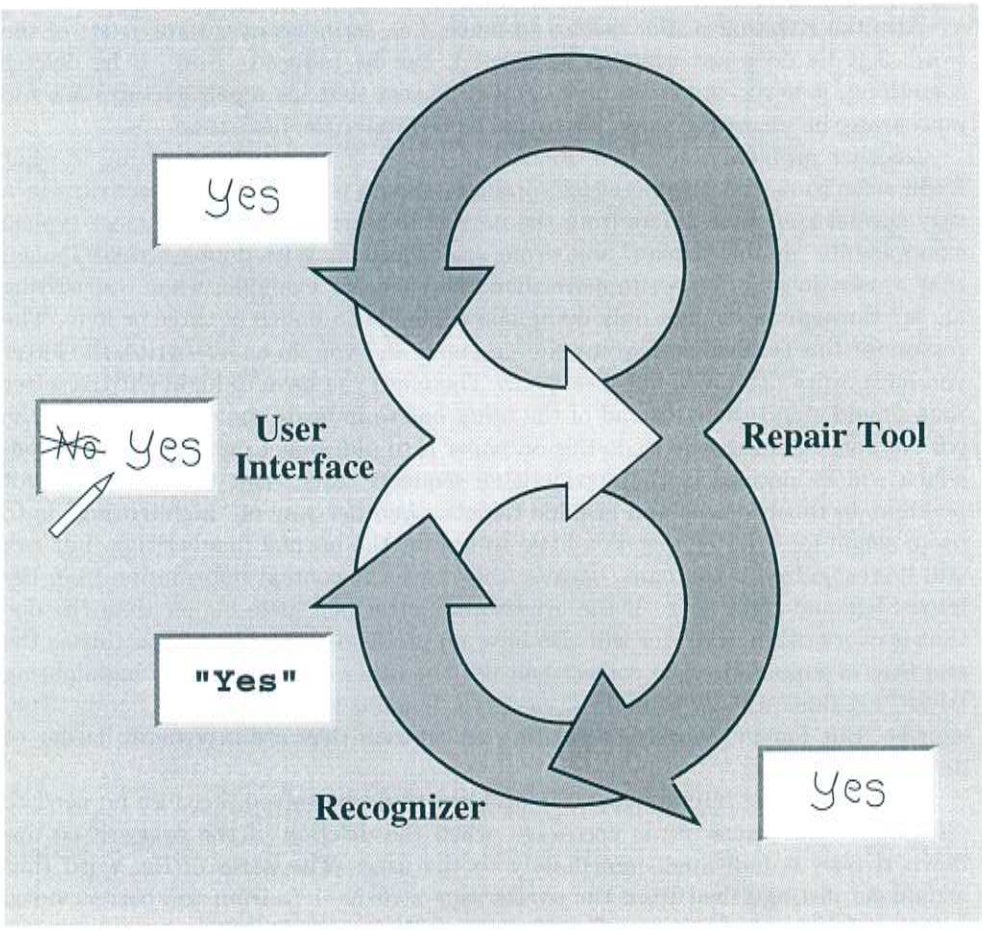

Figure 6.30: Illustration of the repair process with user interaction.

soon will be detected and indicated. The hope now is, that the user realizes, that the more strokes he makes the faster his deletions are detected and handled. In that way he might adapt to the interface and always make the right gestures for deletion, that can be recovered by the repair heuristics. It should be noted, that this seems also like a very comfortable way of deleting. Since no restrictions are made to use a special kind of shape for his gesture, the user can just stay with his pen in the area to delete and continue making his pen movements till the part to delete disappears (roughly speaking: if you are just scribbling in an unpredictable way over an area, it should not matter, if you make 5 or 10 pen movements more or less, until the deletion is indicated).

Now imagine the following situation. A user makes a deletion in a word, but does not cross out every part he really wants to be deleted. With no user interaction the repair handling tools or the recognition algorithms have to deal with this problem. But if the deletion is indicated immediately to the user, he sees, that there are still some parts not deleted that should be and can cross them out, too. Therefore an error in case of a "fuzzy" repair is avoided. Note that with the criterion used here to detect a deletion, this action can also be done very simple. Since the user has already made a lot of repair strokes, the remaining ones needed to delete the rest he wants to remove should not be to much. If you use a gesture instead, for example, a cross, you have to do this whole gesture again, maybe on a very small part of the word, what seems to be very uncomfortable.

Also the risk that a user deletes to much, i.e., scribbles over some parts of the word that he does not want to be deleted, can be reduced. Sure, if he deletes something, it never will come back. But if he sees that his repair gestures are too inaccurate, he might try to do his repair more proper the next time.

Another problem that often appears in repair of human handwriting is what I will refer to as the "highlighting"-problem. Some users do their overwrites in a very special kind that differs from the normal handwriting. The two most typical examples are "double strokes" and wrong scaled letters. With double strokes I mean that a user does an overwrite more than one time, for example, when overwriting an "a" through an "o" not only doing one circle, but a bunch of three or four. The reason for this is obvious: if you write on paper and you do an overwrite, the letter you have overwritten will not disappear. Therefore you have to highlight the letter that should stand there instead of the other one to indicate somehow, which is the correct one. Another way to do this on paper is to put higher pressure on your pen, which will be noticed in the downwritten sequence later. But this is usually not possible on touchscreens and graphic tablets. Another way of "highlighting" is to use a shape for a letter that is a little to big for the normal handwriting, but can still be recognized by humans, because they have the context information from the letters left and right of it. If the overwriting letter is a little higher than the one that is overwritten, a reader will also have no problems with identifying this as the one that is supposed to be correct and not the other one. All these "highlighting tricks" are done usually to make it easier for humans to read repaired handwritten signals. But for automatic handwriting recognizers they are in general harder to detect and recover.

With the use of interactivity this problem could be solved, because no need to "highlight" this overwrite is necessary, when the deletion of the overwritten up-down strokes is indicated immediately to the user. The parts of the word that should be distinguished from the overwriting strokes, i.e., from the correct ones, will be deleted automatically. Therefore no confusion can happen.

So there are some strong arguments that let you hope to overcome at least some of the problems described above by taking profit of the use of interactive repair handling. Therefore I implemented such an immediate repair indication into my repair handling tools. In the next chapter I will evaluate them and discuss the results.

6.3.4 Evaluations

evaluation

To evaluate the repair tools with interactivity, additional data had to be collected. The setup for this **data collection** was the following: the "clean" data from the data collection described in section 6.3.2 was presented again to the same writers that have written them the first time. Now they were asked to do the same corrections again (like: "correct: *hair* instead of *chair*"). But this time, feedback from the repair tool was given. Deletions and overwritings were shown immediately to the user. Therefore the writer was able to see, what was going on. The hope was not only to reduce the error and improve the recognition result, but also to give the user an idea for how the repair works, what can be handled and what not.

Since I was mainly interested in how the interactive repair indication influences the user behavior, I did not include run-on recognition in the data collection interface. Note that the early indication of a part of the result can also influence the user behavior and the way he does repair. But here the most important question was how a user reacts to an interactive repair tool and how this influences repair handling.

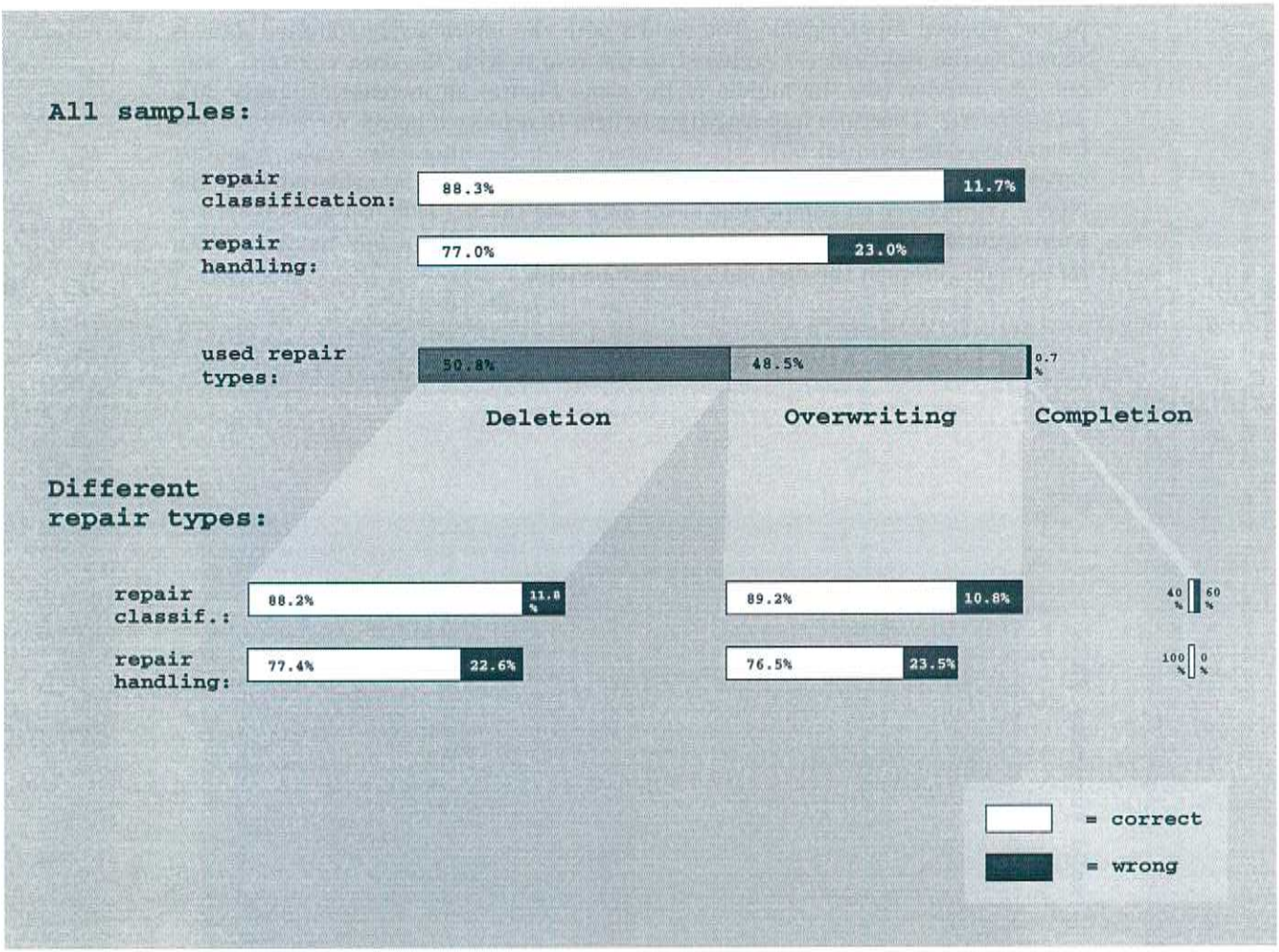

Figure 6.31: Evaluation of the repair handling with interactive repair indication

For the on-line repair calculation and indication I used the Version v6 for the overwriting checks (see page 59) and Version v2 for the insertion checks (see page 65), since they received the highest results with the databases tested in the previous chapters. To avoid repair classification problems due to a wrong setting of the threshold φ to classify a deletion the responsible value was set extremely high, i.e., to 2.8 (instead of 1.8 used in the evaluations with the data collected without interactivity).

The results of this evaluation can be found in Figure 6.31 and 6.32. The first one analyzes the repair handling, the second one shows the received word recognition with "interactive repair", compared to the none interactive case from section 6.3.2.

The analysis of the repair handling (Figure 6.31) shows that nearly 90% of the data was classified correct, and that in about 75 to 80% of the cases the repair handling was ok. Note, that a repair that is not handled completely correct, still can be recognized well. For example, if only one up-down stroke is deleted to much, the recognizer might still be able to calculate the correct result. There was no significant difference in repair classification and handling between the repair types deletion and overwriting. Both received about the same results.

Figure 6.32 shows the word accuracy received when the recognition was applied

to the repaired input signal. The results with the interactively collected data is shown on the right side. Compared to the results with the data collected without interactivity (see the middle of the same Figure) an increase of nearly 30% was achieved. Therefore interactivity can help to achieve a better recognition performance. The received 65% word accuracy with the interactive repair handling cover about 75% of the recognition performance that can be achieved with the NPen^{++}recognizer on comparable clean data (see the left side of Figure 6.32; the word accuracy here was calculated on the clean data without any repair, written by the same users in the first data collection step).

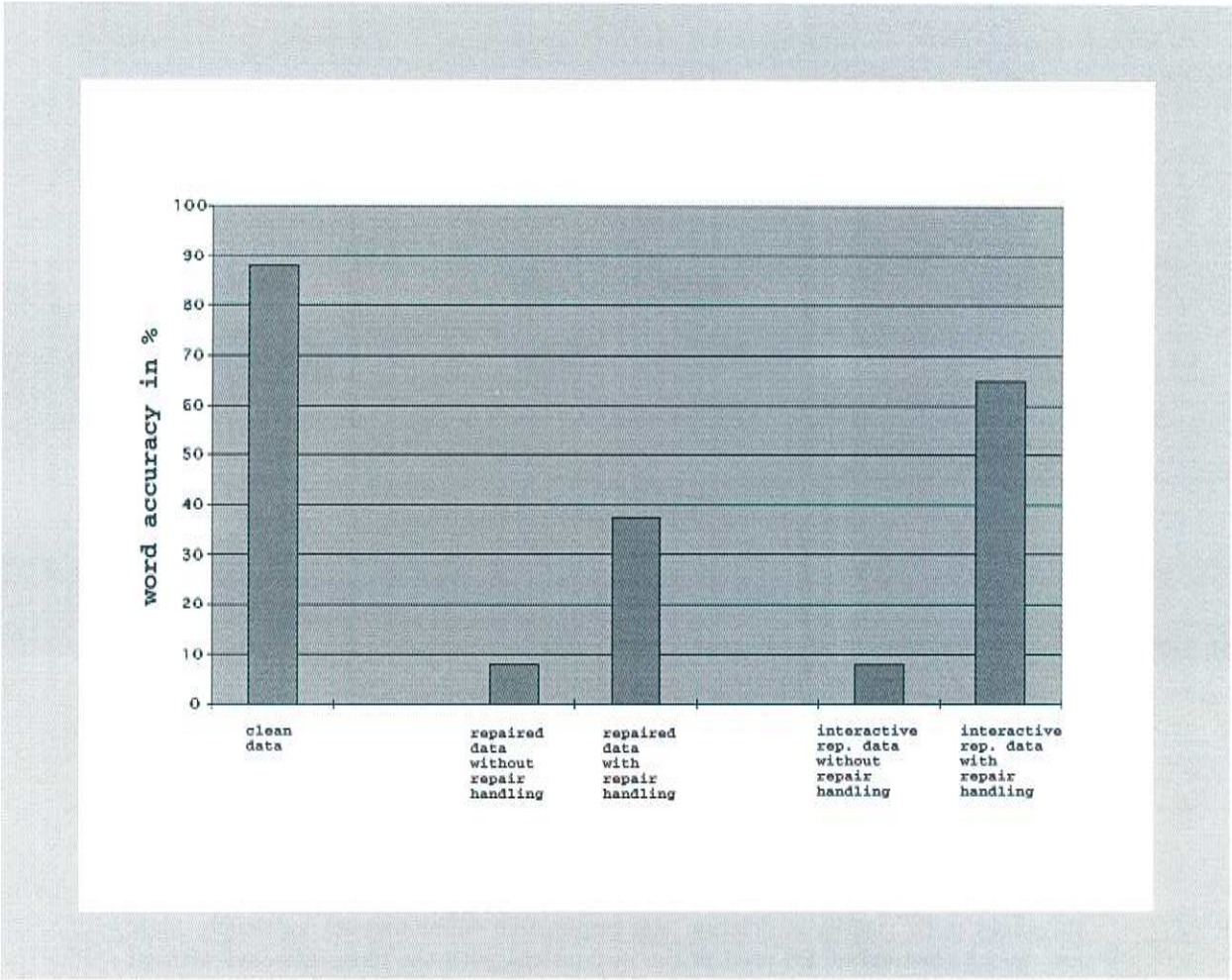

Figure 6.32: Word accuracy with the interactively collected data (right) compared to the none interactive results (middle) and the recognition performance on comparable "clean" data (left).

discussion

Since about 75% of the recognition accuracy with comparable "clean" data has been received with this interactive approach, the word accuracy results are very satisfying. But like already mentioned at the beginning of this report, recognition accuracy should not be the only criterion in the design of a usable, satisfying interface for human-computer interaction. Also user acceptance plays an important role. Therefore I asked the people who wrote the data used in this evaluation, how they liked the offered repair possibilities and what they think about them (note

that with only four writers no statistical relyable statement can be made, so these user opinions can only show a first trend). All asked persons agreed, that the overwriting works very robust and is pleasant to use. They also all had the opinion, that the deletion is kind of critical, because it sometimes can take to long, until it is discovered by the repair tools and indicated to the user. This is a consequence of the setting of the threshold φ for the classification test in the deletion case to an extremely high value. It is interesting to note, that the users had the impression, that the deletion case is not that robust like the overwriting case, even if the classification and repair handling results in both cases have been about the same (compare Figure 6.31). Therefore the high recognition performance on one hand was achieved by lowering user acceptance on the other hand in the case of deletion, which is very critical, especially since one of the motivations to take care of the repair task was to increase recognition accuracy.

Another interesting thing I detected while analyzing the data was, that it seems that the users kind of adapted to the interface. That is, they made corrections in a way users usually don't do, but that are ok and can be handled very well with this interface and repair handling tools. For example, instead of overwriting two wrong letters with the correct ones, also the letter before this two was overwritten, although there was no need for it, since this letter was written correctly. The reason why the user did this was, that these three letters appeared very narrow to each other and it was easier in this case to overwrite all of them instead of trying to "hit" the right position to overwrite only the two wrong letters. This probably would have resulted in a repair handling error in this case. So it seems like the users got a feeling for the repair handling algorithms, realized what works and what does not, and adapted to it, without feeling unpleasant about it.

Summarized the interactive repair handling approach can be seen as a success. It proved to be useful to increase the recognition accuracy in the tests that were made here and helped to reduce the errors that happened without interactivity in case of wrong or fuzzy repair by the user. In this context it should be noted, that this approach corresponds to some of the remedies that were proposed by L. Schomaker in [Sch94] to solve the problems and errors that usually appear in human handwriting input (see page 37). The main demands that are covered through the interactive repair tool are "clarify to the user what is going on" and "give up the paper metaphor".

Through direct indication of the repair handling, user behavior can be influenced in a way that first increases the performance of the repair handling and the recognition and second can be comfortable and easy to use, if implemented in the right way. Here the main problems appeared in the proposed heuristics. Even though the deletion handling was implemented in an extremely general way, that also seems to be very intuitive to use, users felt uncomfortable with it. The reason for this was, that acceptable repair handling and recognition accuracy could only be reached with a high threshold φ for the test, if a deletion occurred or not. This problem destroyed the main advantage the proposed method has, that is, being very general and intuitive to use (roughly speaking: it is intuitive to make a lot of strokes to cross something out, but it looses its intuitivity, if you have to make too many strokes). Therefore it might seem as a good approach just to cover overwriting and completion with the heuristics proposed here and handle the deletion case by introducing an additional gesture that is more restrictive to the user and therefore easier and more robust to detect. But it should be noted, that the repair handling step, i.e., the question which parts of a word have to be deleted after a deletion is classified, is not easy to solve with this approach, too. In fact, it is probably even harder then with the heuristic used here, if you expect the user to make a gesture that has a special shape. Also this gesture should be easy to remember and intuitive to use.

Otherwise a decrease in user acceptance will be the consequence in this case, too. But if you take care of these criteria, this method seems to be promising for future approaches.

Chapter 7

Repair of Printed Letters

In the last chapters different heuristics and algorithms to deal with the problem of repair in a handwritten input signal have been proposed, discussed, and evaluated. Now I like to face the problem of repair in printed text, i.e., corrections of the recognition result, that is indicated to a writer.

There are good reasons, why this possibility of repair should be offered to a user in a comfortable interface. First of all it seems more natural to correct an error where it occurs. And when a user writes a correct input that looks legible to him, but for some reasons can not be handled and recognized correctly by the recognition algorithms, then to him the error occurs in the result indicated at the screen and not in his input (even if his handwriting might be the real reason for the wrong recognition). There are also some error situations, that just can not be solved through correction of the input signal. For example, imagine you are writing the word "hell" and the recognition result indicates you "hello". How should you correct or improve your handwriting to correct this error? The best and most intuitive way here seems to just cross out the wrong additional letter from the indicated result.

The problem of repair in the printed recognition result is much easier than the problem of repair in handwritten input. It appears to be less complicate, because the case of repair handling, usually does not have to deal with problems, like, for example, "Which part of the word should be remove after a deletion?", etc., because the borders of the single letters are well known. But it should be noted that "easier" does not mean, that this is an easy problem. There are still a lot of issues and difficulties remaining, that have to be solved. Therefore I will introduce some repair handling heuristics for corrections in printed text in the next section.

7.1 Algorithms and Heuristics

One big advantage of repair in printed text, compared to corrections done in handwritten input, is, that it is usually done on letter or word level. Things, like correcting only one half of a letter or changing its shape by adding some small strokes, usually do not happen. Of course, there are some cases, that can be thought of, where things like this can occur. For example, the letter "l" can be changed to the letter "h" just by adding one small stroke. But these cases are very rare. The reason for this is first, because there are not much letters that can be "changed" in such a way and second, because usually the font size is relative small, so it is easier just to write the whole new letter than to correct the wrong one. Also you should keep in mind, that the purpose of the heuristics, that will be proposed here, is to correct a wrong recognition result, not to create a perfect editing system for printed text. Therefore it should be ok to restrict the repair handling algorithms

to be done on letter level only.

This and the problem, that it is very hard to distinguish between what is an overwrite and what is a deletion, is the reason, why I implemented **a version that only handles deletion and rewriting of some letters**. This might seem very restrictive, but it can be useful nevertheless. The reason for this is, that the font to indicate the recognition result is usually not very large. Therefore overwriting of a letter that stands between some others is kind of uncomfortable for the users (maybe not on paper, but definitely with the hardware usually used today for pen based computer input). It seems easier just to cross out this letter and rewrite it at another position, where more space is available and you are not forced to adapt your handwriting to a specific font size.

If deletion and a following insertion is the only repair allowed, two steps that have to be done in the repair handling process of handwritten input disappear. First no **repair detection** has to be done. Since the handwriting to correct is done in an area, where usually only printed text is written, every handwriting in this part of the input interface can be interpreted as a repair. Also **the classification step** reduces to the problem of distinguishing between deletion and insertion. But since no overwriting is allowed, this check is very easy: everything written over printed text can be seen as a deletion, everything left, right, over, or under it as an insertion.

If you look at a text string of printed letters, there are several **possible positions for a deletion and an insertion**:

- beginning
- middle (middle)*
- end

The star (*) means that no, one, or more than one deletion or insertion can occur at this position. Here other problems arise. Since no overwriting is allowed, insertions have to be done at another part of the area where handwritten input is allowed. Where should such a handwriting be inserted? Further restrictions to the user can help to overcome this problem.

different approaches

The **most general way** is to make no restrictions at all. But then you do not only face the problem of inserting a repair at the right position. Also the recognition process becomes very hard. If you write whole words, the search for the correct result in the recognizer has to be done only on the allowed dictionary, which usually has a limited size[1]. But if now some parts of this word are corrected by rewriting them, theoretically every substring that is contained in the original dictionary can be written. As a consequence a search on every such substring has to be done, what makes the recognition problem much harder and usually decreases the performance. There are some techniques and simplifications, that can be thought of, to overcome this problem. For example, the writing of a single letter should result in a relatively short sequence of coordinates, so the search should not have to be done on extremely large substrings. But these techniques all have their problems and disadvantages, too (e.g., how should "relatively short" in the previous example be defined?). Therefore I tried another approach here: by making more restrictions

[1] It is also possible to allow any word as input, if the recognition is done on letter level, but obviously this problem is much harder and therefore usually leads to much lower recognition performances. Also the NPen^{++}recognizer used in this thesis operates with a fixed dictionary size.

to the user I will reduce the size of possible solutions for the handwritten input. Then the recognition has only to be done based on a new dictionary with a much smaller size, which should obviously increase the recognition performance.

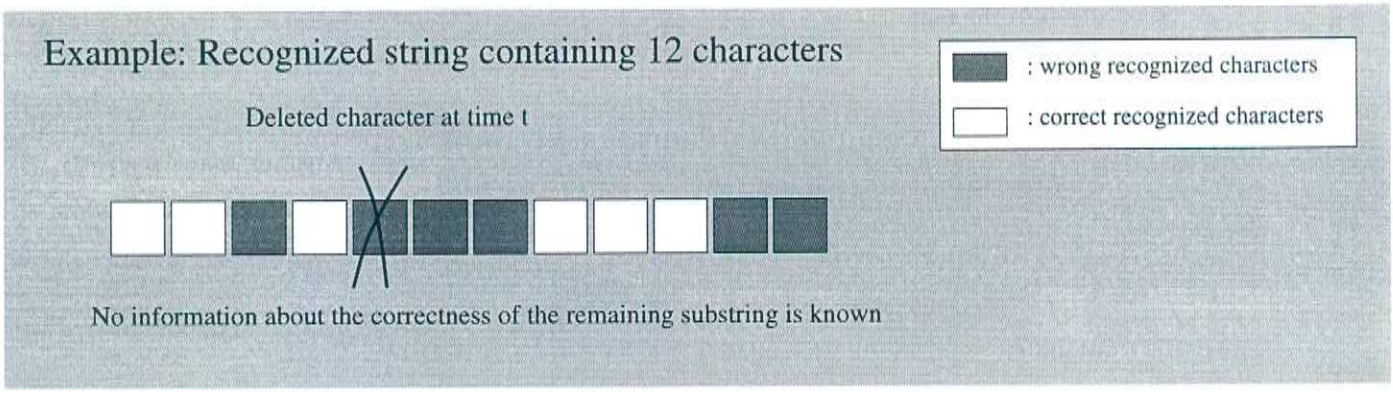

Figure 7.1: Example for the most general case: no restrictions are made at all.

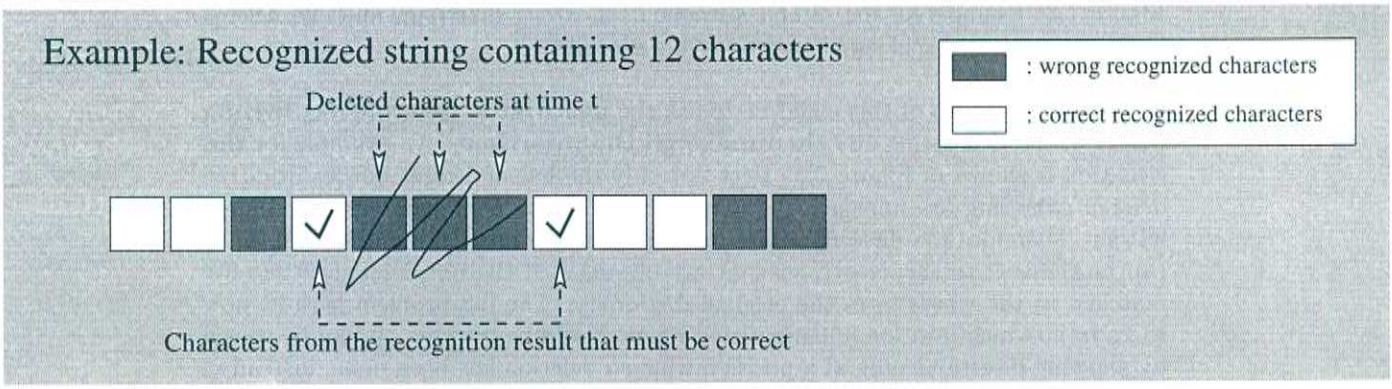

Figure 7.2: Example for the first restriction: a whole substring must be deleted at once.

The idea here is to take profit of the parts of the word that are not deleted. If you make a **first restriction**, that a user is only allowed to **delete wrong substrings as a whole**, you know the left and right border of the string to be inserted at this position, since the whole correct substring is inserted. An illustration of this can be found in Figure 7.1 and 7.2. Here a text string containing twelve letters is represented through one square for each letter. The grey squares indicate wrong letters, the white ones stand for correct characters. Figure 7.1 shows the case, where no restriction is made at all. Since the user can delete and insert when and whatever he wants, you have no information about the correctness of the substring after a first deletion. The only thing you know is, that the recognized word was wrong, otherwise a correction does not make any sense. Theoretically a search on every substring of the original dictionary should be done, if the user writes an insertion now. But with the restriction, that a whole substring has to be deleted at once, you can get some information from the part of the word that is not deleted. Figure 7.2 illustrates this example. If all the wrong letters are deleted, the ones left and right of the deleted substring must be correct. Therefore a search on a handwritten input signal, that is supposed to be inserted here, has only to be done on the substrings from the original dictionary that appear between these two letters.

Depending on what characters these are an enormous reduction of the search space can be achieved, promising higher recognition performances.

It should be noted, that another restriction is made here: it is assumed, that the user first deletes the wrong substring and then writes the correct part of the word to be inserted. This restriction also solves the problem, where to insert a handwritten sequence into the text.

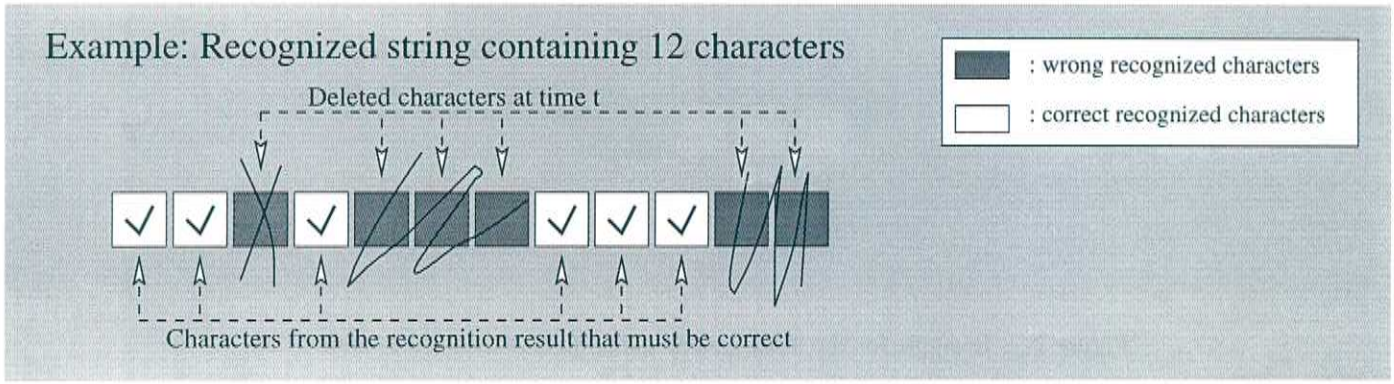

Figure 7.3: Example for the second restriction: all wrong substrings must be deleted in a first step before making all insertions in a second step.

A **modification** of this could be to ask the user **first** to **delete every wrong letter** of the text string and **then make all the insertions**. An example for this situation is shown in Figure 7.3. Here not only the left and right border from the deleted substring are known to be correct, but the whole remaining, not deleted letters. Therefore the dictionary size for the search on inserted handwritten input can be reduced to the substrings that are obtained, if the not deleted letters are matched to the words from the original dictionary. The big problem here is, how to know at which position a handwritten input should be inserted. It can be solved by allowing insertions only at a position where a deletion has been done, instead of writing over or under the word. This should be an acceptable restriction, since the insertion and deletion is divided in two phases anyway. But the problem with this solution is the size of the font usually used. In most of the cases handwriting is much bigger than printed text. If you expect a user to write and insert between printed letters, you must use an unusual big font, otherwise he might feel uncomfortable with it, because he has to write much smaller than he is used to.

The version I implemented in my interface is the most restrictive: **deletion in a sequence of printed letters is allowed only once**. This means, if you want to correct a wrong recognition result by deletion and insertion, you have to delete a substring that contains every wrong letter, since no second deletion is allowed. Therefore it can happen, that also some correct letters have to be deleted and rewritten. The dictionary size reduces to the number of different substrings occurring in the original dictionary in the words that match the undeleted one (if deletion at the beginning or the end) or two (if deletion it the middle) substrings. An illustration of this approach can be found in Figure 7.4.

The restrictions made here sound very strong and in fact, I believe that they would be to hard for a text editing system. But keep in mind that here we have the situation, that the users are correcting a wrong recognition result. Therefore situations where wrong letters and substrings appear on different positions in the

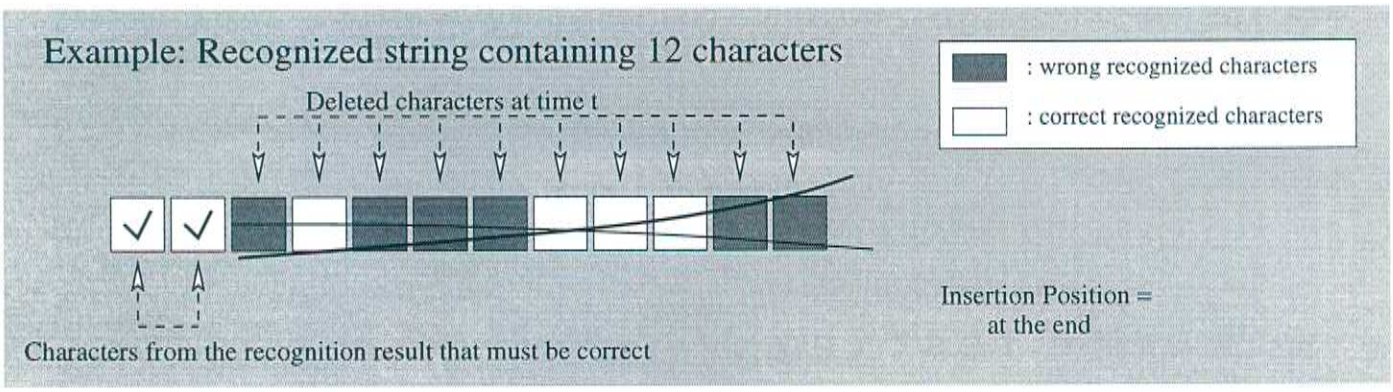

Figure 7.4: Example for the third restriction: only one connected string can be deleted.

word are very rare. Usually a wrongly recognized string contains just one substring. As a consequence of this I think the proposed restrictions are not that much of a problem and should not make the interface to uncomfortable for a user.

On the other hand this approach has the advantage that it solves the problem of where to insert a handwritten signal (see below) and also removes the restriction, that a deletion and an insertion have to be done in a specific order.

The **implementation** of a system restricted in such a way appeared to be relative simple. Like already mentioned, no repair detection step is required. If the interface is designed in a way that the recognition result appears in an area that is separated from the place, where the recognition input is made, every handwriting in that area can be seen as a repair. Also the classification which repair type happened is easy: everything over a printed letter is seen as a deletion, everything written in a "reasonable" distance of the printed word is interpreted as an insertion.

Deletion handling is done by comparing the bounding boxes of the printed letters with the bounding boxes of the strokes that are written over them. An example for this can be found in Figure 7.5. The bounding box of each letter is compared to the thresholded bounding box of the handwriting strokes. If an overlap occurs, the according character is deleted. This is a very easy algorithm, but it works very robust. Failures can be handled through direct indication of the deleted parts of the word. If one or more letters at one border are wrongly deleted, they can just be reinserted by writing them again. If letters are not deleted that should be, the user can just add a few strokes to delete them.

Insertion can be done at any position in the area that is supposed for this input and that does not contain any printed text. If a deletion happened before, the insertion is done at that position. If not and the handwriting is done left or right of the printed word, it is seen as a word beginning or end, respectively. If a deletion is done after such an insertion, the recognition is reset and a new one is done, whose result will be entered at the position of that deletion.

After a deletion is finished and an insertion begins the recognizer is reinitialized with a new dictionary, that contains only words, i.e., substrings of the words from the original dictionary, that can be possible replacements for the deleted string.

These algorithms seem all very easy and user restrictive. But implemented in the right way and integrated in a "fitting" application, they work pretty robust and still do not decrease comfort and usability for the user to much.

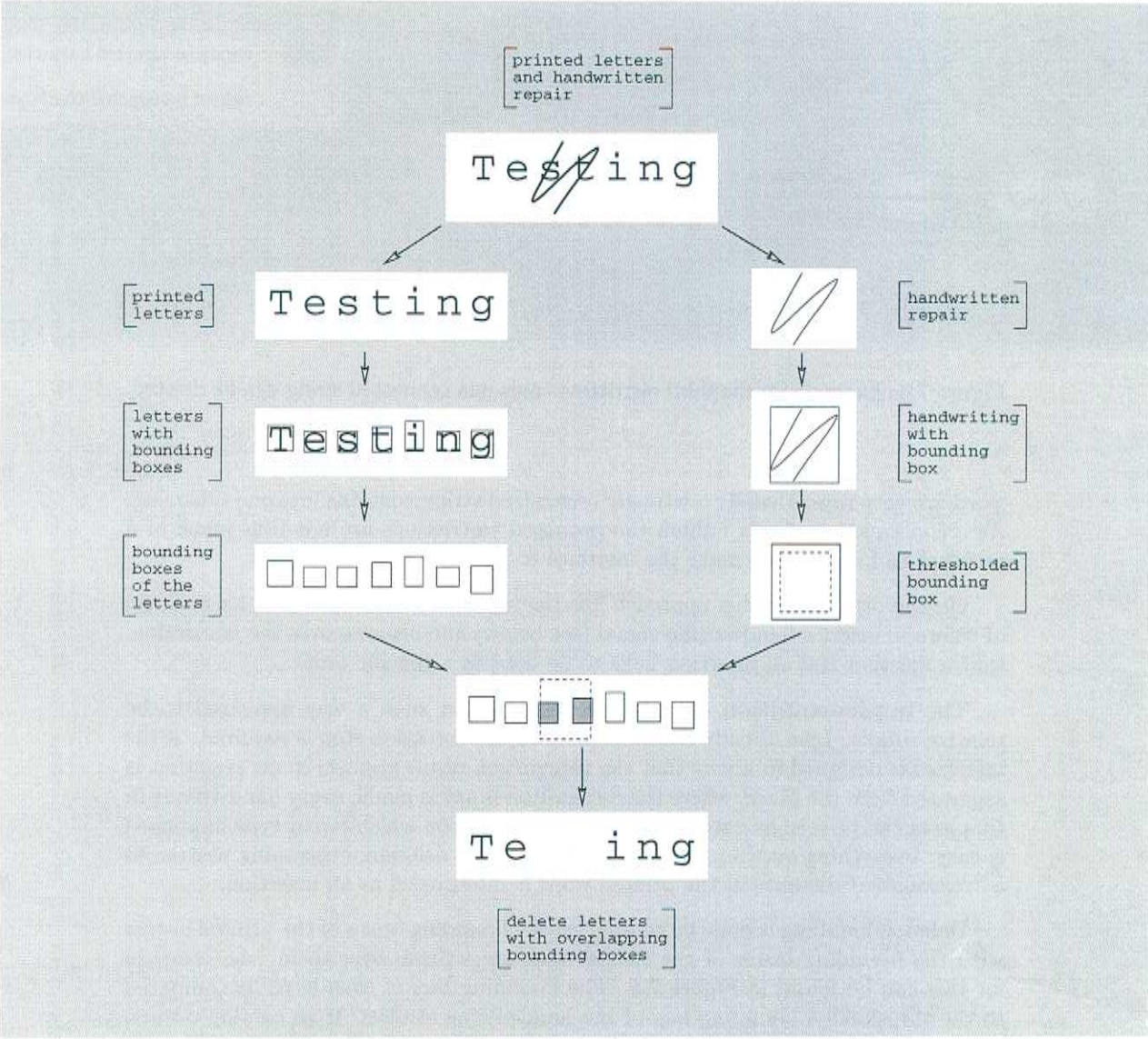

Figure 7.5: Example for the deletion of some parts of a text string.

7.2 Evaluations

In the following I will show some examples, how the dictionary size is decreasing as a consequence of setting different restrictions to a user, like they were discussed in the previous section. I did not test my system with handwritten input data, since in the version I implemented the dictionary size for the inserted handwriting signal depends extremely on the example you choose. Therefore it is hard to find test examples that provide objective and statistical useful measures.

evaluation

Instead I showed on an **example, how the dictionary size reduces**, if you assume several conditions concerning the user behavior in the case of a deletion.

For this purpose I took a dictionary with an original size of 51866 words. Half of them start with an upper case letter, the others are the same, but begin with a lower case letter. If you make **no restrictions** at all, the recognition result for the inserted handwritten signal is one substring from all the words in the original

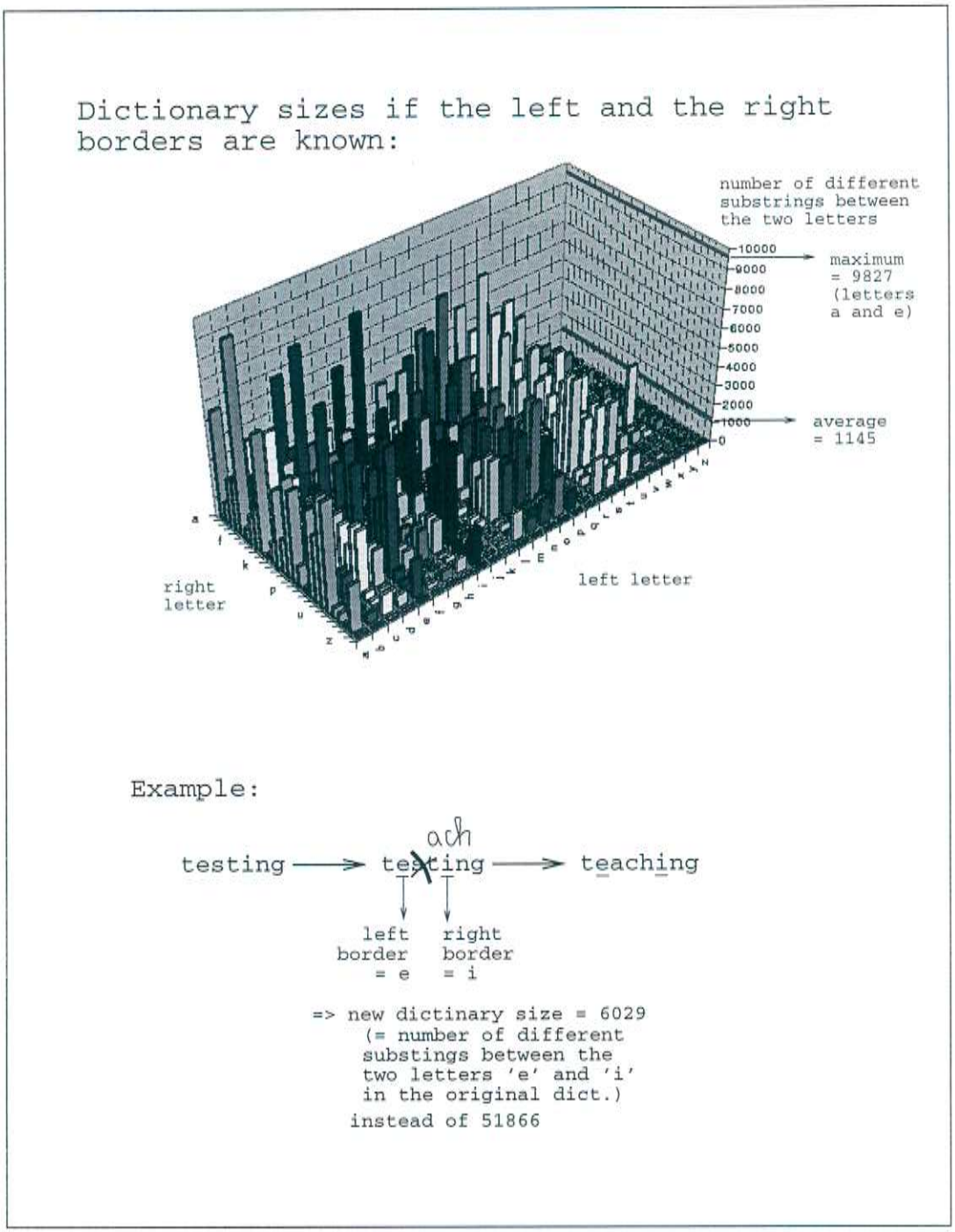

Figure 7.6: Example for the reduction of the dictionary size, if the left and the right border of the deleted substring are known (and the left one is a lower case letter).

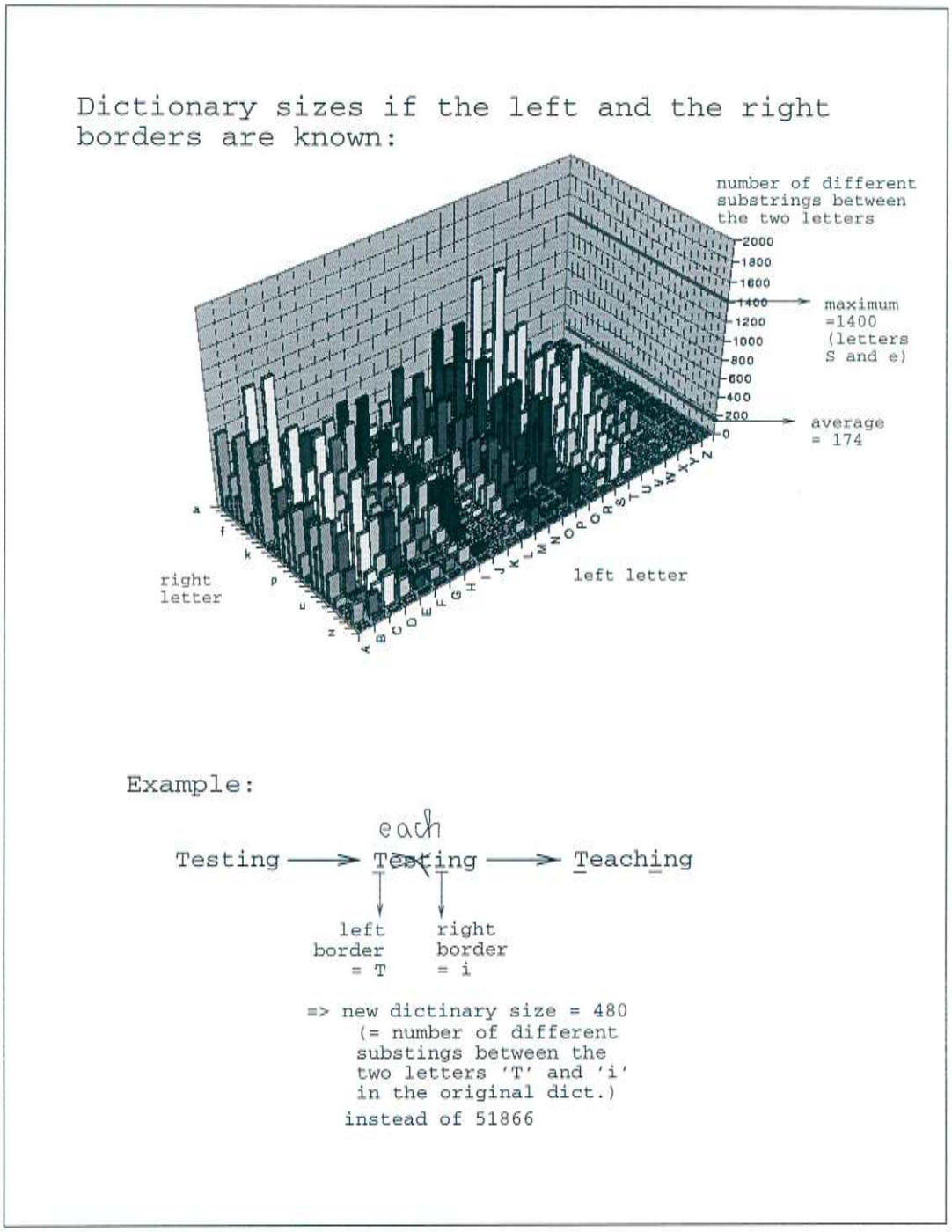

Figure 7.7: Example for the reduction of the dictionary size, if the left and the right border of the deleted substring are known (and the left one is an upper case letter).

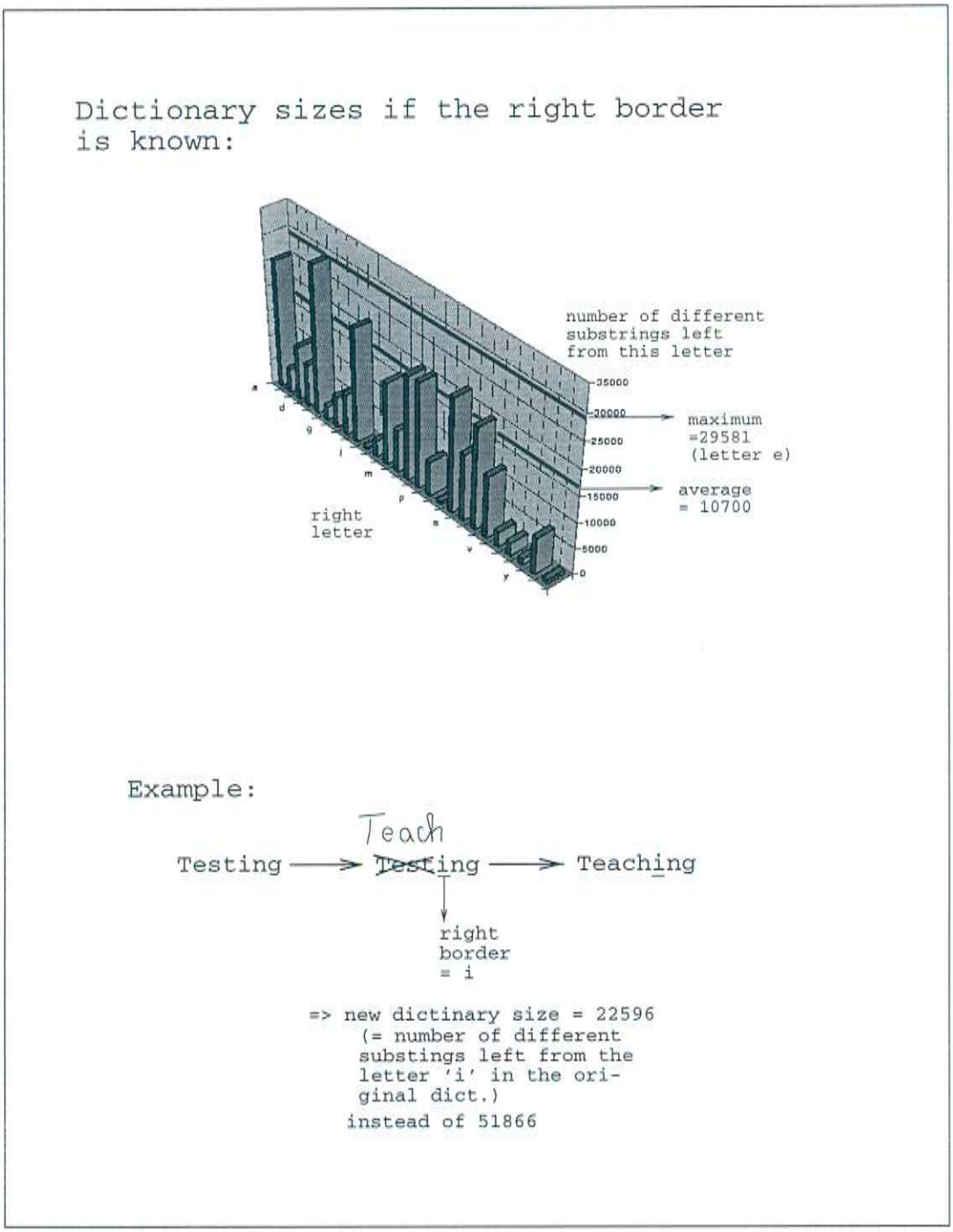

Figure 7.8: Example for the reduction of the dictionary size, if the the right border of the deleted substring is known and the word beginning is deleted.

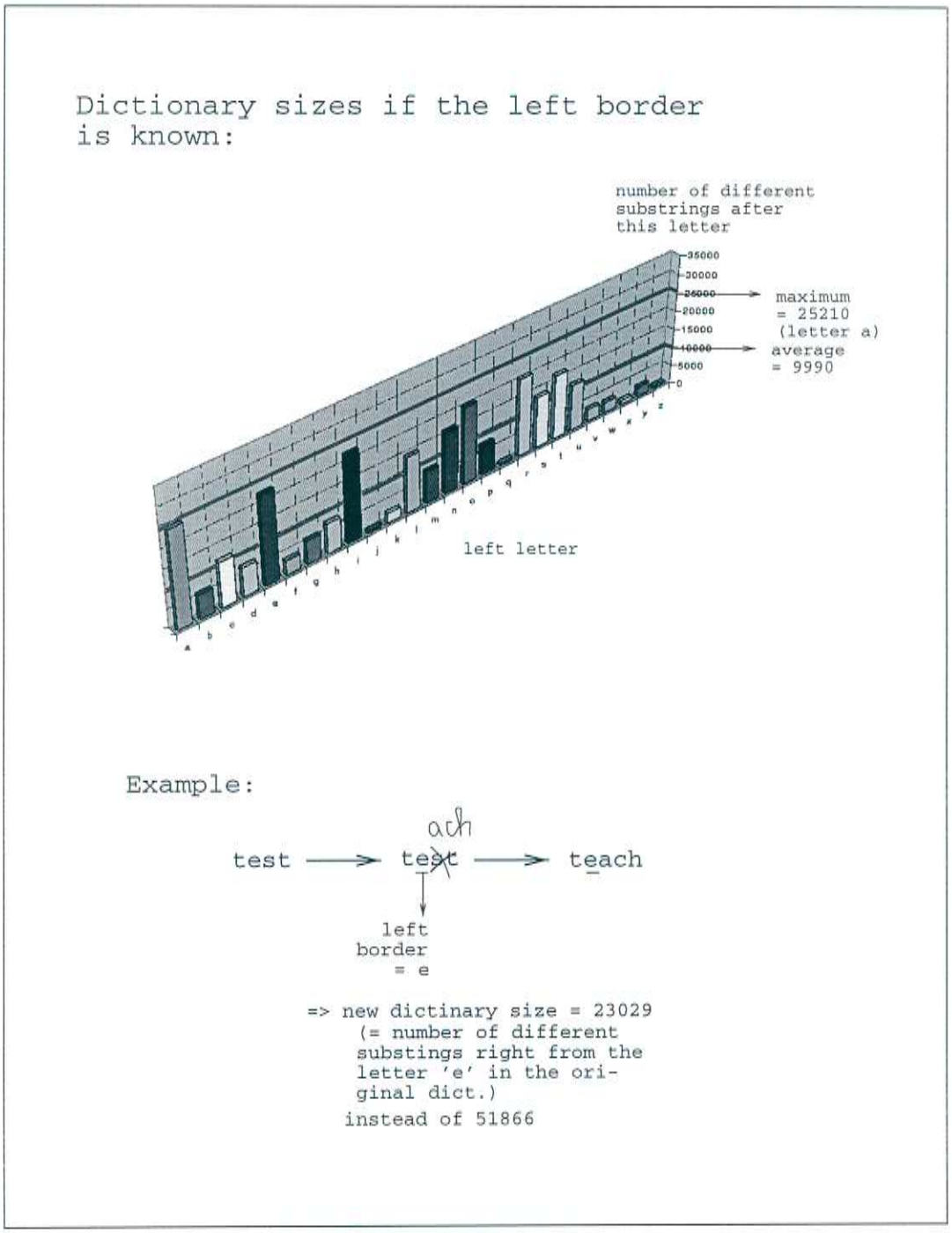

Figure 7.9: Example for the reduction of the dictionary size, if the the left border (lower case letter) of the deleted substring is known and the word end is deleted.

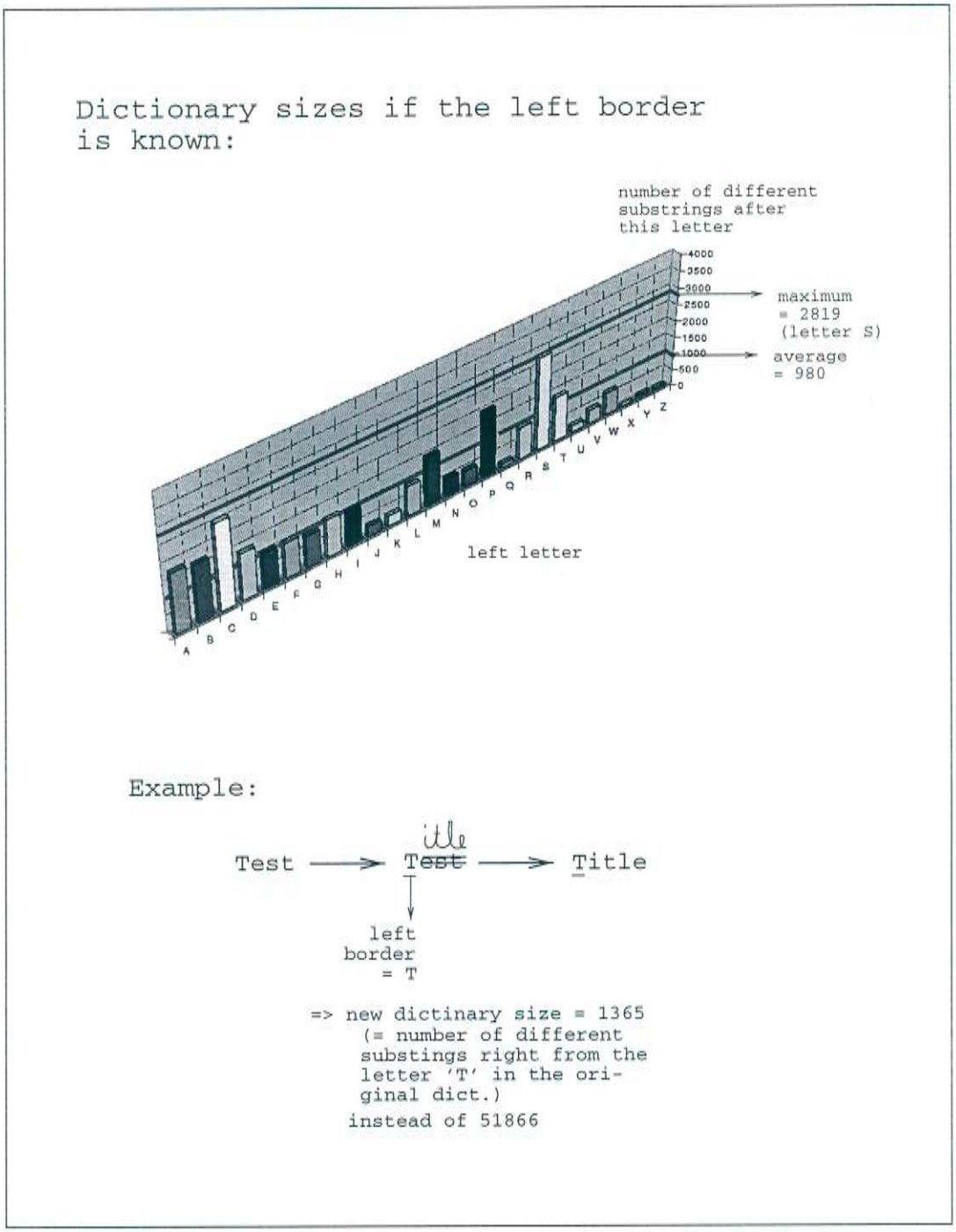

Figure 7.10: Example for the reduction of the dictionary size, if the left border (upper case letter) of the deleted substring is known and the word end is deleted.

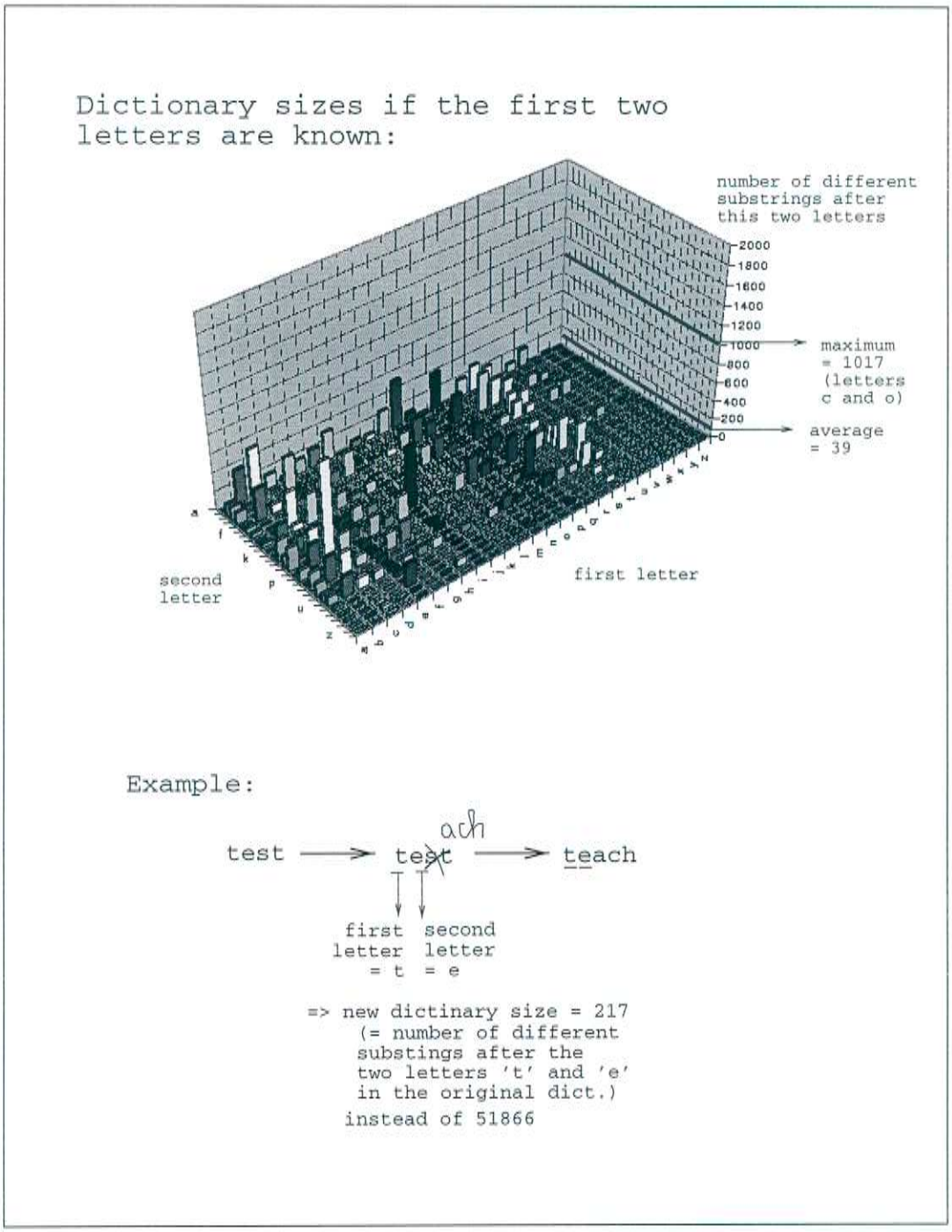

Figure 7.11: Example for the reduction of the dictionary size, if the first two letters of the whole word are known and the word starts with a lower case letter.

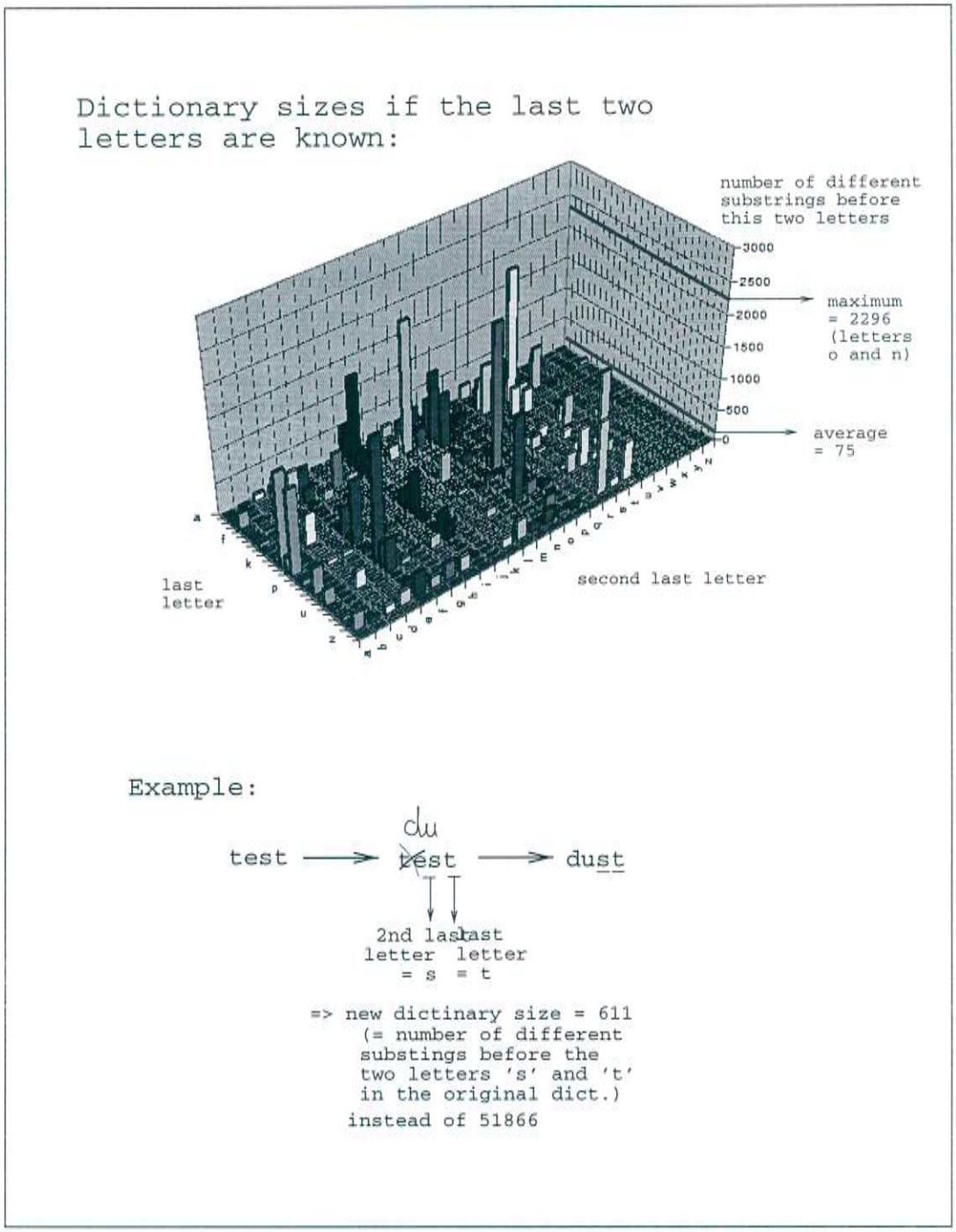

Figure 7.12: Example for the reduction of the dictionary size, if the last two letters of the whole word are known.

dictionary. In this example there are 288 610 different substrings. This means you are trying to recognize a handwritten input on a dictionary that is about 5.5 times larger than your original one. Without any further modifications this problem seems to be very hard and a decrease in recognition performance definitely will happen. In this context it should be noted, that the recognizer I am using was designed to recognize whole words, not single letters. The recognition accuracy on single characters is usually lower than the average performance of the system. This might be a consequence of the baseline calculation in the preprocessing step, that usually requires a certain length of the word to offer "reasonable" results (compare Figure 6.5). This makes this problem even harder. But if you restrict the user to do, e.g., only one deletion, like I did, then the dictionary size will reduce very much. Especially if the deleted substring is very short, usually more correct letters are left. But the more parts of the word are known, the more the search space reduces in general. Therefore even with a low recognition performance there is a good chance to get a correct result.

Even with the weakest **restriction** proposed in the previous section, i.e., **deletion (insertion) only of whole wrong (correct) substrings**, an enormous reduction of the search space can be received. Three cases can happen here at one step: deletion at 1. the begin, 2. in the middle, and 3. at the end of a word. In the first and the last case the right or left letter respectively from the deleted substring can be assumed to be correct. Therefore the inserted signal must be one of the substrings from the original dictionary that can be found before or after the occurrence of such a letter. In the second case, deletion in the middle, both, the left and the right, borders are known. The search for the correct inserted word can be done on a dictionary that contains all different substrings that appear between this two letters in the original dictionary. To show how this reduces the search space I calculated the size of this new dictionaries for every letter combination of the alphabet. The results, if both borders are lower case letters, can be found in Figure 7.6 and in Figure 7.7, if the left border is an upper case letter. It can be seen that even in the worst case (that is "a" as left and "e" as right border) the dictionary size reduces more than five times, i.e., from 51866 to 9827 words. If only the right border is known, i.e., the beginning of a word is deleted, the reductions are not that big, but still there is a remarkable increase in the size of the search space (see Figure 7.8). Figures 7.9 and 7.10 show the reductions, if the left border is known (Fig. 7.9 for lower, Fig. 7.10 for upper case letters).

For **the next two restrictions** I proposed, i.e., **delete all wrong letters at once** and **deletion of only one substring is allowed** the examples to show reductions of the dictionary size can not be found and evaluated so easy, since the shown reduction depends extremely on the used samples. Of course, the more correct letters you know, the smaller the dictionary size usually gets. The examples I will show here are the cases where everything except the first or the last two letters of a word are deleted. Figure 7.11 and 7.12 show the respective results of this analysis. If the first two letters are known, the maximum dictionary size reduces to 1017 for the two letters "co". If everything but the last two letters is deleted, the inserted sequence must be one of maximum 2296 (if the letters are "on"). Note that at the start lower and upper case letters are allowed for each word. Therefore the maximum reduced dictionary size for deletion at the word beginning is about twice as much as for deletion at the word end. It can be seen, that even if only two correct letters are known, the size of the search space is decreasing extremely.

This is a good example, how some restrictions in the interface, that are still acceptable for the user and do not miner the comfort of using the interface to

much, can help increase the probability of a higher recognition accuracy. Even with very simple and easy algorithms recognition performances can be improved.

Chapter 8

Conclusion

In the first part of this report empirical studies and the discussion of different issues concerning errors and repairs that usually happen in human handwriting of textual input were presented. The topic of this part was the problem of repair handling and the question, how pen based user interfaces can be designed to fulfill the two important aspects of a good recognition rate on one hand and a high usability on the other. Two approaches, that both have their needs and advantages, have been discussed here: repair of an input signal written by a human user and corrections of the recognized result, i.e., repair in a handwritten sequence of coordinates and in printed ASCII-text. **Two concepts** have been introduced to deal with errors and corrections in human handwriting: first, the concept of **error handling**, and second, the concept of **error avoidance**. The goal of error handling is to recover from occurring errors and mistakes by offering several tools and algorithms for that purpose. The concept of error avoidance contains all the techniques and methods used to avoid or reduce the occurrence of errors, for example, by the "clever" design of an interface and/or recognition algorithm.

For the case of **repair in a handwritten input signal** different heuristics have been proposed, discussed, and evaluated for the three repair classes, deletion, overwriting, and completion, introduced in part I. Deletions have been restricted to two typical deletion actions, i.e., relative long strokes and a lot of strokes in a narrow area. These gestures are not only the most common ones found in the database that was analyzed in part I, but also very general and intuitive to use. Classification has been done based on comparing the length and size of the repair strokes with the overwritten parts of the word. Overwrites are handled through comparing their bounding boxes with the ones of the overwritten strokes. Different heuristics have been tested to check the overlap of these bounding boxes and to decide, which strokes have to be removed from the original input signal. It turned out, that the one that removes the most strokes from the input signal received the best repair handling and recognition results. Several heuristics have been discussed and tested to insert the overwriting strokes into the remaining handwriting and to handle the repair type "completion".

The proposed heuristics are all very simple. This makes it easy to implement them in a way that handles repair very fast, which turned out to be an important issue. To evaluate the different methods some data was collected, where the user was asked to do some corrections. With this data an analysis of the repair handling heuristics was done. The achieved results showed some problems, which occurred with the proposed algorithms and with the repair handling task in general. For example, users tend to do their corrections in a "fuzzy" way, that is, their repair gestures do not fit exactly into the original input signal.

To solve this problems an interactive approach was evaluated, i.e., a repair tool was implemented, which uses the same heuristics but indicates the repair handling immediately to the writer. With the use of this interactivity several problems can be solved. A data set that contained repair types that are handled by the proposed heuristics was tested with a recognizer that achieves a word accuracy of about 88% with comparable "clean" data. In 65% of all cases the right word was recognized. Nearly 80% of the performed repair gestures were handled correctly. The methods for the repair type deletion achieved about the same results as the ones used for handling an overwrite. But they seemed to be to uncomfortable for the users, because it sometimes takes too long for the interface to detect and classify them. Since user acceptance is an important issue in the design of human-computer interfaces, this turns out to be a big problem. For that reason this should be the point where to start some future work in this task of repair in handwritten input signals.

It should be noted that the "pure" error handling approach, that is, the use of the proposed heuristics without interactivity, did not achieve satisfying results. But with the use of the immediate indication of the repair handling results, a significant improvement in the recognition accuracy was reached. This is a good example, how techniques that refer to the concept of error avoidance can help to overcome problems that are very hard to solve or impossible to solve at all with methods from the concept of error handling.

The other possibility where to offer some repair handling tools in automatic handwriting recognition is to allow corrections of the recognized result, i.e., **repair of printed text**. This has been done in the last chapter of this report. Some heuristics have been implemented to cover the repair cases of deletion and insertion to correct a wrong recognized word. Due to the nature of this task the assumption was made, that a restriction to these two kinds of corrections does not lower user acceptance and satisfaction in the special application that was the topic of this report. It turns out that repair in printed text is easier than the task of repair in a handwritten input signal, because some problems from the later one do not happen here. Repair detection and classification reduces to some very simple tests due to the restriction to two repair types (deletion and insertion) and the nature of the problem. The borders of a correction are much easier to specify in printed text than in cursive handwriting. Also the repaired entities (i.e., the printed text) has not to be divided from the repair signal (i.e., the handwriting), like it has to be done in the case of repair in a handwritten input. It was shown that some new arising problems, like an enormously increasing search space and the correct specification of a position for an insertion, can be solved by setting several restrictions to the user, i.e., allowing only some specific repair actions. With these conditions the problem of a lower recognition rate due to this additional problems can be solved very easyly. To avoid a low user acceptance, it was taken care, that these restrictions are chosen in a way that makes them not to uncomfortable for the users. With these modifications the search space for the correction of the wrong recognized output can be restricted, what usually influences recognition accuracy positively. The evaluation of the average recognition rate in this setup is very critical, since it depends extremely on the samples used for testing. It is very hard to specify a data set that guarantees statistically independent, relyable results. Therefore such tests have not been done here. Alternatively the thesis that the different restrictions made to the user result in a reduction of the size of the search space has been proven by numerous representative examples. The decrease of the search space size was that high that a corresponding increase in the recognition accuracy can be assumed to be true.

Summarized it can be said, that the repair task appeared to be a very hard one. The data analysis indicated, that there are problems that can not be solved by applying some simple heuristics to the handwritten input signal. In fact, there were examples in the different databases analyzed here, that make it doubtful, that this task can ever be solved completely, if applied in a separate step prior to the final recognition (at least with the common techniques and algorithms for handwriting recognition used today).

But it was also shown here, that repair handling in on-line handwriting recognition can be done. The key is not only to handle errors, but to try to avoid them. I introduced these two concepts, error handling and error avoidance, in the discussion about different error and repair handling issues in the introduction of this part of the report. In both cases, repair of handwritten input and of printed text, the "pure" error handling heuristics did not work very well. But with the additional use of some techniques to avoid errors satisfying results have been received. This is done in several ways, like, for example, using interactivity or allowing only some special repair actions that can be recognized easier than the general cases.

The more restrictions are made to a user, the more "rules of behavior" are proposed to him, the better the repair handling results and recognition rates can get. The problem is, that restrictive rules and guidelines usually lower the user acceptance. Therefore it was taken care in this work, that the restrictions and handicaps made here are still comfortable and intuitive to use for the writer. It should be noted in this context, that also in handwriting recognition of "clean" data, not containing any repair, several restrictions are usually made. For example, basing the recognition on a dictionary with a fixed size, i.e., the restriction, that the user is only allowed to write a word from this data set. Therefore it can be seen as a common approach to increase the recognition performance by assuming a special user behavior. The big challenge is to find a reasonable way between the two extremes "no restrictions and assumptions to the user behavior at all", usually resulting in high user acceptance, but low recognition accuracy, and "a lot of restrictions", usually resulting in low user acceptance, but high recognition accuracy.

Appendix A

Used Databases

The samples that were taken from the database introduced in section 2.1 to evaluate the different parameter settings and heuristics for repair handling are listed in Table A.1 till Table A.3. This data contains repair performed by the users without being asked to do some corrections. Table A.1 shows the samples used to evaluate the parameters for the case of detecting and handling a deletion. Note that the original signal was a handwritten word. Therefore an error does not necessary result from a misspelling. For example, the "Y" in the word "Ypon" (entry number 12 in the table) should not be seen as a wrong written letter, but as the character "U" having a "bad" or uncommon shape that looks more like a "Y" than a "U". Therefore the bold printed letters in the table represent the characters that a human reader would most likely recognize. It was not necessaryly the intention of the corresponding users to write these letters. Table A.2 indicates the data samples that contain an overwriting. Examples used to evaluate the parameter settings and the different heuristics in the completion case can be found in Table A.3.

A list with the data that was collected with the invitation to the writers to do some repair (see section 6.3.2) can be found in the Tables A.4 and A.5. The instructions given to the user were the following:

- First instruction:
  " Please write the word 'wrong word' "
- Second instruction:
  " Please correct: 'correct word' instead of 'wrong word' "

where *'wrong word'* is one of the words shown in the left columns of Table A.4 or A.5 and *'correct word'* is the corresponding one from the right column. No order of how to correct was given to the user.

In the interactive data collection (see section 6.3.4) the word collected in the first collection step from above was presented to the user. Then the second instruction was shown him again to do some corrections using the immediate indication of the performed repair actions.

Note that in the data collection process these lists have been randomly mixed to avoid an influence of the repair behavior of the different writers.

	target	writing
1.	es	**Es**
2.	imagin	imagine
3.	Services	**Faces** Services
4.	you in	**on you** you in
5.	sprang	**ap** sprang
6.	rum	**s** rum
7.	Nikoloff	**M** Nikoloff
8.	come	**l** come
9.	long	longer
10.	Fitz	Fritz
11.	voice	vocice
12.	Upon	**Ypon** Upon
13.	look	looke
14.	among	amoung
15.	puzzling	puzzeling
16.	destroying	**dayliq** destroying
17.	prosperous	propsperous
18.	newspaper	newspapers
19.	that	thlat
20.	like	**ll** like
21.	won	wone
22.	Palomares	**Po** Palomares
23.	on	**of** on
24.	upon	**on** upon
25.	the	**w** the
26.	whispered	**wis** whispered
27.	ostracism	**osta** ostracism
28.	narrow	**ma** narrow
29.	away	**an** away
30.	Princess	**B** Princess
31.	name among	narme among
32.	called	**coul** called
33.	must	**u** must
34.	Cologne	Colongne
35.	just	**js jcst** just
36.	feel	**se** feel
37.	beginner	**beginer** beginner
38.	screwdriver	**screwdrw** screwdriver
39.	knew	**ke** knew
40.	happened	**a** happened
41.	Three	**three** Three
42.	ten	**the** ten
43.	million	**mile** million
44.	workers	**working** workers
45.	like	**lif** like
46.	emergency	**energ** emergency
47.	she	shee
48.	wistfully	**wh** wistfully
49.	boundaries,	**boundr** boundaries
50.	nuclear	**ne** nuclear
51.	Ziegler	**S** Ziegler
52.	Ursuline	Ursucline
53.	high	heigh
54.	Yes	Yess
55.	guardhouse	**ga** guardhouse
56.	that	thate
57.	epela	**epell** epela
58.	is	**will** is
59.	cell	**cle** cell
60.	you	**g** you
61.	went	**when** went
62.	the	**to** the
63.	Melchisedec	**Mele** Melchisedec
64.	upon	**o** upon
65.	chamois	chamoise
66.	when	**the** when
67.	what	**that** what
68.	Dr	**S** Dr
69.	farad	**g** farad
70.	veterans	**events** veterans
71.	were	**are** were
72.	speak	**s**lpeak
73.	something	**to** something

Table A.1: Words that contain deletions. The bold printed letters in the right columns indicate the parts of the words that were crossed out.

	word	overwrite
1.	which	hich
2.	inaptitude	d
3.	Yugoslav	o
4.	resistive	s
5.	put	p
6.	handsom	ds
7.	license	c
8.	primose	o
9.	condescension	sc
10.	Salerno	le
11.	diminutive	(first) i
12.	its	its
13.	behind	bin
14.	than	h
15.	elegance	a
16.	believe	(first) e
17.	spewed	e
18.	responsibility	b
19.	repairmen	e
20.	ears	a
21.	eaving	e
22.	carp	p
23.	cage	g
24.	beadle	ead
25.	with	th
26.	workhorse	h
27.	documentation	e
28.	last	t
29.	did	d
30.	her	h
31.	did	(first) d
32.	and	n
33.	pupil	i
34.	scattergun	g
35.	March	a
36.	considered	ed
37.	governance	n
38.	dium	um
39.	Balkan	k
40.	Guiana	(half) G
41.	optometry	t
42.	prairie	ie
43.	par	r
44.	escapee	a
45.	bicep	c
46.	adsorb	d
47.	circus	r
48.	tortoiseshell	(first) e
49.	they	e
50.	feign	g
51.	miscreant	i
52.	mayor	y

Table A.2: Words that contain an overwrite. Left: the whole word; right: the part of the word that was overwritten.

	word	corrected letter
1.	that	a
2.	experiment	x
3.	expunge	x
4.	expose	x
5.	existence	x
6.	always	s
7.	patriarchy	h
8.	we	w
9.	exclusive	x
10.	approximately	a and x
11.	expansive	x
12.	devotion	d
13.	that	a
14.	loving	v
15.	lovers	v
16.	change	g
17.	the	t
18.	return	r
19.	despair	e
20.	expansive	x
21.	same	m
22.	Redstone	d
23.	bagpipe	p
24.	narrowly	o
25.	from	o
26.	existential	x
27.	benzene	e
28.	exportation	x

Table A.3: Words that contain a completion. Left: the whole word; right: the part/letter of the word that contained the completion.

wrong word	correct word
abee	bee
aback	back
caching	aching
chair	hair
emanuel	manuel
emission	mission
factor	actor
leach	each
beach	each
plane	lane
gab**b**le	gable
hal**e**comb	halcomb
igono**m**miny	igonominy
icke	ike
Approaching	Aproaching
lig**h**num	lignum
mag**h**dalene	magdalene
ob**s**tain	obtain
tab**b**oo	taboo
wa**c**ken	waken
Abscissae	Abscissa
cocoon**s**	cocoon
zimmerman**n**	zimmerman
stone**s**	stone
wolf**f**	wolf
absent	sent
aboriginal	original
accent	cent
across	ross
became	came
binuclear	nuclear
debase	base
mcdonald	donald
obsession	session
upside	side
back**e**nup	backup
cess**i**nna	cessna
dur**e**pont	dupont
li**r**aza	liza
plen**t**ium	plenum
bee**c**h	bee
austral**i**a	austral
automat**i**c	automat
bonn**e**t	bonn
complain**e**d	complain
epigram**m**e	epigram
here**t**o	here
mack**e**y	mack
pessimist**i**c	pessimist
real**l**y	real

wrong word	correct word
diffusion	fusion
innperuse	peruse
propose	pose
rimroad	road
forsign	sign
unamoment	moment
sig**for**ma	sigma
sim**s**enon	simon
war**r**anty	warty
zircon**con**	zircon
abram**son**	abram
diet**ary**	diet
mission**ary**	mission
write**up**	writ
villa**ger**	villa
turn**off**	turn
there**for**	there
steal**thy**	steal
polo**gon**	polo
nor**way**	nor
shight**see**	sight
highnoon	noon
withnell	nell
fromnorm	norm
highjag	jag
jack**boot**	jack
aver**ring**	aver
blan**chard**	blanc
consent**oray**	consent
ear**drum**	ear
dynamo**tics**	dynamo
free**hand**	free
full**back**	full
ground**work**	ground
house**boat**	house
in**forcitement**	incite
ku**nidoma**	kudo
onkuhning	kuhn
collabs	lab
laet**ther**	latter
mil**lter**	milt
nur**c**leons	nucleon
postordered	order
ra**na**mand	raman
sieg**fri**del	sigel
surly	– (whole word deleted)
truth	– (whole word deleted)
turbulent	– (whole word deleted)
rainwater	– (whole word deleted)
Coup	– (whole word deleted)

Table A.4: List with the collected data containing repair. Left: the "wrong" word written in the first data collection step; right: the "correct" word, i.e., the target of the corrections in the second data collection step. The wrong parts of a word are printed in bold style.

wrong word	correct word
zanbia	zambia
yosenite	yosemite
synnetry	symmetry
squauder	squander
spougy	spongy
rowhey	rowley
preeude	prelude
hazehut	hazelnut
fulhess	fullness
emersan	emerson
embraider	embroider
eaohan	eachan
drcam	dream
authcntic	authentic
blackyuard	blackguard
bluthe	blythe
bououza	bonanza
callegraph	calligraph
capetal	capital
capitol	capital
checher	checker
delinguent	delinquent
requlate	regulate
drauidian	dravidian
extrauagant	extravagant
fechless	feckless
fhnt	flint
joeene	joanne
krehs	krebs
laey	lacy
manlate	mandate
nickd	nickel
ovilorm	oviform
pany	pang
perleaps	perhaps
represe	reprise
shoy	shoji
lruger	kruger
eadle	ladle
nost	most
magma	magna
nase	nose
eption	option
qarquet	parquet
peck	perk
plartic	plastic
caster	raster
reserue	reserve
ureck	wreck
tasi	taxi

wrong word	correct word
jale	yale
wisard	wizard
sublle	subtle
pompa	pampa
pakietan	pakistan
roost	roast
audille	audible
esoape	escape
snycler	snyder
tokm	token
uninorm	uniform
upcrade	upgrade
velicle	vehicle
axeral	axial
baia	baja
bale	bake
cartin	carlin
dilnore	dilmore
diplonat	diplomat
dena	elena
errer	error
slop	stop
freguent	frequent
green	grain
grest	grist
hetch	hutch
innume	immune
bat	hat
rat	cat
prenorm	perform
degibity	debility
keeser	kaiser
policemen	policeman
postmister	postmaster
servize	service
shallow	shadow
simanese	siamese
wilderniss	wilderness
zombee	zombie
revue	reuse
roclkand	rockland
walh	wahl
sned	send
abel	able
tabel	table
smlet	smelt
feild	field
sumner	sunmer
rigth	right
recieved	received

Table A.5: List with the collected data containing repair (Cont.). Left: the "wrong" word written in the first data collection step; right: the "correct" word, i.e., the target of the corrections in the second data collection step. The wrong parts of a word are printed in bold style.

List of Figures

List of Tables

Bibliography

[AKLP93] Oscar E. Agazzi, Shyh-Shiaw Kuo, Esther Levin, and Roberto Pieraccini. Connected and degraded text recognition using planar hidden markov models. *Proceedings of the IEEE*, 1993.

[BBNN] E. J. Bellegarda, J. R. Bellegarda, D. Nahamoo, and K. S. Nathan. A fast statistical mixture algorithm for on-line handwriting recognition. *Proceedings of the IEEE.*

[BDS92] John Bear, John Dowding, and Elizabeth Shriberg. Detection and correction of repairs in human-computer dialog. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 1992.

[BKZ94] Jerome R. Bellegarda, Dimitri Kanevsky, and Wlodek Zadrozny. Error correction and add-word strategy in automatic speech recognition. *Research Report, IBM Research Devision, T.J. Watson Research Center*, 1994.

[Bli97] Conrad H. Blickenstorfer. A new look at handwriting recognition. *Pen computing magazine*, (4):76–81, 1997.

[BM93] Uli Bodenhausen and Stefan Manke. Automatically structured neural networks for handwritten character and word recognition. In *Proceedings of the International Conference on Neural Networks, San Francisco*, 1993.

[BMW93] Uli Bodenhausen, Stefan Manke, and Alex Waibel. Connectionist architectural learning for high performance character and speech recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Minneapolis*, 1993.

[CKJ90] Chang-Keng and Bor-Shenn Jeng. On-line recognition of handwritten chinese characters and alphabets. *Proceedings of the IEEE*, 1990.

[ea95] R. A. Cole (et al), editor. *Survey of the State of the Art in Human Language Technology.* published in the WWW under http://www.cse.ogi.edu/CSLU/HLTsurvey/HLTsurvey.html, 1995.

[FHM95] Clive Frankish, Richard Hull, and Pam Morgan. Recognition accuracy and user acceptance of pen interfaces. In *Conference on Human Factors in Computing Systems*, CHI '95 Proceedings, 1995.

[Gro97] Ralph Groß. Incremental preprocessing and recognition in an on-line handwriting recognition system. Studienarbeit, Universität Karlsruhe, Institut für Logik, Komplexität und Deduktionssysteme, 1997.

[Hol81] J.M. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:337–372, 1981.

[KA93] S. Kuo and O. E. Agazzi. Visual keyword recognition using hidden marcov models. *Proceedings of the IEEE*, 1993.

[Kas95] Robert Howard Kassel. *A Comparison of Approaches to On-Line Handwritten Character Recognition.* Phd thesis, Massachusetts Institute of Technology, 1995.

[LMT96] Arjan B. M. Lelivelt, Ruud G. J. Meulenbroek, and Arnold J. W. M. Thomassen. Mapping abstract main axes in handwriting to hand and finger joints. In *Handwriting and Drawing Research: Basic and Applied Issues*, IOS Press. M.L. Simner, C. G. Leedhan, and A. J. W. M. Thomassen, 1996.

[MB94] Stefan Manke and Uli Bodenhausen. A connectionist recognizer for on-line cursive handwriting recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Adelaide*, 1994.

[MCvMK95] Thomas P. Moran, Patric Chiu, William van Melle, and Gordon Kurtenbach. Implicit structures for pen-based systems within a freeform interaction paradigm. *CHI Proceedings*, 1995.

[MFW94] Stefan Manke, Michael Finke, and Alex Waibel. Combining bitmaps with dynamic writing information for on-line handwriting recognition. In *Proceedings of the International Conference on Pattern Recognition, Jerusalem*, 1994.

[MFW95a] Stefan Manke, Michael Finke, and Alex Waibel. NPen^{++}: A writer independent, large vocabulary on-line cursive handwriting recognition system. In *Proceedings of the International Conference on Document Analysis and Recognition*, IEEE Computer Society, August 1995.

[MFW95b] Stefan Manke, Michael Finke, and Alex Waibel. The use of dynamic writing information in a connectionistic on-line cursive handwriting recognition system. In *Advances in Neural Information Processing 7*, MIT Press, Cambridge (MA), 1995.

[MFW96] Stefan Manke, Michael Finke, and Alex Waibel. A fast search technique for large vocabulary on-line handwriting recognition. In *Proceedings of the International Workshop on Frontiers in Handwriting Recognition, Colchester, England*, 1996.

[MSY92] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7), 1992.

[NBS+95] Krishna S. Nathan, Homayoon S. M. Beigi, Jayashree Subrahmonia, Gregory J. Clary, and Hiroshi Maruyama. Real-time on-line unconstrained handwriting recognition using statistical methods. *Proceedings of the IEEE*, 1995.

[Pom92] D.A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* Phd thesis, Carnegie Mellon University, Pittsburgh, February 1992.

[Rab90] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in Speech Recognition, Alex Waibel and Kai-Fu Lee (ed.)*, pages 267–296, 1990.

[Sch94] L. R. B. Schomaker. User-interface aspects in recognizing connected-cursive handwriting. In *The Institution of Electrical Engineers*, Proceedings of the IEE Colloquium on Handwriting and Pen-based input, March 1994.

[SHTA93] L.R.B. Schomaker, Eric H. Helsper, H.L. Teulings, and G.H. Abbink. Adaptive recognition of online, cursive handwriting. 6th International Conference on Handwriting and Drawing (ICOHD'93), Paris, France, July 1993.

[SMW96] Bernhard Suhm, Brad Myers, and Alex Waibel. Interactive error recovery for speech user interfaces. Proc. ICSLP, Philadelphia USA, 1996.

[ST94] Y. Singer and N. Tishby. Dynamical encoding of cursive handwriting. *Biological Cybernetics*, 71, 1994.

[VHY+95] M.T. Vo, R. Houghton, J. Yang, U. Bub, U. Meier, A. Waibel, and P. Duchnowski. Multimodal learning interfaces. *ARPA Spoken Language Technology Workshop*, 1995.

[VW93] M. T. Vo and A. Waibel. Multimodal human-computer interaction. *Proceedings of ISSD*, 1993.

[WD94] A. Waibel and P. Duchnowski. Connectionist models in multimodal human-computer interaction. *Proceedings of the Government Microcircuit Applications Conference (GOMAC)*, 1994.

[WHHS90] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, and Kiyohiro Shikano. Phoneme recognition using time-delay neural networks. *Readings in Speech Recognition, Alex Waibel and Kai-Fu Lee (ed.)*, pages 393–404, 1990.

[Win96] Hans-Jürgen Winkler. HMM-based handwritten symbol recognition using on-line and off-line features. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1996.

[WMO92] T. Wakahara, H. Murase, and K. Odaka. On-line handwriting recognition. *Proceedings of the IEEE*, 80(7), 1992.

[WVDM95] A. Waibel, M.T. Vo, P. Duchnowski, and S. Manke. Multimodal interfaces. *Artificial Intelligence Review Jounal, special issue*, 1995.

Diplomarbeit

(deutsche Kurzfassung)

Repair in On-Line Handwriting Recognition

von

Wolfgang Hürst

angefertigt an der

Carnegie Mellon University
Pittsburgh PA 15213
U.S.A.

Betreuer:

Prof. Dr. Alex Waibel
und Dr. Jie Yang

März 1997

Repair in On-Line Handwriting Recognition

Wolfgang Hürst

Interactive Systems Laboratories
Carnegie Mellon University, Pittsburgh PA, U.S.A
Universität Karlsruhe (TH), Karlsruhe, Germany

Im folgenden sind die wesentlichen Inhalte, Ideen, Algorithmen und Ergebnisse meiner Diplomarbeit "Repair in On-Line Handwriting Recognition" zusammengefaßt, die an der Carnegie Mellon University in Pittsburgh, PA, in den U.S.A. angefertigt wurde. Für weitere Ausführungen sei auf die englische Ausarbeitung verwiesen. Dort ist auch eine ausführliche Liste der verwendeten Literatur zu finden.

Abstract

Im Bereich der Mensch-Maschine Kommunikation bieten multimodale Benutzerschnittstellen große Vorteile. Typische Beispiele und Bereiche sind Sprach- und Handschrifterkenner, Gesichtsverfolgung, Gestikerkennung, etc. Bei der Entwicklung solcher Systeme gibt es zwei wichtige Aspekte: Erkennungsleistung und Benutzerzufriedenheit/-akzeptanz. Nur wenn beide Bereiche in einer vernünftigen und brauchbaren Art und Weise realisiert sind, wird ein solches Interface sinnvoll und für die tägliche Anwendung nutzbar sein. Während bei den Erkennungsraten bereits beachtliche Erfolge erzielt werden konnten, wurde der zweite Bereich in der Vergangenheit eher vernachlässigt. Ein Ansatz zur Verbesserung des Benutzerkomforts ist es, verschiedene Methoden zur Behandlung von Verbesserungen und Korrekturen (Repair) in ein System zur multimodalen Mensch-Maschine Kommunikation zu integrieren. Vorliegende Arbeit beschäftigt sich mit dem Thema von Korrekturen bei der automatischen on-line Handschrifterkennung einzelner Worte. On-line bedeutet, daß die Erkennungsalgorithmen nicht nur auf der Basis der Bitmap einer Handschrift, sondern auch unter Zuhilfenahme dynamischer Schreibinformation ablaufen. Ausgehend von der These, daß Korrekturen in von Menschen geschriebenen Texten und Fehler in den von der Maschine gelieferten Ergebnissen vorkommen (und immer vorkommen werden), werden hier einige Heuristiken und Algorithmen vorgestellt, um die sich dadurch ergebenden Probleme zu beheben. Vorangehende These wird durch diverse empirische Studien belegt. Diese Untersuchungen dienen auch als Basis für eine Klassifikation typischer Fehler und Korrekturen in der automatischen Handschrifterkennung. Algorithmen und Heuristiken für zwei Arten von Korrekturen, erstens Verbesserungen in einem handgeschriebenen Signal, zweitens in gedrucktem Text, also die Korrektur eines Erkennungsergebnisses, werden vorgestellt. Diese Heuristiken werden an diversen Beispieldaten evaluiert, und einige Ideen für einen besseren Interfaceentwurf in der automatischen Handschrifterkennung werden diskutiert.

1 Einführung

Multimodale Benutzerschnittstellen, wie beispielsweise Handschrift-, Sprach- oder Gestikerkenner, bieten große Vorteile im Bereich der Mensch-Maschine Kommunikation. Beim Entwurf solcher Systeme spielen zwei Dinge eine wesentliche Rolle: Erkennungsleistung (recognition accuracy, [6]) und Benutzerzufriedenheit (user acceptance). Da die sinnvolle Nutzung eines Interfaces ohne hinreichende Erkennungsleistung nicht praktikable ist, wurde das Hauptaugenmerk bei der Entwicklung von Sprach-, Handschrift-, Gestikerkennern, usw., in der Vergangenheit hauptsächlich auf den ersten Punkt, das Erzielen einer möglichst hohen Erkennungsrate gelegt. Mittlerweile wurden hier Ergebnisse erzielt, die einen Nutzen in der praktischen Anwendung zulassen. Dadurch erhält der zweite Bereich, die Benutzerzufriedenheit und Akzeptanz durch den Anwender, eine höhere Bedeutung. Selbst mit hundertprozentigen Erkennungsraten wird ein Modul zur Mensch-Maschine Kommunikation nicht benutzt werden, wenn es dem Anwender keine Flexibilität und keinen Komfort bietet, sowie keine einfache Bedienung erlaubt.

Vorliegende Arbeit beschäftigt sich mit dem Thema von Korrekturen (Repair) in der On-Line Handschrifterkennung. Bei der automatischen Handschrifterkennung wird versucht, ein von einem Menschen handgeschriebenes Eingabesignal in einen maschinenlesbaren Code (ASCII-Text) umzuwandeln. Man unterscheidet hier zwischen Optical Character Recognition (OCR) und On-Line Erkennung ([1, 2]). Bei der OCR liegt das Eingabesignal nur in Form einer Bitmap vor, d.h. ein geschriebener Text wird nur durch eine Menge diverser x,y-Koordinatenpaare repräsentiert. Bei der On-Line Erkennung dagegen existiert ebenfalls Information über den Zeitpunkt, wann ein bestimmter Koordinatenpunkt geschrieben wurde. Dieses zusätzliche Wissen kann im Erkennungsprozeß sinnvoll genutzt werden und führt somit in der Regel zu besseren Ergebnissen.

Wie bereits erwähnt, werden im Bereich der Erkennungsleistung bereits beachtliche Resultate erzielt. Die Möglichkeit, Korrekturen zu erkennen und zu behandeln wurde allerdings in den vergangenen Jahren eher spärlich behandelt. Eine Motivation, warum dies jedoch ein extrem wichtiges Thema ist, und eine Untersuchung, welche Formen von Fehlern und Korrekturen in handgeschriebenen Worten üblicherweise vorkommen, wird im nächsten Kapitel gegeben. In den folgenden Abschnitten werden verschiedene Heuristiken und Algorithmen besprochen, die versuchen, dieses Problem zu lösen.

2 Fehler und Korrekturen bei der maschinellen Handschrifterkennung

Fehler bei der automatischen Handschrifterkennung können in zwei Bereichen vorkommen: zum einen auf der Benutzerseite und zum anderen auf Seiten der Maschine, bzw. des Erkennungsalgorithmus. Die Tatsache, daß selbst Menschen nicht in der Lage sind, eine hundertprozentig richtige Erkennung auf zufällig ausgewählten handgeschriebenen Daten durchzuführen (dies wurde in [5] durch empirische Studien bewiesen), läßt darauf schließen, daß es unwahrscheinlich

ist, daß je eine "perfekte" maschinelle Erkennung erzielt werden kann. Im Rahmen dieser Arbeit wurde eine empirische Studie mit einer Datenbank diverser handgeschriebener Texte durchgeführt, die bewies, daß Fehler und Korrekturen in von Menschen geschriebenen Worten vorkommen. Die Tatsache, daß Verbesserungen in den Elementen dieser Datenbasis auftraten, obwohl die jeweiligen Schreiber nicht dazu aufgefordert wurden, Korrekturen durchzuführen, und es keinen Feedback eines etwaigen Erkennungsalgorithmus gab, läßt die Folgerung zu, daß man bei der Entwicklung und dem Einsatz von Modulen zur automatischen Handschrifterkennung immer mit solchen Eingabedaten rechnen muß. Ein weiteres Argument für die Notwendigkeit diverser Korrekturmechanismen ist die Tatsache, daß selbst eine Tastatur, die üblicherweise zur Dateneingabe benutzt wird, verschiedene Möglichkeiten anbietet, Verbesserungen durchzuführen (z.B. durch den Einsatz einer "backspace"-Taste), obwohl in der Regel eine hundertprozentige Erkennungsrate vorliegt.

Die erwähnte Datenbasis wurde dahingehend untersucht, welche Fehler typischerweise in einer Handschrift und deren Erkennung vorkommen. Durch Literaturrecherche konnte folgende Klassifikation von gängigen Fehlern bei der automatischen Handschrifterkennung gefunden werden (vgl. [5]):

- "device-generated errors"; z.B. technische Störungen im Eingabegerät.
- "badly spelled words"; z.B. ein Wort, das einen Buchstaben zuviel oder zuwenig enthält.
- "input legible by humans but not by the algorithm"; z.B. sich überschneidende Buchstaben, wenn ein Erkennungsalgorithmus verwendet wird, der darauf konzipiert wurde, nur sauber voneinander getrennte Buchstaben zu erkennen.
- "badly formed shapes"; z.B. der Buchstabe "n", geschrieben wie ein "u".
- "unknown words"; z.B. Zahlen, wenn das Erkennungsvokabular auf Buchstaben eingeschränkt ist.
- "discrete noise events"; z.B. Linien und Punkte, die dadurch entstanden sind, daß der Stift über der Schreiboberfläche fallen gelassen wurde.
- "canceled material"; z.B. Ausstreichungen bestimmter Wortteile oder ganzer Worte.

Die so gefundenen fehlerhaften Daten wurden daraufhin untersucht, ob in irgendeiner Weise Korrekturen und Verbesserungen von Seiten des Benutzers auftraten. Diese "Repair-Daten" dienten als Basis für eine Klassifikation verschiedener Korrekturarten, die in handgeschriebenen Texten üblicherweise vorkommen. Die einzelnen "Repair-Klassen" sind:

- "Deletions" (Auslöschungen): der Benutzer macht undefinierbare Striche in einem Bereich, in den er zuvor geschrieben hat, mit der Absicht, die entsprechenden Wortteile zu löschen.

- "Completions" und "Insertions" (Vervollständigungen und Einfügungen): der Schreiber fügt einen oder mehrere Striche zu den bereits geschriebenen Wortteilen hinzu.
- "Overwriting" (Überschreiben): bestimmte Teile eines Wortes werden überschrieben, mit dem Ziel, den darunterliegenden Text durch die Korrektur zu ersetzen.

Es sei darauf hingewiesen, daß weitere Klassifikationen möglich und die Grenzen zwischen den verschiedenen Klassen häufig nicht eindeutig sind. Insbesondere die Unterscheidung zwischen "Completion" und "Overwriting" ist oft nicht ohne weiteres möglich. Die Einteilung in diese drei Klassen orientiert sich hauptsächlich daran, welche Reaktionsart ein Tool, das diese Korrekturen vor einer eigentlichen Erkennung behandeln soll, durchführen müßte. Im Falle eines Löschens oder Ausstreichens ("Deletion") sollten sowohl die überschriebenen Wortteile gelöscht werden, als auch die Teile der Eingabe, die die Korrektur bilden. Beim Überschreiben hingegen werden nur die überschriebenen Striche gelöscht, die "Überschreiber" jedoch müssen in die verbleibende Sequenz eingefügt werden. Der Korrekturtyp Vervollständigen und Eingfügen ("Completion/Insertion") erfordert lediglich, daß die Korrekturzeichen in die Originalsequenz von handgeschriebenen Daten eingefügt werden. Ein Löschen ist in diesem Fall nicht erforderlich.

Das bemerkenswerteste Ergebnis der oben angesprochenen Datenanalyse war die Feststellung, daß, wenn auch nur in geringem Rahmen, Korrekturen und Verbesserungen in den handgeschriebenen Texten vorkamen, obwohl die Schreiber lediglich darum gebeten wurden, einen vorgegebenen Text abzuschreiben. Sie wurden dazu aufgefordert, dies fehlerfrei und ohne Korrekturlesen zu tun. Dies ist ein starkes Argument dafür, daß ein menschlicher Benutzer nicht nur Fehler macht, sondern auch nach einer Möglichkeit verlangt, diese Fehler korrigieren zu können.

Im folgenden werden diverse Heuristiken besprochen und vorgestellt, die sich diesem Problem annehmen. Dabei wird unterschieden zwischen Korrekturen in handgeschriebenen Eingaben und gedrucktem ASCII-Text. Ersteres entspricht einem Verbessern von Fehlern in der eigenen Handschrift. Verschiedene Heuristiken für die eingeführten Korrekturklassen werden vorgestellt und besprochen. Das Verbessern von gedrucktem Text entspricht einer Korrektur des vom Computer angezeigten Erkennungsergebnisses durch eine handgeschriebene Benutzereingabe. Beide Korrekturarten haben in unterschiedlichen Situationen und unter verschiedenen Voraussetzungen ihre Vorteile und ihre Existenzberechtigung.

3 Korrekturen in handgeschriebenem Text

Allgemein gibt es für multimodale Benutzerschnittstellen zwei verschiedene Vorgehensweisen, das Problem auftretender Fehler und daraus resultierender Korrekturen in den Griff zu bekommen. Diese wurden in dieser Arbeit in zwei Konzepten manifestiert:

1. das Konzept der Fehlerbehandlung ("the concept of error handling"), d.h., der Versuch, auftretende Fehler zu erkennen und zu beheben, sei es durch Integration in den regulären Erkennungsprozeß oder beispielsweise durch das Anbieten diverser Möglichkeiten zur Korrektur.

2. das Konzept der Fehlervermeidung ("the concept of error avoidance"), d.h., durch einen "geschickten" Entwurf des Interfaces wird das Auftreten von Fehlern verringert oder ganz unterdrückt.

Beim Entwurf eines brauchbaren und zuverlässigen Interfaces muß man sich verschiedener Methoden aus beiden Bereichen bedienen, um ein für den Benutzer verwendbares und "angenehmes" Tool zur Verfügung stellen zu können. In Kapitel 2 wurde, basierend auf einer Datenbankanalyse, eine Definition verschiedener Klassen von Korrekturen in handgeschriebenen Texten eingeführt. Im folgenden sollen nun diverse Heuristiken vorgestellt werden, die sich dieser Problematik annehmen.

3.1 Klassifikation

Verschiedene Gründe (vgl. die Ausführungen in Kapitel 6 der englischsprachigen Ausarbeitung), legen es nahe, die Behandlung von Korrekturen in einem getrennten Vorverarbeitungsschritt durchzuführen. Hierzu ist zunächst eine Trennung des ursprünglichen Eingabesignals von den Teilen, die die Verbesserung bilden, durchzuführen. Diese Klassifikation in "reguläre" Handschrift und Korrektursignal wird hier mittels zweier Heuristiken durchgeführt.

Ein typisches Merkmal für das Auftreten eines "Repairs" in einem handgeschriebenen Eingangssignal ist, daß der Schreiber mit seinem Stift zurückgeht in einen Bereich, in den er bereits zuvor geschrieben hat, und zusätzliche Striche zu seiner Handschrift hinzufügt. Das heißt, zeitlich relative weit auseinanderliegende Teile des Eingabesignals haben einen relativ geringen Abstand voneinander bezüglich der x-Koordinatenrichtung. Dies wird in einer ersten Heuristik zur Klassifikation von Korrekturen genutzt. Mit der aktuellen Stiftposition wird ein Schwellwert entgegen der Schreibrichtung mitgeführt. Beginnt ein Benutzer nun links von diesem Schwellwert zu schreiben, werden alle folgenden Koordinatenpunkte als Korrektur interpretiert, bis der Stift wieder abgesetzt wird. Um unabhängig von verschiedenen Schreibstilen zu sein, wird der Schwellwert adaptiv an die aktuelle Schriftgröße angepaßt. Dieser dynamische Schwellwert berechnet sich aus der maximalen Breite der letzten fünf "up-down strokes" zuzüglich eines von Hand eingestellten Offsets. Ein "up-down stoke" sei hier definiert als alle Koordinatenpunkte des handgeschriebenen Textes zwischen zwei aufeinanderfolgenden lokalen Extrema. Eine solche dynamische Einstellung des Schwellwertes ist nötig, da nicht nur der Schreibstil unterschiedlicher Benutzer, sondern selbst der einzelner Personen sich ändern kann und zwar nicht nur über mehrere Worte hinweg, sondern sogar innerhalb eines einzigen Wortes. Ein wohlbekanntes Phänomen ist beispielsweise die Tatsache, daß die Schriftgröße einzelner Buchstaben bei vielen Schreibern zum Wortende hin kleiner, dafür aber breiter wird.

Alternative Ansätze wurden ebenfalls untersucht und ausgewertet, konnten jedoch keine vergleichbaren Erfolge erzielen, weshalb sie hier nicht aufgeführt werden. Der interessierte Leser sei auf die englischsprachige Ausarbeitung verwiesen. Ferner sei hier nocheinmal darauf hingewiesen, daß wir hier an Korrekturen für einen on-line basierten Erkennungsprozeß interessiert sind. Da die Erkennung in diesem Fall auf dem zeitlichen Eingabesignal basiert, kann *jedes* Zurückgehen in x-Koordinatenrichtung als "Repair" interpretiert werden. Dies ist im Falle einer OCR (Optical Character Recognition) nicht der Fall.

Mit oben eingeführter Methode lassen sich Verbesserungen im letzten Buchstaben eines Wortes nicht erkennen. Es sei hier bemerkt, daß in diesem Fall von den im vorherigen Abschnitt eingeführten Korrekturklassen nur "Löschen" und "Überschreiben" vorkommen können. Diese werden klassifiziert, indem die Anzahl der "up-down strokes" *rechts* von dem oben definierten dynamischen Schwellwert betrachtet wird. Liegt diese über einer von Hand vorgegebenen Anzahl, die sich am Durchschnittswert in "normaler" Handschrift ohne Korrektur orientiert, ist dies ein deutliches Indiz für das Vorkommen einer Verbesserung. Im folgenden Kapitel werden Heuristiken für die Klassifikation und Behandlung der einzelnen Korrekturtypen besprochen.

3.2 Behandlung unterschiedlicher Korrekturarten

3.2.1 Löschen und Ausstreichen

Zur Behandlung der ersten Korrekturklasse, Löschen und Ausstreichen, wurden die entsprechenden Beispiele aus oben erwähnter Datenbasis dahingehend untersucht, welche Merkmale für diesen Typ von Verbesserung typisch und daher für eine Klassifikation geeignet sind. Zwei charakteristische Sorten von Auslöschungen konnten gefunden werden:

- im Vergleich zur "normalen" Handschrift relativ lange Striche

und

- viele Striche auf einem relativ kleinen Bereich.

Zur Klassifikation der ersten Klasse wurden jeweils die Höhe und Breite (die Ausdehnung in x- und y-Koordinatenrichtung) der "up-down strokes" berechnet. Liegt einer dieser Werte erheblich über dem Durchschnitt, werden die entsprechenden Koordinaten als "Ausstreicher" und alles von ihnen Überschriebene als gelöscht interpretiert.

Eine weitere Heuristik deckt den zweiten Fall, viele Korrekturstriche im gleichen Bereich, ab. Hierbei wird die Gesamtlänge der zuvor klassifizierten Korrekturzeichen mit der Länge, bzw. Anzahl, der von ihnen überschriebenen Koordinaten verglichen. Ist die Anzahl der Korrekturkoordinaten wesentlich höher als die der entsprechenden "normalen" Handschrift, wird ein Löschen angenommen. Wie im obigen Fall werden nun sowohl die Korrektur- als auch die ausgestrichenen Koordinaten vom ursprünglichen Eingangssignal entfernt. Nach einer

Skalierung der x-Koordinatenwerte der Handschrift rechts von der Löschstelle, kann das korrigierte Signal an den Erkenner weitergeleitet werden. Diese beiden Ausstreicharten haben gegenüber den Gestiken, die herkömmlicherweise zur Erkennung von Auslöschungen verwendet werden, den Vorteil, daß keine Vorgabe bezüglich einer bestimmten Form gemacht wird. Dem Benutzer wird diesbezüglich eine relativ große Freiheit eingeräumt. Lediglich an bestimmte Merkmale und Eigenschaften der verwendeten Korrekturzeichen werden gewisse Anforderungen gestellt. Form, Größe und Aussehen sind für die Erkennung irrelevant.

3.2.2 Überschreiben

Für den Korrekturtyp Überschreiben wurden verschiedenen Heuristiken implementiert, verglichen und ausgewertet. Sämtliche Ansätze basieren auf einem Vergleich der Bounding Boxes der Korrektur- mit denen der ursprünglichen Handschriftdaten. Die Bounding Box einer Menge von Koordinaten ist definiert als das kleinstmögliche Rechteck mit Seiten parallel zur Horizontalen und Vertikalen, das alle Koordinatenpunkte der Menge beinhaltet. Die Bounding Boxes aller "up-down strokes", die von den Korrekturgestiken überschrieben worden sind, werden mit denen der Verbesserungen verglichen. Existiert eine "hinreichend große" Überlappung der beiden Bereiche, wird der überschriebene "up-down stroke" (= alle Koordinaten zwischen zwei aufeinanderfolgenden lokalen Extrema) aus dem Handschriftsignal entfernt. Es hat sich herausgestellt, daß ein Löschen oder "Herausnehmen" einzelner Wortteile auf der Basis von "up-down strokes" eine sinnvolle Realisierung darstellt, die bessere Ergebnisse liefert, als ein verallgemeinerter Ansatz, der es erlaubt, beliebige Wortteile und Koordinatenmengen zu löschen. Die behandelten Heuristiken funktionieren alle nach diesem Prinzip. Einziger Unterschied ist eine unterschiedliche Interpretation der Eigenschaft "hinreichend große" Überlappung. Insgesamt wurden sechs verschiedene Ansätze untersucht. Diese Methoden liegen zwischen den beiden Extremen "entferne möglichst wenig" und "entferne möglichst viel" vom ursprünglichen Signal. Im ersten, sehr restriktiven, Fall bedeutet dies: "lösche einen "up-down stroke" nur dann, wenn er vollständig von einer Bounding Box überdeckt wird". Im zweiten Fall dagegen wird ein "up-down stroke" bereits gelöscht, wenn seine Bounding Box sich nur ein bischen mit der eines Korrekturzeichens überlappt. Ferner wurden unterschiedliche Ansätze mit diversen Schwellwerten, etc., implementiert und getestet. Bei der Auswertung stellte sich heraus, daß die Methode die besten Ergebnisse liefert, die den zweiten Extremfall darstellt, d.h., die am wahrscheinlichsten einen überschriebenen "up-down stroke" aus dem ursprünglichen Signal entfernt. Dieses Resultat gilt sowohl für die erzielten Erkennungsraten als auch für die reine Behandlung der Korrektur in einem optischen Check. Damit ist die Frage gemeint, ob das durch die Korrektur entstandene neue Eingabesignal vom Menschen richtig erkannt werden kann (salopp gesagt: werden genau die Teile des Wortes gelöscht, die gelöscht werden sollen oder nicht?)

Nachdem auf diese Weise diverse Teile der ursprünglichen Handschrift entfernt wurden, muß nun die Korrektur in das verbleibende Signal eingefügt

werden. Dies geschieht mit einer der Heuristiken, die im nächsten Abschnitt erläutert werden.

3.2.3 Vervollständigen und Einfügen

In Falle einer Vervollständigung muß kein Teil des ursprünglichen handgeschriebenen Signals gelöscht werden. Lediglich ein Einfügen der Korrektur ist erforderlich. Dies entspricht der gleichen Situation wie im Falle eines Überschreibens, nachdem die überschriebenen Teile des Wortes entfernt wurden. Deshalb werden hier auch die gleichen Heuristiken verwendet. Insgesamt wurden zehn unterschiedliche Methoden implementiert, die aufgrund diverser Kriterien Einfügungen an unterschiedlichen Stellen vornehmen. Diese wurden anhand einer Datenbasis ausgewertet. Es stellte sich heraus, daß die Ansätze, die sich an den jeweiligen x-Koordinatenwerten orientierten, die besten Ergebnisse erzielten. Eine Korrektur wurde hierbei so zwischen zwei "up-down strokes" eingefügt, daß die Summe aus der Strecke zwischen dem vorangehendem "up-down stroke" und dem Anfang der Korrektur und der Strecke zwischen Ende der Korrektur und dem folgenden "up-down stroke" minimal ist. Mit Strecke ist hier der Abstand bezüglich der x-Koordinatenwerte gemeint.

Es sei hier darauf hingewiesen, daß im letzten Fall der vorgeschlagenen Korrekturklassifikation, Vervollständigungen und Einfügungen, hier nur Verbesserungen *innerhalb* des bereits geschriebenen Wortes behandelt werden (d.h., das Einfügen durch Schreiben, z.B. eines Buchstabens, über oder unter ein Wort wird mit den vorgestellten Heuristiken nicht abgedeckt).

3.3 Zusammenspiel der einzelnen Heuristiken

Nachdem im vorangegangenen Abschnitt verschiedene Heuristiken für die einzelnen Korrekturklassen getrennt voneinander eingeführt und ausgewertet wurden, soll nun beschrieben werden, wie die einzelnen Verfahren zusammen in ein einziges Korrektur-Tool integriert werden können.

Zunächst wird mit den in Abschnitt 3.1 eingeführten Methoden festgestellt, ob eine Verbesserung stattgefunden hat, und im positiven Fall das Korrektursignal von der ursprünglichen Handschrift getrennt. Daraufhin werden nacheinander die Fälle "Löschen", "Überschreiben" und "Vervollständigen/Einfügen" mit den jeweiligen Heuristiken behandelt. Zunächst wird überprüft, ob ein Löschen vorliegt. Dies geschieht gemäß der oben beschriebenen Verfahren durch diverse Längen- und Größenvergleiche. Wird ein Ausstreichen festgestellt, werden sowohl die ausgestrichenen Teile als auch die "Ausstreicher" vom Eingangssignal entfernt und die x-Koordinatenwerte rechts der Löschstelle um deren Breite nach links verschoben. Die Behandlung der Korrektur ist damit abgeschlossen. Findet kein Löschen statt (d.h., ist das Ergebnis sämtlicher Längen- und Größenvergleiche negativ) wird getestet, ob ein Überschreiben vorliegt. Dabei werden die überschriebenen Teile des Wortes durch Bounding Box-Vergleich gemäß einer der implementierten Heuristiken aus dem ursprüngliche Signal entfernt. Die Korrektur wird nun in die verbleibende Handschrift eingefügt. Dies

geschieht, wie bereits erwähnt, mit den gleichen Methoden, die angewendet werden, sollte kein Überschreiben auftreten.

Durch die Ausführung der einzelnen Tests nacheinander und ein Abbrechen der Korrekturbehandlung im Erfolgsfalle ist keine explizite Klassifikation und Trennung der einzelnen Korrekturklassen erforderlich. Ein Überschreiben kann nur vorkommen, wenn kein Löschen vorliegt. Wurden auch keine Wortteile überschrieben, muß die entsprechende Verbesserung ein Vervollständigen, bzw. Einfügen sein.

Um die implementierten Heuristiken auswerten zu können, wurden einige Daten gesammelt, die Korrekturen enthalten. Dabei wurden die einzelnen Schreiber, im Gegensatz zu der Datenbasis, die für die empirischen Studien über Fehler und Verbesserungen verwendet wurden, dazu aufgefordert, Korrekturen durchzuführen. Dies geschah, indem der jeweilige Benutzer zuerst ein Wort "falsch" schreiben und es anschliessend verbessern sollte. Hierzu wurden ihm Anweisungen der folgenden Art gegeben: 1. Bitte schreiben Sie das Wort "chair", 2. Bitte korrigieren Sie: "hair" anstelle von "chair". Die zweite Aufforderung wurde dem Schreiber erst angezeigt, nachdem das Wort der ersten vollständig geschrieben war. Es sei hier darauf hingewiesen, daß solch eine Datensammlung extrem schwierig ist, da jegliche Aufforderung an einen Benutzer, eine Verbesserung durchzuführen, sein Korrekturverhalten beeinflussen kann und somit möglicherweise zu einem verfälschten, statistisch nicht repräsentativen Ergebnis führen kann. Aus diesem Grund wurde hier besonderer Wert darauf gelegt, die Daten so auszuwählen, daß eine gute Bewertung der vorgestellten Heuristiken möglich ist (z.B. durch eine zufällig Auswahl der Worte, Einstreuen von Buchstaben, die gelöscht werden sollen, in gleichem Maße am Anfang, in der Mitte und am Ende der Worte, etc.). Auf diese Weise ergab sich eine Liste fehlerhafter Worte, beispielsweise durch zusätzliche oder fehlerhafte Buchstaben, Vertauschungen zweier Buchstaben, etc. Die Einstreuung von Fehlern orientierte sich auch an den typischen Beispielen, die in der ersten Datenbasis gefunden wurden (vgl. Kaptitel 2).

Die derartig gesammelten Daten wurden mit sämtlichen erwähnten Heuristiken getestet. Das Erkennungsergebnis der mit den entsprechenden Methoden korrigierten und anschliessend getesteten Daten lag im besten Fall knapp unter 40% richtig erkannter Worte. Dies stellt zwar eine Steigerung gegenüber der Erkennungsrate dar, die ohne jegliche Behandlung der Korrekturen erzielt wurde (knapp 8%), ist alles in allem jedoch eher enttäuschend. Deshalb wurden optische Auswertungen durchgeführt, die Aufschluß darüber bringen sollten, ob und wie der jeweilige Korrekturtyp von den entsprechenden Heuristiken behandelt wurde. Es stellte sich heraus, daß nur 26% richtig klassifiziert und verbessert wurden. In vielen Fällen wurde eine Korrektur bereits falsch klassifiziert, d.h. ein Löschen als Überschreiben interpretiert und umgekehrt. Dies lag daran, daß der Schwellwert, der entscheidet, ob eine Korrektur "wesentlich" mehr Koordinaten enthält als der überschriebene Teil des Wortes, nicht optimal eingestellt war. Dieser Wert ist jedoch im wesentlichen dafür verantwortlich, ob eine Verbesserung als Löschen oder Überschreiben klassifiziert wird. Eine Handoptimierung dieses Schwellwertes für die falsch erkannten Beispiele erhöhte die Anzahl

korrekt behandelter Korrekturen von 26% auf 49%.

Insgesamt konnten im wesentlichen drei Probleme festgestellt werden, die für das schlechte Ergebnis verantwortlich sind:

- Die Einstellung des bereits erwähnten Schwellwertes, der für eine korrekte Klassifikation von Löschen oder Überschreiben benötigt wird. Bei der Analyse der Daten stellte sich heraus, daß es sehr schwer, wenn nicht unmöglich ist, diesen Parameter auf einen optimalen Wert zu setzen. Beispiele konnten gefunden werden, die einen relativ hohen Wert erforderten, andere, für die ein relativ geringer nötig gewesen wäre. Es ist unwahrscheinlich, daß eine "perfekte" Einstellung dieses Parameters gefunden werden kann.

- Viele Korrekturen wurden von den Schreibern in einer sehr ungenauen Art und Weise gemacht. Beispielsweise wurden Ränder eines Bereiches, der komplett gelöscht werden sollte, nicht ausgestrichen. Auf der anderen Seite wurden Teile von Buchstaben, die nicht ausgelöscht werden sollten, durchgestrichen. Einem menschlichen Leser bereiten solche "ungenauen" Korrekturen keine allzu großen Probleme, da er in der Lage ist zu verallgemeinern und Kontextwissen besitzt, das er in der Klassifikation von geschriebenen Texten einsetzen kann. Für einen maschinenausgeführten Algorithmus ist es jedoch recht schwierig zu entscheiden, ob beispielsweise ein nur zur Hälfte ausgestrichener Buchstabe gelöscht werden oder in der Koordinatensequenz verbleiben soll. In diese Kategorie fällt auch ein Phänomen, das ich "Highlighting-Problem" nennen möchte: dadurch, daß ein Schreiber eingefügte Striche mehrfach macht, versucht er, sie besonders hervorzuheben, um einem eventuellen Leser anzuzeigen, daß dieser Teil der Handschrift der korrekte ist und nicht der überschriebene. Ebenso wurde in diversen Fällen für eine Korrektur eine unterschiedliche Skalierung als für die "normale" Handschrift verwendet. Dies macht es einem Menschen wesentlich einfacher, das richtige Wort zu erkennen, erschwert den automatischen Erkennungsprozeß jedoch ungemein.

- Ein weiteres Problem ist die Tatsache, daß die eingeführten Heuristiken wenn auch einen großen Teil, so doch nicht jegliche Art von Korrekturen abdecken. Diverse Verbesserungen, z.B. das Ausstreichen mit wenigen Strichen, können damit alleine nicht behandelt werden. Außerdem existieren Fälle, die widersprüchliche Parametereinstellungen bei einigen Methoden erfordern würden. Ein Beispiel wurde bereits unter Punkt 1 mit dem Schwellwert für eine Klassifikation eines Löschens gegeben.

Insbesondere die Tatsache, daß Beispiele in der Datenbasis gefunden wurden, die es zweifelhaft erscheinen lassen, daß eine optimale Parametereinstellung, die ein hinreichend großes Spektrum möglicher Korrekturen abdeckt, gefunden werden kann, legt die Vermutung nahe, daß eine zufriedenstellende Behandlung von Verbesserungen mit den vorgestellten Heuristiken nicht durchgeführt werden kann. Aus diesem Grund wurde eine Erweiterung dieses Ansatzes implementiert, die verspricht, die obigen Probleme zu umgehen. Sie wird im nächsten Abschnitt beschrieben und ausgewertet.

3.4 Interaktiver Ansatz

Die in den vorausgegangenen Abschnitten angewandten Techniken und Methoden, um Korrekturen und Verbesserungen zu behandeln, lassen sich ausnahmslos in das am Anfang dieses Kapitels erwähnte Konzept der Fehlerbehandlung ("the concept of error handling") einordnen. Da die erzielten Resultate nicht zufriedenstellend waren und einige Probleme aufgetaucht sind, die es höchst zweifelhaft erscheinen lassen, daß eine akzeptable Erkennungsrate mit den vorgestellten Heuristiken erzielt werden kann, soll im folgenden eine Erweiterung vorgestellt werden, die nach dem Prinzip der Fehlervermeidung ("the concept of error avoidance") arbeitet. Das heißt, daß durch einen "geschickten" Entwurf des Interfaces versucht wird zu erreichen, daß Probleme, wie sie mit obigen Heuristiken entstehen können, in diesem Maße gar nicht auftreten oder zumindest eingeschränkt werden.

Der hier durchgeführte Ansatz profitiert davon, daß der Benutzer bei der on-line Handschrifterkennung in der Regel direkt vor den Ein-/Ausgabemedien sitzt und somit die Möglichkeit besteht, ihm unmittelbaren Feedback der Repair- und Erkennungsalgorithmen zukommen zu lassen. Daher werden dem Schreiber hier im Falle einer Korrektur alle ausgeführten Verbesserungen direkt angezeigt. Teile, die als Ausstreichen klassifiziert wurden, werden sofort in der für den Benutzer sichtbaren Handschrift gelöscht. Ein überschriebener Buchstabe wird aus dem ursprünglichen Signal unmittelbar entfernt. Im Falle einer Vervollständigung, bzw. eines Einfügens erhält der Schreiber keinen Feedback, da keine für ihn sichtbaren Aktionen durchgeführt werden müssen. Das Ein- und Umsortieren diverser Wortteile in der zeitlichen Eingabesequenz verändert lediglich das Eingangssignal für den Erkennungsalogrithmus, nicht die Bitmap-Darstellung, die dem Benutzer vorliegt.

Dieser interaktive Ansatz verspricht die Lösung diverser Probleme, die bei der vorangehenden Auswertung aufgetreten sind:

- Das Problem, den Schwellwert für die Klassifikation eines Ausstreichens einzustellen, kann umgangen werden, indem dieser relativ hoch angesetzt wird. Damit wird ein Löschen allerdings erst dann als solches erkannt, wenn relativ viele Ausstreichungen gemacht wurden. Doch durch die sofortige Anzeige, weiß ein Benutzer, ob seine Korrektur erkannt wurde oder nicht und kann entsprechend reagieren, d.h., durch das Hinzufügen weiterer Striche dafür sorgen, daß die von ihm gewünschten Wortteile "verschwinden". Die Gestiken, die hier als Löschen interpretiert werden, sind so ausgelegt, daß dieses "Mehr an Ausstreichungen", das nun vom Schreiber gefordert wird, in einer für ihn nicht störenden und vertretbaren Art und Weise erfolgen kann (salopp gesagt: da keine bestimmte Form für die Löschgestik vorgegeben wurde, sollte es keine allzu großen Probleme bereiten, ob ein paar Striche mehr oder weniger gemacht werden müssen, bis ein Ausstreichen schließlich als solches erkannt wird).

- Eine falsche Behandlung einer Korrektur infolge einer ungenauen Benutzereingabe wird dem Schreiber sofort angezeigt. Dadurch erkennt er even-

tuelles Fehlverhalten der Korrekturmethoden und kann entsprechend reagieren. Damit verbindet sich die Hoffnung, daß eine "zweifelhafte" und ungenauen Verbesserung, die von den verwendeten Heuristiken nicht eindeutig behandelt werden kann, gar nicht erst vorkommt, da der Benutzer sich mit der Zeit an die gegebenen Korrekturmöglichkeiten anpassen und demzufolge keine ungenauen Verbesserungen machen sollte.

- Das "Highlighting-Problem", das in Abschnitt 3.3 erläutert wurde, sollte nicht mehr auftreten. Da eine durchgeführte Verbesserung sofort ausgeführt und dem Benutzer angezeigt wird (beispielsweise wird im Falle eines Überschreibens der überschriebene Teil eines Wortes sofort gelöscht) wird die Notwendigkeit, Korrekturen besonders hervorzuheben, überflüssig. Ziel dieser Hervorhebungen war es ja, den Korrekturtext von den darunterliegenden Teilen des Wortes abzusetzen. Dies ist durch deren augenblickliches "Verschwinden" nicht mehr erforderlich.

Zur Auswertung dieses Ansatzes wurde noch einmal die gleiche Datensammlung mit den selben Schreibern wie in oben erwähntem Ansatz durchgeführt. Dazu wurde jedem Anwender das Wort, das er im ersten Schritt geschrieben hatte (d.h., das falsch geschriebenen Wort) mit der gleichen Aufforderung zur Korrektur nocheinmal präsentiert. Allerdings wurde den Benutzern nun das Ergebnis der Behandlung ihrer jeweiligen Verbesserungen sofort angezeigt, so daß sie gegebenenfalls unmittelbar darauf reagieren konnten.

Die auf diese Weise gesammelten und mit den jeweiligen Heuristiken korrigierten Daten wurden an den Erkenner geschickt. Die erzielte Erkennungsrate ("word accuracy" = relative Anzahl richtig erkannter Worte) lag bei 65%. Dies läßt zwar noch einigen Spielraum für Verbesserungen offen, ist jedoch für einen ersten Ansatz ein zufriedenstellendes Ergebnis, insbesondere, wenn man beachtet, daß der verwendete Erkenner auf vergleichbaren "sauberen" Daten (also richtig geschriebenen Worten ohne Korrekturen) eine Erkennungsleistung von 88% besitzt. Auch eine optische Analyse, ob die jeweiligen Korrekturhandlungen korrekt durchgeführt wurden, lieferte ein zufriedenstellendes Ergebnis: knapp 80% aller Verbesserungen wurden richtig analysiert und dem Benutzer korrekt angezeigt.

Allerdings ergibt sich auch mit diesem Ansatz ein Problem. Wie bereits anfangs in der Motivation für diese Arbeit erwähnt, ist für die Nutzbarkeit und Qualitätsbeurteilung eines Interfaces zur Mensch-Maschine Kommunikation nicht nur eine in hohem Maße richtige Interpretation der Eingabe entscheidend, sondern auch Eigenschaften wie Benutzerkomfort und -akzeptanz. Hier liegt bei vorgestelltem Ansatz das Problem. Alle Schreiber wurden hierzu befragt und kamen einhellig zu dem Urteil, daß die Behandlung von "Überschreibern" komfortabel und robust realisiert war, die eines Ausstreichens jedoch nicht. Dies resultiert daraus, daß der Schwellwert, der zum Längenvergleich im Falle eines Löschens benötigt wird, relativ hoch (genaugenommen zu hoch) angesetzt wurde. Demzufolge mußten die einzelnen Anwender häufig zu oft über eine Stelle, die sie löschen wollten, schreiben, bevor sie verschwand. Dies wurde meist als störend und unkomfortabel empfunden. Interessant in

diesem Zusammenhang ist die Feststellung, daß die reinen Erkennungsraten im Falle von Löschen vergleichbar mit denen bei einem Überschreiben waren. Eine Verschlechterung trat hier nicht ein, obwohl nach Aussage der einzelnen Schreiber Überschreiben wesentlich robuster behandelt wurde als Ausstreichen. Dies ist ein deutliches Indiz dafür, daß die alleinige Tatsache einer hohen Erkennungsrate nicht ausreicht, um die Qualität eines Benutzerinterfaces beurteilen zu können.

Hier liegt demzufolge ein möglicher Ansatzpunkt für zukünftige Forschungen. Beispielsweise würde es sich anbieten, nur diverse Gestiken für das Löschen zuzulassen. Einschränkungen bezüglich der Form verschiedener Ausstreichungen versprechen eine höhere Erkennungsrate. Allerdings ist darauf zu achten, daß diese Zeichen von den Benutzern in einer einfachen und leicht zu merkenden Art und Weise machbar sind. Ansonsten stünde man vor dem gleichen Problem, daß zwar eine hohe Erkennungsrate erzielt wird, die Nutzbarkeit und Akzeptanz von Seiten des Benutzers aber darunter leidet.

4 Korrekturen in gedrucktem Text

Neben Korrekturen im handgeschriebenen Eingangssignal bietet es sich auch an, Verbesserungen in der Ausgabe des Erkenners zuzulassen, d.h., die Möglichkeit anzubieten, gedruckten ASCII-Text via Handschrift zu korrigieren. Es gibt sogar Situationen, in denen ein Fehler des Erkenners nicht durch Verbessern der Handschrift behoben werden kann. Wie sollte beispielsweise das handgeschriebene Wort "hell" korrigiert werden, wenn das Erkennungsergebnis "hello" lautete?

Im folgenden wurde eine Version implementiert, die Löschen und Einfügen auf Buchstabenebene erlaubt. Da in der Regel ein wesentlicher Größenunterschied zwischen handgeschriebenem Text und gedruckten ASCII-Zeichen besteht, gibt es häufig keinen allzu großen Bedarf, die Korrekturtypen "Überschreiben" und "Vervollständigen" bei dieser Aufgabenstellung anzubieten. Es wird hier davon ausgegangen, daß es für einen Benutzer nicht wesentlich unkomfortabler ist, den Repair-Typ "Überschreiben" durch zwei Korrekturen (Löschen und Einfügen) zu ersetzen, als seinen Schreibstil, bzw. die Größe seiner Handschrift, einem vorgegebenen Font anzupassen. Dies mag zwar in Ausnahmefällen nicht zutreffen, sollte jedoch keine allzu große Einschränkung sein.

Auf der anderen Seite erleichtert solch eine Restriktion den Entwurf geeigneter Korrekturalgorithmen ungemein. Beispielsweise entfällt der Klassifikationsschritt, der zwischen den einzelnen Verbesserungen unterscheidet, fast völlig. Da kein Überschreiben erlaubt ist, können sämtliche Gestiken über einem gedruckten Text als Löschen interpretiert werden. Jegliche Handschrift in einem Bereich der ausgezeichneten Eingabefläche, in dem keine ASCII-Zeichen stehen, wird dem Repair-Typ Einfügen zugeordnet.

Es sei hier darauf hingewiesen, daß auch der Klassifikationsschritt, der zwischen korrigiertem Signal und Korrekturgestik unterscheidet und ohne den man bei Verbesserungen innerhalb einer handschriftlichen Eingabe nicht auskommt,

nicht erforderlich ist. Diese Trennung entspricht hier der Aufteilung in ASCII-Text auf der einen und den Koordinaten der Handschrift auf der anderen Seite.

Mittles Vergleich der Bounding Boxes der als Löschen klassifizierten Teile der Handschrift und denen der gedruckten Buchstaben läßt sich die Behandlung von "Ausstreichen" recht einfach realisieren. Überlappt die Bounding Box eines Korrekturzeichens die eines Buchstabens, wird das entsprechende ASCII-Zeichen aus der Buchstabensequenz entfernt. Etwaige Ungenauigkeiten an den Rändern können durch direkte Anzeige der Korrektur, d.h., durch das sofortige Löschen der entsprechenden Buchstaben, und die interaktive Reaktion des Benutzers, ausgeglichen werden.

Allerdings stellen sich hier auch zwei neue Probleme, die im letzten Kapitel, bei Korrekturen innerhalb eines handgeschriebenen Signals, nicht aufgetreten sind. Dies sind im einzelnen

- das Finden der richtigen Position für ein Einfügen

und

- ein explosionsartiges Anwachsen der Größe des Suchraumes in der Erkennung.

Letzteres Problem läßt sich wie folgt erklären. Bei der "normalen" Erkennung ist die Suche auf ein vorgegebenes Vokabular eingeschränkt. Das heißt, die verwendeten Algorithmen sind nur in der Lage einen bestimmten Wortschatz fester Größe zu erkennen. Eine Eingabe von Seiten eines Benutzers, die nicht von dieser Menge abgedeckt wird, würde unweigerlich zu einem Fehler führen. Erlaubt man nun Korrekturen des Erkennungsresultats, genauer gesagt: das Einfügen von Wortteilen in ein falsch erkanntes Ergebnis, so kommt als neue handschriftliche Eingabe eines Anwenders potentiell jeder möglichen Teilstring in Frage, der im Originalvokabular existiert. In einem Beispiel wurde gezeigt, daß sich damit die Zahl der möglichen Lösungen bei einer Vokabulargröße von 51866 auf 288610 (Teil-)Worte erhöht. Dies stellt eine mehr als Verfünffachung der Größe des Suchraumes dar, was eine korrekte Erkennung wesentlich erschwert.

Hier wurde nun versucht, diese beiden Probleme zu umgehen, indem verschiedene Einschränkungen bezüglich der zugelassenen Korrekturen gemacht wurden. Bestehen keinerlei Anforderungen an den Benutzer bezüglich der erlaubten Ausstreichungen und Einfügen, kann jede vom Benutzer eingegebene Handschrift prinzipiell an jeder beliebige Stelle des als ASCII-Text dargestellten Ergebnisses eingefügt werden und einen beliebigen String aus der Menge aller im zugrundeliegenden Erkennungsvokabular vorkommenden Substrings darstellen.

Mit einer ersten Einschränkung lassen sich die beiden entstehenden Probleme beheben, bzw. vermindern. Wird vom Anwender verlangt, daß er eine Ausstreichung (und darauffolgende Einfügung) nur auf der Basis von kompletten falschen Substrings des Erkennungsergebnisses durchführt, kann aus den verbleibenden, nicht gelöschten ASCII-Zeichen Information gezogen werden, die im Erkennungsprozeß sinnvoll genutzt werden kann. Wird ein zusammenhängender

falscher Substring des Erkennungsergebnisses in einem Schritt gelöscht, können die Buchstaben, die links und rechts der Löschstelle verbleiben als korrekt angenommen werden. Dadurch läßt sich nun die Menge möglicher Lösungen auf alle Substrings des ursprünglichen Vokabulars einschränken, die zwischen diesen beiden Zeichen vorkommen, bzw. auf die entsprechenden Wortanfänge oder -enden, wenn ein Löschen am Beginn oder Ende einer ASCII-Zeichen-Sequenz stattfand. Analysen mit dem oben erwähnten, 51866 Worte umfassenden, Vokabular haben gezeigt, daß sich auf diese Weise eine enorme Reduktion der Größe des zugrundeliegenden Suchraumes erzielen läßt. Dies wirkt sich in der Regel positiv auf das Erkennungsergebnis aus. Durch die Restriktion bezüglich der Korrekturreihenfolge (1. Löschen, 2. Einfügen) läßt sich das Problem, die richtige Einfügestelle zu finden, ebenfalls umgehen. Eingefügt wird grundsätzlich nur an einer Stelle, an der unmittelbar zuvor ein Ausstreichen stattgefunden hat (bzw. am Wortanfang oder Ende im Falle, daß kein vorheriges Löschen stattfand, jenachdem, ob die Einfügung links oder rechts des ASCII-Textes gemacht wurde).

Eine weitere Möglichkeit, das Problem einer Explosion der Größe des Suchraumes einzuschränken, besteht darin, den gesamten Korrekturprozeß eines Wortes in zwei Phasen aufzuteilen: (1.) Lösche *alle* falschen Buchstaben aus der dargestellten ASCII-Zeichen-Sequenz, (2.) Führe gegebenenfalls Einfügungen durch. Nach dem Ende der 1. Phase, der "Ausstreich-Phase", ist bekannt, daß alle verbleibenden Zeichen des dargestellten Ergebnisses korrekt sind. Im vorangegangenen Fall konnten nur die jeweils linken und rechten Einzelbuchstaben an den Rändern einer Löschstelle als richtig angenommen werden, da es erlaubt war, nach einem Ausstreichen an einer bestimmten Position innerhalb des Wortes, eine weitere Korrektur in einem anderen Wortteil durchzuführen. Mit dieser erweiterten Einschränkung läßt sich nun der Suchraum auf alle möglichen Substrings des Originalvokabulars einschränken, deren entsprechenden Gegenstücke den nicht gelöschten Zeichen des ASCII-Textes entsprechen. Daraus ergibt sich eine weitere beträchtliche Reduktion der Größe des Suchraumes, wie anhand diverser Beispiel mit bereits erwähntem, 51866 Worte großen, Vokabular bewiesen wurde. Allerdings stellt sich hier wieder das Problem, daß mehrere Möglichkeiten für ein potentielles Einfügen bestehen. Dies ließe sich beispielsweise dadurch lösen, daß diverse Gestiken zur Markierung der Einfügestellen eingeführt werden. Weitere Ansätze wären die Verwendung eines sog. Cursors (= Schreibmarke), der die genaue Einfügestelle spezifiziert, oder die Abhandlung aller Einfügungen in einer vorgegebenen Reihenfolge.

Bei der im Rahmen dieser Arbeit implementierten Version wurde eine weitere Einschränkung an das Korrekturverhalten des Benutzers gemacht. Dabei ist nur noch ein einziges Ausstreichen im kompletten Wort erlaubt. Das heißt, in einem Löschvorgang für ein Wort darf genau ein *zusammenhängender* Substring ausgestrichen werden, der *alle* falsche Buchstaben enthalten muß. Das Wissen, daß die verbleibenden Zeichen alle korrekt erkannt wurden und die Tatsache, daß nur an der (in diesem Fall eindeutigen) Löschstelle eingefügt werden darf, Verringern eine Explosion der Suchraumgröße und lösen das Problem, die richtige Einfügestelle zu bestimmen. Allerdings müssen hier unter Umständen auch Wortteile gelöscht werden, die korrekt sind. Dies stellt eine Verschlechterung

des Benutzerkomforts dar, die beispielsweise bei Texteditiersystemen zu Problemen führen könnte und daher unter Umständen nicht hingenommen werden kann. Da es sich in vorliegender Anwendung jedoch um die Korrektur von Texten, bzw. Worten, die als Ergebnis einer automatischen Handschrifterkennung geliefert werden, handelt, fällt dieses Problem nicht so sehr ins Gewicht. Der Grund dafür ist, daß in einem falschen Erkennungsresultat die nicht korrekten Buchstaben in den seltensten Fällen über das ganze Wort verteilt sind. In der Regel treten Fehler nur als zusammenhängende Substrings auf. Dadurch tritt der oben erwähnte Konfliktfall, daß ein Ausstreichen richtiger Wortteile erforderlich ist, um eine Korrektur ordnungsgemäß durchführen zu können, in der Praxis (zumindest mit dem in dieser Arbeit verwendeten Handschrifterkenner) recht selten auf.

Da eine Auswertung auf der Basis einer korrekten Erkennungsrate extrem von den verwendeten Wort- und Korrekturvorgaben abhängt und beeinflußt werden kann, ist eine statistisch unabhängige Evaluation extrem kritisch und schwierig durchzuführen. Aus diesem Grund wurden in vorliegender Arbeit keine derartigen Analysen durchgeführt, sondern lediglich die These, daß sich durch die diversen Einschränkungen eine enorme Verringerung der Größe des Suchraumes ergibt, anhand diverser Beispiele nachgewiesen. Die Beispiele wurden so ausgewählt, daß statistisch repräsentative Ergebnisse erzielt werden konnten. Die auf diese Weise erhaltenen Reduktionen waren so beachtlich, daß eine Verbesserung der jeweiligen Erkennungsraten als sicher angenommen werden kann.

5 Fazit

In vorliegender Arbeit wurden diverse Verfahren und Methoden für die Korrektur bei der automatischen on-line Handschrifterkennung eingeführt, ausgewertet und diskutiert. Ausgangspunkt war die These, daß jegliche Art von Interface zur Mensch-Maschine Kommunikation die Möglichkeit zur "Reparatur" und Verbesserung von Fehlern bieten muß, um von einem potentiellen Nutzer angenommen zu werden. Diese These stützt sich darauf, daß es erstens unwahrscheinlich ist, daß eine immer hundertprozentig korrekte Erkennung einer Benutzereingabe je erreicht werden kann, und zweitens, daß selbst in einem solchen Falle Fehler immer vorkommen und auch immer vorkommen werden, beispielsweise durch fehlerhafte Benutzereingaben, technische Ausfälle, etc. Punkt zwei wurde hier anhand einer Datenbankanalyse belegt. Diese empirische Studie diente auch als Basis für eine Klassifikation typischer Fehler und Korrekturen. Sie gab Aufschluß auf die Fragen, welche Arten in handgeschriebener Texteingabe üblicherweise vorkommen und wie man diese hinsichtlich einer möglichen Realisierung diverser "Repair"-Algorithmen klassifizieren kann. Zu diesem Zweck wurden die Korrekturklassen "Löschen", "Vervollständigen/Einfügen" und "Überschreiben" eingeführt.

Für die beiden Fälle "Korrektur innerhalb eines handgeschriebenen Textes" und "Verbesserungen des in ASCII-Text gedruckten Ergebnisses" wurden diverse Heuristiken vorgestellt und ausgewertet. Es stellte sich bei beiden Alter-

nativen als recht schwierig heraus, Algorithmen und Heuristiken zu entwerfen, die das Korrekturproblem in einer akzeptablen Art und Weise lösen. Probleme, wie ungenaue Benutzereingaben oder eine Explosion der Größe des Suchraumes, waren die Folge. Durch diverse Erweiterungen, Einschränkungen und Modifikationen bezüglich der Eingabe und des Design des Schnittstellen-Interfaces konnte jedoch gezeigt werden, daß eine akzeptable Behandlung von Korrekturen und Verbesserungen möglich ist. Diese beiden Vorgehensweisen, die sich gegenseitig ergänzen und nur zusammen ein sinnvolles Ergebnis liefern, wurden in zwei Konzepten zusammengefaßt: das Konzept der Fehlerbehebung ("the concept of error handling") und das Konzept der Fehlervermeidung ("the concept of error avoidance"). Letzteres beinhaltet beispielsweise die unmittelbare Darstellung einer ausgeführten Verbesserung im Falle von Korrekturen in einem handgeschriebenen Text, wodurch sich diverse Probleme vermeiden lassen. Aufgrund verschiedener Einschränkungen, die an den Benutzer bei der Korrektur von gedrucktem ASCII-Text gemacht wurden, konnten ebenfalls diverse Probleme, wie zum Beispiel die explosionsartige Vergrößerung des Suchraumes, behoben oder zumindest einschränkt werden.

Zwei für das Design jeglicher Art von Interface zur Mensch-Maschine Kommunikation wichtige Kriterien sind eine hohe Erkennungsrate und Benutzerakzeptanz. Obige Implementierungen und Untersuchungen im Falle der automatischen Handschrifterkennung haben versucht, eine Lösung zwischen den folgenden zwei Extremen zu finden: (1) keinerlei Einschränkungen werden bezüglich des Korrekturverhaltens an den Benutzer gestellt und (2) jegliche Art von Verbesserung im Text ist durch Einschränkungen und vorgegebene Verhaltensweisen reglementiert. Punkt (1) resultiert üblicherweise in einer hohen Benutzerakzeptanz, da dem Anwender bezüglich seiner Eingabe völlige Freiheit gewährleistet wird, aber auch in einer niedrigen Erkennungsrate, da eine vollständig richtige Erkennung einer beliebigen Eingabe (noch ?) nicht möglich ist. Das zweite Extrem hingegen wird durch seine diversen Reglementierungen eine hohe Erkennungsrate aufweisen, die Benutzerfreundlichkeit und damit die Akzeptanz eines möglichen Anwenders wird aber zu wünschen übrig lassen. Die große Herausforderung beim Entwurf multimodaler Mensch-Maschine Schnittstellen ist es demnach, zwischen den beiden Extremen einen Mittelweg zu finden, der hohe Erkennungraten garantiert, dem Nutzer jedoch gleichzeitig ein brauchbares und für ihn akzeptables Interface zur Verfügung stellt.

Danksagung

Zum Schluß möchte ich mich hiermit noch bei allen bedanken, die mir bei dieser Arbeit mit Rat und Tat zur Seite gestanden haben. Besonders erwähnen möchte ich Prof. Alex Waibel, der mir die Möglichkeit gegeben hat, diese Arbeit in seinem Arbeitskreis in den U.S.A. durchzuführen, sowie meinen Betreuer Jie Yang.

Ein weiterer Dank geht an Ralph Groß, Bernhard Suhm, Matthias Denecke und Jie Yang, die die Zeit und Geduld aufgebracht haben, mir Daten zu spenden, die ich für die Evaluation der Heuristiken benötigte.

Literatur

[1] S. Mori, C.Y. Suen, K. Yamamoto: Historical review of OCR research and development. Proceedings of the IEEE, 1992

[2] T. Wakahara, H. Murase, K. Odaka: On-line handwriting recognition. Proceedings of the IEEE, 1992

[3] S. Manke, U. Bodenhausen: A connectionist recognizer for on-line cursive handwriting recognition. Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, 1994

[4] S. Manke, M. Finke, A. Waibel: NPen^{++}: A writer independent, large vocabulary on-line cursive handwriting recognition system. Proceedings of the International Conference on Document Analysis and Recognition, 1995

[5] L.R.B. Schomaker: User-interface aspects in recognizing connected-cursive handwriting. Proceedings of the IEE Colloquium on Handwriting and Pen-based input, 1994

[6] C. Frankish, R. Hull, P. Morgan: Recognition accuracy and user acceptance of pen interfaces. Proceedings of the Conference on Human Factors in Computing Systems, 1995