Adaptive Text Entry for Mobile Devices

Proschowsky, Morten Smidt; Schultz, Nette; Hansen, Lars Kai

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Adaptive Text Entry for Mobile Devices

Morten Proschowsky

Kongens Lyngby 2008
IMM-PHD-2008-209

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

The reduced size of many mobile devices makes it difficult to enter text with them. The text entry methods are often slow or complicated to use. This affects the performance and user experience of all applications and services on the device. This work introduces new easy-to-use text entry methods for mobile devices and a framework for adaptive context-aware language models.

Based on analysis of current text entry methods, the requirements to the new text entry methods are established. Transparent User guided Prediction (TUP) is a text entry method for devices with one dimensional touch input. It can be touch sensitive wheels, sliders or similar input devices. The interaction design of TUP is done with a combination of high level task models and low level models of human motor behaviour. Three prototypes of TUP are designed and evaluated by more than 30 users. Observations from the evaluations are used to improve the models of human motor behaviour. TUP-Key is a variant of TUP, designed for 12 key phone keyboards. It is introduced in the thesis but has not been implemented or evaluated.

Both text entry methods support adaptive context-aware language models. YourText is a framework for adaptive context-aware language models that is introduced in the thesis. YourText enables different language models to be combined to a new common language model. The framework is designed so it can be adapted to different text entry methods, thereby enabling the language model to be transferred between devices.

YourText is evaluated with a corpus of mobile text messages. The corpus is created by collecting all sent and received messages from 12 persons in four

weeks. The corpus contains 25,000 messages. A model of text entry speed for TUP is created from the observations in the evaluations. The model is used to predict the performance of TUP, used together with different YourText language models.

Resumé

En af de største udfordringer ved mobiltelefoner er at skrive tekst på dem. Ofte er det enten en kompliceret eller langsommelig opgave. Det går ud over brugervenligheden og brugernes oplevelse. Denne afhandling introducerer nye og nemme måder at skrive tekst på telefoner og for andre mobile elektroniske produkter.

Transparent User guided Prediction (TUP) er en af disse nye metoder. Den er beregnet til produkter med berøringsfølsomme hjul, striber eller lignende endimensionelle inputenheder. TUP er blevet implementeret i tre forskellige prototyper og evalueret af mere end 30 personer. TUP er blevet forbedret ved at anvende statistiske metoder på data fra evalueringerne. Det er lykkedes at forbedre skriveastigheden med næsten 30% og reducere antallet af fejl betydeligt. TUP-Key er en variant af TUP der er lavet til at skrive tekst på mobiltelefoner. TUP-Key er beskrevet i afhandlingen, men er ikke blevet implementeret eller evalueret.

Både TUP og TUP-Key understøtter brug af sprogmodeller der tilpasser sig til brugerens kontekst. En sådan sprogmodel, med navnet YourText, bliver introduceret i afhandlingen. YourText benytter en kombination af andre sprogmodeller til at lave en ny og bedre sprogmodel. Sprogmodellen er meget fleksibel og kan benyttes i mange forskellige typer af produkter.

25.000 sms beskeder fra forskellige brugere er brugt til at teste kvaliteten af YourText. YourText kan bruges til at forbedre TUP. En matematisk model er blevet opstillet for at finde effekten af at bruge TUP sammen med YourText.

Preface

This thesis was prepared at DTU Informatics, the Technical University of Denmark in partial fulfilment of the requirements for acquiring the Ph.D. degree in engineering.

The thesis deals with adaptive text entry methods for mobile devices. Focus is on development of novel text entry methods, adaptive context-aware language models and optimization of text entry methods with a combined quantitative and qualitative interaction design process.

Lyngby, December 2008

Morten Proschowsky

Acknowledgements

I want to thank all the people who have contributed to this work.

My supervisor, Nette Schultz for your help during PhD. Thanks for many good comments and discussions and for fighting the PhD programme committee when it has been needed. I also want to thank my other supervisor, Lars Kai Hansen, for preparing the assessment of my work.

A special thank goes to Niels Ebbe Jacobsen and Rod Murray-Smith. I am very grateful that you have taken the time to follow my work. Your advices and suggestions have guided me through this work.

Thanks to my colleagues from DTU. Especially Andrius Butkus, Su-En Tan, Michael Kai Petersen, Jakob Eg Larsen, Dan Saugstrup and Kristian Kristensen. Thanks to John Williamson and Andrew Crossan for taking good care of me in Glasgow. I also want to thank Mikkel Proschowsky for good discussions and help with the prototypes, and Thomas Troelsgård for your comments on the language model sections.

A big thank you goes out to the more than 40 persons who have been helping me by participating in usability evaluations or by sharing their mobile text messages with me. The work is built on the empirical data you have provided, and would not have been possible to do with out your help.

Finally I want to thank Marie-Louise Smidt for her love and support. I could not have done this work without you. I am looking forward to spending more time with you and David.

Contents

Chapter 1

Introduction

The need for fast and easy text entry on current mobile devices has grown rapidly the last 20 years. The number of mobile devices is increasing, and the devices are becoming more advanced. The popularity of mobile text messages have been the main driver. In 2007 an estimated 2.5 trillion text messages were sent between the 2.5 billion subscribers on GSM networks [GSM Association, 2007]. Recently the demand for text entry have been boosted by new services such as mobile email, multi media messages (MMS), instant messaging and web browsing [GSM Association, 2006]. Many mobile devices also come with personal information management software such as calendars, to-do lists, notepads and office suites. The usability and user experience of all these applications and services are greatly affected by the ease and speed of mobile text entry. Improved text entry methods will likely enable more people to enter text into mobile devices, and thereby enable them to communicate more with other people. Improved text entry methods will enable novice and experienced users to type faster, commit fewer errors and have better user experience.

1.1 Challenges of mobile text entry

Text entry on mobile devices is slow and difficult. The small form factor of many mobile devices makes it undesirable to have a separate key for each character.

Instead the characters are entered by combinations of key presses. This requires more key presses per character and increases the risk of making errors. Some text entry methods are applying language models to improve performance. These methods have a more complex conceptual model, and can be hard to understand for novice users. The performance can decrease dramatically, if the text the user wants to enter does not match the language model.

1.2 Improvements to mobile text entry

This thesis introduces a novel text entry method called Transparent User guided Prediction (TUP). TUP is easy to use and can be fitted in small mobile devices. The method's conceptual model is designed to be easy to learn. A language model is used to improve performance. To avoid increasing the complexity, the language model is hidden from the user. The interaction design of TUP is done with a combination of high level task models and low level models of human motor behaviour.

TUP is evaluated and improved in an iterative interaction design process. It is shown how methods from statistical learning can be used to improve text entry methods.

The language model is a critical component of many text entry methods. YourText is a new context-aware adaptive language model that is created to better suit the needs of mobile text entry. The language model will adapt to the users language and to the current context of the user. YourText is evaluated with a corpus of mobile text messages.

It is shown how empirical evaluation data can be used to estimate the performance of new language models. This method is used to estimate the performance of YourText used together with TUP

The following list describes this thesis' four key contributions to mobile text entry research:

TUP. Novel text entry method for touch input devices. Its key strength is a simple conceptual model that makes it easy to learn for novice users. TUP uses a language model to improve performance, but the model is hidden from the user. TUP can be used together with other types of input devices. TUP-Key is a version of TUP adapted to be used on a normal mobile phone.

Novel language models. New language models called YourText and PPM*D have been created. YourText uses linear interpolation of multiple PPM*D models to estimate character probabilities. PPM*D is an unbounded length PPM model, with a decay factor. It will adapt to the user's language and YourText will adapt to the context of the user.

Statistical Learning approach to optimization of text entry. It is shown how statistical learning can be used to optimize text entry methods. Observations from usability evaluations of TUP have been used to create improved models of human motor behaviour. The improved models are implemented in TUP, and have been verified in a new usability evaluation.

Method to estimate performance of language models. It is shown how the performance of new language models can be estimated from empirical data from usability evaluations.

Text entry is closely related to language. Different languages use different scripts. Text entry methods for one type of languages can be impossible to use with other types of languages. This thesis will only cover western European languages based on the Roman alphabet.

1.3 Content of thesis

2 Text Entry on Mobile Devices introduces existing text entry methods and language models. It is shown how language models can be used to improve the usability and user experience of the text entry methods. The trade offs between size, speed and complexity of text entry methods are discussed.

3 Evaluation of Text Entry methods describes the methodologies that have been used in the text entry evaluations presented in this thesis.

4 Transparent User guided Prediction presents a novel text entry method. The method is improved by an iterative interaction design process, where the method is redesigned, prototyped and evaluated. A key based variant of TUP is introduced.

5 YourText - a context-aware adaptive language model presents a novel language model. Combinations of language models are used to create a context aware language model. The language model is evaluated with a corpus of mobile text messages.

6 Modelling performance of language models and text entry methods shows how empirical data from usability evaluations can be used to estimate the performance of novel language models.

7 Conclusions summarises the concussions from the preceding chapters.

Chapter 2

Text entry on mobile devices

This chapter introduces different text entry methods for mobile devices. It explains how language models can be used to improve the text entry performance. Different types of language models are introduced and compared. Some text entry methods have complicated conceptual models. These text entry methods can be difficult for the users to understand, and thereby causing problems.

2.1 Introduction to mobile text entry

Text entry is the task of entering textual information into computer systems. On desktop computers it is done using a keyboard with 80 - 100 keys. Each key has 2-3 characters or symbols laid out according to the QWERTY layout. There exist different key layouts, but QWERTY is dominant and have not changed for many years. Besides the character keys, the keyboard have modifier keys such as shift, alt and ctrl that are used to modify the meaning of the character keys.

For mobile devices, the text entry methods are a lot more diverse. There exist many different methods that all have their strengths and weaknesses. The main reason for the diversity is that mobile devices are designed with different form

factor, functionality and for different types of usage. A mobile phone designed for email and web browsing requires a fast and easy text entry method while a music player requires little or no text entry at all. Most text entry methods are bound to specific input devices. The size of the input devices is constrained by the form factor and size of the mobile device. The product design of the mobile device might add additional constraints to the input device like choice of materials and placements of buttons. Most text entry methods require a display or another output device to give feedback to the users. The dependency between the output device and the text entry method is not as large as the dependency of the input device.

When looking at mobile text entry systems it is therefore necessary to look at the text entry method in conjunction with the mobile device and the expected usage. Many text entry methods can be used on different devices, but it often requires some adaption of the method. It is unlikely that we will see a unified text entry method for all mobile devices in the near future. Unlike the desktop computer, the diversity of mobile devices and their usage is very large. The text entry methods used for PDAs and advanced mobile phones would not fit a music player or vice versa.

2.1.1 Key based text entry methods

Many mobile devices with support for text entry are utilizing physical keyboards. Most laptops have keyboards similar to desktop computers, with the exception that the numeric keypad often is removed. Other mobile devices often have smaller and fewer keys. The size and number of keys influence the choice of text entry methods. Fewer keys often leads to more complicated text input methods where it requires multiple key presses to enter a single character. The metric Key Strokes Per Character (KSPC) [MacKenzie, 2002a] is often used to describe the efficiency of key based text entry methods. KSPC is the average number of keystrokes that are required to write a character. KSPC can both be used as a characteristic of text entry methods or as a dependent variable in text entry evaluations. The KSPC value is often dependent of the language being used. All reported values in this section are for English. KSPC is only one of the characteristics that influence typing speed. A low KSPC alone is no guarantee for fast text entry. Three presses on the same key can for example be made faster than two presses on different keys.

Another metric is Words Per Minute (WMP) which is used to describe how fast a text entry method is. WPM is normally found from evaluations, but can also be predicted as shown by Silfverberg et al. [2000]. Early work used the

actual number of words to calculate WPM. All recent papers use a common word length, where a word is defined as five characters including spaces.

2.1.1.1 Miniature QWERTY keyboards

Many smart phones and advanced mobile devices are equipped with a miniature QWERTY keyboard. They have the same key layout as desktop computers, but each key is much smaller. The rows with function keys (F-keys) and numeric keys are often removed to save space. Each key press enters a character so KSPC = 1.0. Language-specific characters can either be assigned to their own keys or be entered by a combination of key presses. Since even a miniature QWERTY keyboard takes up a lot of space, the keyboards are often hidden on the mobile devices. The keyboard can be revealed by transforming the device. Figure 2.1 shows different examples of miniature QWERTY keyboards and how they can be implemented in phones. Miniature QWERTY keyboards are sometimes called thumb keyboards, because it is very common to hold the device with two hands and type with the thumbs.

Miniature QWERTY keyboards are easy to learn to use because the users can transfer their skills from desktop computers to miniature QWERTY keyboards. Studies show that novice users who know the QWERTY layout can type over 30 WPM. Expert users can type up to 60 WPM [Clawson et al., 2006, Clarkson et al., 2005, Roeber et al., 2003]. The typing speed is dependent on the size of the keyboard and whether one or two hands are used while typing. The main problem with miniature QWERTY keyboards is that the small keys lead to many errors. It is mainly off-by-one errors where a key next to the target key is pressed [Clawson et al., 2006].

2.1.1.2 Half QWERTY keyboards

Even with the small keys found on miniature QWERTY keyboards, the keyboards take up a lot of space on the mobile devices. If the keys are made smaller, it is likely that the number of errors will increase. A solution to the problem is to put two letters on each key. This is called half QWERTY because there are half as many keys as on a full QWERTY keyboard. With two letters per key, there need to be some way to disambiguate between the letters. An implementation by Matias et al. [1993, 1994] uses a modifier key to let the user select between the letters. Other half QWERTY keyboards have two domes placed beneath each key. This makes it possible to detect whether the user pressed on the left or right side of the key. Another solution is to use

(a) BlackBerry Curve 8320

(b) Nokia E90i

(c) HTC TyTN II

(d) Siemens SK65

Figure 2.1: Different implementations of miniature QWERTY keyboards in phones. In (a) the keyboard is always visible. With (b) the phone needs to be opened to access the keyboard. In (c) the keyboard is slide out from the back of the phone. The keyboard in (d) can be used by rotating the back of the phone.

a language model to disambiguate between the letters. With a language model, the phone software will select the letter that is most likely, given the previous letters. Disambiguation with language models are described in section 2.2.2. Performance of half-QWERTY keyboards is reported to be around 35 WPM for experienced users [Matias et al., 1993].

2.1.1.3 12 keys ITU-T keyboards

Most mobile phones are equipped with a 12 keys ITU-T keyboards. Figure 2.2 shows an example of such a keyboard. Besides the 12 keys most phones also have keys for navigating in the menus, call handling and special purpose keys for controlling music playback, volume settings, shutter-release for built-in cameras and other features.

Figure 2.2: Keyboard from Nokia 6300. Besides the 12 keys ITU-T keyboard there is also keys for navigating the user interface and call handling.

ITU-T E.161 [ITU, 2001] specifies the layout of the 10 numbers and the * and # symbol. The layouts are specified for 4 x 3, 6 x 2 and 2 x 6 key arrays, but the 4 x 3 array is the only layout that is widely used. ITU-T E.161 also specifies how the basic 26 Latin letters are grouped and assigned to the keys. Table 2.1 shows the specified layout.

1	2 abc	3 def
4 ghi	5 jkl	6 mno
7 pqrs	8 tuv	9 wxyz
*	0	#

Table 2.1: Recommended layout of letters on a 12 keys keyboard [ITU, 2001].

Multitap text entry method The standard method for text entry with ITU-T keyboards is multitap. With multitap, consecutive presses on the same key are used to select characters. A single press on key 2 will input *a*. Two consecutive presses give *b*, three presses *c* and so on. The numbers are placed after the letters. To write *7* in the text editor, the 7 key have to be pressed 5 times. On some implementations numbers can also be written by making a long press on the key.

Words with subsequent characters on the same key can be problematic. There needs to be a way of dividing the key presses between the two characters. The word *hi* consists of *h* and *i* that both are placed on key 4. The sequence of key presses 44444 can be interpreted in many different ways, resulting in words like *hi, ih, hhg, gig* and *ggggg*. To avoid this ambiguity, multitap includes a character separator. It is often implemented as a timeout. If the time between two key presses on the same key is larger than 1-2 seconds, then the second key press will be interpreted at the first key press of the next character. Research by Marila and Ronkainen [2004] has shown that users can learn the length of the timeout very fast, if sufficient feedback is given. Some implementations use a special NEXT key to indicate new characters in a sequence of key presses. With a NEXT (N) key *hi* will be entered as 44N444. KSPC for multitap is reported to be 2.0342 by MacKenzie [2002a].

Performance of multitap for novice users is reported to be between 6 and 8 WPM by Butts and Cockburn [2002], James and Reischel [2001]. Expert performance is estimated to be up to 20 WPM by Silfverberg et al. [2000]. The world record is beyond 40 WPM [Haines, 2006].

Chording methods Chording methods describe text entry methods where 2 or more keys are pressed at the same time. As the name indicates, this is like playing music where a chord can be made by pressing multiple keys on a piano. The use of the shift key on a desktop keyboard is a simple chording method. The shift key and 26 character keys enable the user to write 52 different characters. Most chording keyboards use much more complicated patterns to enable text entry with few keys. Twiddler by [Lyons et al., 2004] is a chording method that uses 12 keys in a 4 x 3 layout. It uses a special character layout. The 8 most used characters can be entered by a single key press, and the rest can be entered by two simultaneously key presses. Novice performance with Twiddles is 6-8 WPM while experienced users can type at 26 WPM. KSPC for Twiddler is reported to be 1.4764. Since the keys can be pressed simultaneous, it cannot be directly compared to other KSPC values.

Language-specific characters ITU-T E.161 only specifies the location of the basic 26 Latin letters. Most European languages contain special language-specific characters. The European Telecommunications Standards Institute (ETSI) have created the ES 202 130 standard [Laverack and von Niman, 2007] that describes these language-specific characters and their assignments to ITU-T keyboards. The most recent version have character repositories and key assignments for 101 languages, including languages of the European countries, minority languages, immigrants' languages and other languages of relevance. The characters are assigned to keys based on a number of principles [Böcker et al., 2006]. The language-specific characters are assigned to the same key as the character they are derived from. The Danish å is placed on the same key as a and the German ß is placed on the same key as s. The language-specific characters are divided into two groups. Type A for essential characters for the language and Type B for characters that are used, but not essential.

The key assignments from ITU-T E.161 are used as the basis for all European languages, even if some of the letters are not used in a language. The numbers are placed after the basic Latin letters. Type A and B language-specific characters are placed after the numbers. Table 2.2 shows the assignment of characters for the Danish language. Space and special characters are not included in the specification, but it is recommended that they are placed on the 1 key.

Key	Characters
2	a b c 2 æ å á à ä
3	d e f 3 é ð
4	g h i 4 í
5	j k l 5
6	m n o 5 ø ö ó
7	p q r s 7
8	t u v 8 ú ü
9	w x y z 9 ý

Table 2.2: Recommended layout of characters on a 12 keys keyboard for Danish [Laverack and von Niman, 2007]. It is recommended to place symbols on the 1 key.

The ITU-T E.161 standard are used on almost all mobile phones, but the ETSI ES 202 130 extension is not followed by all manufacturers. Table 2.3 shows the layout used in the Nokia N95 phone.

The number of characters on each key increases when language-specific characters are included. This has a large impact on methods like multitap, which requires many key presses to select the language specific characters.

Key	Characters
1	. , ? ! 1 @ ’ - _ ( ) : ; & / % * # + < = > " $ £ § ¥ ¤ ¡ ¿
2	a b c 2 æ å ä à á â ã ç
3	d e f 3 é ð è ë ê
4	g h i 4 î ï ì í
5	j k l 5 č
6	m n o 5 ø ö ô ò ó õ ñ
7	p q r s 7 ß $
8	t u v 8 ü ù û ú
9	w x y z 9 ý Þ
0	space 0 new-line

Table 2.3: Layout used in Nokia N95 in multitap mode and Danish language.

2.1.1.4 Few keys text entry

Some devices do only have room for 2-5 keys, and still need to support text entry. It could be pagers, wrist watches and similar devices. There exist numerous text entry methods for these kinds of devices.

Date stamp The date stamp method is named after the tellers date stamp. A date stamp is equipped with a series of rubber wheels that can be adjusted to stamp a date. The text entry method, named after the date stamp, consists of a list of characters, a way to highlight a character on the list, and a way to select the highlighted character. One of the most common and simple implementations consist of a display and three keys labelled Next, Previous and Select. The characters are shown on the display and the Next and Previous keys will change the highlighted character to the next or previous character on the list. The Select key will input the currently highlighted character. Another implementation uses a wheel with a key in the centre. The wheel is used to change the highlighted character and the key will input the highlighted character. Some implementations allow the highlight to loop forth and back between the last and the first character on the list. Other implementations support typamatic events, where the next or previous key can be held down for repeated stream of virtual key presses. KSPC for Date stamp is about 9-11 for the different implementations.

The performance for date stamp with three keys is around 9 WPM [MacKenzie, 2002b]. Tarasewich [2003] evaluated a variation where the characters (including numbers and special characters) were divided in four groups. The user should first select a group of characters and then select a character from this group.

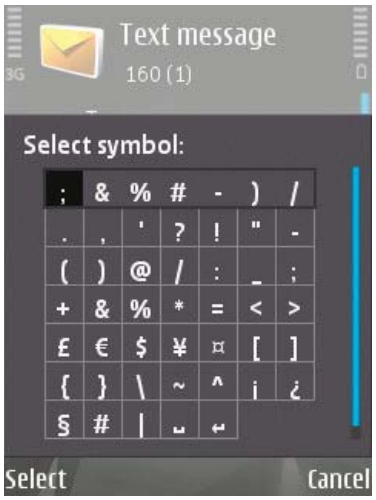


Figure 2.3: Selection keyboard for entering symbols on Nokia S60 devices. The recently used symbols are placed in the top row.

This variant have a KSPC value of about 6, but was found to be significantly slower than the normal date stamp method.

Selection keyboards Selection keyboards are similar to the date stamp method, but the characters are placed in a grid on a screen. Four keys or a joystick are used to highlight a character on the screen. Another key is used to enter the highlighted character. Selection keyboards are not commonly used to enter text on mobile devices. On some mobile devices, such as Nokia S60 devices, selection keyboards are used to enter symbols. Figure 2.3 shows the selection keyboard from a Nokia N95. The selection keyboard gives a better overview of the symbols, than tapping through the list of symbols shown in table 2.3.

2.1.2 Touch based text entry methods

A growing number of devices are equipped with touch sensitive input devices. It can either be input-only devices such as touch pads, touch sensitive sliders and wheels or combined input-output devices such as touch screens. Touch sensitive input devices can be based on different technologies such as resistance and capacitive sensing. The different technologies are optimized for stylus or finger use. Some implementations can also detect the force of the touch. Prior

to 2008 most touch sensitive input devices could only sense one touch at a time. The introduction of iPhone and iPod Touch by Apple made multi touch technology widely available. Multi touch can sense multiple touches at the same time, thereby enabling more advanced interaction.

2.1.2.1 On-screen keyboards

On-screen keyboard is a text entry method for touch screen devices. A virtual keyboard is drawn on part of the screen, and text can be written by pressing the virtual keys. Keyboards designed for stylus use can be very small, while keyboards for finger use require more space to be usable. On-screen keyboards are implemented in the device software, which is why they are sometimes referred to as soft keyboards. They have a number of advantages over physical keyboards. The keyboard can be hidden completely when there is no need for text entry. This leaves the entire screen available to the active application. The key layout of the on-screen keyboard can be changed to suit different contexts. Localized keyboards with language-specific characters can be implemented in software which is much cheaper than a physical keyboard. One problem with on-screen keyboards is the lack of tactile feedback from the touch screen. Hoggan et al. [2008] compared miniature QWERTY with on-screen keyboards. When typing without a stylus, the on-screen keyboard is half as fast as the miniature QWERTY and the users committed more errors on the on-screen keyboard. Using actuators to create tactile feedback for the on-screen keyboard, improved both performance and error rates.

2.1.2.2 Character recognition

Another text entry method for touch devices is character recognition It can either be directly on a touch screen or on a touch pad. Software in the device will perform the character recognition, which converts the drawing into a character. The character recognition is often erroneous, so it is important with an easy method to correct mistakes. Most character recognition systems can be trained to better recognize each user's unique handwriting.

The text entry speed is mainly limited by the human performance and the recognition error rate. The average speed of handwriting is about 15 WPM [Devoe, 1967], so with a perfect character recognition algorithm, it should be possible to reach this level.

The performance can be improved by faster drawing of characters or fewer recognition errors. A number of special alphabets have been made to improve the performance [Goldberg and Richardson, 1993, Wobbrock et al., 2003, Castellucci and MacKenzie, 2008]. They all use simpler strokes to represent the letters. The simple strokes are faster to write and easier to recognize.

2.2 Optimizing text entry methods with language models

All the text entry methods introduced in the previous section are universal in the context of the used character sets. All words from all languages than can be constructed from the character sets can also be written with these methods. Some words might take more effort to write than other words. With multitap *tag* can be written by pressing 824 while the word *oil* requires you to press 666444555.

By using knowledge of how a language is constructed, text entry methods can be optimized to that language. Frequently used words and characters can be written with less effort than rarely used words and characters. Knowledge of the language is modelled by a language model, which is used to optimize the text entry method. This section will show how text entry methods can be optimized.

2.2.1 Optimizing character layouts

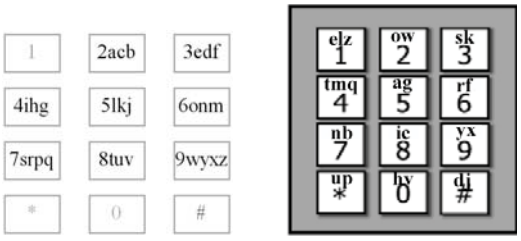
Most text entry methods are using an alphabetic or QWERTY layout of the characters. None of these layouts are optimized for speed. The speed of the text entry methods can be increased, by rearranging the characters so the most frequently used characters are easier to enter. Two such layouts are the OPTI keyboard by MacKenzie and Zhang [1999] and the Metropolis keyboard by Zhai et al. [2000].

For ITU-T keyboards with multitap, the order of the characters has a large affect on the performance. Less-Tap by Pavlovych and Stuerzlinger [2003] sorts the characters on each key, so the most used characters are placed first on the keys. The character layout is shown in figure 2.4(a). Compared to multitap, this reduces KSPC from 2.0342 to 1.5266.

In LetterEase by Ryu and Cruz [2005] all the characters have been rearranged. The 12 most likely characters were placed in the first position of each key. The

next 12 characters were placed in the second position and two keys got an extra character in the third position. Space is placed on a dedicated key. The layout can be seen in figure 2.4(b). An evaluation of LetterEase gave KSPC = 1.32, but it was found to be slower than multitap.

One problem with the optimized key layouts is that users have to learn them before they can benefit from the expected performance improvements. Until the new layout is learned, the performance will be lower than with a well known keyboard. Zhai and Smith [2001] shows that an improved layout algorithm with a term that bias alphabetical ordering can produce key layout that is easier to learn, and with a very small drop in the expected performance. Studies by Ryu and Cruz [2005] and Gong and Tarasewich [2005] confirm this.



(a) Less-Tap (b) LetterEase

Figure 2.4: Optimized layout for phone keyboards.

Dynamic rearranging of characters The introduced optimized key layouts have all been static. This have the advantage that the new layouts can be learned by training. Bellman and MacKenzie [1998] have published work on a method called Fluctuating Optimal Character Layout (FOCL) where the characters are rearranged after each entered character. The currently entered text and a language model are used to predict the likelihood for each character, to be the next entered character.

FOCL was evaluated with the Date stamp method by MacKenzie [2002b]. The FOCL layout was compared to an alphabetic layout. Both methods used Snap-to-home, which means that the cursor will return to the first character on the list after each entered character. Calculations show that Date stamp with FOCL requires half as many key presses as Date stamp with alphabetic layout. The evaluation showed no significant difference between the two methods. FOCL uses less key strokes, but requires extra time to locate the position of the characters.

Nokia used a combination of FOCL and alphabetic layouts in the Nokia 7280 and 7380 phones. FOCL was used to place the five most likely characters at the head of the list. All characters were appended to the list in alphabetic order. When writing a normal phrase 1/3 of the needed characters will be placed at the top of the list, and can be entered by one key press. Another 1/3 of the needed characters will be placed among the remaining 4 characters from the FOCL list. The last 1/3 of the characters can be found in the alphabetic part of the list, where the user will know approximately where they are placed. There have not been published any evaluations of text entry on Nokia 7280 and 7380.

2.2.2 Disambiguation from Language Models

The text entry methods presented until now, all uses extra key presses to select the correct character. There exist a number of text entry methods were characters are entered with only one key press per character. This class of text entry methods is called single-tap. They all use language models to select the correct character. Sometimes the language models select the wrong character. In these cases, the user will have to press extra keys to select the correct character.

2.2.2.1 Character level disambiguation

A simple type of disambiguation is prefix-based disambiguation as implemented in the LetterWise text entry method by Eatoni Ergonimics, Inc. With LetterWise the text is entered character by character just as multitap. When a key is pressed LetterWise will use a language model and the prefix characters to find the probability of all the characters on the pressed key. The most likely character will be written. If this is the wrong character, the user can press a next key to iterate through the characters on the key, sorted by probability. To write *sun* the user will first press 7. This will enter *s*. A press on key 8 will enter *t*, because *st* is very common in English. A press on the next key will change it to *su*. The last press will be on 6 and it will enter a *n*. In this case 4 key presses were needed to write the word. Kober et al. [2001] found that KSPC = 1.15 for LetterWise. The number of key presses needed to write a word depends on how well the word fits with the language model. Words that are well described by the language model will require few or none presses on the next key. Words that are unlikely or not known by the language model will require more presses on the next key. Novice performance of LetterWise is 6-8 WPM, which is similar to multitap. Expert performance after 10 hours is reported to be around 21 WPM [Kober et al., 2001].

2.2.2.2 Word level disambiguation

All the methods that have been introduced until now, have been able to write all possible sequences of characters. There exist $26^3 = 17,576$ different words with three letters. Only 570 of these words are widely used in the British National Corpus [Oxford University Computing Services, 2001] [1].

By limiting text entry systems to only enter known words, the number of key presses can be reduced.

For 12 key keyboards, 8 keys have letters assigned to them by the ITU-T layout shown in table 2.1. Each key represents 3-4 letters. Word level disambiguation, as implemented T9 by Tegic [James and Longé, 2000] and iTap by Motorola, is the most common type of disambiguation. It is sometimes called dictionary-based disambiguation. Each key is pressed once. To write *sun*, the user will press 786. This key sequence is ambiguous and can be any of the $4 \cdot 3 \cdot 3 = 36$ words that can be made by combining the characters on the keys: (7 pqrs) (8 tuv) (6 mno). A language model is used to create a list of the most likely words among all the possible words. Table 2.4 shows the five possible three letters words from the British National Corpus and their probabilities.

Word	Probability
run	69.3%
sun	27.0%
sum	1.4%
quo	1.3%
rum	1.0%

Table 2.4: Most likely words if the user has pressed 786 on an ITU-T keyboard.

The user is presented for the list and can choose the correct word. Some implementations show the entire list to the user and enable the user to navigate on the list. Other implementations only show the most likely word. A next key will then allow the user to iterate over the words on the list. If the desired word is not present on the list, it has to be entered by a fallback method such as multitap. Silfverberg et al. [2000] estimate that the most likely word matches the user's target word 95% of the time. Based on this Kober et al. [2001] estimates that KSPC = 1.0072 for word-based disambiguation like T9. This estimate is very low, and requires that the user only writes words that are known by the language model. The penalty for writing unknown words is high. First the user

[1] There are a total of 17,132,597 three letter words in the British National Corpus. 6596 of these are unique. Only words that occurs more than 171 times (0.001% of all three letter words) are counted as words that are widely used.

has to try entering the word using the normal text entry method. As the word cannot be recognized, the user needs to re-enter the word using the fallback text entry method. James and Reischel [2001] reports novice performance of T9 to be around 10 WPM and expert performance up to 25 WPM.

Word level disambiguation has also been used on devices with fewer keys. Dunlop [2004] presents a text entry method for wrist watches with only 4 character keys and one next key. Results show that novice performance with the method is only 40% slower than a standard mobile phone. In another study by Tanaka-Ishii et al. [2002] participants averaged on 14 WPM for a four key text entry method after 300 minutes of training.

2.2.2.3 Character or word level disambiguation

Word level disambiguation will always be more accurate than character level because more information is available for the language model. More information means that less key presses are needed to select the correct word. The word *sun* required 4 key presses for both word and character level disambiguation. If we instead write *run* it requires 3 key presses for word level and 5 key presses for character level disambiguation. Table 2.2.2.3 shows the process of writing *run* with the different methods.

Gutowitz [2003] and Eatoni Ergonomics Inc. [2001] describe a number of problems with word level disambiguation methods such as T9. The major problem with word level disambiguation is that the word shown in the display is very unstable during writing. Each time a new key is pressed the word will change. Even though the words are based on the same key presses, they can be very different. The example from table 2.4(a) shows how the suggested word changes from *st* to *run* with one key press. It can be very confusing that the characters in the display do not look like the target word. This holds especially for novice users. With character level disambiguation only the last character might be different from the target character. The word stability problem also makes it very difficult to detect and fix typing mistakes. One wrong key press is enough to write *slondp* instead of *summer*. It is very difficult to find the wrong key press, so often the user has to delete the entire word and write it again.

Another problem with word level disambiguation is that only words known by the language model can be written. As mentioned above, new words need to be added to the language model by a fallback text entry method. Character level disambiguation does not have this problem. All words can be entered, but rare words will require extra key presses.

(a) Word level

Key press	Shown in display
7	s
8	st
6	run

(b) Character level

Key press	Shown in display
7	s
next	p
next	r
8	ru
6	run

Table 2.5: Key presses for entering *run* with word and character level disambiguation

The problem with unknown words in the language model is bigger for some languages than others. One of the important factors is the use of compound words in the language. Compound words can be in closed form (*firefly, keyboard*), hyphenated form (*daughter-in-law*) or in open form (*post office, real estate*). Compound words in open or hyphenated form do not have to be in the language model, if the individual words are present. If the compound words are in closed form, then both the compound words and the individual words need to be present in the language model. Often there is not space enough for all the words, so only the most frequently used compound words are included.

2.2.2.4 Optimization of character layout for disambiguation

Like the character layout can be optimized for multitap, it can also be optimized for disambiguation. The optimal character layout will group the characters, so the sum of character probabilities on each key is the same for all keys. The optimal solution for a 4 key keyboard is a probability of 1/4 for each key. It will give $H(1/4, 1/4, 1/4, 1/4) = 2$ bits on information for each key press. If the characters is distributed differently we might get $H(1/2, 1/4, 1/8, 1/8) = 1.75$ bits or $H(3/4, 1/8, 1/16, 1/16) = 1.19$ bits for each key press. Gong and Tarasewich [2005] presents optimal character layout for ambiguity for keyboards with 1 to 12 keys. They investigated both alphabetically constrained and unconstrained layouts. They found that the difference between constrained and unconstrained layouts was very small, why alphabetically constrained layouts is preferred because of the familiarity.

2.2.3 Additional Text Entry Features

Language models can be used to improve many current text entry methods. Most of the features presented in this section can be applied to all text entry methods.

2.2.3.1 Automatic correction of spelling mistakes and typos

A language model can be used to identify and correct mistakes in text. It can either be spelling mistakes or typos. Typos are very likely on mobile devices, due to the small keys on physical and on-screen keyboards. Kukich [1992] gives an overview over different techniques for automatically correcting words in text. When the correcting is done while the user is entering text, extra information can be extracted from the text entry process. Goodman et al. [2002] implemented an error correction method for on-screen keyboards. On-screen keyboards can detect exactly where a key is touched. If the touch is near the boundary of the key, the user might intend to hit the neighbour key. The positions of the touch and a language model are used to correct errors.

Whiteout++ by Clawson et al. [2008] uses the timing of key presses to find errors. If the same key or two adjacent keys are pressed within 50-100 ms, then one of the key presses might be a typo. A language model and a decision tree are used to determine if one of the key presses should be ignored.

Like all other types of automation, automatic correction can be very frustrating for the user if it does not work. Too many false positives, where correct words are changed, will affect the user experience and performance of the text entry method.

2.2.3.2 Word completion

Word completion will suggest entire words based on few characters. A language model is used to create a list of candidate words that are presented to the user. It is implemented in many commercial and academic text entry methods. One of the first implementations was The Reactive Keyboard by Darragh et al. [1990]. It was intended for desktop computers. A list of text snippets was shown in a separate window, and the user could choose how much of the snippets that should be inserted into the text editor. The method required a computer mouse for the interaction, so it was mainly useful for novice users. MacKenzie et al. [2006] used word completion in the Unipad text entry method. Their analysis

shows that the average KSPC drops from 1.0 to 0.5 if word completion was used with five or more words in the candidate list. The average KSPC will drop if more words are included in the candidate list. The overall performance is very likely to decrease, because of the time required to scan all the candidate words.

Some implementations also suggest the next word or phrase based on the current text. AdaptTex by Dunlop et al. [2006] is one example of such implementations.

2.2.4 Problems with text entry and language models

There are a few problems that are general for all methods that use language models. The language model is bound to a language. This means that their need to be a language model for all supported languages. Language models can be very expensive to create and will also take up storage in the devices where they are used. For Europe alone there exist more than 250 languages [Laverack and von Niman, 2007].

To save time and money many commercial text entry systems only supports the major languages. The rest of the users will have to use the language that is closest to their language variant or dialect.

The quality of the language model is very important. If the language model does not reflect the user's language, it wills affects the performance and user experience of the text entry method. Kober et al. [2001] shows how the performance of T9 degrades fast when the target words are not represented by the language model. If 15% of the words are unknown, the performance in WPM will drop with more than 30%.

Several studies of mobile text messaging [Grinter and Eldridge, 2001, Rathje and Ravnholt, 2002] have found that the language used in text messages have evolved. In mobile text messages it is common to use non-dictionary words like cu l8er (see you later), *smiling*, ASAP (as soon as possible), LOL (laughing out loud) and smileys like ;-) and <:-(.

Field interviews by Gutowitz [2003] revealed that unknown words were the most frequent problem with the T9 method.

2.3 Language modelling

Many of the introduced text entry methods require a language model. Most language models are statistical models that can be used to estimate the probability that a given character sequence will occur in the language. This section will give an overview of different kinds of language models.

2.3.1 Information theory and language modelling

The field of information theory was initiated by Shannon [1948], with his mathematical theory of communication. Information theory has mainly been used in the fields of data transmission and compression, but has also been applied in language modelling. The entropy is the basic measure for information. If X is a discrete random variable with alphabet Q and distribution p, then the entropy in bits is defined as:

$$H(X) = -\sum_{x \in Q} p(x) \log_2 p(x) \tag{2.1}$$

A language model M can be defined as an information source that will output symbols $\mathbf{m} = m_0 m_1 m_2 ...$ from an alphabet Q.

Often the language model is used to estimate the probability of a given text. The probability of a text is the same as the probability that the language model will generate the same series of symbols. The following notation is used to give the probability of the text $\mathbf{x}$ being generated by the language model M:

$$M(\mathbf{x}) \equiv P(\mathbf{m} = \mathbf{x}) \tag{2.2}$$

Language models are often used to find the probability of a word or a character α, given the prefixed text $\mathbf{h}$.

$$P(\alpha|\mathbf{h}) \equiv \frac{M(\mathbf{h}\alpha)}{M(\mathbf{h})} \tag{2.3}$$

2.3.1.1 Assessment of language models

A common way to assess language models is to see how well a language model can describe text in a test corpus. The cross entropy between the language model and test corpus is used as a metric. If X and Y are a discrete random

variables with alphabet Q and distributions p and q, then the cross entropy in bits is defined as:

$$H(X,Y) = -\sum_{x \in Q} p(x) \log_2 q(x) \tag{2.4}$$

The cross entropy measures how many bits that are needed to describe X by Y. X is the *true* distribution of text from the test corpus. Y is the distribution of text from the language model. If $Y = X$, then $H(X,Y) = H(X) = H(Y)$.

The quality of language models are often described by the *perplexity* metric. It is defined as 2 to the power of the cross entropy of the test corpus T and language model M:

$$2^{H(T,M)} \tag{2.5}$$

If $T = \alpha_1..\alpha_n$, $n = |T|$, it can be described as:

$$2^{-\sum_{i=1}^{|T|} \frac{1}{|T|} \log_2 P(\alpha_i|\mathbf{h})}, \mathbf{h} = \alpha_1...\alpha_{i-1} \tag{2.6}$$

When measuring the entropy of text, the entropy is often reported in *bits per character* (bpc). A typical user might use up to 100 different characters when writing text. It includes lower and upper case letters, numbers, spaces, punctuation marks and language-specific characters. A complete random text where all 100 characters in the alphabet have the same probability will have $100 \cdot -\frac{1}{100} \log_2 \frac{1}{100} = \log_2 100 = 6.64$ bpc.

2.3.1.2 Information in Language

When writing normal text in a language, some characters will be likely and some will be very unlikely. For English characters like *e, t, h* and *space* will be very likely and characters like *z*, *{* and *** will be unlikely. Because of the skew distribution of the characters, the entropy of English is a lot lower than 6.64 bpc.

Shannon [1951] estimated an upper bound of the entropy of English on 1.3 bpc. Participants were presented for a snippet of text and were told to guess the next characters. By counting the number of guessed characters, he was able to calculate the entropy. Cover and King [1978] used a more refined method, where the participants could gamble on the next character in the text. By doing this they archived an upper bound on 1.25 bpc. Both these estimates were found by using humans to guess the next characters. The test corpora they had to guess were small and they only had to guess between 26-27 characters.

Brown et al. [1992b] presented an estimate of 1.75 bpc obtained by using a trigram language model (see section 2.3.3). The language model was trained on a corpus with about 587 million words. The entropy estimate was found by using the language model on a test corpus of one million words. All 95 printable ascii characters were used. Teahan and Cleary [1996] replicated the early work by Shannon and Cover, but used a PPM language model (section 2.3.6.2) instead of humans. They estimated the entropy to 1.46 for Shannons text and 1.726 for the text used in Covers work.

2.3.1.3 Information theory and text entry

The entropy of language is very relevant for text entry. A keyboard can be seen as a source of information. If the keyboard has n keys, that are all equally likely, the entropy of a key press is $\log_2 n$ bits. If chording (concurrent key presses) is supported, then the entropy will be $\log_2(2^n - 1)$. The -1 is because at least one key needs to be pressed. If the keyboard has n_m modifier keys (like Shift, Alt and Ctrl) then the entropy will be $2^{n_m} \cdot \log_2 n$ bits per key press.

The relation between the bits per key press, KSPC and bpc is described in equation 2.7.

$$\text{bpc} \leq \text{KSPC} \cdot \text{bits per key press} \tag{2.7}$$

If an alphabet has 32 characters, then you will need $log_2 32 = 5.0$ bits of information to write a character. Using a normal keyboard, this will require $2^5 = 32$ keys to be able to write one character per key press. A text entry method that requires two key presses to write a character (KSPC = 2) will need 2.5 bits per key press. This can be done with a keyboard with $2^{2.5} = 5.7 \approx 6$ keys. If KSPC = 3 then the $2^{5/3} = 3.2 \approx 4$ keys are needed. If the keyboard supports chording, then it is possible to enter a character with a single press on a keyboard with $\log_2(2^5 + 1) = 5.04 \approx 6$ keys.

All these examples give the minimum number of keys that are required to enter a character. The coding of key presses to characters needs to be optimal to achieve this minimum. Often it can be desirable to use a suboptimal coding, if it is easier to learn for the user [Zhai and Smith, 2001, Gong and Tarasewich, 2005, Ryu and Cruz, 2005].

If a language model is used, fewer bits are required to write a character. This can be lead to text entry systems with fewer keys or a lower KSPC value.

There are two pitfalls when using language models:

- The first is that the bpc is an average for the entire text. The entropy of a single character can be larger than the average bpc. This is often the case for the first letters in a word. This is due to the simple fact that it is a lot more difficult to guess the first letters of an unknown word, than it is to guess the fifth letter if the first four letters are known. The experiment by Shannon [1951] also shows this effect. This is also the reason that word level disambiguation in general requires less key presses than character level disambiguation as described in section 2.2.2.3.

- The other problem that can occur is when the user writes text that does not match the language model. The bpc of this text will be a lot higher than expected, and can lead to a severe degrade in performance of the text entry method. A good example is T9, where the user is forced to use multitap to write words the language model does not know.

2.3.2 Creating a language model

Most language models are created by compiling a large text corpus. The text in the corpora is often taken from the Internet where it is easy to collect a large amount of text. It could be from sites such as Project Gutenberg (`http://www.gutenberg.org/`) that has more than 25,000 free electronic books or from the online encyclopaedia Wikipedia (`http://wikipedia.org/`). Prior to the wide adoption of the Internet, many corpora were based on books and articles from newspapers.

There are other corpora, where there is more information about the texts. The British National Corpus consists of more than 100 million words, which have been tagged with their word class and other information. Examples of Danish language are found in Korpus 90, Korpus 2000 and KorpusDK from the Society for Danish Language and Literature (Det Danske Sprog- og Litteraturselskab).

It is important that the corpus reflect the users language. Early Danish mobile phones with T9 could not write 'knus' (hug). Instead the word 'lovs' (law's) was suggested. *Hug* is probably not frequently used in newspapers compared to *law's*.

2.3.3 Simple n-gram and n-graph models

n-gram and n-graph are simple statistical language models based on word or character frequencies in the corpus. n-grams describe words and n-graphs describe characters. This section will mainly describe n-grams, but everything can be transferred directly to n-graphs. The n describes the sequence length that is used in the model. For n = 1, 2 or 3 the words unigram, digram or trigram are typically used. A unigram model is a list of single words and their frequencies. A digram model is a list of word pairs and so on. When n increases the size of the model increases exponentially. A simple unigraph model can be implemented in 27 bytes, if one byte is used for for storing the probability of each of the 27 English characters including space. A digraph model will require $27^2 = 729$ bytes and a trigraph model $27^3 = 19,683$ bytes. n-gram models take up a lot more space because of the many words, and because they cannot be enumerated the same way characters can.

The probability of a word w can be found by dividing the frequency with the total number of words. Here $C(w)$ is the count of word w in the corpus and N is the total number of words in the corpus.

$$P(w) = \frac{C(w)}{N} \tag{2.8}$$

For digrams, the probability of a word w, given the previous word w_p can be calculated as shown here. W is the set of all words from the corpus.

$$P(w|w_p) = \frac{C(w_p w)}{\sum_{w_i \in W} C(w_p w_i)} \tag{2.9}$$

2.3.4 Smoothed n-gram and n-graph models

A common problem with n-gram and n-graph models is that it is very likely that some n-grams or n-graphs do not occur in the corpus. Of the more than 100 million words in the British National Corpus, there exist only 20 trigraphs that start with `xk` and end with a letter. They are all shown in table 2.6. Based on this table the probability $P(\texttt{xka}) = 0.0$. In the real world $P(\texttt{xka})$ might be very small, but it should never be 0. In this section the trigraph `xka` have already occurred three times. Another problem is that $P(\texttt{xkx})$ is a lot lower than $P(\texttt{xky})$, even though `xky` only occurred once more than `xkx` in the around 500

million trigraphs in the British National Corpus. Both these problems can be avoided by using smoothing or discounting techniques to give better estimates of the probabilities.

Trigraph	Count	Probability
xki	2	0.10
xkl	5	0.25
xku	6	0.30
xkv	2	0.10
xkx	2	0.10
xky	3	0.15

Table 2.6: Trigraphs from British National Corpus that start with xk and end with a letter.

2.3.4.1 Add one smoothing

Add one smoothing is a simple smoothing function that adds 1 to all the counts and multiply with $N/(N+V)$ [Jurafsky and Martin, 2000]. It is based on Laplace's law of succession. N is the count of all n-grams and V is the number of possible n-grams. The probability function with add one smoothing is:

$$P(w) = \frac{C(w)+1}{N+V} \tag{2.10}$$

Add one smoothing solves the problem of some n-grams having 0 probability. But the value 1 is arbitrary chosen, and could also be set to any other value. Large values will favour unknown n-grams, while small values will favour known n-grams. Another problem with Add one smoothing is that you need to know the number of n-grams. With n-graphs this will be limited by the used character set, but for n-grams it can be impossible to know the upper limit of n-grams.

2.3.4.2 Witten-Bell discounting

A better solution introduced by Witten and Bell [1991] is now known as Witten-Bell smoothing. The sum of probabilities for all unknown n-grams is set to:

$$\sum_{w,C(w)=0} P(w) = \frac{T}{N+T} \tag{2.11}$$

T is the number of unique n-grams seen so far. The probability for known n-grams is:

$$P(w) = \frac{C(w)}{N+T} \tag{2.12}$$

If there are Z unknown n-grams, the probability for each unknown n-gram is:

$$P(w) = \frac{1}{Z}\frac{T}{N+T} \tag{2.13}$$

Z will often be unknown, but the sum of all unknown n-grams can always be calculated. This is called the escape probability.

2.3.4.3 Good-Turing discounting

Good-Turing was introduced by Good [1953]. It is based on the assumption that the counts of n-grams that occurs few times (less than 5-10 times) in the corpus are not reliable. They are replaced by a smoothed count C^*:

$$C^*(w) = (C(w)+1)\frac{N_{C(w)+1}}{N_{C(w)}} \tag{2.14}$$

N_x is the count of n-grams that occur x times. It can be seen as the frequency of frequencies.

2.3.5 Combinations of language models

It is very common to combine language models to get a better model. It could be different types of models or models of different orders. A common example is to combine a general language model with a specific language model. The specific language model can either cover a special domain or it can be an adaptive personal language model that reflects the user's vocabulary.

2.3.5.1 Linear Interpolation

Linear interpolation estimates the probability by a weighted sum of the probabilities from all the language models. If M is the set of all the language models, it can be expressed as:

$$P(\alpha) = \sum_{m \in M} \omega_m P_m(\alpha) \tag{2.15}$$

If we combine different orders of the same language model, the weights could be calculated by the blended context model (first described by Cleary in 1980, here taken from Bell, Witten, and Cleary [1989]). Here i is the order of the model from 0 to l:

$$\omega_i = (1 - e_i) \prod_{j=i+1}^{l} e_j \tag{2.16}$$

$$\omega_l = (1 - e_l) \tag{2.17}$$

e_i is the escape probability for a model of level i. This is the probability for an unknown word or character. It can be calculated by the discounting functions from the previous section. Sometimes the combined model is extended with a model of level -1, where all words or characters have the same probability.

2.3.5.2 Backoff

Backoff by Katz [1987] and blended context models with exclusions are other methods to combine different orders of the same language model. They only uses the model of highest order with nonzero probability to estimate the probability.

If the combined model consist of a unigram, digram and trigram language model, the two previous words $w_{pp}w_p$ will be used to calculate the probability of a new word w, $P(w|w_{pp}w_p) = \alpha_3 P(w|w_{pp}w_p)$. If the trigram $w_{pp}w_pw$ does not occur in the corpus, the model wills backoff to a lower level model. In this case $P(w|w_{pp}w_p) = \alpha_2 P(w|w_p)$. If w_pw does not occur the probability will be estimated by a unigram model $\alpha_1 P(w)$.

Backoff will typical be used together with a discounting function. The values of α depends on the choice of discounting function.

2.3.6 Adaptive Language models

Adaptive language models are useful because they can adapt to the user's language. This is good because new words are added to the language and the frequencies of known words will change over time. Studies by Grinter and Eldridge [2001] also show that many non-dictionary words are used in mobile text messages. Words like *millennium* and *y2k* were used a lot around year 2000 but have not been used much before and after. Other words like *snow* or *Christmas* are bound to different seasons. The symbol @ was not used much before the early nineties. Today it is used by all people who are writing emails.

Most research has focused on creating adaptive language models by combining a general corpus with a domain specific corpus. Examples of such models are given by Bellegarda [2004] and Wandmacher et al. [2008].

2.3.6.1 Move to front

Move to front is a simple adaptive language model, where the words or characters are sorted after their last occurrence in the text. Each time a word or character occurs, it is moved to the front of the list. The actual implementation depends on the text entry method. If move to front is used with a word-level disambiguation method like T9, then each key sequence will have its own candidate list. The candidate list for the key sequence 786 is shown in table 2.4. It is `run, sun, sum, quo, rum`. Each time the user enters a word from the list, the given word is moved to the front of the list. Move to front is better than a static candidate list if the static list does not reflects the users' language. Otherwise the performance of move to front is likely to be worse than a static list. Each time a rare word is used, it will be moved to the front on the list, thereby making it more difficult to write likely words. Another problem is that many experienced users have learned the order of the candidate lists for the most used words. Move to front changes the order very often, which will disturb experienced users. The frequent changes in the candidate lists require extra visual attention from the user, to verify that the entered word is correct.

2.3.6.2 Prediction by Partial Match

Prediction by partial match (PPM) is a data compression technique made by Cleary and Witten [1984]. It can be described as an adaptive n-graph model with backoff. Different smoothing techniques can be used to estimate the probability of unknown n-graphs. The optimal value of n is around 5 when it is used for

data compression. A version of PPM with unbounded variable length n-graphs are presented in Cleary et al. [1995]. In data compressions PPM is always used with n-graphs. In language modelling it can both be used with n-graphs and n-grams. Deligne and Bimbot [1995], Niesler and Woodland [1996], Kneser [1996] all describes how variable length n-grams can be implemented.

2.3.7 Other language models for text entry

This section has mainly described statistical language models. There exist many other types of language models that make use of semantic and syntactic information.

Brown et al. [1992a] used information theory to group words from a corpus into different classes. One class could be weekdays, nationalities or similar. A class-based n-gram model was created, such that the model contained information about the classes and not the actual words. This requires less storage space, and is able to model the semantic of the corpus.

Stocky et al. [2004] used common sense from OMCSNet, a large-scale semantic network, (`http://openmind.media.mit.edu/`), as a language model. OMCSNet can find words related to a query word. The word *chilli* might return words such as *food, hot, red, etcetera*. It was used to disambiguate between words on a single-tap mobile phone text entry method.

Trigger models are an extension to n-gram models. They are similar to n-gram models, but there is a distance between the n-1 first words and the last word. The first words are said to trigger the last word. It is based on the assumption that if some words have occurred recently, then other words are more likely. For example the word *airport* might trigger words as *ticket, flight, security control, etcetera*.

2.4 Trade-offs between size, speed and complexity

The different text entry methods that have been presented in this section all have their strengths and weaknesses. Figure 2.5 shows the relations between the text entry methods, according to their relative speed, complexity and physical size of the input device. The complexity is estimated as a combination of the learnability and memorability of the text entry methods.

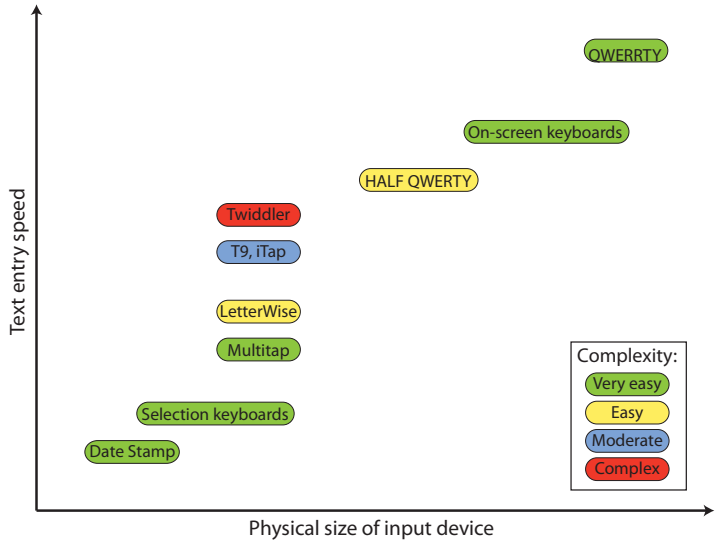


Figure 2.5: Text entry speed, complexity and physical size of input device for different text entry methods.

The figure shows two clear trends: There is a correlation between text entry performance and size of input device and between text entry performance and the complexity of the text entry methods.

2.4.1 Physical size of input device

For the *very easy* text entry methods there is a correlation between the size of the input device and the text entry speed. Only key based methods are included in the figure. This means that the size of the input device can be replaced with the number of keys used to enter text. If fewer keys are used, the bits per key press will be lower. As shown in equation 2.7, this will require the text entry method to have a higher KSPC value. Higher KSPC values mean that more key will have to be pressed to enter the text. This will result in lower text entry speed.

Common for all the text entry methods in the *very easy* group, is that they are deterministic. Each key press performs a well defined action that can be learned very fast. No language model is used in any of the methods.

2.4.2 Complexity

The multitap, LetterWise, T9, iTap and Twiddler methods are all made for ITU-T keyboards. The reported text entry performance of the methods range from 6 to 26 WPM. Novice performance for all methods is 6-8 WPM. The methods with higher complexity can reach higher expert performance.

Multitap and Twiddler are both deterministic. The reason for the high complexity of Twiddler is that the user has to learn chording and remember all the chording combinations. The study by Lyons et al. [2004] shows that it takes 2 hours of practice with Twiddler before it is as fast as multitap. Even after 6 hours of training, Twiddler is only 1/3 faster than multitap.

The two other methods are uncertain. It is not possible for a user to know exactly what will happen, when a key is pressed. This is due to the use of language models. The words or characters shown in the display are dependent on input from the user and the language model. If the language model is static, it will be possible to remember the predicted words for frequently used key combinations. As described in section 2.2.2.3 word level disambiguation is more difficult to learn and use than character level disambiguation. This makes the complexity of T9 and iTap higher than LetterWise.

2.4.3 Skill acquisition

The complexity of text entry methods was defined as a combination learnability and memorability. These factors are very important for how easy a new text entry method is to learn and to master.

Interaction with computers can be seen as a control process (figure 2.4.3). The user has an intention to do something with the computer. It could be to enter the word *sun* with the multitap method. As the user presses the first key (7) the state of the computer changes. The display will show an *r*. The feedback from the computer is perceived by the user. Since *r* is not the first character in *sun* the user will continue to press the 7 key. After four presses the feedback from the computer will match the intention and the user can continue with the rest of the word.

The complexity of multitap is very low, so it is easy for novice users to learn the method. Based on the feedback from the computer the user will start to create a model of how multitap works. Norman [1998] calls this model the User's model. The combination of the computer system and the documentation called the

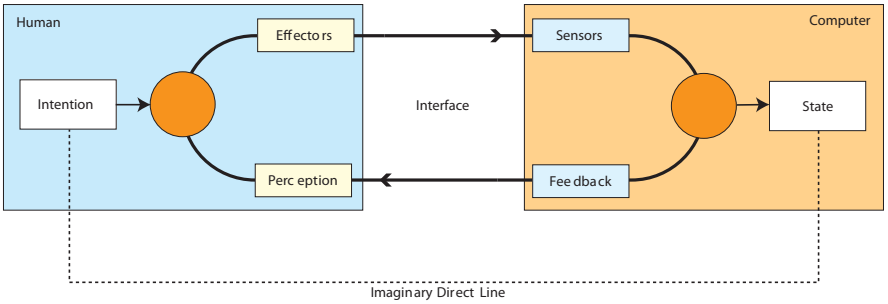


Figure 2.6: Human-computer interaction as a closed-loop control process. From Williamson [2006]

System image. Sharp, Rogers, and Preece [2007] use the terms conceptual and mental model. The conceptual model is created while the system is designed, and it explains how the system works. The mental model is created in the brain of the user when interacting with the system. If a user's mental model does not reflects the conceptual model, then the user will experience difficulties when using the system.

Studies of T9 and similar methods by Eatoni Ergonomics Inc. [2001], Gutowitz [2003] showed that many people did not have a correct mental model of how T9 works. They were confused by the predicted characters. The feedback were perceived, but could not be interpreted correctly. When the feedback cannot be interpreted, it becomes difficult to compare the state with the intention and to select which actions to perform.

The problems can be solved by giving users an introduction to the text entry methods. This works well when doing usability evaluations. For commercial products it is unlikely that users will invest a lot of time to learn new text entry methods. It is therefore important that text entry methods have a simple conceptual model. This will enable the users to create a correct mental model when they are using the text entry methods.

Creating a correct conceptual model is the first step to become efficient with a new text entry method. A correct conceptual model of multitap enables the user to find the required number of key presses. *s* is the fourth character on key 7 (*pqrs*), so four presses are needed to write *s*. With this knowledge the user can enter *s* by making four consecutive key presses on 7. This is faster than before where the user had to perceive the feedback after each key press.

The next step to become more efficient is to use the system. When the

same actions are performed repeatedly, automatic processing will be developed [Schneider and Shiffrin, 1977]. Automatic processing will decrease the attention demands and improve the performance. The user will learn that *s* requires four consecutive key presses on 7. Some of the actions needed to enter text will change from closed-loop processes to open-loop processes. Open-loop means that the need for feedback is gone. Other actions will have reduced their need for feedback. A user that has learned the ITU-T layout can enter text without visual feedback. Tactile feedback is still used to position the finger on the keys and to detect when the keys have been pressed.

With all deterministic text entry methods the entered text is only dependent of the pressed keys. This makes it possible for the user to enter text without feedback from the text entry method. If the text entry method contains a language model it will be nondeterministic. The user needs feedback to compare the state with the intention. It is possible for experienced users to learn parts of a language model. For example that *sun* is the second word in the candidate list for 786. Just like the novice user can learn that *s* is the fourth character on 7. Adaptive language models can be very confusing for experienced users, if the learned order of candidate words changes.

The QWERTY keyboard is good example of a successful text entry method. The conceptual model is very simple and can be learned fast. The layout of the characters will be learned after some time. Frequent use will develop automatic processing, where text can be entered with very little attention. This leaves more attention to the most important task: What to write.

The design of TUP in section 4 is an attempt to create a text entry method for mobile device that is easy to learn and use.

2.5 Conclusions

There exist many different mobile text entry methods. The diversity of the methods are very large. So are the diversities in their speed, complexity and size requirement. The physical size of the input device is important for how fast and easy a text entry method is to use. It is therefore advisable to use as much space as possible to the input device. This makes it difficult to create unified text entry methods that can be used on a range of different mobile devices.

Language models can be used to increase the performance of text entry methods. It will often be at the cost of increased complexity. It is likely that the users will write new words that are unknown by the language models. It is

therefore important that new words can be written easily, without a large drop in performance. This is not the case with many text entry methods to day.

Language models should be able to adapt to the user's language. This will make the predictions from the language model more precise, and thereby increasing performance and user experience of the text entry methods.

Chapter 3

Evaluation of text entry methods

This chapter describes the methodologies used to evaluate the text entry methods in this thesis.

3.1 Evaluations

Evaluations of new and improved text entry methods are an important step to assess the quality of the methods. It is a central part of most interaction design processes. The quality of a text entry method is a combination of the usability and user experience. Usability is the effectiveness, efficiency, safety, utility, learnability and memorability of a method [Sharp et al., 2007]. The performance of a text entry method is part of the usability. Performance is measures of text entry speed, error rates and similar. User experience is everything users experience from a product. It could be from advertisements, buying the product, using the product, etcetera. For text entry methods it will mainly be the users' experience while using the text entry method. Examples of user experiences could be satisfying, enjoyable, helpful, challenging, boring, frustrating or annoying. Usability and user experience is closely related. If a

text entry method have poor performance, it is very likely that the user will feel frustrated of annoyed.

Evaluations can be divided in three different categories:

Usability testing. Made in a controlled environment where users will perform tasks with a product.

Field studies. Done in natural settings together with users.

Analytical evaluations. Expert evaluations and modelling. Conducted with out involving users.

Usability testing traditionally focuses on finding problems with a product or to get performance data. Field studies focus on how products are adopted and used by people in their natural settings. Both usability testing and field studies are done with users. Analytical evaluations are done by experts without involving users. It can either be a walk through of a product, or a model of the interaction between the users and the product. Elements from all three types of evaluations are used in the thesis.

3.2 Mobile evaluations

For text entry methods designed for mobile devices there might be a big difference in performance between a lab evaluation and a field study. In the lab the user will typical be sitting down, using two hands, having a controlled climate, good light conditions, no noise and be able to focus only on the text entry task. In the real world, the user will often have to input text outside, while driving or walking around, in the shop or maybe in the bed at night. Oulasvirta et al. [2005] found that the average continuous span of attention to mobile devices were between 4 to 8 seconds when they were used in the field. In a lab the span of attention was about 14 seconds. The environmental conditions are also very different in the field. It might be cold and windy outside or difficult to see the display because of the sun. People are pushing to each other in the shop, and there is no light when writing text in the bed at night. Text entry methods that require a lot of visual attention might work well in a lab or office environment but can be difficult to use in the field.

If the text entry method is intended for mobile use, it is important that it is also evaluated in the users' natural settings. A comparative study by Hoggan et al.

[2008] showed that users can enter text 40% faster in a lab than on a subway train. Despite this most text entry evaluations are done in labs.

To get realistic performance data most of the evaluations in this thesis are done in the field. Elements from usability testing and field studies are combined to create field usability tests. Focus is on gathering performance data in realistic natural settings of the users.

3.2.1 Prototypes for mobile evaluations

Three different interactive prototypes of TUP have been made to evaluate usability and user experience. The first prototype was a mock-up where a touch pad was attached to a cardboard phone and connected to a computer. The prototype was unhandy, and therefore only usable for usability testing in a lap.

The two next prototypes were constructed to be used in field usability tests. Prototypes made for field evaluations need to have a small form factor, be battery driven and be solid enough to be carried around. The TUP prototypes were both based on the Apple iPod with custom software.

3.2.2 User conducted evaluation

Normally an evaluator will conduct the usability testing with the users. The evaluator will be responsible for controlling the prototype and guide the users through the evaluation tasks. For field usability testing it might not be feasible, or even possible, to have an evaluator next to the user all the time. This hold especially for longitudinal studies.

In the two field usability tests in this thesis, the users have to conduct the most of the evaluation by them self. There are briefing and debriefing sessions together with the evaluator. In between them, the users will be on their own. Besides the prototype, they are given a small notebook and a pen, so the users can write comments and feedback to the evaluator. The requirements to quality of the prototype are high, because the prototype has to be usable by the participants without any help.

The data gathering from the evaluations is done by the users and the evaluation software in the prototype. The users will write contextual information and comments in the notebooks. The evaluation software will create logs of all the interaction with the prototype.

3.3 Text entry evaluation design

The goals of the text entry evaluations in the thesis are to find performance data, to find the participants initial reactions and to understand how TUP is used by the participants.

The participants have to write phrases with TUP on the different prototypes. The writing task can be made as a text copy or text creation task [MacKenzie and Soukoreff, 2002]. With text copy the participants are shown a phrase and have to write it. With text creation the participants have to create their own phrases. Text creation mimics every day use, but is difficult to use in text entry evaluations. The intended phrases are only known by the participants, so it is not possible to detect errors. The participants might also be slowed down because the have to create the phrase and enter it at the same time. Therefore text copy is used in all the evaluations in this thesis. The participants are told to memorize the phrases before they start writing. Only short phrases are used, to make it easy to memorize them.

The phrases presented to the user will be called the *presented text*. The phrases entered by the user will be called the *transcribed text*.

The *unconstrained text entry evaluation paradigm* Soukoreff and MacKenzie [2001], Wobbrock and Myers [2006] is used at most evaluations in the thesis. This method allows the participants to enter and edit the text as the like. This gives a more natural interaction style, where errors in the transcribed text are allowed.

3.3.1 Evaluation phrases

The presented phrases have a large effect on the evaluation results, because some phrases are easier to write than others. This holds especially if the text entry method contains a language model. If for example T9 is evaluated it is crucial that the used words are known by the language model. Otherwise the performance will be very poor. This relation between the phrases and the performance can problematic. You might end up testing the language model and not the text entry method. This can be abused to favour certain text entry methods by picking the right phrases. It also makes it difficult to compare different text entry evaluations.

To avoid these problems MacKenzie and Soukoreff [2003] have published a set of 500 short phrases for use in text entry evaluations. The phrases are all written

in English. They are lower cased and without punctuation. This phrase set has been used in many recent text entry evaluations.

The phrases are used in the two last evaluations in the this thesis. The first evaluation uses 5 other short English phrases. The participants in the evaluations are mainly university students and faculty. The participants have different nationalities, but all can speak and write English. A study by Isokoski and Linden [2004] shows the effect of foreign language in text entry performance. In a evaluation with a QWERTY keyboard, Finns were 16% slower when writing in English compared to writing in Finish. They also made significantly more errors when writing in English. In some text entry evaluations the phrase set have been translated to avoid this effect.

All used evaluation phrases in this thesis are English. This is because of the mixed nationalities of the participants and to make it possible to compare the results with other evaluations.

3.3.2 Text entry performance

The performance data from the evaluations will focus on text entry speed and error rates. Methods from Soukoreff and MacKenzie [2003, 2004], Wobbrock and Myers [2006] are used. The difference between the presented and transcribed text will not tell anything about how many errors that have been made and corrected by the users. Therefore it is also necessary to know the input stream. The input stream consists of all inputs made by the participant. It includes characters (correct and incorrect) and editing commands (backspace, delete, cursor movements, etcetera).

Wobbrock and Myers [2006] provide the StreamAnalyzer application that can analyse an input stream and compute text entry speed and error metrics. The different characters in the input stream are tagged with a class:

C Correct character

IF Incorrect character that is fixed

INF Incorrect character that is not fixed

F Editing commands (F stands for Fixes)

The tagged characters can be used to calculate the following metrics:

WPM: Text entry speed in words per minute (3.1)

Uncorrected Error Rate: Errors that are not corrected (3.2)

Corrected Error Rate: Errors that are corrected (3.3)

Total Error Rate: The sum of corrected and uncorrected error rates (3.4)

The metrics are described here.

$$WPM = \frac{|\mathbf{C}| + |\mathbf{INF}| - 1}{\mathbf{S}} \cdot 60 \cdot \frac{1}{5} \tag{3.1}$$

The time in seconds **S** used to enter a phrase is calculated as the difference between the time stamps for the first and last character. The notation $|\mathbf{C}|$ is used for the count of characters in class **C** for a given input stream.

$$\textit{Uncorrected Error Rate} = \frac{|\mathbf{INF}|}{|\mathbf{C}| + |\mathbf{INF}| + |\mathbf{IF}|} \tag{3.2}$$

$$\textit{Corrected Error Rate} = \frac{|\mathbf{IF}|}{|\mathbf{C}| + |\mathbf{INF}| + |\mathbf{IF}|} \tag{3.3}$$

$$\textit{Total Error Rate} = \frac{|\mathbf{INF}| + |\mathbf{IF}|}{|\mathbf{C}| + |\mathbf{INF}| + |\mathbf{IF}|} \tag{3.4}$$

3.3.3 Learning rates

The learning rate of a text entry method describes the relation between text entry rate and how long time a user has used the method. It is based on the power law of learning from Card et al. [1983]. The model has the form

$$\mathrm{WPM}_n = \mathrm{WPM}_1 \cdot n^x \tag{3.5}$$

WPM_n is the text entry speed after n repetitions. WPM_1 is the initial text entry speed and x is the learning rate. n is the number of repetitions. As described by Isokoski and MacKenzie [2003] the learning rate is good to describe observations, but cannot be used to predict the effect of more training. The main reason is that the predicted text entry rate will grow toward infinity if enough training is done.

Chapter 4

Transparent User guided Prediction

This chapter describes Transparent User guided Prediction (TUP). TUP is a novel text entry method for devices without keyboards. TUP is introduced and implemented in two different prototypes. The prototypes are evaluated in two usability evaluations. Interaction logs from the last evaluation are used to improve TUP. The improved version of TUP is implemented in a third prototype and evaluated. A TUP version for keyboards is proposed.

4.1 Introduction to TUP

TUP was developed to enable easy text entry on small devices with touch sensitive wheels. With TUP all characters are assigned to fixed positions on the wheel. Characters can be highlighted by touching the wheel. A language model and model of human behaviour are used to make it easy to highlight the most likely characters.

TUP was designed for touch input devices, but can be used in with all input devices that can detect a continuous absolute input. It can either be used with a combined input/output device such as a touch sensitive screen or isolated devices

such as touch sensitive wheels or sliders and a display. The early development of TUP is described in my master thesis [Proschowsky, 2005b].

TUP is not a single complete text entry system. TUP is mainly a method for selecting characters from a character set. It contains different interaction styles that control how the selected character is entered. Several graphical user interfaces have been created for TUP. The choice of interaction style and graphical user interface is dependent on the capabilities of the device. Some interaction styles require extra keys to be added to the device. The choice of graphical user interface is highly dependent of the screen size in the device.

TUP does not describe any text editing features, such as delete functions, cursor movements, text highlighting and copy/paste. It is easy to add these features to TUP, but the actual implementation is dependent on the device.

4.1.1 Design of TUP

TUP can be seen as an improved version of Date stamp. The characters are shown in a display, and can be highlighted and entered with the touch sensitive wheel. There is a direct mapping between the wheel and the highlighted character. This enables the user to highlight all characters directly, by touching the wheel at the corresponding point. With the normal date stamp method the user has to scroll to highlight a character.

The design of TUP is based on these main requirements:

Simple conceptual model. It should be very easy for novice users to start using TUP. The complexity of the TUP from the user's point of view should be very simple. The transition from novice to expert user should be straightforward.

Character based. Text should be entered one character at a time because this is easier for novice users. Another advantage of this is that the text entry is not limited to known dictionary words.

Hidden use of language model. Language models can speed up text entry performance, but can be complicated for the users to understand. TUP will use a hidden language model, where the user will not have to know about, or interact with, the language model.

4.1.1.1 Use of language models

The high density of characters on the wheel makes it difficult to highlight a target character. To make it easier a language model is used to predict what characters the user will write next. The predictions are used in the character highlighting algorithm in TUP, so the most likely characters are easiest to highlight. The disadvantage of this is that less likely characters will be more difficult to highlight. This is a problem if the language model does not represent the user's language. To reduce the problem, the movement of the users´ finger is used to control the influence of the language model. When the user places his or her finger on the wheel or makes a fast movement the method will make heavy use of the predictions. A slow movement will make very little use of the predictions, enabling less likely characters to be highlighted. This is based on the assumption that the user will perform a fast movement to get to the approximate target area. If the desired character is not highlighted, the user will make a fine finger movement to highlight the correct character.

The use of prediction is transparent to the user. Because of the transparency, the user will not need to have any knowledge about how the prediction works, and will not use any mental resources on the prediction. This is an improvement compared to most other predictive text entry systems, where knowledge of the prediction system is required, and where the user has to interact with the prediction system while writing. One exception is the HEX text entry method by Williamson and Murray-Smith [2005]. HEX uses motions sensors and a hidden language model to let the user enter text.

4.1.1.2 Example of use

To write *top*, the user will first place the finger on the wheel where t is expected to be placed. If the placement is precise, t is highlighted in the display and can be written by pressing the select button. Otherwise the user will need to scroll a little bit clockwise or counter clockwise, until t is highlighted and can be selected. The next character to be written is o. The language model predicts that o is very likely to occur after t, compared to the other characters near o. In this case the prediction algorithm will highlight o if the finger is placed between q and m. Otherwise a fine movement is needed to highlight o. The last character p is more difficult to write, because o is more likely to follow to than p. Even if the finger is placed directly on p, o will be highlighted. A short slow movement clockwise on the wheel will highlight p, so it can be selected.

4.1.2 Variations of TUP

TUP can be implemented on a wide range of different devices. Dependent on the capability of the device, different variants of TUP can be used. The variants cover different interaction styles and graphical user interfaces.

4.1.2.1 Interaction styles

The TUP text entry method is mainly focusing on how the characters are highlighted. The actual entering of a character is done when the user selects the highlighted character. This can be done in different ways; by a dedicated select key, by clicking on the wheel or when the finger is removed from the wheel. Not all methods can be implemented on all devices.

For a touch sensitive wheel the list of possible interaction styles looks like this:

Click-to-Select The highlighted character is entered when the user clicks on a dedicated select key. It could be placed in the centre of the wheel or any other place on the device. If it placed in the centre of the wheel, then it will typically be used by the same finger the users have used on the wheel. This means that the user will have to move the finger from the wheel to the key. If the key is placed somewhere else on the device, then the user will be able to press the key with another finger. This will enable the user to keep the first finger on the wheel while entering the character.

Click-on-Wheel The highlighted character is entered when the user clicks on the wheel. The user can keep the finger on the wheel and enter the character by pressing on the wheel. Some touch sensing devices can register the force of the touch. These devices can apply an algorithm to detect when the touch was strong enough to enter the character. Touch sensing devices without this capability needs to place the touch sensing device on top of a key to get the same results.

Release-to-Select The highlighted character is entered when the user removes the finger from the wheel. This does not require any extra key presses, but the user might enter unintended characters if the finger slips from the wheel. The user might also want to remove the finger from the wheel without entering a character. This can be implemented by an escape gesture. If the speed of the finger is beyond a threshold when it is removed, then no character will be entered.

4.1.2.2 Graphical user Interfaces for text entry

The user interface to TUP can be implemented in many ways. There need to be a text entry area and a way to see the currently highlighted character. Beside that most users will need a list of all the characters or a part of them. Expert users might know the order of the characters, but the rest of the users will need these lists to be able to write text in an acceptable speed. Users with no knowledge of the order of the characters will benefit from having the complete list of characters. For users with some knowledge of the characters it will probably be sufficient with a partial list of characters. The list should contain the current highlighted character and the surrounding characters. The character lists can be designed in all possible ways. To give the best usability, the design of the character list should correspond to the design of the input device. Figure 4.1 shows six examples of user interfaces for TUP.

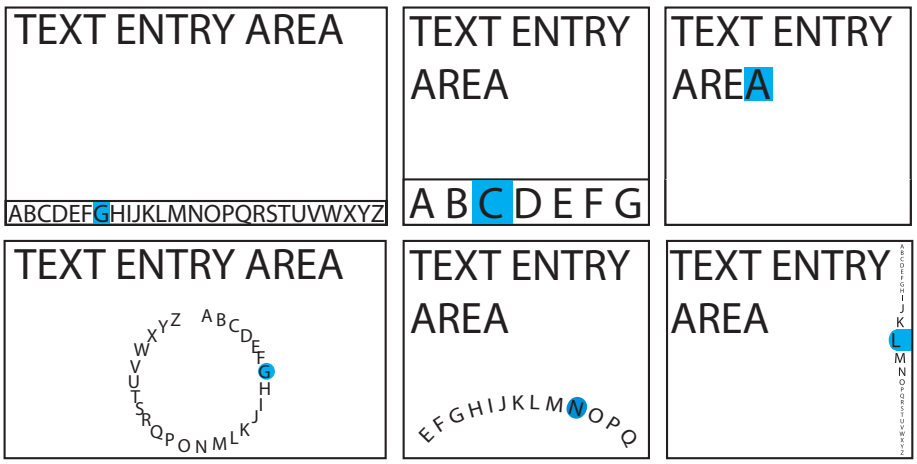


Figure 4.1: Different user interfaces for the TUP text entry method. From the top left: list of characters, partial list of characters, single character, character circle, partial character circle and list of characters with fish eye view.

For use with a touch sensitive the wheel the character circles are the best choice. Their disadvantages are that they take up a lot of space from the text entry area. It might be possible to solve this problem by having a semi transparent character circle.

4.1.2.3 Implementation of text editing and other features

The focus on TUP in this thesis concentrates on the methods to highlight and select characters. A full featured text entry system also needs to implement text editing functions and other features. It could be back space, cursor movements, confirmation of written text, copy/paste and similar features. The implementation of the user interfaces for these features is very dependent of the used hardware. If the device has a lot of keys they can be used to access the features. If the device only have few extra keys they can be used to activate an 'editor' menu, where the touch sensitive wheel is used to navigate in a menu with different text editing options. If the device does not have any extra keys, it is possible to append *virtual keys* to the list of characters. Virtual keys can be selected like any other character. Instead of inputting characters, the virtual keys will initiate an action when they are selected.

4.1.3 Character highlighting algorithm

A central part of TUP is the character highlighting algorithm. It is used to calculate which character to highlight based on the user's gestures. The algorithm has a high impact on the input speed and user experience of TUP. The algorithm is used while the user is performing the gesture, to continuously update the highlighted character.

4.1.3.1 Algorithm design

The algorithm is based on Bayes rule. $\boldsymbol{\alpha}_{1..n}$ is the string of characters entered by the user. α_1 is the first character and α_n is the last. Each entered character is the result of a gesture from the user. It can be a single tap on the wheel or longer gesture. The variable φ_i represents the position of the user's finger at time t_i. $\boldsymbol{\varphi}_{1..i}$ is the positions of all the touches in the current gesture. $\varphi_i \in [0, 2\pi[$. 0 is defined as the top of the wheel and the circumference of the wheel is defined as 2π. The values are increasing clockwise. α_i is the character that is highlighted at time t_i. φ_α is the position of character α on the wheel.

The goal of the character highlighting algorithm is to always highlight the most likely character α_i, given the gesture $\boldsymbol{\varphi}_{1..i}$.

$$\underset{\alpha_i}{\operatorname{argmax}} P(\alpha_i|\boldsymbol{\varphi}_{1..i}) \tag{4.1}$$

By using Bayes rules we get:

$$P(\alpha_i|\boldsymbol{\varphi}_{1..i}) = P(\alpha_i|\varphi_i, \boldsymbol{\varphi}_{1..i-1}) = \frac{P(\varphi_i, \boldsymbol{\varphi}_{1..i-1}|\alpha_i)P(\alpha_i)}{P(\varphi_i, \boldsymbol{\varphi}_{1..i-1})} \tag{4.2}$$

Bayes rule gives that:

$$P(\varphi_i, \boldsymbol{\varphi}_{1..i-1}|\alpha_i)P(\alpha_i) = P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)P(\boldsymbol{\varphi}_{1..i-1}, \alpha_i) \tag{4.3}$$

If equation 4.2 and 4.3 are combined we get:

$$P(\alpha_i|\boldsymbol{\varphi}_{1..i}) = \frac{P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)P(\boldsymbol{\varphi}_{1..i-1}, \alpha_i)}{P(\varphi_i, \boldsymbol{\varphi}_{1..i-1})} = \frac{P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)P(\alpha_i)}{P(\varphi_i)} \tag{4.4}$$

$P(\alpha_i)$ is the probability that character α_i is the next character to be written. It can be calculated in many ways as described in section 2.3. In the prototypes of TUP it is estimated from a trigraphs language model, with out any smoothing functions. It can be seen in equation 4.5. $\alpha_{n-1}\alpha_n$ are the two last characters the user has entered. The language model is created from The British National Corpus [Oxford University Computing Services, 2001].

$$\Theta(\alpha_i) = \frac{C(\alpha_{n-1}\alpha_n\alpha_i)}{C(\alpha_{n-1}\alpha_n)} \tag{4.5}$$

To make it possible to select all characters, a constant of 0.1 is added to all probabilities. The formula for $P(\alpha_i)$ is shown in equation 4.6. $\varepsilon_{\text{predict}}$ can be used to control the influence of the language model. Values less than 1 will favour characters with low probability and values larger than 1 will favour characters with high probability.

$$P(\alpha_i) = \frac{(\Theta(\alpha_i) + 0.1)^{\varepsilon_{\text{predict}}}}{\sum_\alpha \Theta(\alpha) + 0.1)^{\varepsilon_{\text{predict}}}} \tag{4.6}$$

$P(\varphi_i)$ is estimated as constant, since the design of the input device is uniform. No positions of the wheel are more likely to be touched than others. It can therefore be eliminated from equation 4.4.

$P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)$ is estimated as a Gaussian distribution centred around φ_{α_i}.

$$P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i) = \frac{1}{\sqrt{2\ \pi\sigma^2}} e^{-(\varphi_i - \varphi_{\alpha_i})^2/2\sigma^2} \tag{4.7}$$

By combining equation 4.1, 4.4, 4.6, 4.7 and removing constant terms we get:

$$\underset{\alpha_i}{\operatorname{argmax}} \left(\Theta(\alpha_i) + 0.1\right)^{\varepsilon_{\text{predict}}} e^{-(\varphi_i - \varphi_{\alpha_i})^2/2\sigma^2} \tag{4.8}$$

The algorithm consists of two parts. A language model and a low level model of human motor behaviour. The language model can be seen as a task model, that models what the user wants to do. The combination of the models by Bayes theorem gives a unified model of which character to highlight, based on the user's gestures.

4.1.3.2 Parameter estimation

σ^2 and $\varepsilon_{\text{predict}}$ are dependent on the users' movements. They are set to λ_i, which is defined as:

$$\lambda_i = 0.01 + \frac{2}{1 + e^{-|\varphi_i - \varphi_{i-1}| \cdot 0.3}} - 1 \tag{4.9}$$

When the user makes slow movements λ will be close to 0, and when the movements are fast it will be close to 1. Fast movements will make heavy use of the language model and the variance of the Gaussian distribution will be large. Slow movements will make little use of the language model and the variance of the Gaussian distribution will be small.

4.1.3.3 Robustness

It is important for the user experience that the character highlighting algorithm is robust. The highlight should not flicker between characters if the user makes small movements. It is important the highlight only changes when the user makes an intentional movement. The algorithm includes three methods to make it more robust:

Threshold The values from the input device might change very often, even if the user tries not to move the finger. It can be due to electrical inference or to tiny movements of the finger. These movements should not result in a new highlighted character. Therefore the algorithm includes a threshold that defines how much the value from the input device should change before a new calculation of which character to highlight is made.

The threshold for the TUP implementation is set to $\pi/48$ which corresponds to 1/96 of the circumstance of the wheel. This is the same as the resolution of the touch sensitive wheel used en the iPod prototypes.

Direction of scroll Due to the design of the algorithm it might be possible that a small clockwise movement on the touch sensitive wheel will make the highlighting move counter clockwise. This is very unintuitive. It can happen if the user have made a fast movement so $\lambda \approx 1$. This can result in the highlighting of a character α_i far away from the current touch φ_i. If this is followed by a slow movement, a character close to φ_i will be highlighted.

To ensure that the change of highlight will be in the same direction as the movement, equation 4.10 needs to be true before a new character will be highlighted. α_i is the new highlighted character and α_{i-1} is the previous highlighted character.

$$(\varphi_i - \varphi_{i-1})(\varphi_{\alpha_i} - \varphi_{\alpha_{i-1}}) \geq 0 \tag{4.10}$$

Robustness of parameters The problems mentioned in the previous section is due to the fact λ_i from the character highlighting algorithm is directly dependent on $|\varphi_i - \varphi_{i-1}|$. To make the algorithm more robust λ_i from equation 4.9 is replaced in the algorithm. It is replaced by a running weighted average of λ as shown in equation 4.11. Informal evaluations showed that $\beta = 0.3$ was a reasonable value to use.

$$\overline{\lambda_i} = \beta \cdot \lambda_i + (1 - \beta) \cdot \overline{\lambda_{i-1}} \tag{4.11}$$

λ_0 is defined as the inverse entropy of the next character [Cover and Thomas, 1991]. Each time the user enters a new character, the user needs to make a choice between all the characters. The amount of information in this choice can be calculated. Based on the previous characters, the probability for each character can be found. This can be used to calculate the entropy. The entropy

indicates how much effort the user needs to select the next character. The inverse entropy is used as the initial value of λ as shown in equation 4.12. If there are only few likely characters, then the entropy will be low and λ_0 will be high. This will make it very easy to highlight the few most likely characters. If there are many likely characters the entropy will be high and λ_0 will be low. This means that the user needs to be more precise to touch the wheel at the correct position.

$$\lambda_0 = \frac{1}{\sum_\alpha -\Theta(\alpha) \cdot \log(\Theta(\alpha))} \tag{4.12}$$

4.1.4 Slip error correction algorithm

TUP will input the highlighted character when the user removes the finger from the wheel. Sometimes the user will make a small movement before the finger is removed, which can change the highlighted character to a adjacent character. The slip happens shortly before the finger is released. This will input the wrong character and will confuse the user. To avoid this, a simple slip error correction algorithm is built into TUP. It will enter the previous highlighted character if the following criteria are met:

- The current highlighted character has been highlighted for less than 100 milliseconds.
- The current highlighted character is adjacent to the the previously highlighted character.

The parameters and the values in the algorithm were found by trial.

4.2 TUP - Hardware mockup

The goal for the evaluation is to compare TUP with date stamp and a variant of TUP.

4.2.1 Construction of Prototype

The prototype for the first evaluation carried out in this PhD project consists of a touch wheel input device and a PC with a monitor. The touch wheel input device is made from a touchpad from an IBM Travel Keyboard. A mask with a circular slid and a hole in the middle is placed on the touchpad, to resemble a touch wheel with a centre select key. The touchpad is removed from the keyboard, and placed on the front of a small cardboard box. A post-it note is placed on the box to make the touch wheel input device look and feel more like a mobile phone or mp3 player.

The dimension of the input device is 98 x 48 x 12 mm. Figure 4.2 shows the touch wheel input device. Unfortunately the USB chip set from the keyboard can not fit in the box. Instead the input device is connected to the USB chip set in the keyboard with a 15 cm long wire. The wire put some restrictions on the ergonomic of the input device, by limiting the user's freedom of movement. The user had to hold the device close to the desk where the keyboard was placed.

The keyboard is connected to a PC through a USB cable. The TUP method is implemented on the PC, and the graphical user interface is displayed on the PC monitor. An example of the user interface can be found in figure 4.3. The prototype uses two different click sounds as auditory feedback. One sound were used to signal each time a new character was highlighted and another sound were used when a character were selected. The prototype used the Click-to-Select variant of TUP. Only lower case text entry was implemented in the prototype. The PC software was designed to log all interaction from the input device.

4.2.2 Evaluation methodology

To measure and compare text entry speed, error rates, usability, and user experience for TUP, it was evaluated together with two other methods. The text entry methods in the evaluation were:

- TUP. Implemented with Click-To-Select.

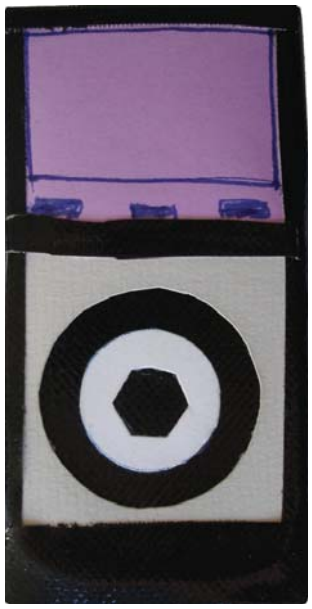


Figure 4.2: Prototype used in first evaluation. A touchpad is hidden under the wheel and connected to a PC

- FIXED. Language independent version of TUP. The characters still have fixed positions like TUP but no prediction is used.

- WHEEL. The date stamp text entry method used with a wheel. Space is placed before 'a' and persistent cursor mode is used.

These methods were chosen to compare TUP against the date stamp method, and to find the effect of the fixed character positions and use of prediction. All three methods were implemented in the same prototype. The evaluation consisted of two parts. An usability test to measure text entry speed and error rates and interviews to find the user's experiences with the different text entry methods.

Six persons were used in the evaluation. Two female and four male, all right handed. They were between 19 and 36 years old. Each participant had to perform three sessions. In each session the participants were asked to write the same five sentences with all three methods. The sentences were between 22 and 35 characters long, with 30 characters as the average length. The sentences were:

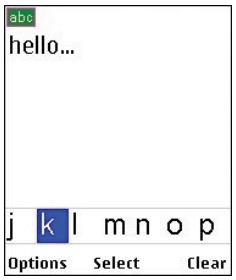


Figure 4.3: User interface shown on the PC monitor

1. what time should we meet tonight
2. let me know if we should wait
3. can i borrow your book
4. should i pick you up at the station
5. when will you be back in town

The participants used the text entry methods in balanced order as shown in table 4.1. They were instructed to have a short break after each method. If they made mistakes while typing, they were instructed not to correct the mistakes, but to continue writing. They were asked to hold the device in their right hand, and use their thumb for writing. Software on the PC logged all input for later analysis of input speed and error rates. After completing all three sessions they were asked to rate five statements about each method. The statements are listed in table 4.2. The statements should be rated on a 7 point Likert scale, where 1 equals disagree completely, 4 equals neither disagree nor agree and 7 equals agree completely.

Participant	P1	P2	P3	P4	P5	P6
Order	WFT	WTF	TFW	TWF	FTW	FWT

Table 4.1: The order in which the participants tried the methods. T(up), F(ixed) and W(heel)

The text entry speed was calculated as the number of correct characters in the transcript minus one, divided by the time between the first and last character. The error rates were calculated as the Minimum String Distance as described by Soukoreff and MacKenzie [2003].

Area	Statements
Speed	Writing with this method is fast
Errors	Writing with this method generates few errors
Learn	This method is easy to learn
Use	This method is easy to use
Overall	This method is the best method for doing text entry

Table 4.2: The users were asked to rate five statements about each text entry method.

Session	TUP		FIXED		WHEEL	
1	4.8	(1.4)	4.4	(1.0)	4.1	(0.8)
2	5.8	(1.5)	5.1	(1.0)	4.6	(0.8)
3	6.2	(1.5)	5.5	(1.2)	4.7	(0.9)

Table 4.3: Text entry speed in WPM from the evaluation. Standard deviation given in parentheses.

4.2.3 Results and discussion

The evaluation was held in August 2005. All data can be found in the evaluation report [Proschowsky, 2005a]. The participants used 5-10 minutes for each method in each of the three sessions. Table 4.3 and 4.4 shows the text entry speed, error rates and standard deviations from the evaluation.

A two-tailed paired t-test was performed on the data from the last session. The results shows that there is a significant difference in text entry speed between TUP and FIXED ($T(5) = 8.28, p < .001$), TUP and WHEEL ($T(5) = 7.02, p < .001$) and between FIXED and WHEEL ($T(5) = 3.31, p < .001$). TUP is the fastest method, and WHEEL the slowest method for all three sessions. The differences between the error rates are not significant between any of the methods: TUP and FIXED ($T(5) = -1.67, p = .100$ns), TUP

Session	TUP		FIXED		WHEEL	
1	1.9	(2.9)	2.1	(3.0)	3.3	(4.4)
2	1.9	(2.8)	2.7	(3.3)	2.1	(3.8)
3	1.2	(1.9)	2.5	(3.9)	2.4	(4.9)

Table 4.4: The total error rates in percent from the evaluation. Standard deviation given in parentheses.

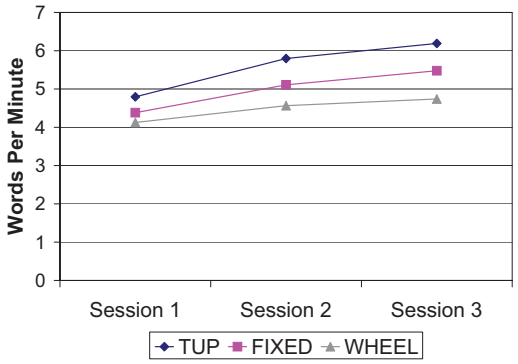


Figure 4.4: Effects of learning on the text entry speed.

Statements about methods	TUP	FIXED	WHEEL
It is fast	4.8	4.5	3.0
I make few errors with it	4.2	4.2	4.5
It is easy to learn	5.3	5.3	6.2
It is easy to use	5.2	5.2	4.3
It is best for text entry	5.7	5.0	2.3

Table 4.5: Rating of statements on a 7 point Likert scale. 1 equals disagree completely, 4 equals neither disagree nor agree and 7 equals agree completely

and WHEEL ($T(5) = -1.01, p = .971$) and between FIXED and WHEEL ($T(5) = 0.03, .987$ns). Figure 4.4 shows the effect of training on the text entry speed. The text entry speed increased for all three methods, as the users were getting more experienced.

Table 4.5 shows the ratings of statements for all three methods. TUP and FIXED are experienced to be fastest, which corresponds to the evaluation data. The error rates are experienced to be almost equal, which corresponds with the evaluation data, where no significant difference was found. WHEEL is experienced to be easiest to learn, while TUP and FIXED are experienced to be easiest to use. Overall TUP is rated at the best method, followed by FIXED. WHEEL is rated to be worse than the two other methods.

4.2.4 Limitations of the Prototype

The main problem with the prototype was the distance from the input device to the PC monitor. Some participants spent lot of time shifting focus back and forth, while other participants mainly looked at the monitor. It is expected that an improved prototype device with combined touch wheel and display, will speed up the text entry rates for all three methods.

4.3 TUP - iPod implementation

One of the problems with the earlier prototypes was that they all needed a computer to run the software. It is necessary to develop a true mobile prototype of TUP, to be able to evaluate TUP in natural settings. The evaluation should focus more on the learning of TUP. Both the participants' initial use of TUP and how their use changes as they are getting more experienced. To be able to study the effect of learning the participants needs to use TUP for a longer period than in the first evaluation. To get more sound results, the number of participants is also increased.

4.3.1 Construction of Prototypes

An important goal for the prototype was that the participants should be able to use the prototype by them self. That would make it easier to test the prototype in the users' natural settings. A number of hardware and software requirements were defined based on this goal.

- Mobile form factor. The prototype should be able to fit in a pocket or a small bag.
- Battery driven. The prototype should be driven by batteries so it can be used wherever the participants want to use it.
- Robust. The risk of breaking the prototype should be very small.
- Usable TUP implementation. The prototype should be powerful enough to use the TUP input method, without any delays in the user interface.
- Easy-to-use evaluation software. The software should enable the participants to carry out the evaluation sessions without help from an evaluator.

4.3.1.1 Hardware design

Different choices of hardware were evaluated for the prototype. The three hardware solutions were:

Hardware mockup Build hardware with a touch sensor, screen, cpu board, etcetera. Very flexible solution, but also expensive and time consuming. The final prototype might be clumsy or fragile.

iPod Use an Apple iPod with special software. High quality screen and touch wheel and nice compact form factor. Limited software development tools.

PDA, Mobile phones Use a mobile phone or pda with a touchscreen and a mask to make it work like a touch wheel. High quality screen and well known user interface. The mask might be fragile. Possible to use wireless connection to communicate with the users and to collect evaluation data. Many software development tools available.

The final choice was the Apple iPod because of the form factor and robustness. The dimension of the device is 103.5 x 61.8 x 10.6 mm and the weight is 138 gram. It has a 2.5 inch backlight screen with a resolution of 320 by 240 pixels. It includes a 30 GB hard drive and has battery for 4-5 hours of operation [Apple, 2006]. A problem with the Apple iPod is that it only has five keys, where four of them are placed on the wheel. This makes it harder to design good interaction with the device.

The touch wheel on the Apple iPod can detect 96 different positions. Position 0 is placed in the top op the wheel. The position number increases clock wise until it reaches 95, which is next to position 0. The touch wheel can only handle one touch at the time. If the wheel is touched multiple places, the outcome will be unstable.

4.3.1.2 Software design

The Apple iPod comes with proprietary software and no software development kit. The software can be replaced with iPodLinux, which is a customized Linux kernel and software stack optimized for embedded devices (`http://ipodlinux.org/`). Besides the kernel there exist a user interface toolkit TTK [Oreman, 2007b] and a user interface [Oreman, 2007a] called Podzilla that resembles the user interface in the Apple software. TTK supports pluggable text entry modules, so the built-in text entry methods can be replaced with other text entry methods. Podzilla can be extended with extra applications by writing plug-ins.

The cpu in the Apple iPod does only support integer arithmetic, because it does not have a floating point unit. The compiler can convert floating point arithmetic to integer arithmetic at compile time. This will slow down the execution time, because each calculation requires more cpu cycles. An implementation of TUP in floating point arithmetic and with no optimization was able to work at a rate of 15-20 updates per second. Each update includes sampling of user input, calculations of probabilities, updating of the graphical

user interface and logging the interaction to a file. This speed is good enough to secure fluent interaction, so no optimizations were made to the code.

TUP was implemented with the Character-Circle user interface. This takes up a lot of screen real estate, but leaves enough space to the Text Entry Evaluation Tool. A photo of the user interface can be seen in figure 4.5. All three interaction styles are implemented, though the participants only will use Select-on-Release. The prototype uses a limited English lower case character set with the 32 most common letters and punctuation marks.

The Text Entry Evaluation Tool was designed to allow the participants to complete the evaluation sessions by them self. Each time the tool is started the display shows the session number, the number of phrases completed in the session, the total writing time for the session and a short user guide (figure 4.6). By clicking the centre key, the user is presented for a new phrase. Beneath the phrase is an input field, where the user can type in the phrase (figure 4.5). When the phrase is completed, the user has to click the Menu key. This will take the user back to the main screen.

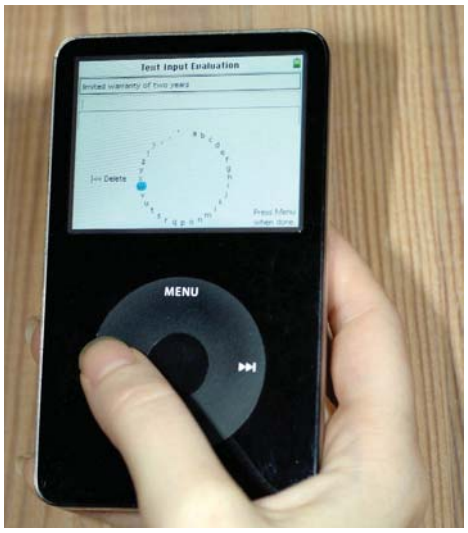


Figure 4.5: The TUP iPod prototype and the Text Entry Evaluation Tool.

4.3.1.3 Problems with the prototype

There was a problem with the power consumption on the prototypes. The Apple software uses undocumented power saving features in the hardware. The Linux

Figure 4.6: The TUP iPod prototype. The left image shows the back of the iPod. The right image shows the main screen in the Text Entry Evaluation Tool.

kernel does not use this features, so it drains the battery a lot faster than the Apple software. The Apple software includes a sleep function, which will put the device into a low power mode when it has been idle for 2-3 minutes. This function was not available in the Linux kernel. When running Linux, the battery time dropped to about 3 hours of operation.

4.3.2 Evaluation methodology

The main goals of the evaluation were to gather performance data for natural mobile use of TUP and to investigate how novice users responded to TUP. The participants would each have the Apple iPod for one week, and was told to complete two 10 minutes sessions each day. It was 10 minutes of active writing, which took 15 minutes to perform for most of the participants. The participants were told not to complete two sessions directly after each other, but to have at least a 30 minutes break in between. They were allowed to perform the sessions where ever they liked.

The participants were instructed to read and memorize the presented phrases

before they started writing. They were told to only correct mistakes if they were discovered shortly after they had been committed. The only editing feature in the TUP iPod prototype is the delete key, which will delete the last character. This makes it expensive to correct mistakes, if too many characters have been written after the mistakes.

To gather information about the context of use, each participant were given a small diary and a pen. The diary contained forms which had to be filled for each session. The form included the session number, date/time of the session, the place or context, light condition, motivation level of the participants and a comment field for further comments and notes. The diary also contained a short guide to the Apple iPod and to the evaluation software. Figure 4.7 shows the front and back of the diary. Figure 4.8 shows the inside of the diary and the form the participants have to fill out for each session.

The gathering of contextual information could have been implemented in the iPod evaluation software. The diary was chosen to give the participants a well known tool for writing notes. The iPod does not have any way to enter text, except from the TUP method that is evaluated. The participants do not know this method, so it would be a bad idea to force the participants to use the method. To make it easy for the participants, the diary had the same size as the Apple iPod and included a small pen.

4.3.2.1 Briefing session

Each participant was invited to a individual briefing session. The session consisted of six parts:

Initial reaction to TUP. The participants were told to write a short sentence or their name with TUP. They were not given any explanation, except that they were told to use the touch wheel. Verbal reports [Boren and Ramey, 2000] from the participants were used to gather information about how they thought TUP worked while they used it. This part of the evaluation was recorded on video for documentation purposes.

Explanation of TUP. The participants got an introduction to TUP, but were not told about the character highlighting algorithm.

Delivery of iPod, charger and diary to the participants. The participants got all the equipment for the evaluation.

Explanation of the input evaluation. The participants were instructed how they should use the iPod and the software. They were told only to correct

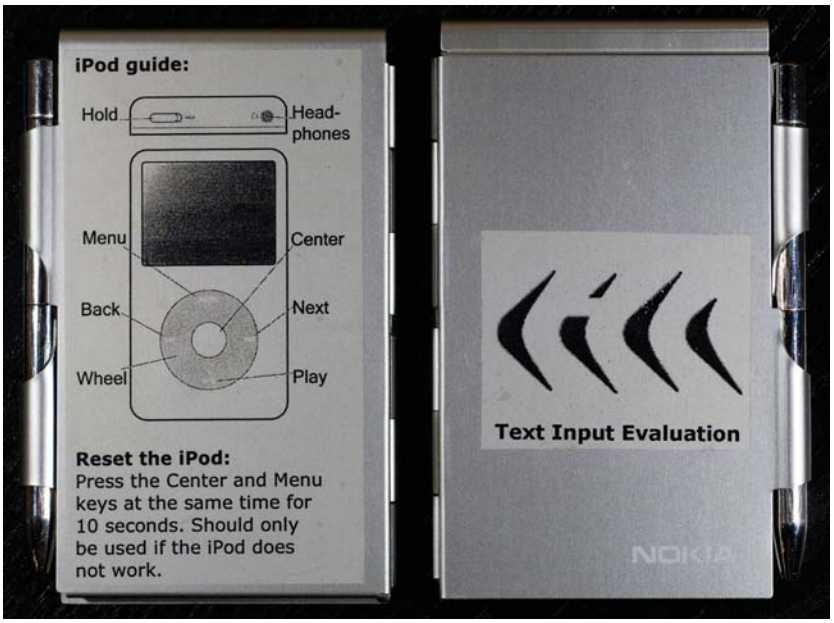


Figure 4.7: Front and back of the evaluation diary

mistakes if they were found shortly after they had been committed. They were explained how to use the notebook and the charger.

Test session. The participants completed a short test session to verify that they could use the iPod, text entry evaluation tool and TUP.

Interview. A short structured interview was made to gather demographic information about the participants. They were asked about there name, sex, age, left/right handed, mobile text entry experience, iPod experience, IT experience and their English skills. The experience and skill levels were later converted to low (L), medium (M) or high (H). The user did not take part in this conversion.

4.3.2.2 Debriefing session

After one week of evaluation sessions, the participants were invited to an individual debriefing session. At the debriefing the participants wrote one phrase on the prototype. The writing was recorded on video for documentation purposes.

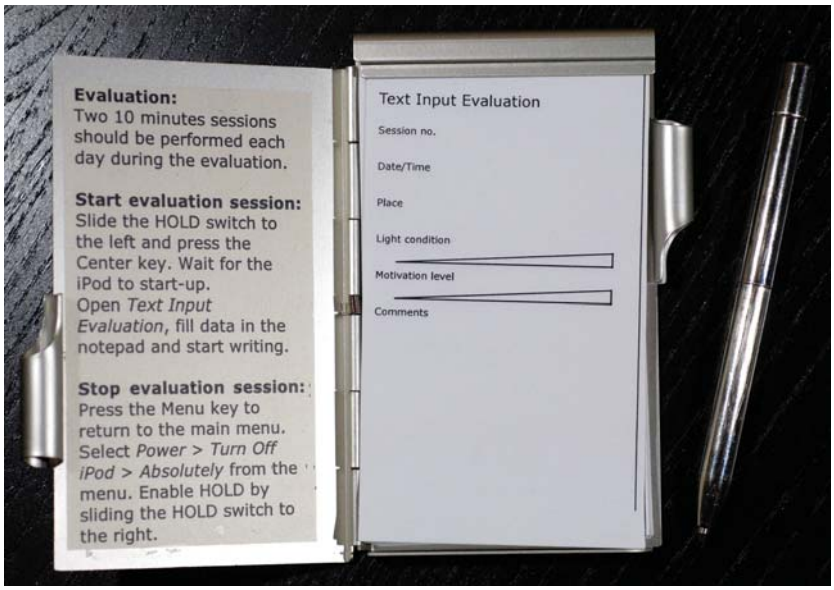


Figure 4.8: Inside of the evaluation diary with the forms

Semi structured interviews were made to gather data about the participants' experience with TUP. The following questions were used as a starting point:

- How was it to write text in the beginning? Which problems did you experience?
- In which way did training affect your writing?
- Describe how you experience TUP at the end of the evaluation. Which text entry situations were difficult?
- The participants were introduced to the character highlighting algorithm. Did you notice the algorithm?
- Do you have any suggestions or improvement ideas?
- The participants were introduced to the other interaction styles (Click-on-Wheel and Click-in-Center). What do you think about the variations?

4.3.2.3 Phrases used for the Evaluation

The phrases used in the evaluation were taken from the set of 500 phrases by MacKenzie and Soukoreff [2003]. The length of the phrases varies between 16 and 43 characters, with an average of 28.61. The phrases do not contain punctuations or other special characters. All phrases were converted to lower-case, because the used implementation of TUP only supported lower-case text entry. The order of the phrases was shuffled for each participant. The participants will not get the same phrase twice, unless they completed all 500 phrases. In this case they will start from the top of the list again.

4.3.2.4 Logging of data

To be able to analyze TUP in depth, an intensive amount of data were logged. For each input session, two different interaction logs were created.

Input log (named `session##_inputlog.xml`) A high level log of the text entry evaluation.

Touch log (named `session##_touchlog.txt`) A low level motor behaviour log of the position of the user's finger on the touch wheel.

4.3.2.5 Slip error evaluation

Two participants (UID: 81, 82) used a special variant of the TUP method. This variant did not include the slip error correction algorithm. This part of the evaluation was needed to create an improved slip error correction algorithm in section 4.4.2.

4.3.3 Results and discussion

The evaluation was carried out from November 2006 to March 2007. A total of 15 persons participated. Their demographics are listed in table 4.6. Most participants completed 12-14 sessions. Two participants only completed 6 and 7 sessions, because they experienced technical problems.

UID	Sex	Age	Hand	Sessions	Text Entry	iPod	Tech	English
1 (pilot)	F	22	Right	7	M	L	M	M
2	M	20	Right	6	M	M	H	M
3	M	32	Right	13	L	H	M	H
4	M	39	Right	13	L	L	H	M
5	F	30	Right	13	H	L	M	H
6	F	44	Right	14	L	M	M	M
7	M	33	Right	12	H	M	H	H
8	F	36	Right	14	M	L	L	M
9	M	46	Right	13	M	L	M	H
10	M	51	Left	12	M	L	M	H
11	M	26	Left	12	H	H	H	H
12	F	24	Left	13	M	L	M	H
13	F	23	Right	10	H	M	M	M
81 (slip)	M	49	Right	14	M	H	H	M
82 (slip)	M	23	Right	12	H	M	H	M

Table 4.6: Participants in the evaluation of TUP iPod implementation. The last four columns show the participants previous experience with mobile text entry, iPods, technology and their English skills. They were all expressed in words and later converted to low (L), medium (M) or high (H).

4.3.3.1 Initial performance and user experience

The following section is based on the observations of participants writing with TUP, on the interviews and on the diaries.

When the participants were introduced to the evaluation they were only told that they could use the touch wheel to select the characters. The TUP iPod prototype uses two special interactions methods, which differ from the common norm. The first is the direct mapping between the touch wheel and the character circle. The norm is to use a relative mapping, which is also the case for the Apple software. The second thing is the use of Select-on-Release. In almost all devices you have to push, flip or touch to perform an action. With the Select-on-Release the action will be performed when the finger is removed from the touch wheel.

Most of the participants found the method very difficult for the first 20-30 seconds. The highlight seemed to jump forth and back, and many participants entered a lot of character with out noticing. This is due to the special interactions methods. After another 20-30 seconds approximately 2/3 of the participants learned the absolute mapping between the touch wheel and the character circle.

Half of the participants learned how to use Select-on-Release by them self. A 1/4 of the participants thought they had to push the centre key to input the highlighted character. Pressing the centre key requires you to remove the finger from the wheel. This means that the highlighted character will be entered and the participants are likely to think that it is the result of pressing the centre key. On the iPod prototype the centre key was not mapped to any action. The last 1/4 of the participants did not learn the method during the first 2-3 minutes. Most of the participants were able to use the delete key without any instruction.

Most of the participants said they needed 2-4 sessions before they felt comfortable with TUP. Many had problems in the beginning with the touch wheel being too sensitive, so the highlight changed very quickly. The participants used two different input strategies; scrolling and tapping. Scrolling means that the finger is placed near to where it was removed from the wheel and then scrolled towards the target character. In tapping the finger is placed directly at the target character. All participants started out with scrolling, but most of them changed to tapping.

Some of the fastest participants were using tapping, with out visual confirmation that the target character was highlighted. This resulted in more errors, but they found that they could write a lot faster with this approach. 1/3 of the participants noticed the prediction in the character highlighting algorithm. They understood that the algorithm was relaxing the requirement to their precision. Other participants had noticed something, but could not explain it or use it. Some felt that the highlight stuck to some characters, and that it could be difficult to write sequences like ‘ a ‘ (space-a-space). In general many participants reported problems with hitting space. Some suggested that space should be located multiple places on the character circle. Other wanted a separate physical key to input space. There were also reports on problems with double letters, because the probability changed when the first character were entered. This can result in two sequential taps at the same place on the touch wheel, will produce two different characters.

Many participants mentioned that they learned the position of the characters in the circle, and that it helped them a lot. Some participants mentioned the small font size as a problem. The height of the characters is 2-3 mm, so they can be difficult to read. A number of participants mentioned the lack of feedback when characters were entered. They would like some sound or visual feedback to indicate that a character was entered. It is very likely that better feedback would have made it easier for the participants to learn the method during the initial session.

Many of the participants accidentally aborted the evaluation while writing. They were hitting the Menu key when they were about to highlight space. This

will end the entry of the current phrase, and return to the main screen of the text entry evaluation tool. This was mainly a problem during the first sessions.

The participants had been using the prototype in many contexts. The most common places were their offices, in the couch or using public transportation like trains or buses. Multiple participants mentioned that writing was difficult in the buses because of the bumping and accelerations. They did not notice any differences between the other contexts.

Most of the participants preferred the Select-on-Release interaction style, because they could omit key presses. The preference for Select-on-Release is strongly influenced by the fact the all participants have used the method during the evaluation. The tactile feedback from the key press with Click-to-Select and Click-on-Wheel were appreciated by many participants. They would like to have similar feedback with Select-on-Release, to make it easy to know when a character had been entered.

4.3.3.2 Text entry speed and error rates

The input log were analysed to find the entry speed and error rates. In total the 15 participants had entered 59,070 characters, in 2099 phrases.

Filtering of data Some participants accidentally aborted the evaluation or were disturbed in the middle of entering a phrase. These phrases have a very high error rate, because they are missing a lot of characters. It was decided to filter out these phrases, to get a more realistic overview of the error rates. It can be difficult to judge whether a phrase was aborted, without inspecting each phrase manually. To avoid it, the relative phrase length was used as a filter. The relative phrase length is defined as the length of transcribed text divided by the length of the presented text as shown in 4.13. Figure 4.9 shows how many phrases that were completed, dependent on the relative length. There is a very steep step around 1, which indicates that the most transcribed phrases had the same length at the presented phrases. Most of the phrases have a relative length where $0.9 < l_{\text{rel}} < 1.1$. This is caused by small mistakes where a few extra characters are written or omitted from the transcribed text.

$$l_{\text{rel}} = \frac{|T|}{|P|} \tag{4.13}$$

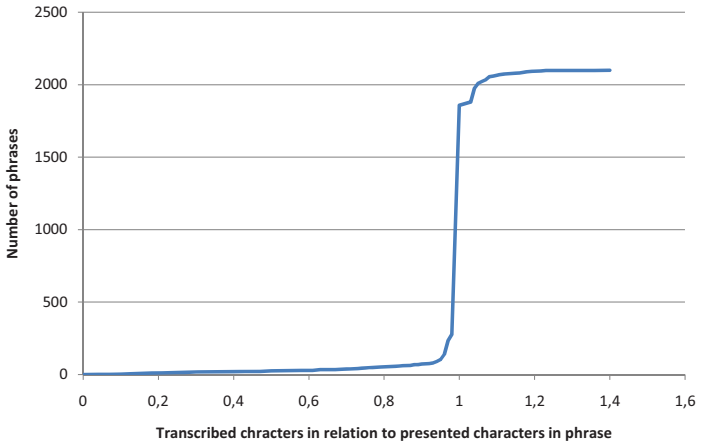


Figure 4.9: Accumulated number of written phrases in relation to the completeness of the phrases. 0.9 means that the length of the transcribed text was at least 90% of the length of the presented text.

It was decided to filter all data where $l_{\text{rel}} < 0.8$. This filtering removed 53 phrases from the data set. Figure 4.10 shows the average text entry speed for all participants and all sessions. Most participants performed 3-6 WPM in the first session. In the participants last session the majority performed between 6 and 10 WPM. The fastest personal average for a session was 10.9 WPM and the fastest phrase was written at 13.9 WPM. For the rest of the analysis, the data from the pilot and slip test (UID: 1, 81, 82) have been excluded from the data set.

Figure 4.11 shows the average text entry speed and error rates for each session. The average text entry speed starts with 4.8 WPM and ends with 8.2 WPM. The error rate is highest for the first session. In the following sessions the corrected error rate slowly decreases from 0.08 to 0.06. The uncorrected error rate decreases from 0.03 to 0.01 - 0.02. There is a clear effect of learning as the participants gets more experienced. To test whether the text entry speed also increases in each session, the increase in WPM between all subsequent phrases for each session is calculated. Figure 4.12 shows the average increase in text entry speed for each phrase over the first phrase in each session. The figure also shows the total number of completed phrases in each session. It can be seen that it takes 2-3 phrases to recall the text entry method. The performance for the rest of the session is more stable. The large variance from phrase 20 and beyond is due to the fact the very few sessions consisted of 20 or more phrases.

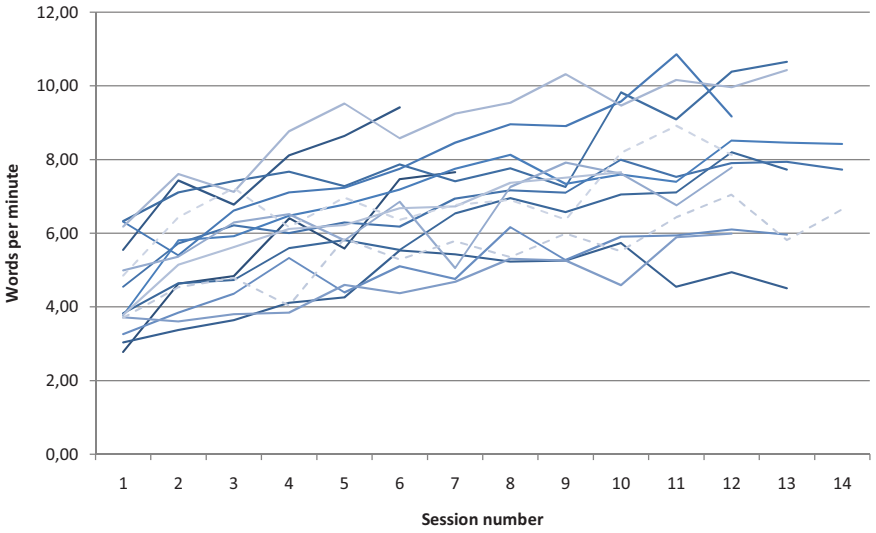


Figure 4.10: Text entry speed for each participants. The dotted lines represents the participants from the slip error evaluation without the slip error correction algorithm.

4.3.3.3 Learning and gestures

From the previous figures it can be seen that the participants learned to use TUP. This resulted in faster text entry and fewer errors. The learning rate is found to be:

$$\mathrm{WPM}_n = 4.13n^{0.2415} \tag{4.14}$$

The details behind the estimate of the learning rate are shown in appendix C.

To investigate how learning improves performance the participants' gestures are analysed. First the time to write each character is divided into three groups:

Finger off wheel is the time where the participants did not touched the wheel.

Finger on wheel is the time the participants touched the wheel until the target character is highlighted.

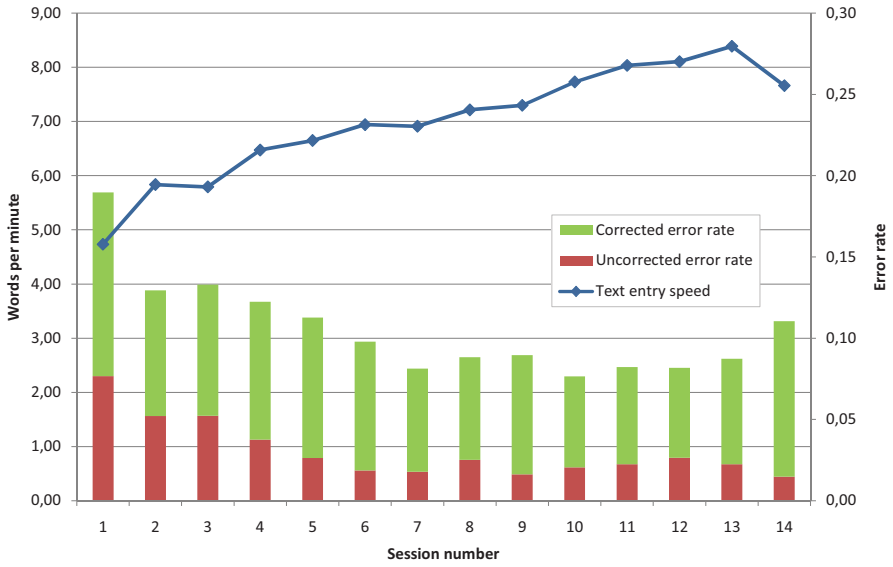


Figure 4.11: Average input speed and error rates. Most participants performed 12-14 sessions, so the validity of the data is largest for the first 12 sessions.

Finger on character is the time the participants touched the wheel at the target character. If the target character was highlighted multiple times during a gesture, only the final touch will be included in this group.

Figure 4.13 shows the average distribution of the time used to input characters for each session. The **Finger off wheel** time drops from 1500 to 750 milliseconds. This could be due to the users feeling more comfortable with the TUP method. Another explanation could be that people learned the placement of the characters on the circle. A visual scan for the target character will take up a lot of time. If the participants had learned the placement of the characters, then the visual scan could be avoided. **Finger on wheel** drops from 350 to 200 milliseconds. This is mainly due to the participants getting better to hit the target character faster. Either by placing the finger near the character or by being able to scroll to the target character fast. **Finger on character** drops from 600 to about 500 milliseconds.

Strategies for highlighting characters The participants used different strategies to highlight the target characters. Based on the observations at the

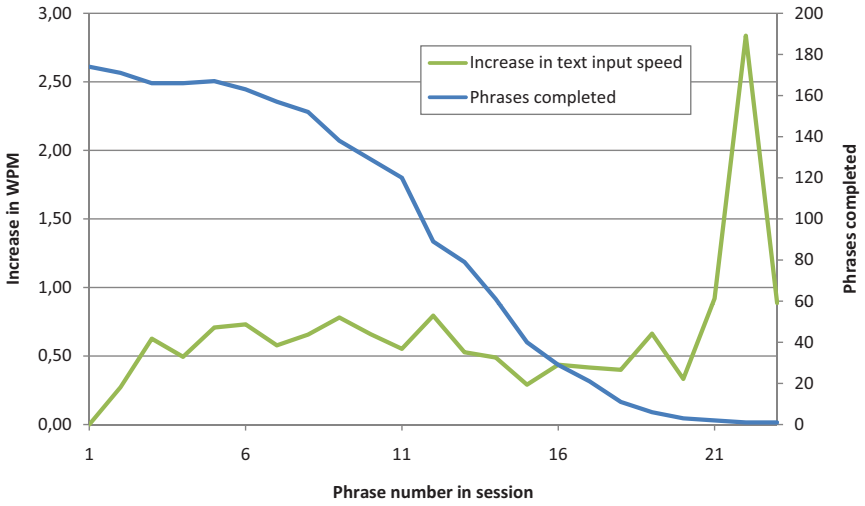


Figure 4.12: Average increase in text entry speed during each session. The plot shows the average increase from the first phrase in each session. The number of completed phrases is also shown.

briefing and debriefing and on the interviews, the following three strategies were identified:

Tap direct: The finger is placed on the position of the target character. Small adjustments are made if the target character is not highlighted.

Scroll fixed: The finger is placed on the same position each time. The finger scrolls toward the target character.

Scroll latest: The finger is placed on the position of the last written character. The finger scrolls toward the target character.

Some participants mentioned that the changed strategy during the evaluation. The participants' strategy and precision can be revealed by the log files. Figure 4.14 shows a 'strategy plot' for session 9 by participant 11. Each strategy has its own indicator function. The indicator function with the highest peak tells which strategy the participant mainly used in the session. The indicator functions are described here:

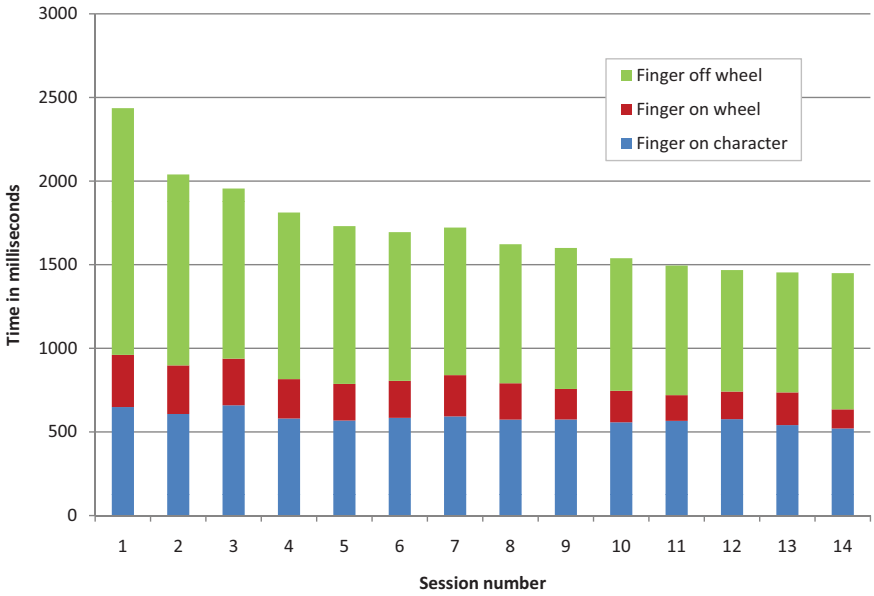


Figure 4.13: Average distribution of time used to input each character. **Finger off wheel** is the time where the participants did not touch the wheel. **Finger on wheel** is the time the participants touched the wheel until the target character was highlighted. **Finger on character** is the time the target character was highlighted before the participants lifted their fingers off the wheel.

Tap direct: For each written character, the difference between the position of the initial touch and the target character is calculated. The indicator function is the distribution of these differences. A peak around 0 means that the user placed the finger close to the character that was produced by the gesture.

Scroll fixed: The indicator function is the distribution of the initial touches for each character. A large peak value of this function will mean that the user have placed the finger the same place on the wheel many times. If the participant used the tapping strategy, the distribution of touches will be equal to the distribution of characters. This explains the small peaks of this indicator function.

Scroll latest: For each written character, the difference between the position of the initial touch and the previous target character is calculated. The indicator function is the distribution of these differences. A peak around 0 means that the user placed the finger close to the position of the previously entered character.

Figure 4.15 shows strategy plots for all sessions by participant 11. It can be seen that the participant changes strategy from **scroll latest** to **tap direct** as he get more experienced. Table 4.7 shows the text entry speed and strategy. It can be seen that the tapping strategy is faster than scrolling.

Session	1	2	3	4	5	6	7	8	9	10	11	12
Strategy	S	S	S	T	S	T	S	T	T	T	T	T
WPM	5.0	5.4	6.3	6.5	5.8	6.9	5.1	7.3	7.9	7.6	6.8	7.8

Table 4.7: Character highlighting strategy and text entry speed for participant 11. T means tapping and S is scroll latest. The text entry speed is reported in WPM.

The relation between the peak values for the different strategies can be used to show how the user changes strategy between sessions. Figure A.1 in the appendix shows the strategies for each user in each session.

Precision Most of the participants mentioned that they learned the positions of the characters, and could highlight them more easily.

The average distance between the first touch and the target character reveals the participants ability to hit the correct position on the wheel. Figure 4.16 shows the average distance for each session.

It can be seen that the data supports the users' experience.

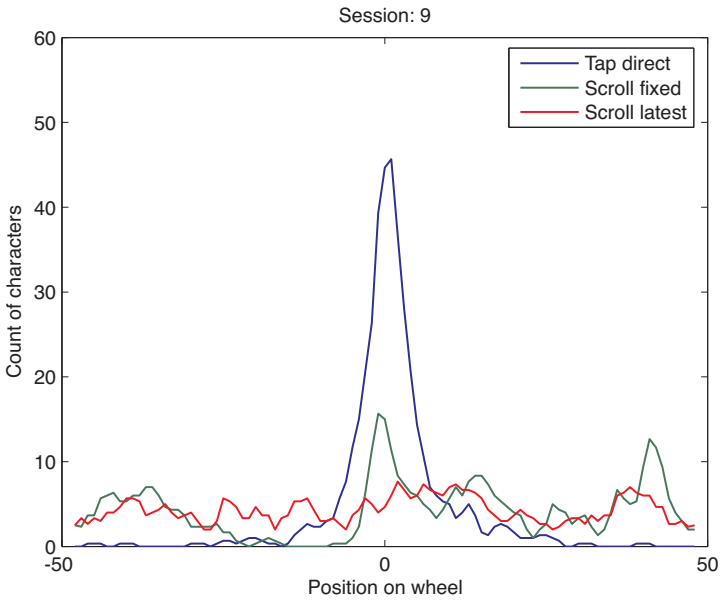


Figure 4.14: Strategy plot that shows the character highlighting strategy for participant 11 in session 9. The strategy with the highest peak is the strategy used by the participant.

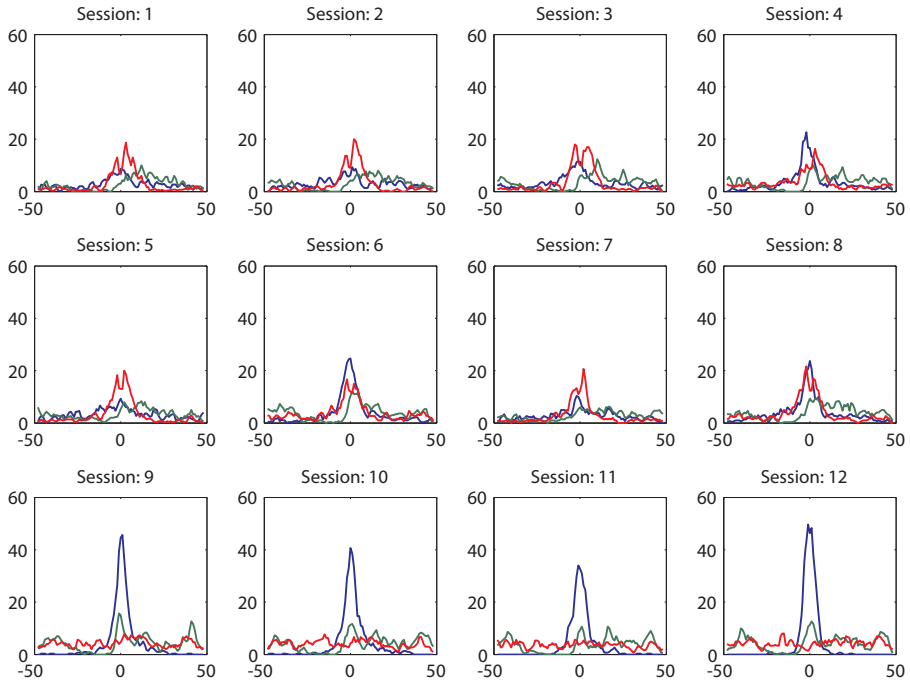


Figure 4.15: Strategy plot that shows the character highlighting strategy for participant 11. The strategy with the highest peak is the strategy used by the participant.

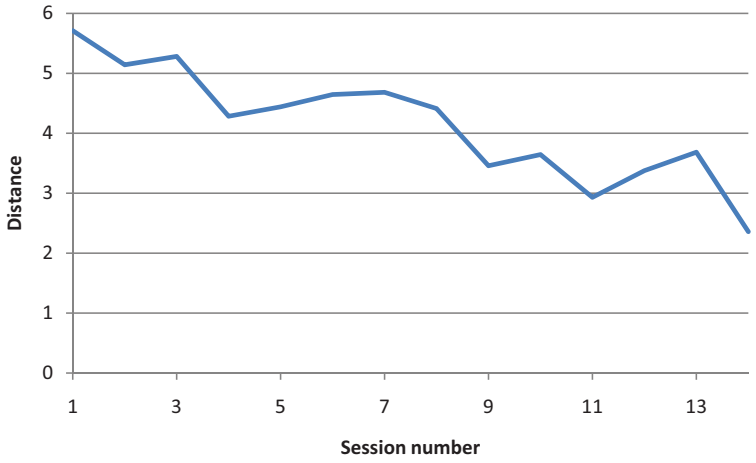


Figure 4.16: The average distance between the first touch and the target character.

Preparation time Most of the participants used the Tap strategy. When using this strategy, the participants need to find the target character before the finger can be placed on the wheel. The time elapsed between a character is entered and the finger is back on the wheel will be called *preparation time*. An analysis of the preparation time shows that it is mainly dependent on the target character. Figure 4.17 shows the average preparation time for each character. The sessions are grouped together 3 by 3 to make it easier to read the figure. It can be seen that the participants improved between the sessions. The observations in each session seem to follow two trends:

- Frequently used characters are found faster. Both the participants' native languages and the evaluation language are expected to have influence on the preparation time. Most of the participants were Danish. W is not very common in Danish, and has a high preparation time during the first sessions. As the participants are getting more experienced with TUP, they learn the position of frequently used characters.
- Characters from the beginning of the alphabet are found faster than characters from the back. This could be because the participants are

doing a visual scan when they are looking for characters. Or it could be because people can remember the characters in the beginning of the alphabet better than the ones at the end.

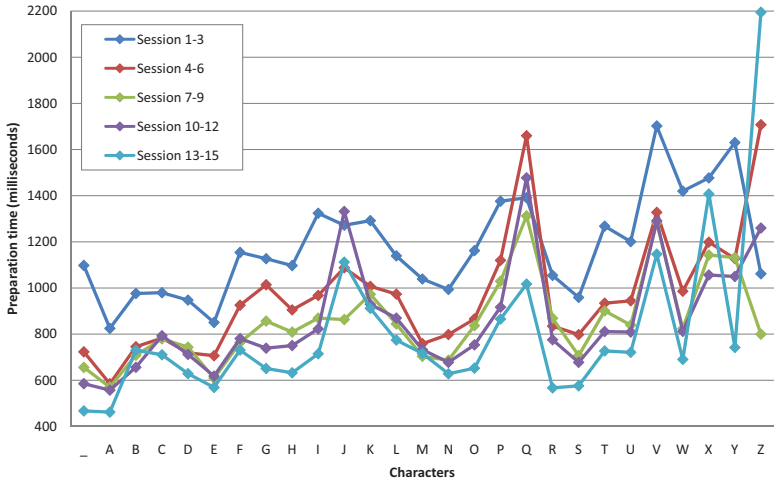


Figure 4.17: Preparation time from a character is entered to the finger is back on the wheel.

4.3.3.4 Effect of language model

The language model is an important factor in the character highlighting algorithm. It speeds up the use of TUP, by making it easier to highlight the most likely characters. To investigate the value of the language model, the relation between probabilities of characters and the input speed is found. All written characters are sorted by probability ($\Theta(\alpha_i)$ from equation 4.5), and divided into 100 bins. For each bin the count of written characters and the average duration of the gesture is found. The gesture time is divided into touch on wheel and touch on target character. A plot of the data can be found in figure 4.18. The duration of gestures are weighted by the count of characters in the bin and neighbour bins. E.g. the time for characters with $P(\alpha) = 50\%$ is calculated as the average for characters with $P(\alpha) = [49\%, 50\%, 51\%]$ weighted by the

count of characters in each bin. This is done because some bins with very few characters had high variance.

The top plot shows the distribution of all written characters, based on their probabilities. There are a few peaks in the plot at 65% and 68%. They are both due to the word *the*. $P(space|he) = 65.2\%$ and $P(e|th) = 68.2\%$.

The time used to write the characters are divided into two parts like in figure 4.13. The time interval where the finger is off the wheel is not included in this plot. It is because the time interval can not be associated with a single character probability. The time interval could be associated with the previous or the next written character, but I have chosen not to do this. It can be seen that the time for **Finger on character** is between 500 and 600 milliseconds for most characters, like in figure 4.13. The time for **Finger on wheel** is largest for characters with low probability. This corresponds very well to the design philosophy of TUP, where the most likely characters should be easiest to highlight. The time for **Finger on wheel** drops to less than 50 milliseconds for the most likely characters. If the target character have high probability, then it is very likely that it will be highlighted as soon as the user touches the wheel.

4.3.3.5 Further statistical analysis

The analysis of how the text entry speed is dependent on the user previously experience and the place of the evaluation is done in section 4.5.3.5. The observations from this evaluation are combined with observations from the improved TUP iPod implementation. This is done to get more significant results.

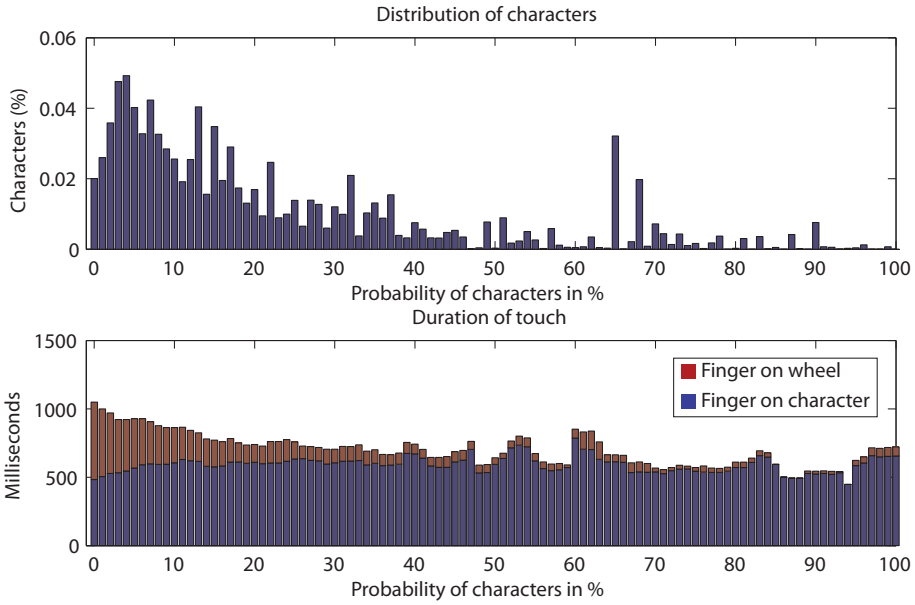


Figure 4.18: Written characters ordered by their probability. All written characters have been sorted by probability and put into 100 bins. The top plot shows the distribution of characters between the bins. The plot in the bottom shows the average duration of gestures.

4.4 Using statistical learning to improve text entry

This section will give to examples of how statistical learning can be used to optimize text entry methods. Both the character highlighting algorithm and the slip error correction algorithm in TUP are based on low level models of human motor behaviour. The algorithms can be improved by updating the low level models. During the evaluation of the TUP iPod prototype (section 4.3) a large number of interaction logs were generated. They contain information about presented phrases, transcribed phrases, errors and the participants' gestures on the touch wheel. The interaction logs will be used to optimize the low level models of human motor behaviour, and thereby improve the character highlighting algorithm and the slip error correction algorithm in TUP. The improved algorithms have been implemented in the TUP improved iPod prototype and evaluated in section 4.5.

In the definition of TUP and its algorithms, the positions on the wheel were defined as $\varphi_i \in [0, 2\pi[$. In this section the positions will be defined as $\varphi_i \in [0, 95[$. 0 is still defined as the top of the wheel and the values are increasing clockwise. The redefinition of φ is done because the data in this section relies on the interaction logs from the iPod prototype. The new values of φ are similar to the output from the iPod wheel.

4.4.1 Optimization of character highlighting algorithm

In the TUP text entry method, the character highlighting algorithm is responsible for selecting which character to highlight. A central part of the algorithm is the function $P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)$. This is the probability of the user's finger being on φ_i, if the user will highlight α_i and have made the gesture $\boldsymbol{\varphi}_{1..i-1}$. In the original algorithm $P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)$ was estimated as a Gaussian distribution with $\mu = \varphi_{\alpha_i}$. The variance σ^2 is a function of the speed of the user's finger on the touch device as described in equation (4.9) and (4.11). Since the distribution is the same for all characters, it can be expressed as the probability of a distance between the current touch and the highlighted character:

$$P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i) = P(\varphi_{\alpha_i} - \varphi_i|\boldsymbol{\varphi}_{1..i-1}) \tag{4.15}$$

The best character to highlight is the character the user wants to write. Observations from the TUP iPod evaluation are used to find the quality of

the current character highlighting algorithm.

Figure 4.19 shows plots of $P(\varphi_\alpha - \varphi|\varphi')$ for three different values of φ'. φ is the current position of the user's touch. α is the target character that is written by the gesture. φ' is the difference between the current and previous touch of the finger. It is defined as:

$$\varphi' = \varphi_i - \varphi_{i-1} \tag{4.16}$$

The plots show the distribution of the distances between the target characters and the position of the users' fingers. The different plots show the distributions for different velocities of the users' fingers. If the finger is moving clockwise ($\varphi' > 0$) then the target character will most likely be clockwise to the position of the user's finger.

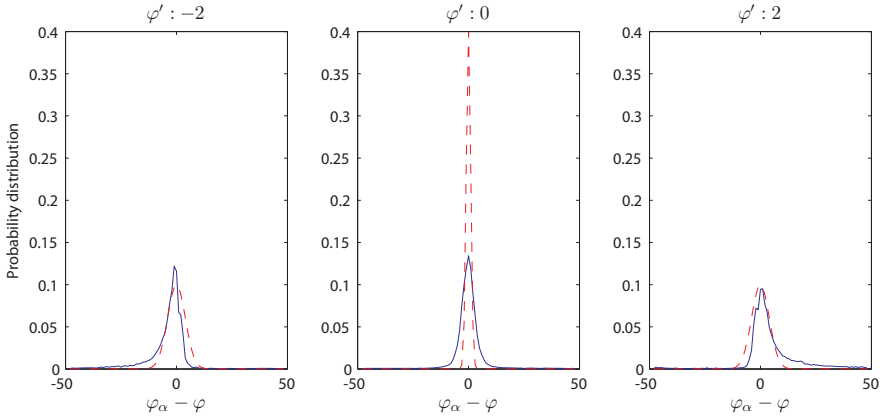


Figure 4.19: Distribution of the distances between the target characters and the position of the users' fingers for different velocities of the fingers. The blue line is the observation data. The red line is the Gaussian distribution used in the original character highlighting algorithm.

It can be seen that the measured distribution is asymmetric. The original Gaussian distribution changed variance dependent on the velocity, but it was always symmetric. All three plots show that the shape of the distribution is very different from the Gaussian distribution. Lilliefors test for normality rejects the hypothesis that any of the three measured distributions is normal distributions.

Figure 4.20 shows the measured distribution of $P(\varphi_\alpha - \varphi|\varphi')$ for all values of φ'. The data have been scaled, so the maximum value in each row is 1.0. It is very clear to see that the distributions not are symmetric around $\varphi_\alpha - \varphi = 0$.

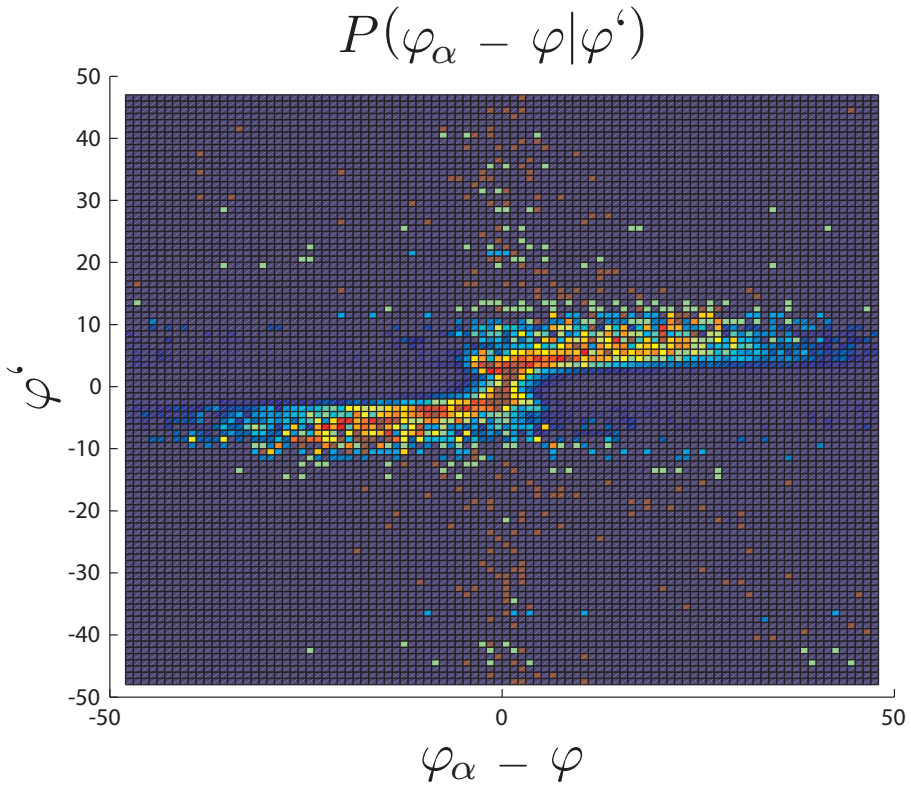


Figure 4.20: Plot of $P(\varphi_\alpha - \varphi|\varphi')$. The data have been scaled so the maximum value in each row is 1.0. Dark blue equals 0 and dark red equals 1.

4.4.1.1 Defining algorithm inputs

The plots in figure 4.19 and 4.20 are both using φ' as one of the input dimensions. φ' is the difference between the previous and current position of the finger. There might be other features that could be useful for defining the new algorithm. The following three features will be evaluated to find out which one that best describe the data.

Movement of finger: φ' . The difference between the previous and current position of the finger as defined in equation 4.16.

Speed: $\frac{\varphi'}{t'}$ Can be found by dividing φ' with the time differences t' between the measurements.

Variation: σ^2 Calculated as in equation 4.9 with an direction factor. The original variance were a factor between 0.01 and 1. Because of the asymmetry of $P(\varphi|\alpha, \varphi')$, the variance needs to be multiplied with a direction factor $\frac{\varphi'}{|\varphi'|}$

Thanks to the log files from the evaluation, it easy to calculate the features for all the data points. Column 22 to 24 in the evaluation dataset contains the extra features, in the same order as above.

All the features will change very fast when the speed of the finger changes. As described in section 4.1.3.3, this makes them unsuitable to be used directly in the character highlighting algorithm. Instead the algorithm should use a weighted running average of the features as described in equation 4.11. The features have been calculated with five different values of $\beta = 0.1, 0.3, 0.5, 0.7, 1.0$. $\beta = 0.1$ will give least weight to the current measurement, while $\beta = 1.0$ only will take the current measurement into account when calculating the input variable.

4.4.1.2 Pre-processing of input

The distribution of values for each feature can be seen in figure 4.21. For all three features the majority of the observations are close to 0. To avoid uncertain predictions for outliers, the 0.5 % outer most observations on each side will be ignored. When the probability of new observations needs to be estimated, observations beyond the limits will be replaced by the limit. The limits for all three features for $\beta = 0.5$ is shown i table 4.8.

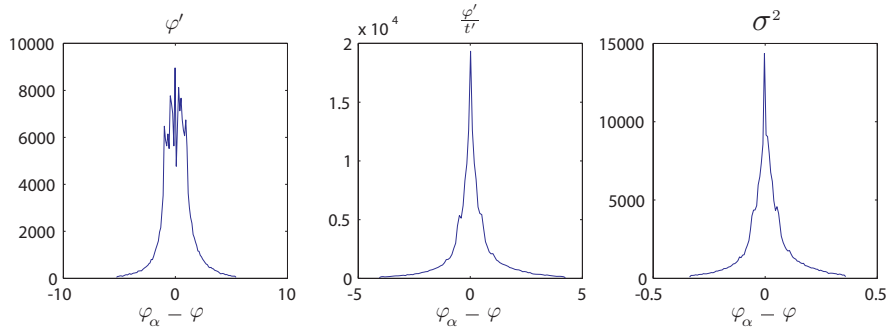


Figure 4.21: Distribution of observation values for three different features for $\beta = 0.5$.

	φ'	$\frac{\varphi'}{t'}$	σ^2
min	-5.25	-4.67	-0.43
max	5.42	4.91	0.45

Table 4.8: Limits for all three features for $\beta = 0.5$.

For the remaining 99% percent of the observations, plots of $P(\varphi_\alpha - \varphi, \textit{feature})$ are calculated. The plots are similar to the one shown in figure 4.20. The plots can be seen in in appendix A.2.1 in figure A.3. All the plots have the same characteristic *s*-shape. From the figure it can be seen that plots with $\beta \leq 0.5$ are smoother than plots with a high β value. This corresponds fine with the theory. A low β value will mean that the data points are averaged over more observations.

4.4.1.3 Fitting a function to the observations

The observations are fitted to a circular sigmoid product function. More details on the function and fitting algorithm can be found in appendix B.

4.4.1.4 Finding the best feature and β value

With the fitting algorithm and error estimate from appendix B it is possible to compare the different features and β values. Besides the five original β values, five extra values are tested. The results can be seen in table 4.9.

errors $\cdot 10^3$	φ'	$\frac{\varphi'}{t'}$	σ^2
$\beta = 0.01$	3.281	3.616	3.731
$\beta = 0.05$	2.887	3.146	3.098
$\beta = 0.10$	2.973	2.943	2.883
$\beta = 0.15$	3.059	2.870	2.874
$\beta = 0.20$	3.366	3.015	3.013
$\beta = 0.25$	3.492	2.968	3.043
$\beta = 0.30$	3.623	3.025	2.995
$\beta = 0.50$	4.321	3.235	3.200
$\beta = 0.70$	4.942	3.609	3.585
$\beta = 1.00$	3.634	4.017	3.577

Table 4.9: Errors for different features and values of β. All the errors are multiplied by 10^3 to make them easier to read.

Based on the results it is decided to use $\frac{\varphi'}{t'}$ and $\beta = 0.15$. The input will be limited if it is below -2.826 or above 3.126. The term $\Psi(x|y)$ is used to describe the distribution. It is calculated as the circular sigmoid product function (equation B.3) with the parameters from equation 4.17. x is the distance $\varphi_\alpha - \varphi$ and y is the velocity $\frac{\varphi'}{t'}$. The observed data and the fitted $\Psi(\varphi_\alpha - \varphi|\frac{\varphi'}{t'})$ function can be seen in figure 4.22.

$$[a_1, c_1, a_2, c_2] = [1, \frac{\varphi'}{t'}, (\frac{\varphi'}{t'})^2, (\frac{\varphi'}{t'})^3] \cdot \begin{bmatrix} -0.00620 & -3.0258 & 0.0872 & 2.7969 \\ 0.13595 & 1.8559 & -0.2433 & 2.2619 \\ -0.10389 & -0.7251 & -0.1132 & 1.1676 \\ 0.00568 & 0.1143 & 0.0201 & 0.0072 \end{bmatrix} \tag{4.17}$$

$\Psi(x|y)$ is replacing the Gaussian distribution in the character highlighting algorithm, as an estimate of $P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)$:

$$P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i) = P(\varphi_\alpha - \varphi|\frac{\varphi'}{t'}) = \Psi(\varphi_\alpha - \varphi|\frac{\varphi'}{t'}) \cdot \gamma_{\frac{\varphi'}{t'}} \tag{4.18}$$

$\gamma_{\frac{\varphi'}{t'}}$ is a term that is constant for each value of $\frac{\varphi'}{t'}$:

$$\gamma_{\frac{\varphi'}{t'}} = \frac{1}{\sum_{\varphi_\alpha - \varphi} \Psi(\varphi_\alpha - \varphi|\frac{\varphi'}{t'})} \tag{4.19}$$

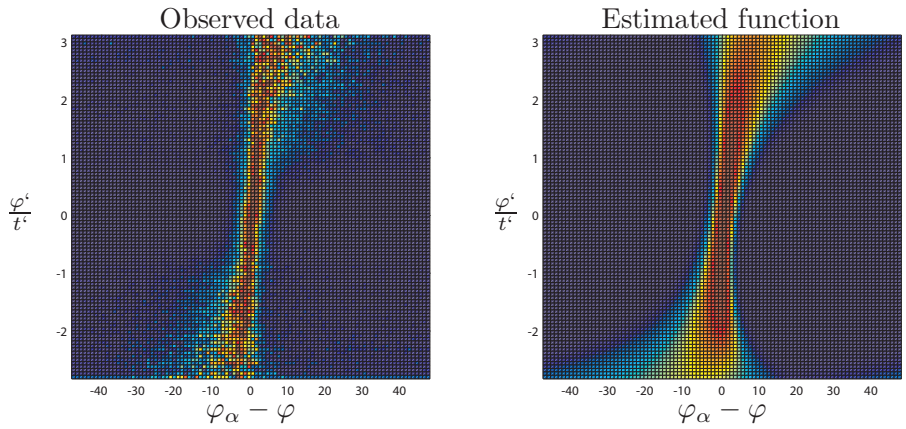


Figure 4.22: The plot to the left shows the observed data for feature $\frac{\varphi'}{t'}$ and $\beta = 0.15$. The plot to the right shows the fitted sigmoid product function.

It is used because the optimized distribution is based on the circular sigmoid product function. It is not a probability function, so the sum of probabilities will not equal 1 with out this constant.

The optimized character highlighting algorithm can be found by combining equation 4.1, 4.4, 4.6, 4.18 and removing constant terms:

$$\underset{\alpha_i}{\operatorname{argmax}} \left(\Theta(\alpha_i) + 0.1\right)^{\varepsilon_{\text{predict}}} \Psi(\varphi_\alpha - \varphi | \frac{\varphi'}{t'}) \tag{4.20}$$

4.4.1.5 Implementation of function

The optimized character highlighting algorithm requires significantly more floating points computations than the old Gaussian based algorithm. To avoid a drop in performance the algorithm is implemented as a lookup table. The values are calculated for 101 values of $\frac{\varphi'}{t'}$ and all 96 values of $\varphi_\alpha - \varphi$.

There is one problem with the new improved character highlighting algorithm. The initial value of $\frac{\varphi'}{t'}$ is 0, which makes the distribution very narrow. The initial distribution should be wide, to enable easy highlighting of likely characters. Figure 4.23 shows the difference between the Gaussian distribution and the optimized distribution from the new character highlighting algorithm. For the Gaussian distribution $\sigma^2 = \lambda_0 = 1/4.15$, as explained in section 4.1.3.3. The entropy 4.15 is found from the character probabilities for the the first character

to be written in a phrase, according to the language model used in the TUP iPod implementation.

The narrow distribution makes it very difficult to hit the correct character in the initial touch. Due to the skewness of the optimized distribution, it is not possible to select a larger initial value of $\frac{\varphi'}{t'}$. The solution is to increase the λ_0 value in the optimized Character Highlighting algorithm. In the optimized character highlighting algorithm, λ is still used to control the influence of the language model because $\varepsilon_{\text{predict}}$ is set to λ. By increasing λ_0, the language model will have more impact on the overall algorithm during the initial touch. When the user scrolls on the wheel $\frac{\varphi'}{t'}$ will increase and λ will decrease to the value described by equation 4.9 and 4.11. By informal evaluations it was found that an increase of λ_0 with a factor 3 gave good initial performance.

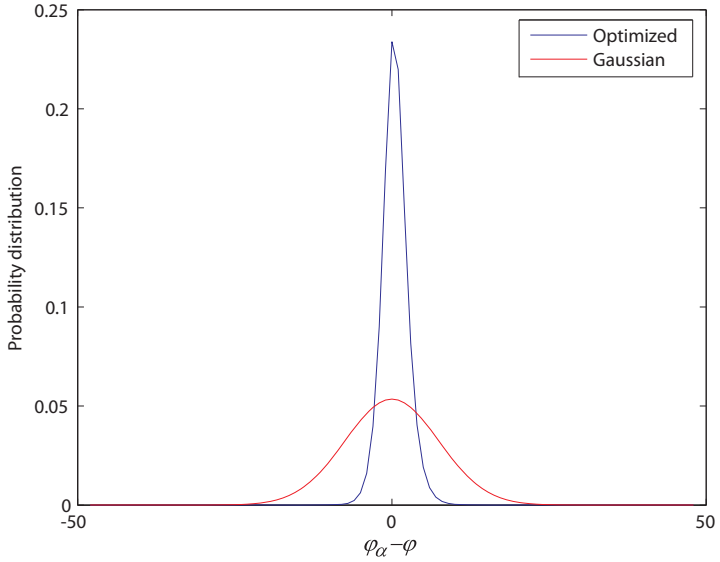


Figure 4.23: The initial distribution of $p(\varphi_\alpha - \varphi|\alpha)$ when $\varphi' = 0$ and $\lambda = 1/4.15$. Both the Gaussian distribution from the first iPod prototype and the optimized distribution are shown.

4.4.2 Optimized slip error correction algorithm

The slip error correction algorithm used in TUP (section 4.1.4) is used to correct some mistakes automatically. It corrects errors where the highlighted character changes while the finger is released from the wheel. The algorithm

uses two parameters: the time the currently highlighted character have been highlighted and the distance between that character and the previously highlighted character. If the highlight time is less than 100 milliseconds and the characters are adjacent to each other, then the previously highlighted character will be written instead of the currently highlighted character. The algorithm and values are based on informal studies and evaluations.

It is expected that the algorithm can be improved, by replacing it with a new classifier. The classifier will be trained on data from the TUP iPod evaluation. It is likely that the original algorithm have affected the participants' interaction. To avoid any possible influence by the original algorithm, only data from participant UID: 81 and 82 are used. These participants used a special prototype, where the slip error correction algorithm was disabled.

Figure 4.24 shows the highlight time and distance for all written characters from the usability evaluation of user 81 and 82. All correct entries are blue and incorrect entries are red. The dashed box marks the parameters used in the fixed error correction algorithm. Table 4.10 lists the number of entries and misclassifications if the original algorithm was used. From a usability point of view, correct characters that are classified as incorrect are a lot worse than incorrect characters that are classified as correct. It is very frustrating for the user, if the systems change correct characters. Therefore the error function (4.21) is designed so misclassifications of correct entries are weighted higher than misclassifications of incorrect characters. The weight is set to 5, meaning misclassification of correct entries will weight five times as much as misclassification of incorrect errors. The value 5 is arbitrarily chosen.

$$\text{Error} = \frac{\text{Misclassification}_{\text{Correct}} \cdot 5 + \text{Misclassification}_{\text{Incorrect}}}{\text{Entries}} \tag{4.21}$$

It should be noted that not all incorrect errors are due to slips while releasing the finger. It can also be typos or other errors.

User	Entries	Correct	Incorrect	Misclassification		Error
				of correct	of incorrect	rate
81	1897	1574	323	17	131	0.114
82	1407	1110	297	18	90	0.128
Total	3304	2684	620	35	221	0.112

Table 4.10: Number of entries for user 81 and 82.

C	Name	Description
13	Highlight time	Time current character has been highlighted
15	Prev. highlight time	Time previous character has been highlighted
16	Highlight difference	Distance in characters between the current and previous highlighted character
17	Highlighted characters	Count of highlighted characters in this gesture
18	Duration of gesture	Duration of the gesture

Table 4.11: List of features that will be used to make an improved slip error correction algorithm. C is the column number. All time measurements will be in milliseconds.

4.4.2.1 Pre-processing of data

Statistical learning can be used to create a better error correction algorithm. It is a classification problem, where the function estimate should classify the gesture as either correct or incorrect. Each gesture consists of a number of touches on the wheel. It is possible to use functional data analysis on the entire gesture or to use feature extraction to convert each gesture to a multidimensional data point. It is chosen to use feature extraction because it requires less processing resources, and therefore is more suitable for mobile devices. The features that are used are listed in table 4.11.

Only rows where the finger is just released from the wheel are used. This means that column 10 should equal 2. This reduces the data set from $35,950$ to 9117 rows. The algorithm cannot be used if only one character have been highlighted during the gesture. Therefore all rows where column 17 equals 1 is removed from the data set. This reduces the number of rows from 9117 to 3304.

4.4.2.2 K-nearest neighbours classifier

To make a better slip error correction algorithm than the one outlined previously, the K-nearest neighbours method is used as a classifier. The algorithm should estimate the values of column 11 based on column 13, 15, 16, 17 and 18. Column 11 equals 1 for correct characters and 0 for incorrect characters. The character probabilities are not used, because they are used in other parts of the character highlighting algorithm. To find out which columns to include in the classifier, the best subset method is used to test all combinations. The classifiers are validated by using cross validation with 20 groups. The error for all possible subsets are found for k=1..50. Figure 4.25 shows the best subset for each value

of k. Table 4.13 in the appendix shows the error for all possible subsets for k=5, 15 and 40.

The lowest error rate is 0.0742 which is found when using $k = 3$ and columns 13 and 15. This is the highlight time and the previous highlight time. The mean error rates for all input dimensions are listed in table 4.12 for $k = 5$. It can be seen that the highlight time is the input dimension that give the lowest error.

	13	15	16	17	18
$k = 5$	0.0867	0.1671	0.1635	0.1735	0.1494
$k = 15$	0.0896	0.1523	0.1492	0.1536	0.1313
$k = 40$	0.0916	0.1402	0.1411	0.1421	0.1263

Table 4.12: Mean error rates for subsets with different input dimensions and values of k.

4.4.2.3 K-nearest neighbours with loss matrix

The problem with the solution from the previous section is that even though the error is lower than the original algorithm, then the number of misclassifications of correct entries is very high. The problem is that the weighting between misclassifications of correct and incorrect characters, only are used in the error function. By extending the K-nearest neighbours method with a loss matrix, the weighting will also be used in the classification of new data points. By using this method, it is possible to specify a loss matrix, which defines the cost of misclassifications. In this case the cost of misclassification is set to 1 for incorrect entries and 5 for correct entries. The 1:5 ratio is chosen to match the error calculation algorithm in (4.21). For K=10, this means that 2 neighbours classified as correct will have more influence than 8 neighbours classified as incorrect.

Figure 4.26 shows the results of the K-nearest method with loss matrix. Again the best subset is chosen for each value of K. By comparing the figure to figure 4.25 it can be seen that the error is approximately the same, but the number of misclassifications of correct entries are much lower with the K-nearest with loss matrix. The lowest error rate is 0.0681 which is found when using $k = 2$ and columns 13 and 15. This is the highlight time and the previous highlight time. It is the same columns that in the normal K-nearest method.

The optimal K-nearest neighbours solution to the problem, is to use K-nearest neighbours with loss matrix, $k = 2$ and only use the highlight time and previous highlight time. With this method, the estimated error rate will be 0.0681.

There will be 24 misclassifications of correct entries and 105 misclassifications of incorrect entries. This is a significantly lower than the original algorithm. $k = 2$ is a very low value. A larger value of k will make the algorithm more stable. Figure 4.27 shows the plot of the best subset for $k = 2$, $k = 10$ and $k = 40$.

The classifier for the improved slip error correction algorithm consists of 3304 data points. Each time the participants remove their fingers from the touch wheel, all 3304 points will need to be compared with the new data point. Even if there are only two dimensions, it will take a long time to compute. It is very crucial that the slip error correction algorithm is fast, so the current solution is not feasible to implement. Even though the K-nearest classifier can not be implemented it provides a good base line for further classifiers.

4.4.2.4 Linear Classification

Another classification method is to use linear regression to classify the data. The observations from the evaluation are organized in a n by p matrix X, where n is the number of observations and p is the number of features. The same features as above (table 4.11) are used. A constant, 1, is added to each observation. The classification of each observation is organized in a n by g matrix G, where g is the number of classes. The classes are binary encoded. $X_{i,1} = 1$ if observation i is classified as *Correct*. Otherwise it is 0. The same way $X_{i,2} = 1$ if observation i is classified as *Incorrect*.

The classifier is found by the normal linear regression formula:

$$\hat{\beta} = (X^T X)^{-1} X^T G \tag{4.22}$$

A new observation, x, is classified by selecting the class with the highest expected value:

$$\underset{i}{\operatorname{argmax}}\, g_i, g = x\hat{\beta} \tag{4.23}$$

As with the K-nearest classifier it is possible to introduce a loss matrix to favour the *Correct* class. The loss matrix is shown in equation 4.24. p is the loss if an *Correct* observation is classified as *Incorrect*. $(1 - p)$ is the loss if an *Incorrect* observation is classified as *Correct*.

$$L = \begin{bmatrix} 0 & p \\ 1-p & 0 \end{bmatrix} \tag{4.24}$$

When using the loss matrix new observations x are classified by selecting the class that will minimize the expected loss:

$$\underset{i}{\operatorname{argmin}}\, l_i, l = x\hat{\beta}L \tag{4.25}$$

A cross validation with 20 groups is used to test the linear regression classifier. Figure A.4 in appendix A.2.2 shows the number of misclassifications and the error rate from equation 4.21. The minimum error is at $p = 0.62$, but the performance of the linear regression classifier is worse than the K-nearest neighbours classifier.

4.4.2.5 Discriminant Analysis

Another solution is to use discriminant analysis to classify the observations [Hastie et al., 2001].

Linear, logarithmic and quadratic discriminant analyses are used on the observations. All are cross validated with the same 20 groups as before. Figure A.5, A.6 and A.7 in appendix A.2.2 show the number of misclassifications and the error rates. The best classification is made by the logarithmic discriminant analysis with $p = 0.65$. It is still not as good as the K-nearest neighbour classifier, but since it is not feasible to implement K-nearest neighbour the logarithmic discriminant analysis will be used in the slip error correction algorithm.

To simplify the slip error correction algorithm the logarithmic discriminant analysis is evaluated with all possible subsets of the input features. The number of misclassifactions and error rates can be found in table 4.14. The lowest error rate, 0.0956, is found when using the highlight time and the previous highlight time. This is the same feature subset that was best for the K-nearest neighbours classifiers.

The found solution does not classify observations better than the K-nearest neighbours classifier, but it requires very few calculations to classify new observations. That makes it more feasible for implementation in mobile devices.

The final classifier is:

$$\alpha = \{ \begin{array}{lll} \text{correct} & if & \kappa \geq 0 \\ \text{incorrect} & if & \kappa < 0 \end{array}$$

$$\kappa = -4.5608 + \textit{highlight time} \cdot 0.0129 + \textit{previous highlight time} \cdot 0.0016 \tag{4.26}$$

The time should be in milliseconds.

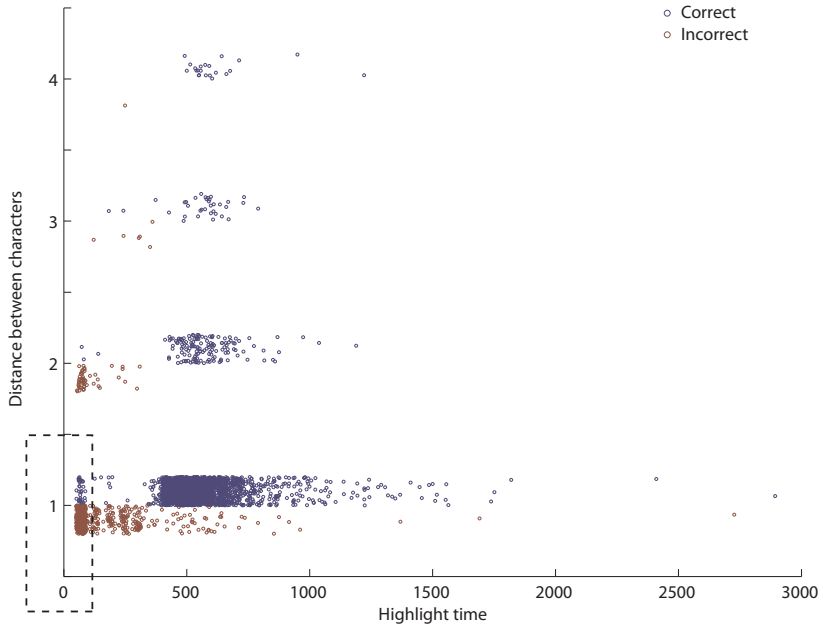


Figure 4.24: Plot of correct and incorrect characters. Highlight time is the time the character was highlighted before it was entered. The distance is the distance between the character and the previously highlighted character. The distance is measured in characters. E. g. the distance between *h* and *i* is 1. The distances are all integer values. To be able to distinguish between the characters a random value between 0 and 0.2 have been added to all correct characters, and a random value between 0 and -0.2 have been added to all incorrect characters. The dotted square indicates the characters that will be corrected by the original slip error correction algorithm. Gestures where only one character was highlighted are excluded form the plot.

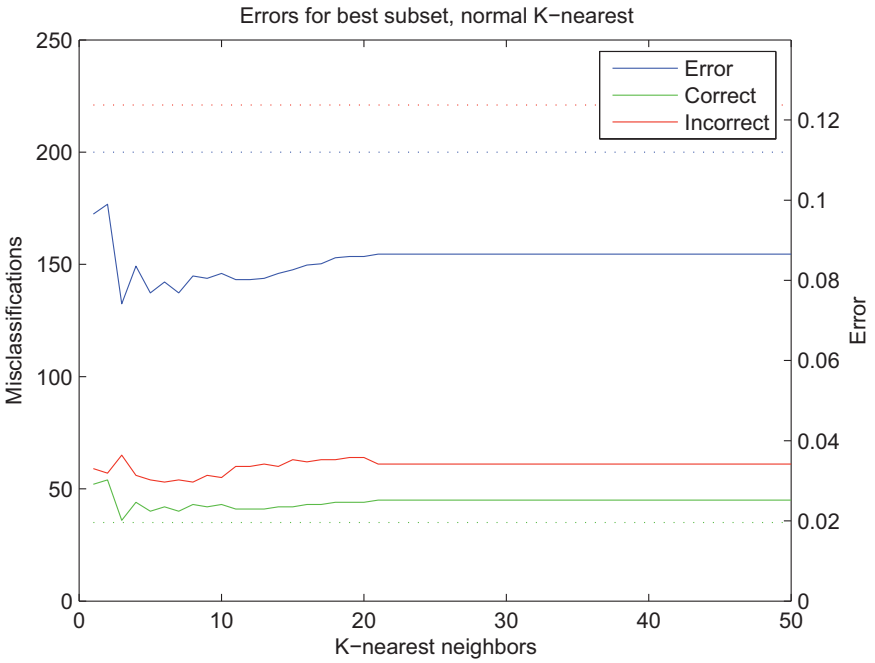


Figure 4.25: Best subset for each value of k. The subset with the lowest error are used. Misclassifications of correct and incorrect entries are plotted. The dotted lines indicates the performance for the old algorithm.

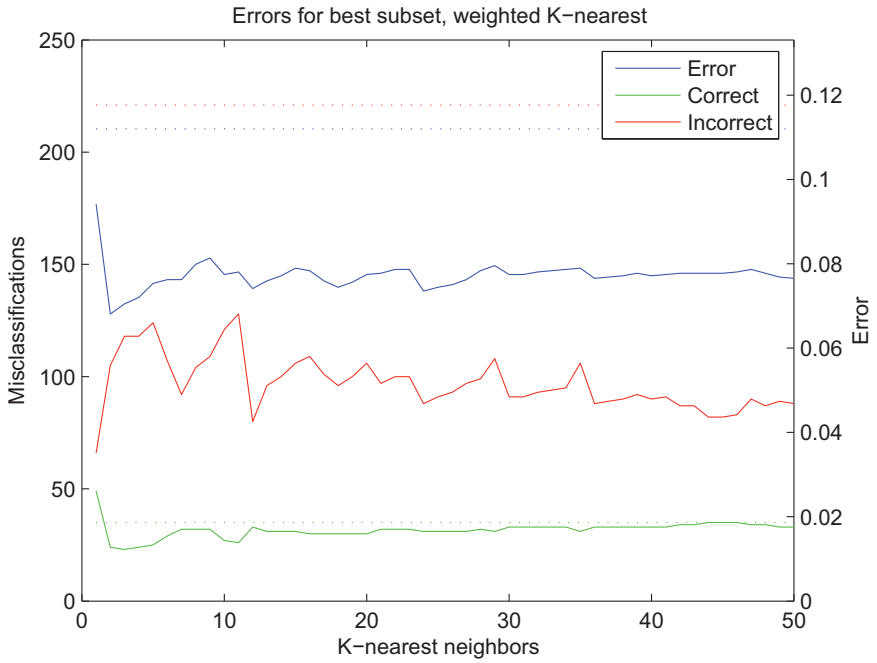


Figure 4.26: Best subset for each value of k. The subset with the lowest error is used. Misclassifications of correct and incorrect entries are plotted. The dotted lines indicates the performance for the old algorithm.

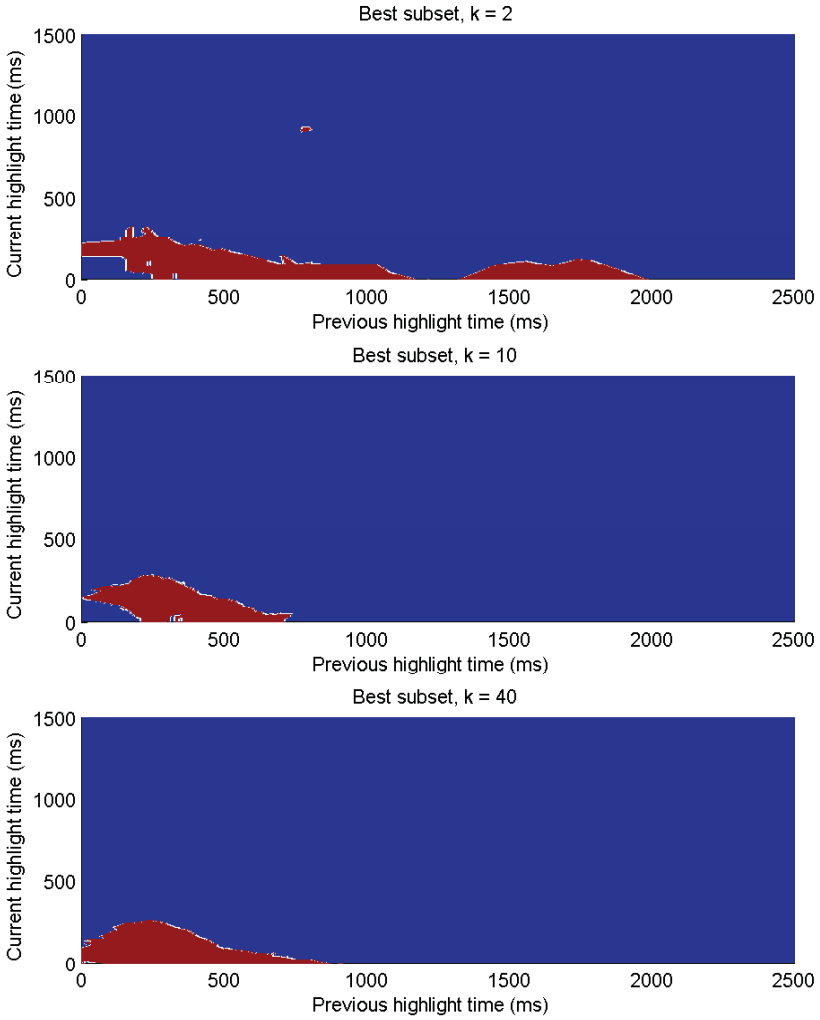


Figure 4.27: Map of the optimal solution for different values of k. The current and previous highlight time were both included in the best subset for all three values of k.

Columns used					K-nearest method		
13	15	16	17	18	K = 5	K = 15	K = 40
1					0.088	0.088	0.088
	1				0.294	0.281	0.261
1	1				0.082	0.092	0.087
		1			0.188	0.188	0.188
1		1			0.090	0.087	0.087
	1	1			0.304	0.282	0.262
1	1	1			0.085	0.091	0.087
			1		0.191	0.188	0.188
1			1		0.102	0.092	0.095
	1		1		0.361	0.306	0.255
1	1		1		0.080	0.091	0.095
		1	1		0.191	0.188	0.188
1		1	1		0.100	0.092	0.101
	1	1	1		0.375	0.314	0.226
1	1	1	1		0.082	0.092	0.095
				1	0.204	0.204	0.183
1				1	0.090	0.089	0.087
	1			1	0.175	0.124	0.127
1	1			1	0.077	0.083	0.087
		1		1	0.203	0.202	0.203
1		1		1	0.090	0.088	0.087
	1	1		1	0.180	0.132	0.128
1	1	1		1	0.077	0.083	0.087
			1	1	0.304	0.224	0.197
1			1	1	0.091	0.091	0.095
	1		1	1	0.169	0.140	0.123
1	1		1	1	0.079	0.091	0.095
		1	1	1	0.307	0.224	0.197
1		1	1	1	0.093	0.091	0.095
	1	1	1	1	0.170	0.143	0.130
1	1	1	1	1	0.081	0.092	0.095

Table 4.13: Error for K-nearest neighbours estimation for different subsets of input data

Columns used					Misclassifications		
13	15	16	17	18	of correct	of incorrect	Error
1					46	99	0.0996
	1				0	620	0.1877
1	1				46	86	0.0956
		1			0	620	0.1877
1		1			46	99	0.0996
	1	1			0	620	0.1877
1	1	1			46	86	0.0956
			1		0	620	0.1877
1			1		46	97	0.0990
	1		1		0	620	0.1877
1	1		1		46	87	0.0959
		1	1		0	620	0.1877
1		1	1		46	98	0.0993
	1	1	1		0	620	0.1877
1	1	1	1		46	90	0.0969
				1	3	600	0.1861
1				1	46	94	0.0981
	1			1	2	671	0.1758
1	1			1	46	88	0.0962
		1		1	2	587	0.1807
1		1		1	46	95	0.0984
	1	1		1	4	539	0.1692
1	1	1		1	46	89	0.0965
			1	1	43	476	0.2091
1			1	1	46	93	0.0978
	1		1	1	56	303	0.1765
1	1		1	1	46	88	0.0962
		1	1	1	41	474	0.2055
1		1	1	1	46	94	0.0981
	1	1	1	1	54	304	0.1737
1	1	1	1	1	46	89	0.0965

Table 4.14: Misclassifications and errors for logarithmic discriminant analysis for all combinations of subsets of input features. p is set to 0.65.

4.5 TUP - improved iPod implementation

The evaluation of the improved iPod implementation was made to test if the improvements since the last evaluation would speed up text entry or improve user experience.

4.5.1 Construction of prototype

The Apple iPod is used as the hardware platform for the prototype as in the previous evaluation. The implementation of the TUP text entry methods is updated to implement the improvements. The text entry evaluation tool is unchanged.

The improvements in the prototype are:

Visual feedback When the user has released the wheel and entered a character, the character will be displayed in the centre of the character circle for 300 milliseconds. This change is included to make it easier for novice users to notice when they enter characters. It it expected that it mainly will help the users during the first few sessions. The change is also expected to make it easier to detect slips and typos for all users.

Character highlighting algorithm The term $P(\varphi_i|\boldsymbol{\varphi}_{1..i-1}, \alpha_i)$ from the character highlighting algorithm have been implemented by the circular sigmoid product function from section 4.4.1.

Slip error correction algorithm The old error classification algorithm (section 4.1.4) has been replaced by a classifier based on logarithmic discriminant analysis (section 4.4.2).

The improved prototype was tested informally on different users before it was used in the evaluation.

4.5.2 Evaluation methodology

The methodology is similar to the first TUP iPod implementation with a few exceptions:

- No pilot study. Since the methodology is similar to the first evaluation there is no need for a pilot study.
- No focus on alternative interaction styles. The participants are not asked about the alternatives to Release-to-Select from section 4.1.2.1.
- No slip error evaluation. The slip error evaluation was done to collect data for the improved slip error correction algorithm. This is not needed in the next evaluation.

4.5.3 Results and discussion

The evaluation was carried out from March 2008 to June 2008. A total of 12 persons participated in the experiment. Their demographics is listed in table 4.15. Most participants completed 14 sessions.

UID	Sex	Age	Hand	Sessions	Text Entry	iPod	Tech	English
1	M	25	Right	11	H	H	M	H
2	M	31	Right	12	H	L	H	M
3	F	27	Right	14	H	L	M	M
4	F	27	Right	15	H	L	M	H
5	M	26	Right	14	M	L	H	M
6	M	26	Right	14	L	H	M	H
7	M	21	Right	14	H	L	H	M
8	F	20	Left	14	H	L	M	L
9	M	21	Right	14	H	H	H	M
10	F	22	Right	15	H	H	M	M
11	M	29	Right	14	H	L	H	H
12	F	20	Right	13	H	L	M	M

Table 4.15: Participants in the evaluation of TUP iPod implementation. The last four columns show the participants previous experience with mobile text entry, iPods, technology and their English skills. They were all expressed in words and later converted to low (L), medium (M) or high (H).

4.5.3.1 Observations and interviews

Most of the participants managed to use TUP without any instruction. 10 out of 12 learned to use Select-on-Release by them self. Some participants accidentally entered many characters the first few seconds, but the visual feedback made them aware of the characters. Almost all participants found and could use the

Delete key. The absolute mapping was found by 2/3 of the participants. In general the participants picked up the method faster than the participants in the first evaluation. The visual feedback is expected to be the main reason for the improved initial use of TUP.

Many participants found the first sessions difficult. They made many errors and were slow. The participants complained especially about space and *a*. After some time they learned the position of the characters, and could speed up the writing. Several participants mentioned that they did not look at the wheel at the end of the evaluation.

Most participants used the tapping strategy. One participant told that she often forgot the Select-on-Release method. When she had placed her finger on a wrong character, she would make another tap closer to the target character. This created a lot of errors.

Several participants experienced difficulties in highlighting characters. Often the highlight jumped forth and back between two characters on each side of the target character. Other participants reported problems when highlighting *a* and space. Especially the word *' a '* caused problems for the participants like in the first evaluation. One participant said that he had created a new strategy to highlight these difficult characters. Instead of making a small correction, he would scroll 4-5 characters away and then back at the target character.

About 1/3 of the participants found the prediction and could explain how it works. Another 1/3 had noticed something, but could not express it. One participant said that it felt like the highlight was faster than him. Another participant was surprised that the highlight was precise, even when she were imprecise.

Some participants had noticed special words that were easy to write. It were words like **thought**, **the**, **you**, **your**, **my** and common part of words like **est**. One participant mentioned that she had learned a special rhythm for entering these words.

4.5.3.2 Text entry speed and error rates

The input log were analyzed to find the entry speed and error rates. The 12 participants had entered a total of 70,748 characters in 2499 phrases.

Similar to the first TUP iPod evaluation, phrases where the relative length is less than $l_{\text{rel}} < 0.8$ were removed from the data set (see equation 4.13). The

filtering removed 36 phrases.

Figure 4.28 shows the average text entry speed for all participants and all sessions. Most participants performed 4-6 WPM in the first session, which is a little better than the first TUP iPod evaluation. In the participants' last session the majority performed between 9 and 12 WPM. This is a lot faster than the first evaluation where the text entry speed was 6-10 WPM. The fastest personal average for a session was 13.1 WPM and the fastest phrase was written at 17.3 WPM.

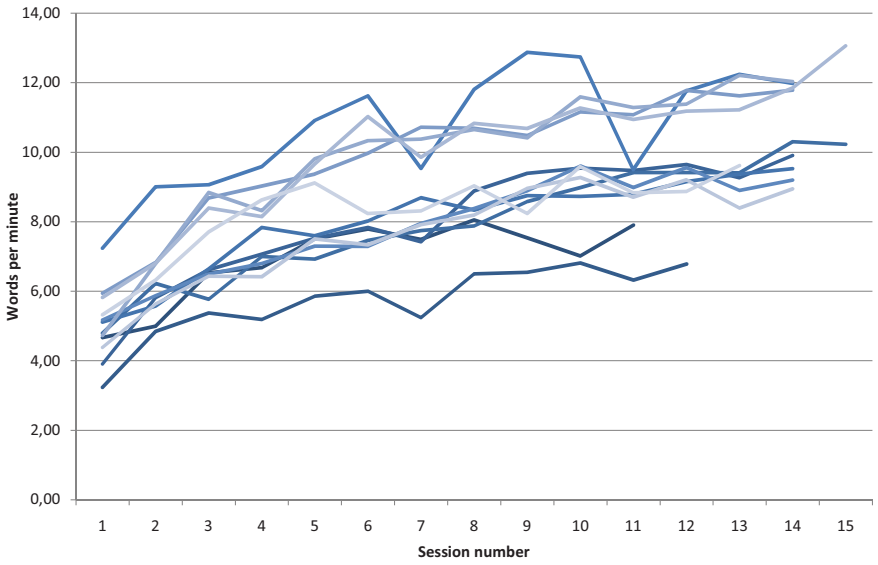


Figure 4.28: Text entry speed for all participants and sessions.

Figure 4.29 shows the average text entry speed and error rates for each session. The average text entry speed starts with 5.1 WPM and ends with 10.7 WPM after 14 sessions. After the first two sessions, the corrected error rate drops to just below 0.04. The uncorrected error rate is also largest for the first two sessions. After session two, it stays between 0.005 and 0.01.

Figure 4.30 shows the difference between the two TUP iPod evaluations. The initial text entry speed is almost the same in both evaluations. After a few sessions the text entry speed for the second evaluation is about 25% faster than the first evaluation. The difference is the same throughout the sessions.

The error rate in the second evaluation is approximately half of the error rate

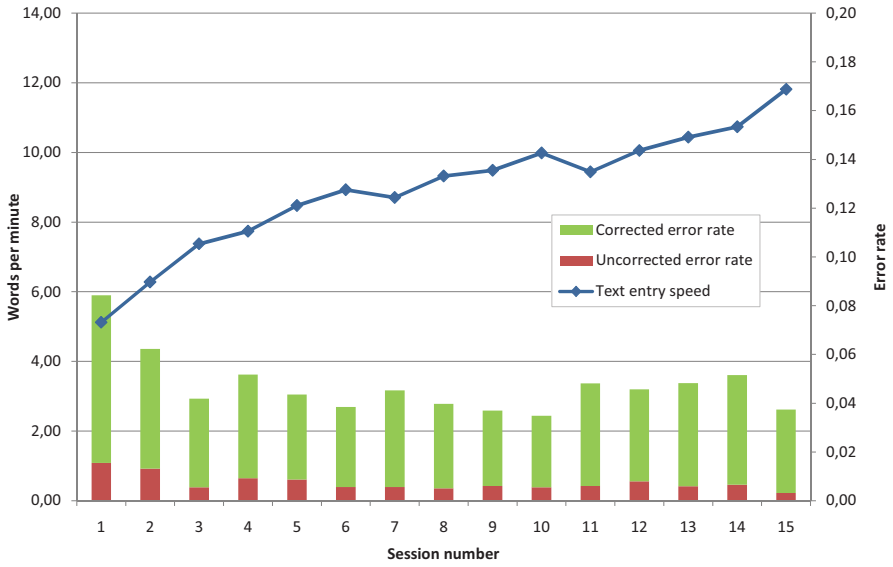


Figure 4.29: Average input speed and error rates. Most participants performed 14 sessions, so the validity of the data is largest for the first 14 sessions.

in the first evaluation. For both evaluations, most of the errors are corrected. The number of corrected errors in the first evaluation is 50% to 100% larger than in the second evaluation. The decrease in corrected errors in the second evaluation is likely to be one of the reasons to the improved text entry speed.

In both evaluations the participants were told to only correct mistakes if they were discovered shortly after they had been committed. The uncorrected error rates in the first evaluation are 3-4 times higher that the uncorrected error rates from the second evaluation. This means that the participants either did not find the mistakes, or that they found them to late to be able to correct them. The improved visual feedback in the prototype is the only change that can account for the big difference in the uncorrected error rates. The visual feedback makes it easy to see whenever a new character is entered. In the first TUP iPod prototype the feedback was limited to the small characters in the input field.

4.5.3.3 Learning and gestures

The learning rate for the participants is:

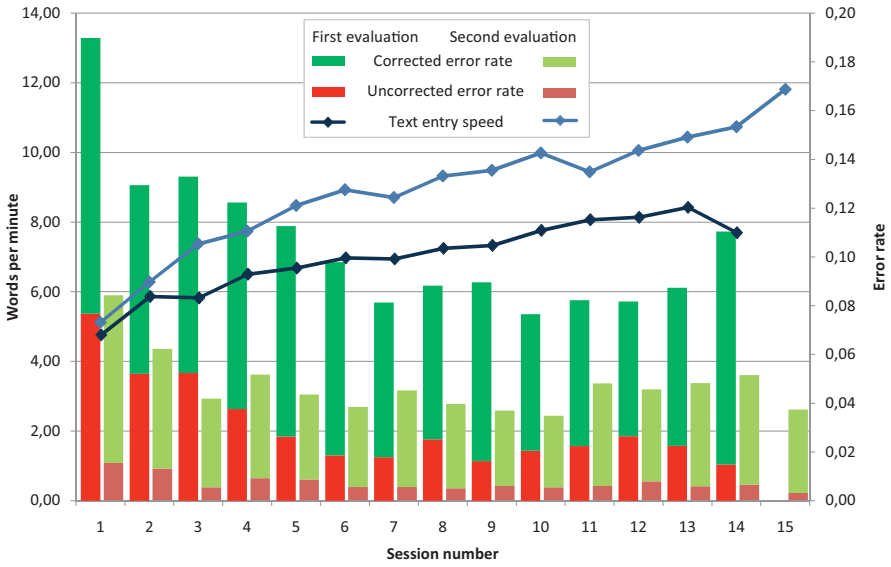


Figure 4.30: Average text entry speed and error rates for both TUP iPod evaluations. Most participants completed 12-14 sessions, so the validity of the data is largest for the first 12 sessions.

$$\mathrm{WPM}_n = 5.32n^{0.2415} \tag{4.27}$$

The details behind the estimate of the learning rate are shown in appendix C.

Figure 4.31 shows the average distribution of the time used to input characters for each session. The **Finger off wheel** time drops from 1300 to 500 milliseconds. Compared to the first TUP iPod evaluation, this is 200-250 milliseconds faster. **Finger on wheel** drops from 500 to 200 milliseconds and **Finger on character** drops from 500 to about 450 milliseconds.

Most participants used the tapping strategy to enter characters. Figure A.2 in the appendix shows the strategies used by each user in each session.

Preparation time Figure 4.17 shows the average preparation time for each character. The trends in the observations are similar to the first TUP iPod evaluation.

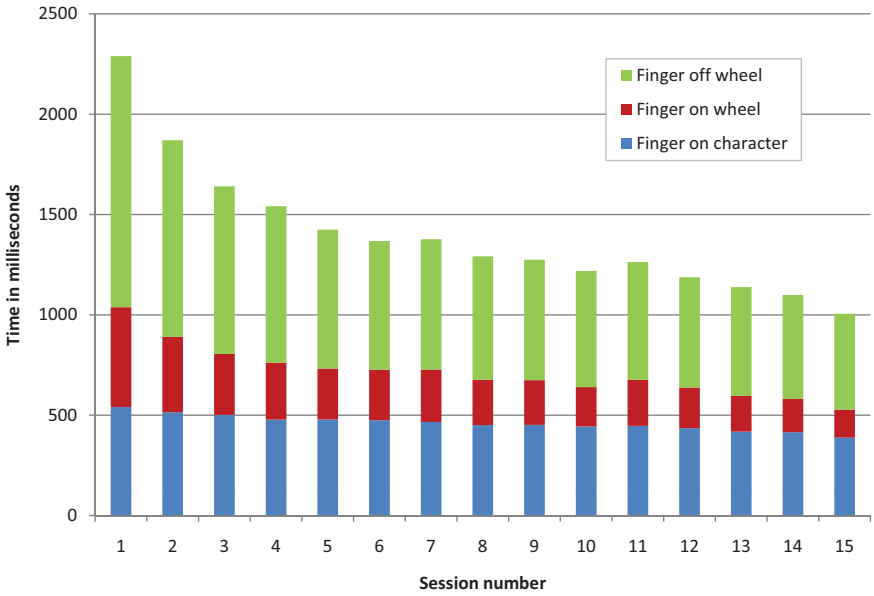


Figure 4.31: Distribution of time used to input characters. **Finger off wheel** is the time where the participants did not touch the wheel. **Finger on wheel** is the time the participants touch the wheel until the target character is highlighted. **Finger on character** is the time the target character was highlighted before the participants lifted their fingers of the wheel.

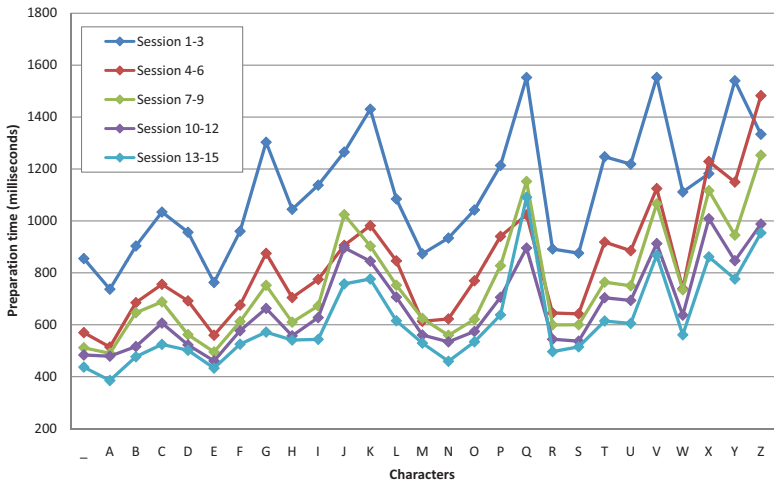


Figure 4.32: Preparation time from a character is entered to the finger is back on the wheel.

4.5.3.4 Effect of language model

Similar to the first TUP iPod evaluation in section 4.3.3.4, the effect of the language model is found. Figure 4.33 shows the average duration of the gestures for the written characters, grouped by probability of the characters.

The time the finger is on the character is close to 500 milliseconds, and unrelated to the probability of the target character. The average time for touching the wheel before the target character is highlighted is 500 milliseconds for characters with low probability. For characters with higher probability the average time is lower. This is expected and is the result of how TUP is designed. Compared to the similar plot for the first TUP iPod evaluation (figure 4.18), the observations from the second evaluation is much more stable. This holds especially for the **finger on character** time. In the first evaluation it varied between 500 and 800 milliseconds.

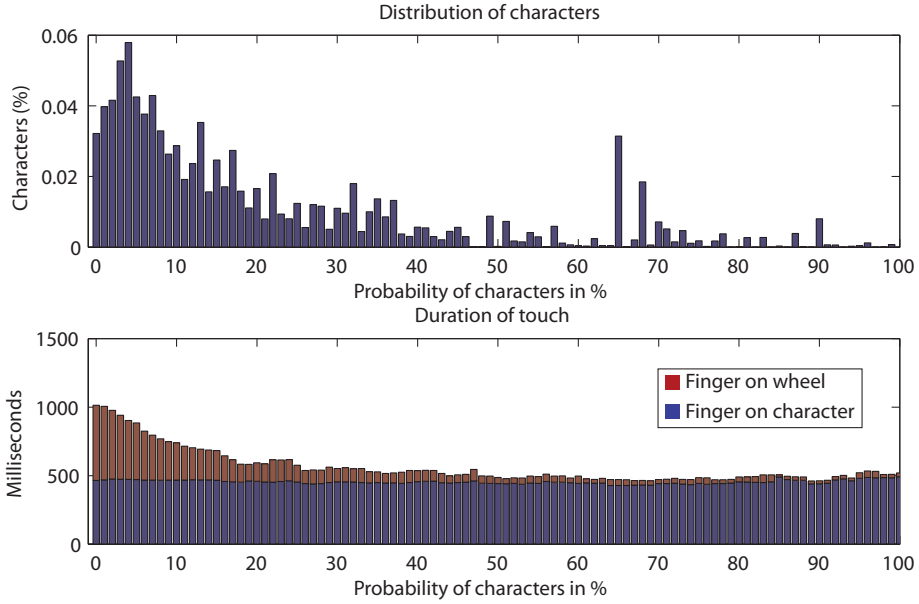


Figure 4.33: Written characters ordered by their probability. All written characters have been sorted by probability and put into 100 bins. The top plot shows the distribution of characters between the bins. The plot in the bottom shows the average duration of gestures.

4.5.3.5 Effects of doing evaluations in different contexts

Observations for both TUP iPod evaluations are used to find the effects of doing evaluations in different contexts. For each session the participants wrote a page in the diary with the place of the session, the light condition and their motivation. These effects of these factors for the text entry speed have been analysed with mixed linear models. The statical model is shown in equation C.10. Appendix C describes the complete analysis.

The light condition has no significant effect on the text entry speed. The motivation level is significant. Sessions where the participants were more motivated have a higher text entry speed.

Table 4.16 shows the effect of doing the evaluation sessions in different places. It has a negative effect on the text entry speed if TUP is used while moving. Both StandWalk and Transport are significantly lower than Bed, Couch and Office. The table can also be used to estimate the performance of TUP in different

situations.

Place	Change in WPM
Unknown	0.0%
$\text{Bed}_{a,b}$	3.3%
$\text{Couch}_{c,d}$	2.7%
Garden	-6.6%
Home (Kitchen)	1.4%
Lecture	7.1%
Livingroom	0.6%
$\text{Office}_{e,f}$ (Desk)	2.9%
$\text{StandWalk}_{a,c,e}$ (Standing, Walking)	-8.4%
$\text{Transport}_{b,d,f}$ (Car, Train)	-2.0%

Table 4.16: Change in text entry speed if the evaluation sessions are done in different places. The places in parenthesis indicate that observations from these places have been combined in the analysis. The subscripts indicate that these places are significantly different using a 5 % level and Tukey-Kramer corrections of p-values. All other places are not significantly different.

4.5.3.6 Effect of participants previous experience

The analysis of the observations shows that the participants' previous experiences and English skills have no significant effect on the text entry speed.

4.5.3.7 Effect of slip error correction algorithm

The slip error correction algorithm (section 4.1.4) is included in the TUP prototypes to correct slip errors. The algorithm was improved for the second TUP iPod evaluation. Table 4.17 shows the effect of the slip error correction algorithms for both TUP iPod implementations.

1198 and 5631 times the slip error correction algorithm changed a incorrect character to a correct character. With out the algorithm all these characters would have been incorrect. The algorithm is mainly used when the target character has low probability. Entering characters with low probability requires extra precision from the user. A small slip is more likely to remove the highlight from characters with low probability than for characters with high probability.

236 and 597 times the corrected characters were incorrect. These can be due to one of two causes: Either the algorithm has changed correct characters to

TUP iPod	Correct	Incorrect	Total
All characters	48,952	3,934	52,886
Slip error	1,198	236	1,434

Improved TUP iPod	Correct	Incorrect	Total
All characters	69,970	2,685	72,655
Slip error	5,631	597	6,228

Table 4.17: Count of correct and incorrect characters entered by the participants. The slip error lines shows how many times the slip error correction algorithm was used, and whether the entered characters were correct or incorrect.

incorrect characters, or the users have entered incorrect characters. This could be typos, wrong words or similar errors. It is not possible to assign the individual error to the causes, with out doing a manual analysis of the error logs. This has not been done because it is very time demanding.

A test with the hypergeometric distribution shows that the fraction of incorrect characters is significantly higher when the slip error correction algorithm has been used. This can be because the algorithm changes correct characters. Another hypothesis is that difference, is due to the users making more errors when the target characters have low probability.

The algorithm was used for 2.7% of the written characters in the first evaluation and for 8.6% in the second evaluation. The big difference is likely due to the fact that the participants adapt to the system. In the first evaluation the participants made many mistakes. This will results in a speed-accuracy trade-off, where the participants try to be more accurate when the enter characters. One way of doing this is to be more concentrated when removing the finger from the wheel. The participants in the second evaluation wrote faster and made fewer errors. Their accuracy will be relaxed, thereby making more slips. Most of the slips are correct automatically, so the participants will not have to slow down.

4.5.3.8 Effect of improved character highlighting algorithm

The improved TUP iPod prototype uses a new algorithm to select which character to highlight. The new algorithm was designed to make it easier to highlight characters. It is expected that this will lead to fewer errors and increased text entry speed.

To analyse the effect of error rates on text entry speed, the mixed linear models from earlier are extended with the corrected and uncorrected error rates. The final model is shown in equation C.18. Appendix C describes the complete analysis.

It is found that there is no significant effect of the uncorrected error rate and the text entry speed. This is as expected because the uncorrected errors are included in the calculation of text entry speed (see section 3.3.2 for details). The participants were told not to correct an error if more than two correct characters would be deleted in the editing process. It is likely that many of the uncorrected errors were newer found by the participants.

There is a significant effect of the corrected error rate and the text entry speed. The corrected errors are not included in the calculation of text entry speed, and it is therefore expected that many corrected errors will decrease the text entry speed. Table C.13 in the appendix shows the relation between corrected error rate and text entry speed.

The new statistical model is used remove the effect of corrected errors from the estimated learning rates. This is done for both evaluations. The estimate for evaluation 1 is shown in equation 4.28 and in equation 4.29 for evaluation 2. These estimates assume that the corrected error rate is 0. A plot of the learning rate can be found in figure 4.34. The estimated learning rates with inclusion of corrected errors are also shown.

$$\mathrm{WPM}_n = 5.00n^{0.2184} \tag{4.28}$$

$$\mathrm{WPM}_n = 5.84n^{0.2184} \tag{4.29}$$

The relation between the estimated learning rates in equation 4.28 and 4.29 expresses the improvement in the task of entering characters. The relation is 1.168 which means that the improved TUP iPod method is 16.8% faster than the original TUP iPod method.

The similar relation between the estimated learning rates in equation 4.14 and 4.27 is 1.288. These learning rates include the effects of faster entering of characters and less errors to correct. By dividing 1.288 with 1.168 the effect of the fewer errors is found to be an increase in text entry speed on 10.3%.

The improvements of text entry speed in the improved TUP iPod is expected to be due to the new character highlighting algorithm and the slip error

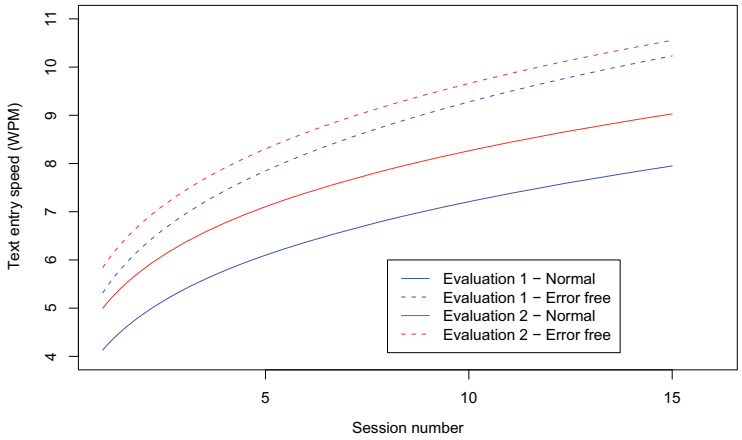


Figure 4.34: Estimated learning rate and error free learning rate for both TUP iPod evaluations.

correction algorithm. It is not possible to estimate the precise effect of each improvement. The improved character highlighting algorithm makes it possible to be less precise when highlighting characters. The slip error correction algorithm allows the participants to commit slip errors. Both improvements relaxed the requirements to the participants' precision, thereby enabling them to write faster. The overall improvement in text entry speed is very significant.

4.6 A key based variant of TUP

This section will introduce a variant of TUP for use with ITU-T keyboards. The conceptual model will be very simple, to make it easy to learn for novice users. Language models are used to improve the performance of the method. Extra features like context-aware adaptive language models and automatic correction of spelling mistakes and typos, can be added without increasing the complexity of the text entry method. The text entry method is not implemented or evaluated.

4.6.1 Summary of problems with current text entry methods

The standard multitap text entry method requires many key presses to write text. It can be optimized by using chording or optimized character layouts. Chording requires either two handed use or physical modification of the device. Optimized character layouts have a steep learning curve, and will require special hardware for each language variant.

Single-tap methods can be divided into character or word level disambiguation methods. Character level methods require the user to verify and correct each character after it has been written. Word level methods suffer from problems with word stability and the fact that all words unknown by the language model, need to be entered with a fallback method.

4.6.2 Requirements to TUP-Key

TUP-Key is designed to be a key based variant of TUP. The same three main requirements are used as a starting point: Simple conceptual model; Character based; Hidden use of language model. Besides that the method should be easy to learn and to use, and it should enable an easy transition from novice to expert use.

To make it easy for the user, a normal ITU-T keyboard with the character layout assigned from Laverack and von Niman [2007] will be used. The keyboard and character layout are known by most people, and will help ensure good novice performance and user experience.

It is difficult to hide the language model completely. A press on a key is a discrete event, and it has to provide some feedback to the user. The user knows exactly which key was pressed, and expects the feedback to correspond to that key. With TUP, the touch of the wheel generates a continuous output value. The high density of characters means the user will not know exactly which character was touched. It is therefore possible to use the language model without the user will notice it. The use of language models in TUP-Key cannot be hidden completely. Is is made to be very simple, while enabling users to interact more with the language model if they want to.

4.6.3 Design of TUP-Key

TUP-Key is designed as a single-tap text entry method where each key press will enter a character. It is using a hybrid between character and word level disambiguation. Besides the ITU-T keyboard, a next, previous and accept key is needed. Many mobile devices have a joystick or 5-way key for navigating the user interface. They down, up and right key can be used as next, previous and accept.

When character keys are pressed, a language model will create a list of all possible character strings that corresponds to the pressed keys. The list is sorted by probability. The most likely string is displayed. At any time the user can choose to accept the predicted string or select less likely strings with the next and previous keys.

The predicted string of characters can be accepted in three ways;

- By pressing the accept key.
- By selecting another predicted string of characters and press accept or any key on the ITU-T keyboard.
- By pressing the space key.

The main difference between TUP-Key and text entry methods like LetterWise and T9 is that the user can choose when the predicted characters should be corrected or accepted. With Letterwise each single character has to be accepted. T9 works on word level, and all characters in the word have to be entered before it can be accepted.

Novice users can choose to accept each character after it has been written. This strategy will remove the problem with word stability. More experienced users

can choose to write the entire word before accepting it, or to divide the word and accept each part by itself. When writing long words, it will often be easier to accept parts of the word instead of the entire word. This will make it easier to find typos and spelling mistakes. This holds especially for languages where most compound words are in closed form without a space in the middle.

4.6.3.1 Writing words with low probability

If the correct keys are pressed, the target word is guaranteed to be on the list of predicted strings. It can be entered by iterating through the list, until the word is found. If the target word has low probability, it will be far from the head of the list. This will make it very time consuming to enter it. It can be entered faster by dividing the word up in smaller parts, and correct and accept each part alone. To be effective, this requires that the user has a suspicion that a word has low probability. Otherwise the user might want to start with writing the entire word at once, before realizing that it has to be divided in smaller parts.

4.6.3.2 Language models for TUP-Key

TUP-Key can be used with most character level language models. An algorithm is used to create a list of all possible strings of characters for a given series of key presses. The list ordered by probability with the language model. Most language models that work on character level are able to do this. An example could be the YourText language model introduced in section 5. Section 5.3.3 describes how it can be used to predict probabilities of strings of characters.

The language model can make better predictions, if it uses the history up to the current characters. Table 2.4 shows that *run* is more likely than *sun*. If the history is *the yellow*, then *sun* is more likely than *run*. The disadvantage of using the history is that it can confuse experienced users. Some users have learned the order of some of the predicted words. If the history is used, then the order of the predicted words will be different for different histories.

4.6.3.3 Automatic corrections of spelling and typing errors

TUP-Key can be extended with automatic correction of spelling and typing errors. This can be done with only a minor increase of the complexity. If the user makes errors while pressing the keys, the target word will not be on the list

of predicted words. This is because the list only contains the words that can be made from the keys pressed by the user. The algorithm that creates the list of predicted words can be relaxed to allow the user to make errors.

4.6.4 Discussion of TUP-Key

One of the main advantages of TUP-Key is that it is very similar to LetterWise and T9. By correcting and accepting the predicted characters at different times, TUP-Key can be used in the way preferred by the individual user. Novice users can choose to correct each individual character, so they always can see what they have written. More experienced users can speed up the text entry, by correcting and accepting larger chunks of characters. If the user only corrects entire words, the method is very similar to T9. It is therefore expected that the performance will be similar to T9 performance for experienced users. The performance of TUP-Key is expected to be better than T9 when writing words that are not present in the T9 dictionary.

All words can be written with the same method. There are no unknown words, but only words with low probability. If the user knows that some word has low probability, then it can be entered easily by dividing it up in smaller parts. Otherwise it will be time consuming to enter words with low probability.

TUP-Key can be used together with adaptive language models. When words with low probability are entered, they language model will adapt to these words. This will make it easier to enter the word next time.

TUP-Key can be extended with extra features such as automatic corrections of spelling and typing errors. It is also possible to make variants of TUP-Key for use with keyboards with less or fewer character keys.

4.7 Conclusions

This chapter has introduced the TUP text entry method. TUP is designed for use with touch sensitive wheels, sliders and similar input devices. The method have been designed to be easy to use for novice users and to provide an easy transition from novice to expert. The method works on character level, so all possible words and character combinations can be written. TUP can be used with adaptive language models.TUP has been evaluated in three evaluations. 33 users have participated in the evaluations and more than 4.600 phrases have

been transcribed. TUP-Key has been introduced. It is a variant of TUP for use with ITU-T keyboards. TUP-Key is not implemented or evaluated.

Both methods can be used with different sizes of input devices. If TUP has more space, then the density of characters will be smaller. This will make it easier to highlight the correct characters. For TUP-Key the size and number of keys can be changed to fill the available space.

TUP was designed by combining low level models of human behaviour with the language model as a high level task model. The low level models were optimized by using methods from statistical learning and data from an evaluation of TUP. The optimized low level models were used to improve the algorithms. TUP was also improved with extra visual feedback, as a result of the qualitative data from the evaluation. An evaluation of the improved version of TUP showed that the text entry speed had increased 29% and the error rate has dropped from 8% to below 5%.

Chapter 5

YourText - a context-aware adaptive language model

This chapter introduces a new language model framework called YourText. It is a context-aware adaptive language model based on a set of PPM models. It is described how different language models can be combined to make a more powerful model. A corpus of sent and received text messages from users are created and used to verify the new language model.

5.1 Language Usage in Mobile Text Entry

Our language varies a lot depending on what we are doing. It is unlikely that a single language model can be applied with success in many different applications or domains. There are numerous examples of how specific language models perform better than general purpose models. Teahan and Cleary [1996] got a 10% reduction of the entropy of classic English literature, when their language model where trained on texts from the same author as the test. Most domain specific language models have been trained with text from a given domain to learn new words. For the mobile text entry language domain, it will probably not be enough to add some extra words to the language model. The structure of the language model has to be changed to match the specific language usage.

Mobile text entry is often used to enter single words or short phrases. It could be mobile text messages, calendar appointments, phone book entries or internet urls. Mobile text messaging is the service that requires the most text entry, even though each single message has a limit of 160 characters. A study by Laursen [2006] shows that people are having long conversations over mobile text messages. Sometimes a conversation can be paused for hours before it is resumed. The study also shows that some of the observed people had 2-4 simultaneous conversations with different people at the same time.

In many ways mobile text entry resembles spoken language more than written language. Each message has a sender and one or more receivers. Often the message is part of a larger information flow between two or more people. The mobility of the devices enables the user to enter text when and where he wants to.

Studies by Grinter and Eldridge [2001] shows that users tend to use a lot of abbreviations to save time when entering text. Many of these abbreviations are not established parts of the language. For example these different abbreviations for tomorrow: *2moro*, *2morra*, *tomor* and *2morrow*. Because of the use of abbreviations mobile text messages include a high number of non-dictionary words.

5.2 Design requirements

To be able to better support text message language usage, the language model should be aware of the context of the text entry. Examples of contexts are when the messages are sent, who receives them, where they are sent from, whether they are replies to another message, etcetera.

The many non-dictionary words and the general evolving of language, make it desirable to have an adaptive language model. Evaluations done by Kuhn and Mori [1990] showed that words used recently are much more likely to be used again soon after. The language model should also support this kind of adaption.

Language models typically adapt to the text they have seen in the past. With language models for mobile text entry, it is possible to let the language model adapt to other sources of information. It could be calendar appointments, names of contacts, GPS locations, received messages, text from visited websites, etcetera. This implementation of YourText will only adapt to sent and received messages, but can easily be extended to include other sources of information.

Even with an adaptive language model that can learn from many sources, it is very likely that the user will write words unknown to the language model. A word-level language model will assign the same probability to all unknown words. A character-level language model will assign a probability based on how close the unknown word resembles the language structure encoded in the model. YourText will be based on a character-level language model, to better support unknown words. Another advantage of character-level language models is that they can be used with both character-level and word-level text entry methods. The opposite, to use word-level models with character-level text entry methods, is much more complicated.

Based on this and the previous sections, a number of requirements to the language model are made.

Adaptive The language model should be adaptive. The probabilities should be continuously updated to better match the users' language. Many sources can be used to adapt the language model; the users own text, text of received messages, device phone book entries, calendar events, and etcetera.

Context-aware The language model should be able to use contextual information to provide better probability estimates. Contextual information could be the receiver of the text message, the time of day, location, etcetera.

Recency The language model should favour words that have been used recently.

Character-based The language model should work on character-level.

5.3 Design and implementation

Focus has been on how to incorporate new design ideas into a simple language model. It is very likely that the language model can be improved, by adding techniques from established and state of the art language models. The language model has been designed so that it can be used in real time on a desktop computer. The implementation has not been optimized in any way regarding performance and memory usage.

YourText is a framework that uses a pool of language models. Each time YourText is needed, a subset of the language models are selected from the pool. Linear interpolation between the chosen language models is used to create the

final probability estimate. Linear interpolation has been chosen because it is fast and easily can be used with all types of language models. The weights are dynamically updated while YourText is used.

The pool of language models is dependent on the configuration of YourText. A typical configuration will have one or more basic language models and sets of language models associated with different contextual information. These sets will be called contextual dimensions. Examples of contextual dimensions could be location, time of day, recipient, and etcetera. Each contextual dimension has a set a language models that corresponds to the state of the contextual dimension. For the recipient dimension, this will be a language model for each recipient that the user will communicate with. When YourText is used, the relevant subset of language models are selected from the pool based on the current context of the user.

YourText can be used with all types of language models. In this implementation it uses a novel adaptive language model called PPM*D.

5.3.1 PPM*D language model

YourText will be based on a set of unbounded length n-graph PPM models (as described in section 2.3.6.2). A character based model is used so YourText can predict the next character given the current history. The model is called PPM*D. The PPM* refers to the compression method by Cleary, Teahan, and Witten [1995], where variable length n-graphs are used. The D refers to a decay function that is build into the language model. The decay function will reduce the counts of the characters, so that characters that have not been used recently will be less likely.

Most adaptive language models are only able to remember. They work by increasing counts of words and characters each time they are seen in the text. A word mentioned 10 times in the beginning of a text will have a higher probability than a word mentioned five times in the same section. By introducing a decay function the language model is also able to forget. There are other examples of language models that are able to forget. Kuhn and Mori [1990] made a cache based language model, where a trigram model is extended with a cache component. The last 200 words from different Parts-Of-Speech classes were saved in a cache. Words from the cache were assigned higher probabilities than words outside the cache. Clarkson and Robinson [1997] made a language model with an exponentially decaying cache component. The probability of the words in the cache is dependent of the distance to the current word. Both these language models were improved by adding cache components.

In PPM*D the decay function is built into the language model, and not in a separate cache component.

PPM*D is defined using an alphabet Q, a decay factor λ and a pruning level δ.

It is implemented in a tree-structure, where the history controls the shape of the tree. The root is the empty history. The child nodes beneath the root have a history of one character. Underneath are nodes with histories of two characters and so on. Each node holds the count of how many times each character have been observed after the current history. The depth of the tree is similar to the length of the n-grams. Nodes at level n contain the count of n-grams.

Figure 5.1 shows a small example tree for a PPM*D language model. To make it easy to find the longest matching history in the tree, the history is encoded backwards. To find the most likely characters to follow *He lost the keys to his ca*, you will have to perform the following steps: Start in the root; go to node *a*; go to node *c*. This node is the longest possible history in the language model. The node holds the counts of all characters that have been observed to follow *ca*.

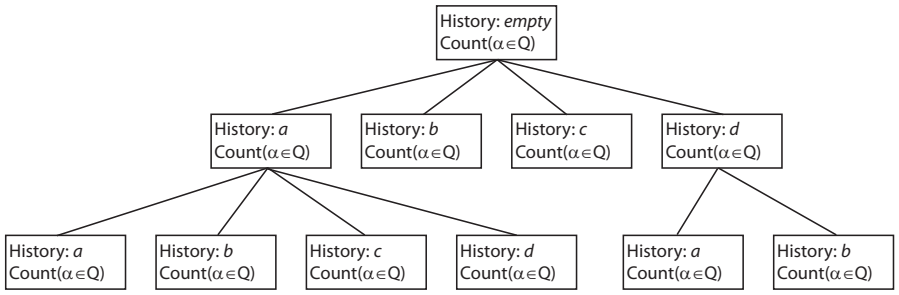


Figure 5.1: Example tree for a PPM*D language model

5.3.1.1 Probability estimates

PPM*D uses backoff [Katz, 1987] to estimate the probability based on the longest possible history. α_0 is defined as the next character the user will enter and $\mathbf{h}_n$ is the history of all the characters the user have entered until now: $\alpha_{-n}\alpha_{-n+1}\alpha_{-n+2}...\alpha_{-2}\alpha_{-1}$.

The length of $\mathbf{h}$ is $|\mathbf{h}| = n$.

$C_c(\alpha|\mathbf{h})$ is defined as the count of how many times α follows the history $\mathbf{h}_c$. Only the last c characters in the history are used.

$$C_c(\alpha|\mathbf{h}) = C(\mathbf{h}_c\alpha) = C(\alpha_{-c}\alpha_{-(c-1)}\alpha_{-(c-2)}...\alpha_{-2}\alpha_{-1}\alpha) \tag{5.1}$$

For example $C_0(\alpha|\mathbf{h})$ is the count of α. Since $c = 0$, the history is not used at all.

The probability of a character α_0 to follow the history $\mathbf{h}$ is

$$P(\alpha_0|\mathbf{h}) = P_c(\alpha_0|\mathbf{h}), c = |\mathbf{h}| \tag{5.2}$$

$$P_c(\alpha_0|\mathbf{h}) = \begin{cases} \omega_c \frac{C_c(\alpha_0|\mathbf{h})}{\sum_{\alpha \in Q} C_c(\alpha|\mathbf{h})} & \text{if } C_c(\alpha|\mathbf{h}) > 0; \\ (1-\omega_c)P_{c-1}(\alpha_0|\mathbf{h}) & \text{if } C_c(\alpha|\mathbf{h}) = 0 \end{cases} \tag{5.3}$$

$$P_{-1}(\alpha|\mathbf{h}) = \frac{1}{|Q|} \tag{5.4}$$

The backoff weights ω_c are found from Witten-Bell discounting [Witten and Bell, 1991]. Backoff and Witten-Bell discounting are described in section 2.3.4 and 2.3.5.

5.3.1.2 Pruning

PPM*D uses an unbounded length n-graph PPM model. Since there is no limit on the length of n-graphs, there need to be another way to limit the size of the language model. PPM*D uses a pruning criterion to decide when to stop adding child nodes to the tree. The criterion is used on each individual node in the tree, allowing the tree to vary in depth. PPM*D uses the Kullback-Leibler divergence by Kullback and Leibler [1951] as a pruning criteria. It has also been used by Stolcke [1998], Gao and Zhang [2001].

It can measure the information gain of adding a extra child node to the tree. It is defined as

$$\mathrm{D_{KL}}(X||Y) = \sum_i X(i) \log_2 \frac{X(i)}{Y(i)} \tag{5.5}$$

The Kullback-Leibler divergence is closely related to the entropy and cross entropy:

$$\mathrm{D}_{\mathrm{KL}}(X||Y) = H(X,Y) - H(X) \tag{5.6}$$

$\mathrm{D}_{\mathrm{KL}}(X||Y)$ is a measure of how much extra information that is required to code a sequence from X with Y instead of X. For PPM*D X and Y are the distribution of the characters with different history lengths. For any two given histories $\mathbf{h}_c, \mathbf{h}_{c-1}$, the information gain of the extra character can be described as:

$$\Delta(\mathbf{h}_c, \mathbf{h}_{c-1}) = \mathrm{D}_{\mathrm{KL}}(P_c||P_{c-1}) = \sum_{\alpha \in Q} P(\alpha|\mathbf{h}_c) \frac{P(\alpha|\mathbf{h}_c)}{P(\alpha|\mathbf{h}_{c-1})} \tag{5.7}$$

If the information gain of a specific history $\Delta(\mathbf{h}_c, \mathbf{h}_{c-1})$ is less than the pruning level δ, the node with history $\mathbf{h}_c$ will be excluded from the model.

5.3.1.3 Initial training of PPM*D

To be able to produce good estimates, the language model needs to be trained on some text. The training of PPM*D is done in the following steps:

1. Create an empty tree with a root element.
2. Set n to 1
3. Iterate through the corpus and count all n-graphs. Add them to the appropriate nodes at level n.
4. Prune tree. Remove nodes at level n if the information gain is below δ.
5. Create child nodes for all nodes at level n
6. Increase n with 1.
7. Repeat from step 3
8. Continue until the tree stops growing

5.3.1.4 Updating of PPM*D

When PPM*D is used in a text entry system, it will adapt to the users writing style and vocabulary. Each time a character is entered, the language model will update the counts of the characters. The count of the entered character will be increased by one, and the counts of all characters will be reduced by the decay factor λ.

If the user enters α after the history $\mathbf{h}$, the count for α is increased with 1:

$$C(\alpha|\mathbf{h}) = C(\alpha|\mathbf{h}) + 1 \tag{5.8}$$

The counts for all characters are reduced by the decay factor λ:

$$C(\alpha|\mathbf{h}) = C(\alpha|\mathbf{h})\lambda \tag{5.9}$$

λ controls how fast the counts will decay. If $\lambda = 1$, the counts will not be decayed. In this case the language model will be similar to a normal n-graph model. Values closer to 0 will increase the decay speed. Typical values of λ will be very close to 1.

The counts are updated for all matching histories in the language model:

$$\mathbf{h}_c,\ c = 0...|\mathbf{h}|\ if \sum_{\alpha \in Q} C(\alpha|\mathbf{h}_c) > 0 \tag{5.10}$$

For the tree implementation, this means that all nodes in the path from the root node to the deepest node in the history are updated.

If the longest matching history in the language model is nondeterministic, then the length of the history is increased with one character. The longest matching history is nondeterministic if at least two different characters have been observed to follow the history (equation 5.11). This criterion is similar to a Kullback-Leibler divergence of 0. It is possible to prune the language model later, but it will not be feasible to prune every time the user enters a new character.

$$\forall_{\alpha_i \in Q} C(\alpha_i|\mathbf{h}) < \sum_{\alpha \in Q} C(\alpha|\mathbf{h}) \tag{5.11}$$

5.3.2 Combinations of language models

The basic configuration of YourText will use two PPM*D models; a long term and short term model. The long term will have a decay factor close to 1. The model will be very stable and will serve as a basic model. The short term model will have a lower decay factor, so data in the short term model will decay faster. The short term model can be seen as a cache component. It is useful because we often use the same words multiple times over a short period of time.

The two language models will be combined using linear interpolation with dynamic weights.

$$P = \frac{P_{\text{long term}}}{Z} + \frac{P_{\text{short term}}}{P_{\text{long term}}} \frac{P_{\text{short term}}}{Z}, Z = \frac{P_{\text{short term}}}{P_{\text{long term}}} \tag{5.12}$$

More language models can be included in YourText. Here P_0 is assumed to be the long term model:

$$P_{YourText}(\alpha) = \sum_i \frac{P_i(\alpha)}{P_0(\alpha)} \frac{P_i(\alpha)}{Z}, Z = 1 + \sum_i \frac{P_i(\alpha)}{P_0(\alpha)} \tag{5.13}$$

This way YourText can be extended with additional language models. It could be language models from different contextual dimensions.

The notation for each individual PPM*D model is:

$$\Theta^{\lambda,\delta}_{\text{identifier}}$$

A configuration of YourText with different PPM*D models is written this way:

$$[\Theta^{\lambda,\delta}_{\text{model0}} \bullet \Theta^{\lambda,\delta}_{\text{model1}} \bullet \Theta^{\lambda,\delta}_{\text{model2}}]$$

5.3.3 YourText and text entry methods

YourText can be used with most text entry methods. For all character-level text entry methods, YourText can directly specify the probability $P(\alpha|\mathbf{h})$ of a character α to follow a given history $\mathbf{h}$.

5.3.3.1 Probability of words

Given a word $\mathbf{w} = \alpha_1\alpha_2...\alpha_i$ where $i = |\mathbf{w}|$, the probability of $\mathbf{w}$ to follow history $\mathbf{h}$ can be expressed as;

$$P(\mathbf{w}|\mathbf{h}) = P(space|\mathbf{hw}) \cdot \prod_{j=1}^{i} P(\alpha_j|\mathbf{h}\alpha_1\alpha_2...\alpha_{j-1}) \tag{5.14}$$

The *space* term is required to get the exact probability of $\mathbf{w}$, and not the probability of all words that starts with the $\mathbf{w}$.

5.3.3.2 Word-level text entry methods

Even though YourText is based on a character level PPM model, it can be used to predict words. This is useful for text entry with word-level disambiguation and for word completion systems.

Many text entry systems will provide a list of candidate words that need to be ranked. The most likely words on the candidate list will be presented to the user, who has to pick one of them to enter. The task of ranking the list of candidate words is trivial given equation 5.14 as long as the list of candidate words is reasonable small.

For text entry systems that uses ITU-T keyboards the list of candidate words will be all possible combinations of the characters on the pressed keys. The candidate list will increase with a factor 3-4 for each key press made by the user. If the user intends to write *summer* the list of candidate words will be all combinations of *(pqrs) (tuv) (mno) (mno) (def) (qprs)*. This gives 1296 possible combinations for the candidate list. Longer words like *introduction* will have 708, 588 possible combinations. If language-specific characters are supported by the text entry method, the branching factor will increase from 3-4 to about 8 for each new key press.

The problem with long list of candidate words can be solved by iterating over the sequence of key presses and frequently pruning of the list. The following algorithm outlines how to create a list of candidate words:

1. Start with an empty list of candidate words.

2. For each key pressed by the user do:
   (a) Create a new list of candidate words with all combinations of the current list and the characters from the key press.
   (b) Calculate the probability for each candidate word, given the history **h**.
   (c) Prune the list of candidate words by one of these criteria:
      Fixed size Keep the n words with highest probability.
      Fixed probability Keep all words with probability larger than p. At least one word should be kept on the list.
3. Rank the words according to probability.

The algorithm will return a ranked list of the most likely candidate words. The order of the list is very likely to be different from the correct list, found by calculating the probability for all possible combinations. The differences will be largest among the words with low probability. A high value of n or low value of p will ensure that many words are included on the candidate list. It will require more computational power, but will reduce the risk of errors among the top ranked words in the candidate list.

Reasonable values for n or p can be estimated by numerical analysis.

5.3.3.3 Word completion

Many text entry methods include word completion, where the system will predict entire words based on the history of text written until now. There are three ways of using word completion:

Complete next word Predict the next word based on the history.

Complete based on partial word Predict the current word based on the characters entered so far and the history.

Complete based on key presses Predict the current word based on the keys pressed so far and the history.

Method 2 and 3 are only different if the text entry method is ambiguous. A normal QWERTY keyboard will use method 2, because the beginning of the word is known from the key presses. Ambiguous text entry methods like T9

will use method 3, because the key presses can represent many possible word beginnings.

A common algorithm can be used to create a list of word completion candidates for all three methods. The algorithm is very similar to the algorithm used to create a list of candidate words from the previous section.

1. The initial list of word completion candidates is dependent on the method:

   Complete next word Empty list of word completion candidates.

   Complete based on partial word The partial word entered by the user is the only word on the list.

   Complete based on key presses The n most likely candidate words from previous section are used as word completion candidates.

2. Repeat until all word completion candidates on the list end with a space character:

   (a) Create a new list of word completion candidates with all words from the current list that end with a space character. Combinations of all words that do not end with a space character and all characters are appended to the list.

   (b) Calculate the probability for each word completion candidate, given the history h.

   (c) Prune the list of word completion candidates by one of these criteria:

   Fixed size Keep the n words with highest probability.

   Fixed probability Keep all words with probability larger than p. At least one word should be kept on the list.

3. Rank the words according to probability.

Reasonable values for n or p can be estimated by numerical analysis.

5.3.3.4 Character set

YourText can be used with different character set. The character set used in with YourText in this thesis contains only lower case letters. The Danish letters æ, ø and å are included because YourText is going to be used with Danish text. The most common punctuation marks (, . ? ') are included. The rest of the punctuation marks are added as one symbol. All digits

are also added as one symbol. The reason for this is that most text entry methods use special techniques for entering punctuation marks and digits. It is therefore not necessary to have a language model that can predict which digit or punctuation mark the user will write. If a text entry method needs the individual probabilities for each digit and punctuation marks, they can easily be added to the character set. The cost of extending the character set will mainly be larger storage requirements.

5.4 Evaluation and parameter estimation

To estimate parameters and evaluate the performance of YourText a number of simulations are conducted. The simulations are made with a corpus of mobile text messages. The message corpus contains more than 25,000 messages that were sent and received by 12 users during four weeks. The corpus can be used to simulate real mobile text communication of the 12 users.

The following configurations of YourText are evaluated:

K2000 PPM*D A single PPM*D model trained on the Danish Korpus 2000 corpus.

K2000 and short term PPM*D As described in 5.12. The optimal λ value for the short term PPM*D is found and the performance of the combined model is estimated.

K2000 and context dependent short term PPM*D As described in 5.12, but with multiple short term PPM*D models. Each model is bound to a given context. The context in this evaluation is the recipient of the message. A short term PPM*D model will be made for each recipient. The optimal λ value for the short term PPM*D is found and the performance of the combined model is estimated.

K2000, short term and context dependent short term PPM*D As above, but with an extra common short term PPM*D model, that are common for all recipients. The λ values from above are used to estimate the performance.

Empty, short term and context dependent short term PPM*D As above, but the basic long term model is not trained on any data.

SE model A single PPM*D model trained on a small corpus of mobile text messages.

SE model, short term and context dependent short term PPM*D As above but with a common short term PPM*D model and a short term PPM*D model for each recipient.

5.4.1 Methodology

The mobile text message corpus was created by collecting sent and received messages from 12 different users for 4 weeks. The users are listed in table 5.1. Details about creation of the corpus are described in appendix D.

User	Sex	Age	Phone	Sent	Received
1	M	25	Nokia E50	193	440
2	M	19	SE W910i	2.383	2.641
3	M	18	SE K800i	463	489
4	M	15	Nokia N95	679	819
5	M	32	Nokia N95 8GB	102	143
6	M	31	HTC S710	527	325
7	F	33	Nokia E65	202	195
8	M	18	SE K850i	1.874	1.912
9	M	19	Nokia E51	1.824	2.067
10	M	21	Nokia 8600	1.123	1.091
11	F	19	SE K850i	1.347	3.114
12	M	24	Nokia N95	82	335
Total				10.799	13.571

Table 5.1: Users that supplied messages to the text message corpus. Sent and received is the number of text messages the users sent during the four weeks.

5.4.1.1 Message corpus

The message corpus is divided into three parts:

Test set Messages from user 1, 6 and 10

Training set Messages from user 4, 5, 7, 9 and 12

SE set Messages from user 2, 3, 8 and 11

The SE set consists of messages from users of Sony Ericsson phones. As described in appendix D, these phones can not save the time stamp of sent

messages, so it is not possible to recreate the message history. The rest of the messages are divided into a training set and a test set. Approximately 2/3 of the messages are in the training set and 1/3 in the test set.

The training set is used for all parameter estimations. The test set is only used for the final performance evaluations.

5.4.1.2 Assessments

The perplexity is used as an assessment for the performance of YourText. Two different measurements are used:

Average perplexity The average perplexity for all messages sent by the user.

Final perplexity The average perplexity for the last 10% of all messages sent by the user.

In both cases the average is weighted by the message length. The final perplexity is necessary because YourText is adaptive. Therefore it is expected to perform better when it has learned the users writing style.

5.4.2 Results

This section holds the results for the different configurations of YourText. For each evaluated configuration, the average and final perplexity are listed. Appendix A.3 has plots of the perplexity as a function of the count of sent and received messages. The plots are useful for studying the relation between perplexity of YourText and the number of messages it has been adapted to. For all PPM*D models $\delta = 0.1$ and the character set Q is defined as in section 5.3.3.4.

5.4.2.1 K2000 PPM*D

A simple version of YourText with a single PPM*D model.

$$[\Theta_{\mathrm{K2000}}^{\lambda=1.0}]$$

The model is trained on Korpus 2000. The corpus consists of approximately 22 million Danish words. The pruning level is $\delta = 0.1$ and $\lambda = 1$.

The model is limited to 6 levels due to performance reasons. Details on the model are shown in table 5.2.

Depth	Possible histories	In corpus	Pruned	In K2000 PPM*D
1	1	1	0	1
2	37	37	0	37
3	1.369	1139	14	1.125
4	50.653	22.322	2.129	20.193
5	1.874.161	168.640	32.429	136.211
6	69.343.957	560.770	192.440	368.330

Table 5.2: Size of the K2000 PPM*D model for each level. At depth n the language model are using n-graphs. The number of possible histories is $|Q|^{n-1}$.

The perplexity of the test data is shown in table 5.3.

Perplexity	User 1	User 6	User 10	Average
Average	20.21	14.60	12.22	15.68
Final	16.65	14.00	11.43	14.02

Table 5.3: Perplexity of test data. YourText with a K2000 PPM*D model.

The perplexity is very high. This is mainly because the language model is trained with a corpus that is based on written Danish language. The messages that people write are more like spoken language. For instance the messages with the highest perplexity from each user in the test data set are: '*Pumping? Today? Five o'clock?*', '*Yep!*' and '*Og?*' (*And?* in Danish). The first message is in English, and therefore has a high perplexity. The two last messages are very common when you speak, but are rarely found in written Danish language. They are also very short and contain punctuation marks. The large difference between the language of the K2000 model and the users, shows the importance of YourText being able to adapt to the language of the users.

5.4.2.2 K2000 and short term PPM*D

In the next evaluation YourText is configured with the same K2000 PPM*D model and a short term PPM*D model. Before the configuration can be evaluated the optimal value of λ needs to be estimated. The parameter estimation of λ is done by calculating the average and final perplexity of the messages for all users in the training data set. Two different estimates are made. One where YourText only adapts to sent messages and one where it adapts to both sent and received messages. The perplexity is only calculated from the sent

messages in both cases. λ values from $1-10^{-1}$ to $1-10^{-10}$ were used. After the initial calculations, extra calculations were made around the expected optimal values of λ. Figure 5.2 shows the average perplexity and figure 5.3 shows the final perplexity for the messages from each user from the training data set.

As expected the final perplexity is lower than the average perplexity. This indicates that YourText adapts to the users' language. The variations between the users are larger for the final perplexity, but it is expected because it is calculated from fewer messages.

The adaption to received messages decreases the average perplexity for most users. The average improvement is about 0.3, but one user improves with 1.0. The similar data for the final perplexity is more unclear. By adapting to received messages, the perplexity decreases for one user, increases for two other users and does not change for the last two users.

The adaption to received messages has a positive effect on the average perplexity for all sent messages. The findings might be affected by the poor performance of the K2000 PPM*D model. By also adapting to the received messages, the short term model will capture more information fast. This is crucial for a low average perplexity. The effect on the final perplexity is more mixed, but the average of all users is neither decreasing nor increasing.

The optimal λ values are shown in table 5.4.

Optimal λ	Average Perplexity	Final Perplexity
Adapt to sent messages	$1-10^{-2.2}=0.9937$	$1-10^{-2.4}=0.9960$
Adapt to all messages	$1-10^{-2.6}=0.9975$	$1-10^{-2.8}=0.9984$

Table 5.4: Optimal λ values for the short term PPM*D model.

When the received messages are included, YourText will adapt to approximately twice as many messages. The information from a sent message will decay faster, because the model is updated twice as often. The fast decay can be neutralised by using larger λ values, because they will slow down the decay of information in the language model.

The estimates of λ values from the evaluation are larger when YourText also adapts to the received messages. The ratio between sent and received characters in the text messages for the training data set is approximately 1:1.5. In this case the information will decay 2.5 times faster when YourText is adapted to the received messages. This information can be used to calculate the relation between the λ value used to adapt to all messages (λ_{all}) and the value used to adapt only to sent messages (λ_{sent}). If the information from sent messages should

decay at the same pace, the two λ values should be equal after n characters have been sent:

$$\lambda_{\text{all}}^{n \cdot 2.5} \approx \lambda_{\text{sent}}^{n} \Rightarrow \lambda_{\text{all}}^{2.5} \approx \lambda_{\text{sent}} \tag{5.15}$$

The estimated λ values fit perfectly in the equation. The values from the average perplexity are $0.9975^{2.5} = 0.9938 \approx 0.9937$ and $0.9984^{2.5} = 0.9960 \approx 0.9960$ for the final perplexity.

Based on the evaluation it is decided to adapt to both sent and received messages. This gives the best overall performance. $\lambda = 0.9980$ will be used. This is the average of the optimal λ values for the average and final perplexity. The configuration of YourText is:

$$[\Theta_{\text{K2000}}^{\lambda=1.0} \bullet \Theta_{\text{short term}}^{\lambda=0.9980}]$$

Table 5.5 shows the perplexity of the test data when $\lambda = 0.9980$ is used as the decay factor.

Perplexity	User 1	User 6	User 10	Average
Average	4.96	4.86	4.31	4.71
Final	4.20	4.79	4.13	4.38

Table 5.5: Perplexity of test data. YourText with a K2000 PPM*D model and short term PPM*D model.

5.4.2.3 K2000 and context dependent short term PPM*D

The common short term model of YourText is replaced by a context dependent short term model. The contextual dimension is the recipient of the message. Each recipient will have unique short term model. It is assumed that all the short term models have the same λ value. It is estimated the same way as in the previous section. Figures 5.4 and 5.5 show the results.

Both the average and final perplexity is lower when YourText also is adapted to the received messages. The optimal λ values are shown in table 5.6.

The final perplexity is almost the same for $\lambda = 0.9975$ and $\lambda = 0.9960$. The optimal solution is to adapt YourText to both sent and received messages and to use $\lambda = 0.9960$ as decay factor. The configuration of YourText is:

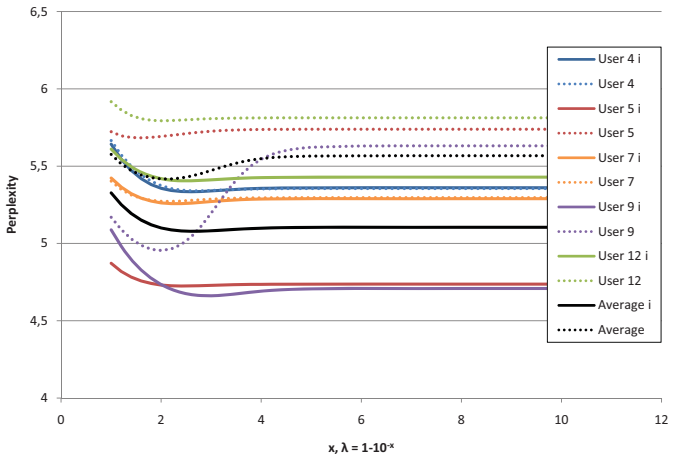


Figure 5.2: Perplexity of messages from users found by YourText with a K2000 and short term PPM*D language model. Different values of λ in the short term PPM*D model have been evaluated. The dotted lines are the perplexity if only sent messages are used to train the short term model. The solid lines are the perplexity if both sent and received messages are used to train the short term model.

$$[\Theta_{\mathrm{K2000}}^{\lambda=1.0} \bullet \Theta_{\mathrm{recipient(i)}}^{\lambda=0.9960}]$$

The index in recipient(i) means that each recipient will have a separate PPM*D model.

Table 5.7 shows the perplexity of the test data when $\lambda = 0.9960$ and each recipient has its own PPM*D model. The performance is worse than with the common short term model. It is very likely that this is due to the poor performance of the K2000 PPM*D model. When separate short term models are used for each recipient it requires more messages before the performance of the short term model can compensate for the K2000 model.

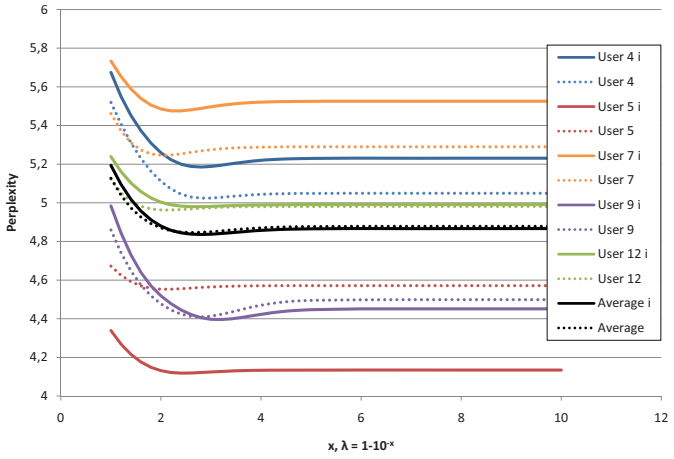


Figure 5.3: Perplexity for the last 10% of all messages from users found by YourText with a K2000 and short term PPM*D language model. Different values of λ in the short term PPM*D model have been evaluated. The dotted lines are the perplexity if only sent messages are used to train the short term model. The solid lines are the perplexity if both sent and received messages are used to train the short term model.

5.4.2.4 K2000, short term and context dependent short term PPM*D

This evaluation was done with a YourText configuration with both a context dependent short term PPM*D model and a common short term PPM*D model. The λ values found earlier are used:

$$[\Theta_{\text{K2000}}^{\lambda=1.0} \bullet \Theta_{\text{short term}}^{\lambda=0.9980} \bullet \Theta_{\text{recipient(i)}}^{\lambda=0.9960}]$$

The perplexity for this configuration can be seen in table 5.8. The performance is better than the previous configurations where either a common or context dependent model were used.

It is likely that better λ values can be found, if both values are estimated at the same time.

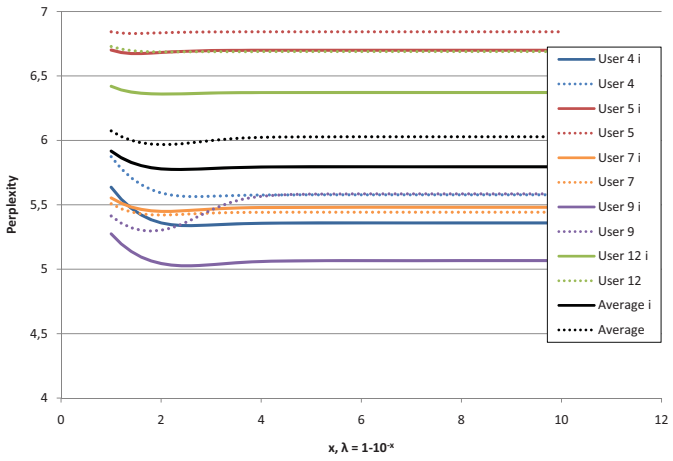


Figure 5.4: Perplexity of messages from users found by YourText with a K2000 and context dependent short term PPM*D language models. Different values of λ in the context dependent short term PPM*D models have been evaluated. The dotted lines are the perplexity if only sent messages are used to train the short term models. The solid lines are the perplexity if both sent and received messages are used to train the short term model.

5.4.2.5 Empty, short term and context dependent short term PPM*D

As mentioned in section 2.2.4 it can be costly to create language models for all possible languages. With adaptive language models, it is possible to put an empty language model in a product, and let the language model learn from the user. To see how YourText performs when it has no prior knowledge of the language, a new configuration is made with an empty long term PPM*D model. The model has $\lambda = 1$ and $\delta = 0.1$. Besides the long term model, YourText is configured with a common short term model and context dependent short term models with the same λ values as above:

$$[\Theta_{\text{Epmty}}^{\lambda=1.0} \bullet \Theta_{\text{short term}}^{\lambda=0.9980} \bullet \Theta_{\text{recipient(i)}}^{\lambda=0.9960}]$$

The results in table 5.9 show that the perplexity increases with about 1 when no prior knowledge are used.

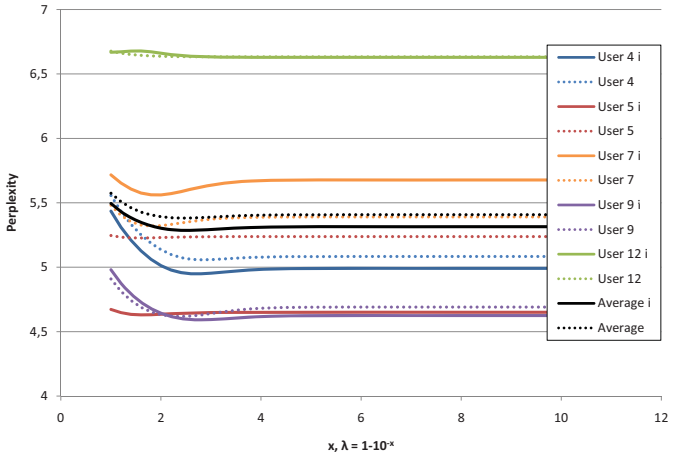


Figure 5.5: Perplexity for the last 10% of all messages from users found by YourText with a K2000 and context dependent short term PPM*D language models. Different values of λ in the context dependent short term PPM*D models have been evaluated. The dotted lines are the perplexity if only sent messages are used to train the short term models. The solid lines are the perplexity if both sent and received messages are used to train the short term models.

5.4.2.6 SE PPM*D

The empty PPM*D model did not perform very well. Another evaluation was made to see how well a language model trained on a small corpus will perform. Small corpora are easier to create. If the performance is acceptable then it will be a good solution for real world usage. Especially if language models for many languages are needed. The small corpus was made from the SE data set. The SE data set contains 768,000 characters, or about 168,000 words. This is less than 1% of the number of words in Korpus 2000 used in the other evaluations. The PPM*D model trained on the SE data set will be called SE PPM*D:

$$[\Theta_{\mathrm{SE}}^{\lambda=1.0}]$$

The first evaluation was made with a YourText configuration with a single SE

Optimal λ	Average Perplexity	Final Perplexity
Adapt to sent messages	$1 - 10^{-2.0} = 0.9900$	$1 - 10^{-2.4} = 0.9960$
Adapt to all messages	$1 - 10^{-2.4} = 0.9960$	$1 - 10^{-2.6} = 0.9975$

Table 5.6: Optimal λ values for the context dependent short term PPM*D models.

Perplexity	User 1	User 6	User 10	Average
Average	5.79	5.75	4.61	5.38
Final	5.79	5.46	4.24	5.16

Table 5.7: Perplexity of test data. YourText with a K2000 PPM*D model and a context dependent short term PPM*D model.

PPM*D model. The results in table 5.10 show the performance. Compared to the similar configuration from section 5.4.2.1, where K2000 is used, the SE model performs a lot better. In general a larger corpus should perform better. The result shows that the language used in Korpus 2000 is very different from what the participants wrote in their messages. It is likely that this is due to the reasons mentioned in section 5.1.

5.4.2.7 SE, short term and context dependent short term PPM*D

To evaluate the SE PPM*D model in a more advanced configuration of YourText, common short term and context dependent short term PPM*D models are added to the configuration. This configuration is similar to that of section 5.4.2.4, except that the basic model is trained on the SE data set instead of Korpus 2000.

$$[\Theta_{\mathrm{SE}}^{\lambda=1.0} \bullet \Theta_{\mathrm{short\,term}}^{\lambda=0.9980} \bullet \Theta_{\mathrm{recipient(i)}}^{\lambda=0.9960}]$$

The results are shown in table 5.11. The performance is not as good as the similar configuration where the K2000 PPM*D model is used as a basic model (table 5.8).

Perplexity	User 1	User 6	User 10	Average
Average	4.38	4.64	4.13	4.38
Final	3.99	4.51	3.90	4.13

Table 5.8: Perplexity of test data. YourText with a K2000 PPM*D model, common short term PPM*D model and a context dependent short term PPM*D model.

Perplexity	User 1	User 6	User 10	Average
Average	5.24	6.23	5.24	5.57
Final	4.74	5.58	4.56	4.96

Table 5.9: Perplexity of test data. YourText with an empty PPM*D model, a common short term PPM*D model and a context dependent short term PPM*D model.

5.5 Discussion

The results from the evaluation of YourText are affected by the poor performance of the K2000 PPM*D model. It makes it more difficult to compare the different configurations of YourText. Table 5.12 summarise all the evaluated configurations.

One of the interesting findings is the relation between the K2000 and SE PPM*D model. If YourText only is configured with one of these basic models, then the SE model performs a lot better than the K2000 model. The perplexity is reduced with 40% - 45% by using the SE model. Performance of both configurations improves when context dependent and common short term PPM*D models are added to the configurations. For the K2000 model the perplexity drops with 70%. The perplexity for the SE model drops with 45%. Adding the extra models to the configurations makes the K2000 configuration perform best. The YourText configuration with a K2000 model has a 5% - 8% lower perplexity than the configuration with the SE model.

These findings indicate that there is a subset of words that are more frequently used in text messages than in the general language. The SE model included these words, and therefore performed better than the much larger K2000 model. When context dependent and common short term PPM*D models are added to the configurations, both configurations adapt to the users' language. The adaption improves both configuration, but is especially successful for the K2000 configuration. After the K2000 configuration have adapted to the language, it outperforms the SE configuration.

Perplexity	User 1	User 6	User 10	Average
Average	10.44	10.57	6.77	9.26
Final	7.37	9.70	6.37	7.81

Table 5.10: Perplexity of test data. YourText with a PPM*D model trained on the SE data set.

Perplexity	User 1	User 6	User 10	Average
Average	4.61	5.27	4.33	4.74
Final	4.00	4.95	4.05	4.33

Table 5.11: Perplexity of test data. YourText with a SE PPM*D model, a common short term PPM*D model and a context dependent short term PPM*D model.

The SE and Test data set have a lot in common, but many words from the Test data set do not occur in the SE set. Most of these words are found in the much larger Korpus 2000. This is the reason for the good initial performance of the SE model, and the improved performance of K2000 configurations when they have been adapted to the language. A PPM*D model trained on both Korpus 2000 and the SE data set is expected to give a good initial and long term performance.

The inclusion of context dependent short term PPM*D models gives a 5% - 7% reduction in perplexity, compared to similar configurations with out these models. The recipient was the only evaluated contextual dimension. Other dimensions might give a larger reduction. The price of adding context dependent short term models is mainly storage space.

The configuration of YourText with an empty, short term and and context dependent short term PPM*D models have a 20% higher final perplexity than the similar K2000 configuration. The average perplexity is increased with 27%. In some cases the drop in performance can be justified from a business point of view. If the users can cope with the poor initial performance it might be a good solution. The risk is that many users will stop using the text entry method before the performance improves. The users from the test data set had sent 193, 527 and 1123 messages. The final perplexity is calculated on the last 10% of the messages, meaning that the performance of the first 90% will be worse. If you write few messages, then it will take months or even years before the performance is acceptable.

YourText with an empty PPM*D model can only be used for some text entry methods. Word levels methods, that require disambiguation or word completion

YourText configuration	Average perplexity	Final perplexity
$[\Theta^{\lambda=1.0}_{\text{K2000}}]$	15.68	14.02
$[\Theta^{\lambda=1.0}_{\text{K2000}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}}]$	4.71	4.38
$[\Theta^{\lambda=1.0}_{\text{K2000}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	5.38	5.16
$[\Theta^{\lambda=1.0}_{\text{K2000}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	4.38	4.13
$[\Theta^{\lambda=1.0}_{\text{Epmty}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	5.57	4.96
$[\Theta^{\lambda=1.0}_{\text{SE}}]$	9.26	7.81
$[\Theta^{\lambda=1.0}_{\text{SE}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	4.74	4.33

Table 5.12: Overview of evaluated YourText configurations

will not work. When no or little information is stored in the YourText model, then the algorithms from section 5.3.3 will fail. The algorithms will still be able to create list of candidate words or word completion candidates, but the lists will mostly contain non dictionary words.

5.5.1 More advanced configurations of YourText

The evaluated configurations of YourText are all very simple. It is possible to create many more advanced configurations. One method is to use more contextual information. In the evaluations the recipient was the only used contextual information. Other information could be time of day, location, and etcetera. There are a number of challenges associated with adding extra language models to the pool. An important problem is how to define and group the contextual dimensions. A sound evaluation of a YourText configuration will also be more difficult if the number of contextual dimensions is large. There are also more technical problems, such as storing a large number of language models.

5.5.2 Better pruning of PPM*D models

The storage problem can partly be solved by using better pruning algorithms. The current adaption and pruning of PPM*D models is done on each individual model. When YourText has been used for a while there will be a lot of redundant information in the different PPM*D models. All models have been adapted to the same common words. A part of the redundant information can be removed by pruning PPM*D models with respect to other PPM*D models. The pruning criteria is identical to original criteria in equation 5.7, but the probabilities should be calculated from a set of PPM*D models. The probability estimate

should be calculated like the overall YourText estimate from equation 5.13. If some words have high probability in a basic PPM*D model, then the information gain of adding them to another PPM*D model will be very small.

There is one important criterion that needs to be fulfilled. The set of language models used when pruning a model must also be available when the model is used. Otherwise important information might get lost. Language models that decay fast (low λ value) should not be used when a model is pruned. If a model for a single recipient is pruned in respect with a short term model, then many words will be removed from the recipient model. Next time the recipient model is used, the short term model might have changed and valuable information has been lost. The best solution is to prune short term PPM*D models with respect to long term models.

5.5.3 Challenges with context dependent language models

One of the biggest challenges with the context dependent configurations of YourText is how to find and group the contextual dimensions. The most important criteria is that contextual information is available and that there are some variations in the information. As an example, not all mobile devices are able to retrieve their current location. A desktop computer might know its current location, but since it is fixed the information is not very useful.

A good contextual dimension needs to be correlated with the user's language. Otherwise YourText will not benefit from including the dimension in a configuration. The best way a to find the added value of a new contextual dimension is to evaluate the dimension. It can be done as in the previous section, where the recipient dimension is evaluated.

For recipients there exists an easy one-to-one match, where each recipient gets a separate language model. For contextual dimensions like location or time it is much mode difficult. The values of these dimensions are continuous and cannot be used directly. They have to be converted to discrete values, by assigning the different values to groups. For the time dimension, the groups could be the weekdays, seasons (winter, spring, summer, autumn), time of day (morning, noon, afternoon, evening, night) or whole hours (0-23). The location dimension could have groups defined by the nearest city. When having discrete values, a language model can be created for each value.

If a group is too large, then the language model for the group might be adapted to many different contexts. If the location dimension is grouped by nearest city, then it will be a problem if you are working, shopping and going out in the same

city. Similar, a group that is too small, will not hold enough information to be useful.

One solution is to create many small groups, and then cluster the language models by a similarity measurement. If this is used on the recipient dimension, then all language models from the recipient dimensions will be clustered by a clustering algorithm. When a user writes a message to one recipient, then all language models in the cluster are used to calculate the probability estimates.

Another solution is mentioned by Clarkson and Robinson [1997]. They used a weighted set of language models where the weights were continuously updated, based on how well each language model described the previously seen text.

5.5.4 Additional contextual dimensions

Some devices will hold additional contextual dimensions that might be very useful for YourText. A mobile phone with an updated calendar can tell whether the user is in a meeting, on holiday or something else. In some phones the contact list holds information about the user's relation to the contacts. It can be very relevant to know whether a contact is a co worker, friend or family member.

5.5.5 Unified language model

One of the advantages of YourText is that it can be used with many different text entry methods. This means that the language model can be transferred across devices. Both from old to new devices, but also across multiple devices that are being used at the same time. Using the same language model in multiple devices means that the language model can adapt to more text. This will improve the quality of the model. It might be a good idea to include the device type as an additional contextual dimension. This will allow the users to have different language usage for different devices.

5.6 Conclusions

A new language model framework YourText was introduced. It uses a combination of PPM*D models to estimate character probabilities. PPM*D

is an unbounded PPM model, that is extended with a decay function. It will adapt to new words, but will also forget words that are not used any more. YourText can be configured in many ways to be aware of the user's context. The contexts could be the recipient of the text, the user's location, time of day, etcetera.

Different configurations of YourText were evaluated. Text messages from a corpus were used to simulate real usage of YourText. The PPM*D model adapted to the users' language and is able to model the higher probability of recently used words. Context-aware configurations where each recipient has an additional private PPM*D model is found to perform better than configurations with only common PPM*D models. Adapting to both sent and received messages performs better than only adapting to sent messages. YourText is only evaluated for Danish, but can be used with all languages that are using the Roman alphabet.

Chapter 6

Modelling performance of language models and text entry methods

A language model was used as a high level task model for the TUP text entry method. Replacing the language model with an improved language model is expected to improve TUP. This chapter will show how to model the text entry rate for TUP as a function of the predictions from the language model. Different configurations of YourText are evaluated to estimate the text entry speed of TUP, if used with these language models.

6.1 Text entry performance models

It is possible to model performance of text entry methods. Often the Keystroke-Level model by Card et al. [1980] or Fitts' law [Fitts, 1954] are used. The keystroke-level model estimates the performance of a task by summing the time of all low level operations needed to complete the task. Fitts' law models the time it takes to complete rapid aimed movements.

Most text entry models are used to estimate performance of new methods or variants of known methods. The optimized character layouts from section 2.2.1 are all found by using models to evaluate the alternative layouts. Models of text entry performance often include language models. Silfverberg et al. [2000] combines a digraph language model and Fitts' law to model the performance of mobile text entry methods. Fitts' law is used to model the movement time from key to key and the digraph model is used to find the distribution of all possible movements between keys.

6.2 Model of TUP performance

The model of TUP will be used to estimate performance of TUP when it is used with different language models. The quality of language models is an important factor in all text entry methods that are using language models. Poor language models can lead to very bad performance. The goals of the model are to:

- Estimate the performance of TUP without doing another evaluation. All the implementations of TUP in the thesis are using the same simple language model. This makes it easier to compare the different evaluations. The effects of improving the algorithms in TUP are found in section 4.5. This model will find the effects of using an improved language model.
- Understand the performance of the different configurations of the YourText language model. In section 5.4, the perplexity is used a assessment of the performance of YourText. The perplexity is good to rank different configurations, but it is difficult to understand how a reduction in perplexity will affect the performance of a text entry method. By using both perplexity and TUP performance, it is easier to understand the difference between the YourText configurations.

6.2.1 Creating a model

The model should be able to estimate the text entry speed of TUP for a given phrase. The data from the improved TUP iPod evaluation will be used to estimate the performance. By filtering the data based on session number, it is possible to model either novice or more experienced performance. The model in this section will use data from session 10-12. This will model performance of users with 90-120 minutes of experience.

The text entry speed is estimated in WPM. The $T(\alpha_i|\boldsymbol{\alpha}_{1..i-1})$ function estimates the time to enter α_i in milliseconds. When writing a phrase $\boldsymbol{\alpha} = \alpha_1..\alpha_n$ with TUP, using the language model Θ, the text entry speed can be estimated as:

$$\mathrm{WPM}(\boldsymbol{\alpha}, \Theta) = \frac{n}{\sum_{i=1}^{n} T(\alpha_i|\boldsymbol{\alpha}_{1..i-1})} \cdot 60,000 \cdot \frac{1}{5} \tag{6.1}$$

The calculation of WPM is done as in equation 3.1. The time to enter a character is estimated as the time to complete three operations:

$$T(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) = T_{\mathrm{prepare}}(\alpha_i) + T_{\mathrm{select}}(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) + T_{\mathrm{confirm}} \tag{6.2}$$

T_{prepare} is the time it takes to prepare to enter a new character. It is calculated as the elapsed time between the previous character is entered and the finger touches the wheel again. This is called preparation time in section 4.3.3.3 and 4.3.3.3. In figure 4.13 and 4.31 the preparation time is labelled **finger off wheel**.

T_{select} is the time it takes to select the target character. It is the time from the initial touch of the wheel until the target character is highlighted. T_{confirm} is the time it takes to confirm the highlighted character. It is calculated from the time the target character is highlighted until the finger is removed from the wheel. In figure 4.13, 4.18, 4.31 and 4.33 T_{select} is called **finger on wheel** and T_{confirm} is called **finger on character**.

6.2.1.1 Estimate of T_{prepare}

The preparation time is mainly dependent on the target character. The probability of the target character or the previous character does not have any great effect on the time. T_{prepare} is therefore estimated from the observations from the evaluation. The preparation times can be seen in figure 4.32. Only observations from session 10-12 in the improved TUP iPod evaluation are used.

6.2.1.2 Estimate of T_{select}

T_{select} is estimated from the average time the participants touched the wheel before the target character was highlighted. It is only dependent on the probability of the target character. Characters with high probability are easier to highlight and have lower T_{select} values.

The observations are approximated with different function. The fifth order polynomial in equation 6.3 is found to give the best fit. Very few observation have character probabilities larger than 0.7 (equation 6.4). This makes the approximation unstable in this area. To make the estimate more stable, all character probabilities will be limited to 0.7. Figure 6.1 shows the observations, the approximate and the final estimate.

$$\begin{gathered} T_{\text{select}}(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) = \\ -14,185p^5 + 38,478p^4 - 39,433p^3 + 19,204p^2 - 4645.2p + 531.9, \\ p = P(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) \end{gathered} \tag{6.3}$$

$$P(\alpha_i|\alpha_{1..i-1}) = \begin{cases} \Theta(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) & if \quad \Theta(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) < 0.7 \\ 0.7 & if \quad \Theta(\alpha_i|\boldsymbol{\alpha}_{1..i-1}) \geq 0.7 \end{cases} \tag{6.4}$$

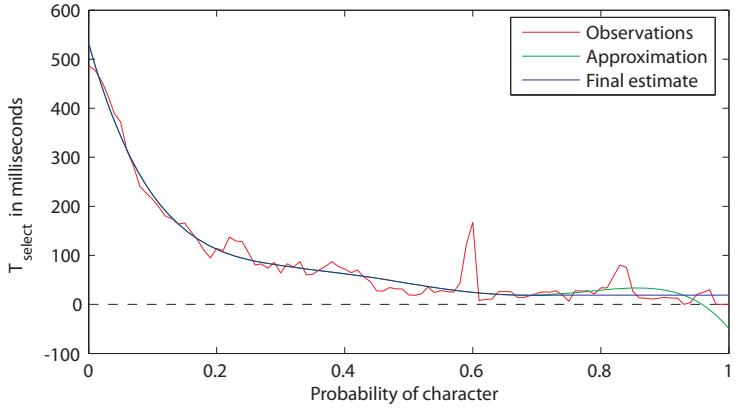


Figure 6.1: Observed values for T_{select}, approximated values and the final estimate.

6.2.1.3 Estimate of T_{confirm}

The time to confirm the highlighted character is independent of the character and the probability. T_{confirm} is estimated as the average of the observation from session 10-12:

$$T_{\text{confirm}} = 442\, milliseconds \tag{6.5}$$

6.2.2 Limitations of the model

There are some limitations of the model. The model does not include the time to correct errors. The model will therefore estimate error free performance. It is possible to extend the model to take the error rate into account. Either by using a fixed error rate or by estimating the error rate from the text and language model. To do this, the relation between probability of target characters and error rates needs to be investigated. This has not been done in the thesis.

The other limitation is in the handling of character sets. The model is based on observations from the TUP iPod evaluation. The prototype from the evaluation uses a limited character set with only 32 characters. The model is extended so it can estimate text entry speed of text written with other character sets. T_{select} and T_{confirm} are not changed. T_{prepare} is only defined for the characters used in the evaluation. It is extended so T_{prepare} for all new characters will be estimated as the average T_{prepare} time from the evaluation. The validity of the extended model is largest if the size of the new character set is close to the size of the character set from the evaluation.

6.3 Validation of TUP performance model

The model is evaluated by estimating the performance of the set of 500 phrases from MacKenzie and Soukoreff [2003]. The language model from equation 4.5 is used. Both language model and phrases are identical to what have been used in the TUP iPod evaluations.

The perplexity of the language model and phrases is 7.7162. The text entry speed is found to be 9.89 WPM. This is slightly higher than the observed text entry speed of session 10-12, which are 9.83 WPM. A bigger difference was expected because the model estimates error free performance. The corrected error rate for session 10-12 is 0.039.

6.4 Evaluation of YourText configurations

The performance of TUP is estimated when used with the different YourText configurations. The configurations and parameters are taken from section 5.4. The configurations are evaluated with the test data set from the message corpus. All text is in Danish. The results are shown in table 6.1.

YourText configuration	Average WPM	Final WPM
$[\Theta^{\lambda=1.0}_{\text{K2000}}]$	10.02	10.02
$[\Theta^{\lambda=1.0}_{\text{K2000}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}}]$	10.41	10.42
$[\Theta^{\lambda=1.0}_{\text{K2000}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	10.32	10.33
$[\Theta^{\lambda=1.0}_{\text{K2000}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	10.45	10.49
$[\Theta^{\lambda=1.0}_{\text{Epmty}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	10.19	10.26
$[\Theta^{\lambda=1.0}_{\text{SE}}]$	10.07	10.09
$[\Theta^{\lambda=1.0}_{\text{SE}} \bullet \Theta^{\lambda=0.9980}_{\text{short term}} \bullet \Theta^{\lambda=0.9960}_{\text{recipient(i)}}]$	10.42	10.46

Table 6.1: Overview of evaluated YourText configurations.

TUP is faster with YourText than with the phrases and language model used in the validation. This is even the case for the K2000 PPM*D configuration that has an average perplexity of 15.68. This is due to the different distributions of used characters. The messages from the test data set have more characters with low preparation time.

The biggest difference between the evaluated YourText configurations is 5.0%. It is likely that the difference will be larger if the model is extended to also model error rates.

The evaluation shows that the performance of TUP can be improved if TUP is used with better language models. The improvements are not very big, but are expected to have a positive influence on the user experience. The evaluation shows that performance of TUP will degrade slowly if the quality of the language model decreases. This is an advantage of the character based language model and text entry method. With word based methods, the performance will be severely damaged when the user writes an unknown word.

6.5 Conclusions

A model of text entry speed for TUP was introduced. It is based on observations from the improved TUP iPod evaluation. It can estimate the text entry speed with TUP for a given text and language model. The model was used to estimate text entry speed of TUP used together with different YourText configurations. The biggest difference between the evaluated YourText configurations is 5.0%.

Chapter 7

Conclusions

The need for mobile text entry is larger than ever. Many of the existent text entry methods are either slow, have high complexity or require much space. Language models are often used to increase the performance of text entry methods, at the cost of higher complexity.

7.1 Novel methods for text entry

The TUP text entry method was introduced in chapter 4. TUP is designed for use with touch sensitive wheels, sliders and similar input devices. The method is designed to be easy to use for novice users and to provide an easy transition from novice to expert. The method works on character level, so all possible words and character combinations can be written. TUP uses a combination of low level models of human motor behaviour and a language model to predict which character the user will enter.

To keep the conceptual model of TUP simple, the language model is hidden from the user. TUP has been evaluated in three evaluations. 33 users have participated in the evaluations and more than 4600 phrases have been transcribed. TUP-Key is introduced. It is a variant of TUP for use with

keyboards. TUP-Key is not implemented or evaluated. Both TUP and TUP-Key can be used with different sizes of input devices. If TUP has more space, the density of characters will be smaller. This will make it easier to highlight the correct characters. For TUP-Key the size and number of keys can be changed to use all the available space.

7.2 Optimizing motor behaviour models

The low level models of motor behaviour were optimized by using methods from statistical learning and data from an evaluation of TUP. Combinations of high level data of the text entry sessions and low level logs of motor behaviour were used. New functions were fitted to the observations, and used to improve the algorithms in TUP. An evaluation of the improved version of TUP showed that the text entry speed had increased 29% and the error rate has dropped from 8% to below 5%.

7.3 Novel language models

Both TUP and TUP-Key are using language models to improve performance. The language models work at character level so all possible words and character combinations can be written. The methods support adaptive language models, so the language model can learn from the users. It could be the YourText language model introduced in chapter 5. It is a context-aware adaptive language model. It uses a combination of PPM*D models to estimate character probabilities. PPM*D is a unbounded PPM model, that is extended with a decay function. It will adapt to new words, but will also forget words that are not used any more. YourText can be configured to be aware of the user's context. The contexts could be the recipient of the text message, the user's location, time of day, and etcetera.

Simulations of adaptive text entry with YourText were made to find the performance of different configurations. Having an additional private PPM*D model for each recipient was found to decrease perplexity. Adapting to both sent and received messages were found to work better than only adapting to sent messages. In chapter 6 a model of text entry speed for TUP was used to estimate text entry speed of TUP used together with different YourText configurations. Improving the language model was found to improve text entry speed with 5.0%. The performance model did not include the language model's

effect on the error rate. It is expected that a better language model will reduce the error rate, thereby increasing performance.

7.4 Assessment of TUP and TUP-Key

The final estimate of text entry speed for TUP with YourText is 10.5 WPM. This is lower than the reported speed of many other mobile text entry methods. Most of these other methods have only been evaluated in labs. The evaluations of TUP were done in the users' natural environment. It is likely that a higher text entry speed can be observed if the evaluations are repeated in a lab, but the estimate will not be very realistic for everyday mobile use. It is also likely that the performance of TUP will be higher, if the evaluation was done in the participants' native languages. TUP-Key has not been evaluated, but the text entry speed is expected to be similar to T9, which is 25 WPM for experienced users. The performance of TUP-Key is expected to be better than T9 when writing words that are not present in the T9 dictionary.

TUP and TUP-Key does not have problems with entering unknown words, which can be troublesome with many other text entry methods. All words and combinations of characters can be entered with the same text entry method. If used together with YourText, it will automatically adapt to all entered text. As YourText adapts to the user's language, text can be entered with less effort.

7.5 Unified text entry

Many users have multiple mobile devices and change devices frequently. Today each device has it own language model. Each time the user buys a new device, all language model adaptations will be lost. This is likely to result in poor initial performance and user experience of the new device. To be able to improve performance and user experience, it should be possible to transfer skills and language models across devices. Both from old to new devices, but also across multiple devices that are being used at the same time.

TUP and TUP-Key is well suited to be used on a range of different devices. TUP can be used on small devices where there is no room for keys, or on devices where keys cannot be used because of aesthetically or environmental requirements. TUP-Key can be used with all key based devices. By combining TUP and TUP-Key with a YourText language model, the user will get easy to

use, context-aware, and adaptive text entry methods. If the language model is shared among devices, the adoption will occur faster and leads to increased performance and user experience.

Increased performance and user experience of text entry have a great influence on the usability and user experience of mobile applications and services. Better text entry methods will empower the users and enable them to better communicate and share information through mobile devices.

Appendix A

Extra plots and figures

This section holds extra plot and figures.

A.1 TUP evaluations

The plots show the strategies used by each user in each session. There are three lines for each user. Each line represents a strategy. The strategies are: Tap direct (blue), Scroll fixed (green) and Scroll latest (red). The line at top shows the dominant strategy for the session. See section 4.3.3.3 for details. Figure A.1 shows the strategies for the first TUP iPod evaluation and figure A.2 shows the strategies for the improved TUP iPod evaluation.

A.2 Improvements to TUP

This section contains extra plots from section 4.4.

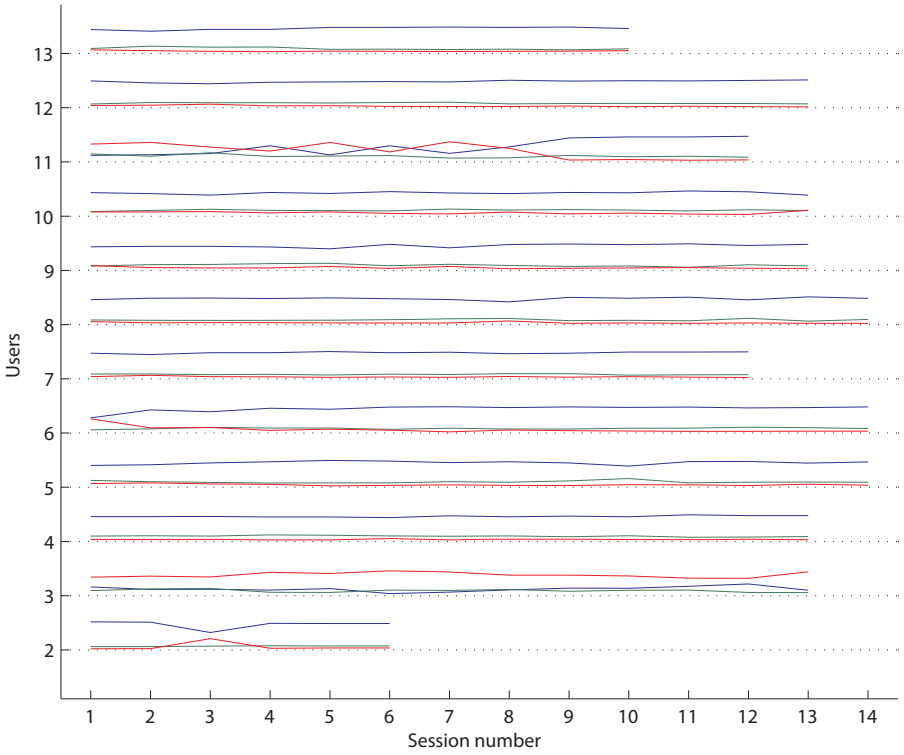


Figure A.1: Strategies for highlighting characters for the users in the first TUP iPod evaluation. The strategies are: Tap direct (blue), Scroll fixed (green) and Scroll latest (red).

A.2.1 Plots of $\varphi_\alpha - \varphi$

The plots in figure A.3 shows the distribution of $\varphi_\alpha - \varphi$ for different features and values of β.

A.2.2 Optimized slip error correction algorithm

Plots of error rates for different classifiers for the slip error correction algorithm. The p value is from the loss matrix in equation 4.24. The classifiers are: linear (figure A.4), linear discriminant (figure A.5), logarithmic discriminant (figure A.6) and quadratic discriminant (figure A.7).

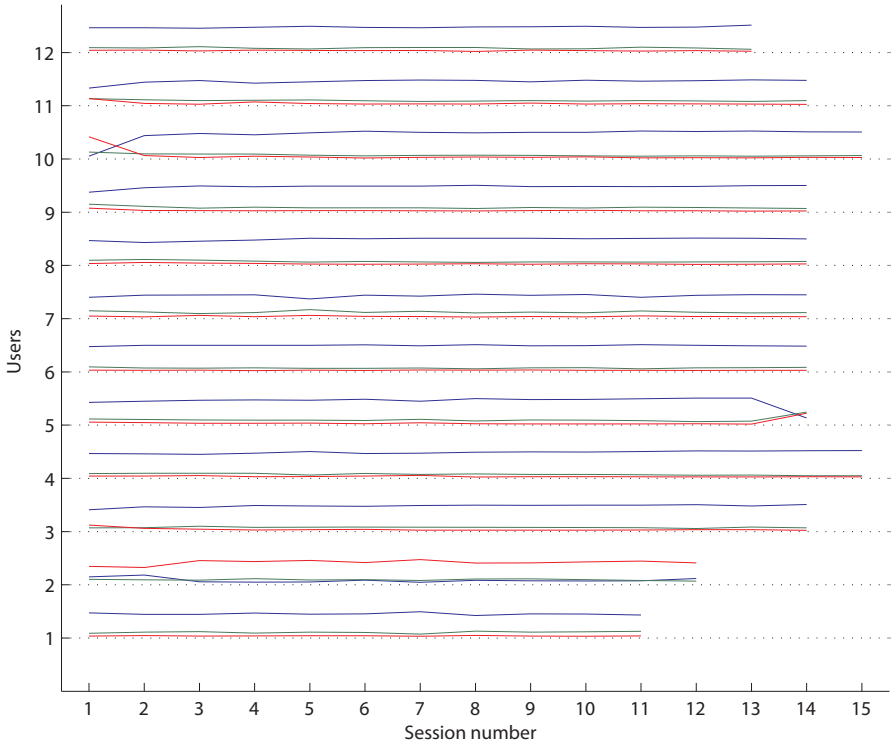


Figure A.2: Strategies for highlighting characters for the users in the second TUP iPod evaluation. The strategies are: Tap direct (blue), Scroll fixed (green) and Scroll latest (red).

A.3 Evaluation of YourText

Plot from evaluation of YourText in section 5.4. The *Perplexity* is the running average perplexity of the last 20 sent messages. The *Text messages* is the number of text messages sent so far. Because YourText is adaptive, it can be seen that the perplexity will drop when more messages have been sent. This is because YourText will adapt to the users' writing style and language usage.

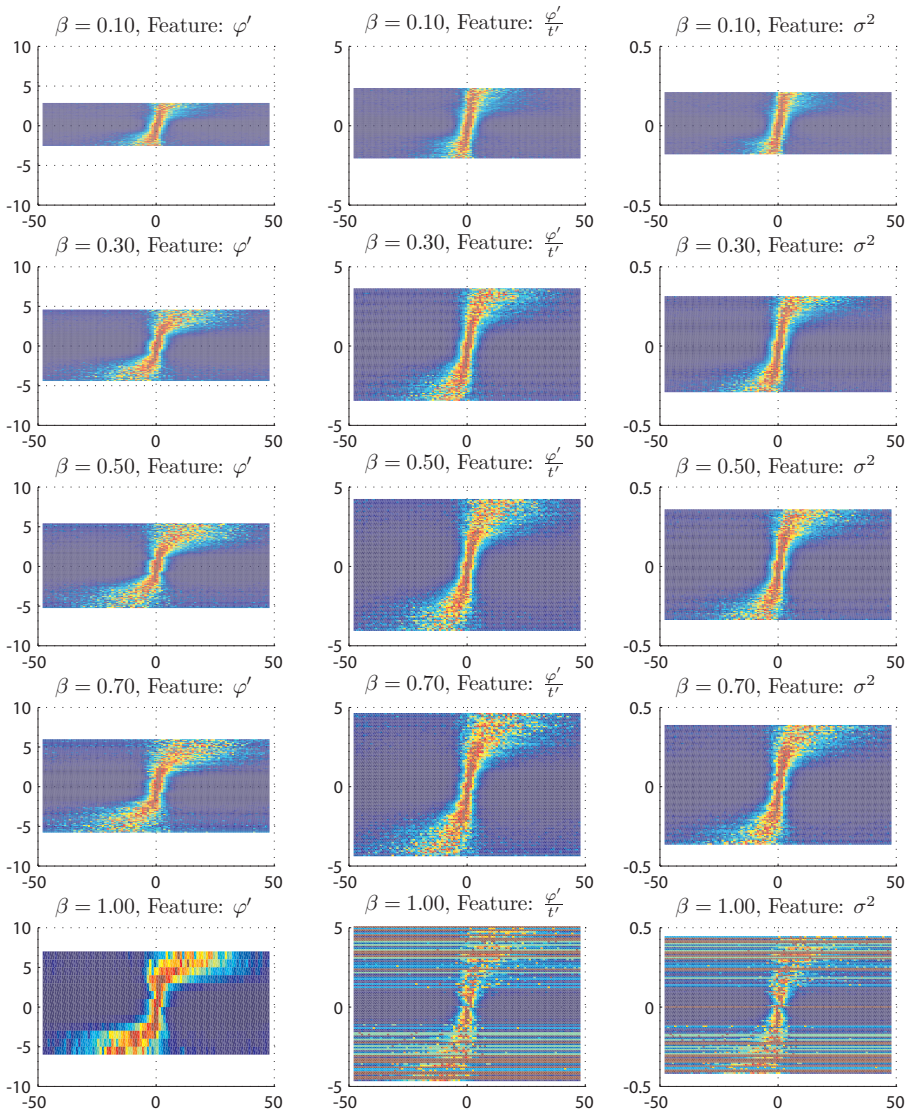


Figure A.3: Plots of $\varphi_\alpha - \varphi$ for different features and values of β. The value $\varphi_\alpha - \varphi$ is mapped to the x-axis and the y axis is the value of the feature. The data have been scaled so the maximum value in each row is 1.0. Dark blue equals 0 and dark red equals 1.

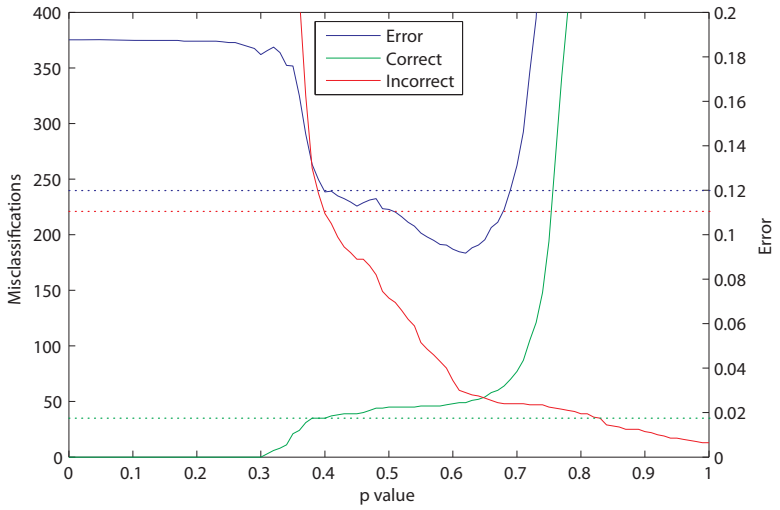


Figure A.4: Linear classification

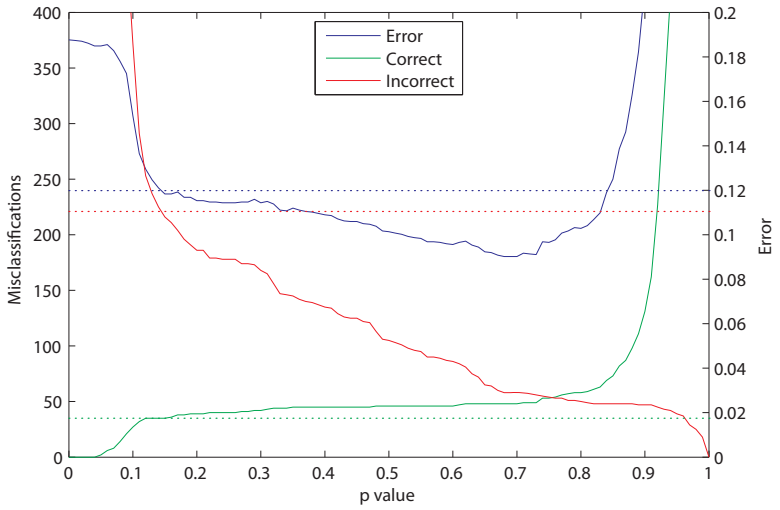


Figure A.5: Linear Discriminant Analysis

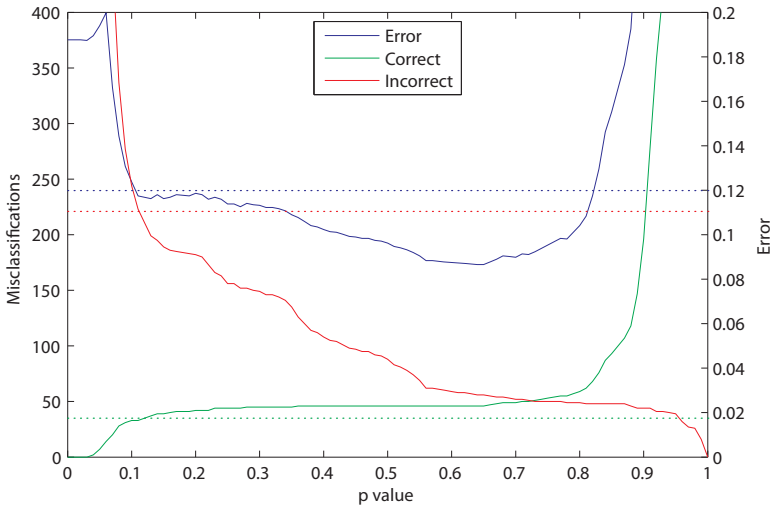


Figure A.6: Logarithmic Discriminant Analysis

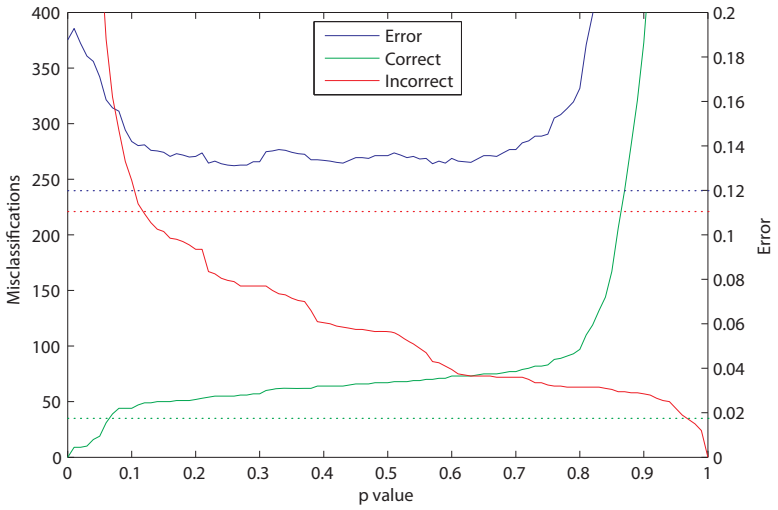


Figure A.7: Quadratic Discriminant Analysis

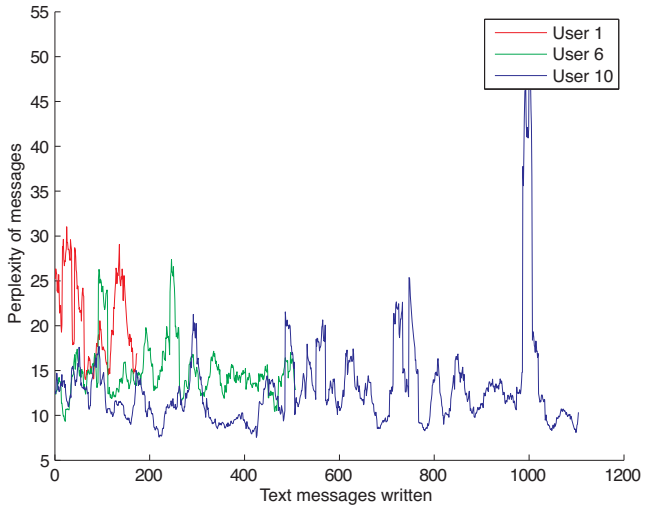


Figure A.8: Perplexity of test data. YourText with a K2000 PPM*D model trained on Korpus 2000 (section 5.4.2.1).

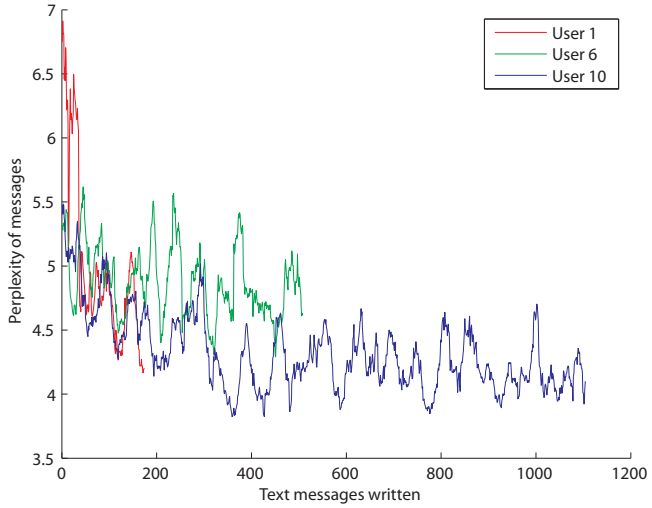


Figure A.9: Perplexity of test data. YourText with a K2000 PPM*D model and short term PPM*D model (section 5.4.2.2).

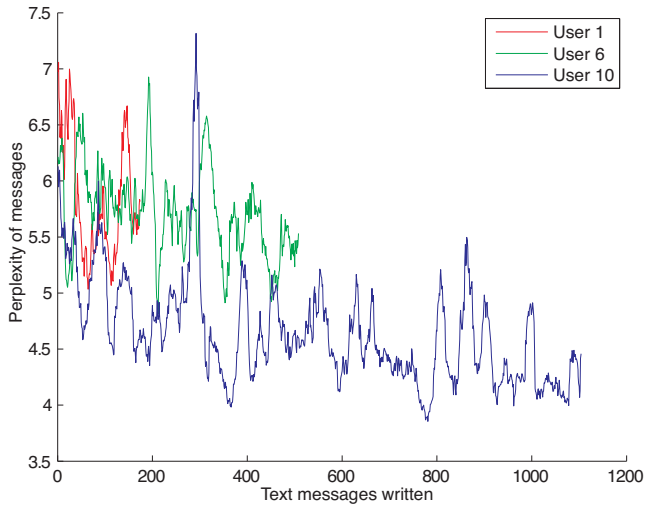


Figure A.10: Perplexity of test data. YourText with a K2000 PPM*D model and a context dependent short term PPM*D model(section 5.4.2.3).

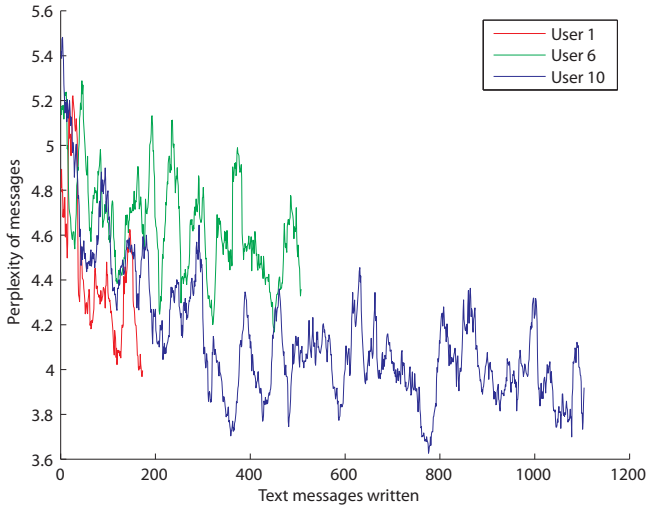


Figure A.11: Perplexity of test data. YourText with a K2000 PPM*D model, a common short term PPM*D model and a context dependent short term PPM*D model (section 5.4.2.4).

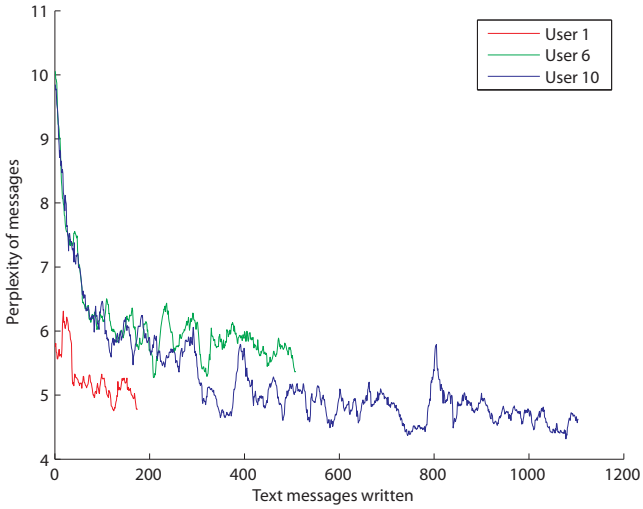


Figure A.12: Perplexity of test data. YourText with an empty PPM*D model, a common short term PPM*D model and a context dependent short term PPM*D model (section 5.4.2.5).

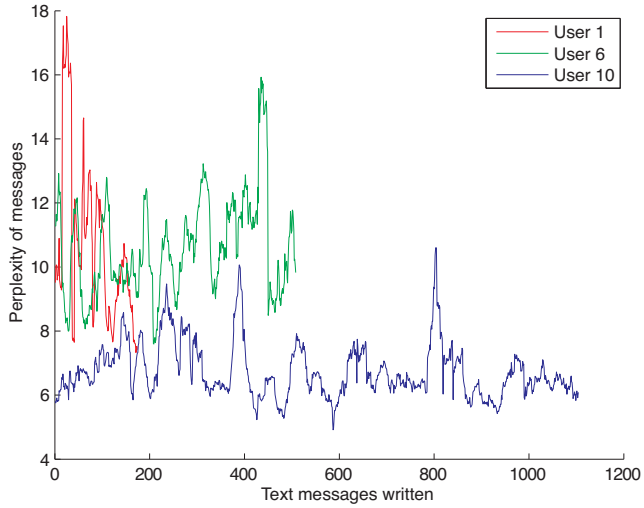


Figure A.13: Perplexity of test data. YourText with a PPM*D model trained on the SE data set (section 5.4.2.6).

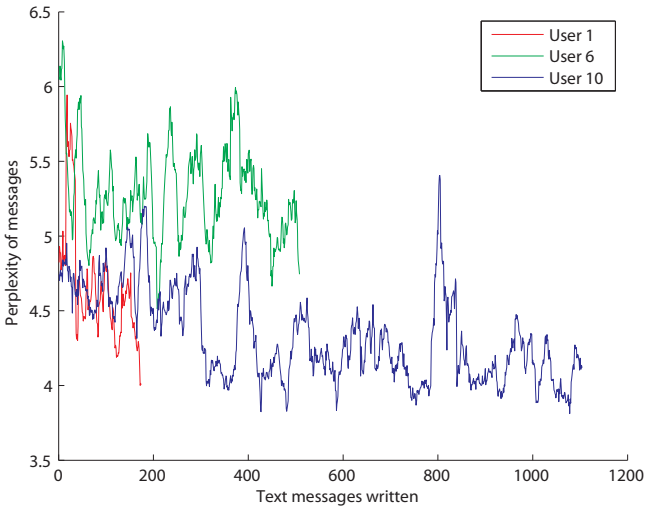


Figure A.14: Perplexity of test data. YourText with a PPM*D model trained on the SE data set, a common short term PPM*D model and a context dependent short term PPM*D model (section 5.4.2.7).

Appendix B

Fitting sigmoid functions to the observations

This section explains how a function is fitted to the observations for the optimized character highlighting algorithm in section 4.4.1.

Based on informal evaluation of different functions, it was decided to use the product of two sigmoid functions for the fitting. The function is fitted to the observations in each bin. The parameters for each function are then replaced by a maximum likelihood estimate based on linear regression.

The formula for a single sigmoid function is shown in equation B.1. a controls the slope of the curve and c controls the position of the slope. The product of two sigmoid functions (equation B.2) has four parameters. If $a_1 > 0$, $a_2 < 0$ and $c_1 < c_2$ the function will have a bell shape. An example of a sigmoid product function is shown in figure B.1. In this case where one of the dimensions are circular, the sigmoid product function needs to be modified. Equation B.3 shows the modified sigmoid product function where the function is calculated as the maximum of three normal sigmoid product functions. For two of the functions the x parameter have been displaced with a full rotation in either clock wise or counter clock wise direction. Figure B.1 shows both the normal sigmoid product function and the circular sigmoid product function.

$$\text{sigmoid}(x, a, c) = \frac{1}{1 + e^{-a(x-c)}} \tag{B.1}$$

$$\text{sigmoid}_{\text{p}}(x, a_1, c_1, a_2, c_2) = \frac{1}{(1 + e^{-a_1(x-c_1)})(1 + e^{-a_2(x-c_2)})} \tag{B.2}$$

$$\text{sigmoid}_{\text{cp}}(x, a_1, c_1, a_2, c_2) = \max \begin{bmatrix} \frac{1}{(1+e^{-a_1(x-c_1)})(1+e^{-a_2(x-c_2)})} \\ \frac{1}{(1+e^{-a_1(x+96-c_1)})(1+e^{-a_2(x+96-c_2)})} \\ \frac{1}{(1+e^{-a_1(x-96-c_1)})(1+e^{-a_2(x-96-c_2)})} \end{bmatrix} \tag{B.3}$$

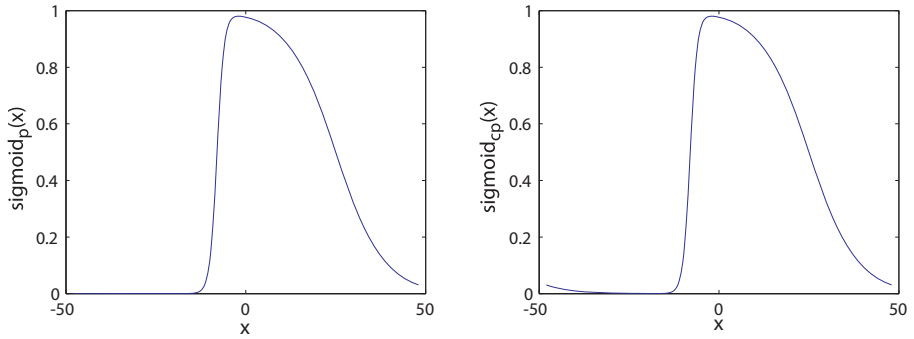


Figure B.1: Plot of the product of two sigmoid function with parameters $a_1 = 1, c_1 = -8, a_2 = 0.15, c_2 = 25$. The left plot is the normal sigmoid product function from equation B.2. The right plot is a specialized circular sigmoid product function, as shown in equation B.3.

The final function has to estimate the term $p(\varphi_\alpha - \varphi|\textit{feature})$. For observations where *feature* is not integer values, the observed features have been divided in 101 bins. For integer values, there will be a bin for each unique value.

A circular sigmoid product function is fitted to the observations in each of the n bins. This will give n times four parameters. The final parameters will be a maximum likelihood estimate from the n times four parameters, by linear regression with a cubic basis. This will give a smooth function.

Finding maximum values of $\varphi_\alpha - \varphi$ To be able to fit the circular sigmoid product function to the data, the most likely value of $\varphi_\alpha - \varphi$ needs to be estimated. This has to be done for all feature value bins i_n. The estimated maximums have been calculated as the averaged of the three most likely values:

$$\max(\varphi_\alpha - \varphi | i_n) = \sum_{\varphi_\alpha - \varphi = -48.}^{48} \frac{P(\varphi_\alpha - \varphi, i_n)}{3}, \textit{if } P(\varphi_\alpha - \varphi, i_n) \textit{is in top three} \tag{B.4}$$

Linear regression is used to create an estimated maximum of $\varphi_\alpha - \varphi$ for any given feature value. A plot of the estimated maximums for the feature φ' and $\beta = 0.5$ is shown in figure B.2.

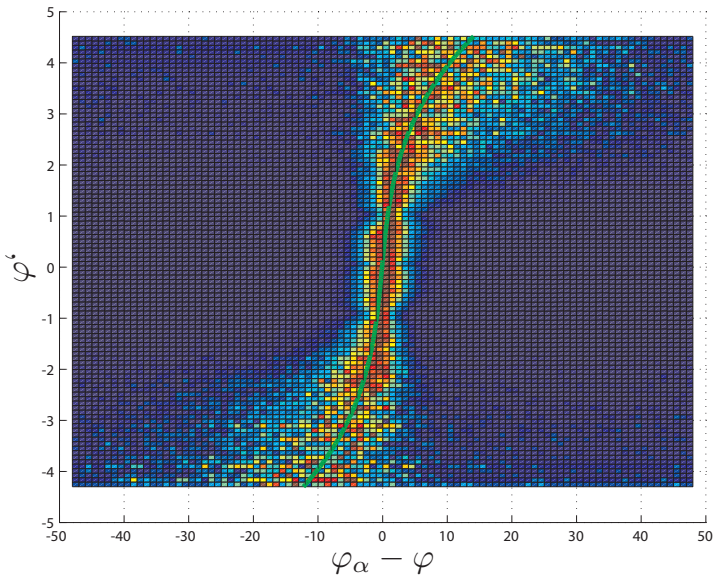


Figure B.2: Estimated maximums of $P(\varphi_\alpha - \varphi)$.

Estimating parameters for one bin The sigmoid product function is fitted to the observations in each bin in a two step procedure

1. Fit a single sigmoid function to all observations where $\varphi_\alpha - \varphi$ is less than the estimated maximum.

2. Fit a circular sigmoid product function to all the observations. The parameters a_1 and c_1 are fixed from the single sigmoid function.

A single sigmoid function is fitted to the observed data up to the estimated maximum. It is assumed that the errors follow a Gaussian distribution. The goal is to find the global minimum of the squared error function.

$$\underset{a,c}{\operatorname{argmin}} \sum_{x} (\text{sigmoid}(x, a, c) - p(x))^2 * p(x) \tag{B.5}$$

No method can guarantee to find the global minimum. The error function is continuous and is not expected to have a many local minimums away from the global minimum. Therefore it is very likely that an algorithm will be able to find the approximate location of the global minimum.

The minimum will be found by an iterative algorithm that will calculate the error for m data points between $a_{\min}$ and $a_{\max}$. The data points are logarithmically equally spaced. The l data points with the lowest errors, and the data points' neighbours will be used for the next iteration. $a_{\min}$ and $a_{\max}$ will be set the the minimum and maximum of the data points and m new data points will be generated. The algorithm will continue to run until $a_{\min}$ and $a_{\max}$ approach each other. If an iteration does not decrease the difference between $a_{\min}$ and $a_{\max}$, the m parameter is increased by one. This will change the position of all the generated data points for the next iteration.

For each estimate of a the c parameter is fitted. Fitting c is the same as distributing the sigmoid function along the x-axis. The signed squared error function in equation B.6 is used. By using the signed error function and bisection method, [Shampine et al., 1997] c is guaranteed to converge.

$$\sum_{x} (\text{sigmoid}(x, a, c) - p(x)) * |\text{sigmoid}(x, a, c) - p(x)| * p(x) \tag{B.6}$$

$c_{\min}$ and $c_{\max}$ values are set to a very large and very small value. The errors for $c_{\min}$ and $c_{\max}$ will have opposite signs. In each iteration the error for a new $c = (c_{\min} + c_{\max})/2$ value is calculated. The new c replaces the old $c_{\min}$ or $c_{\max}$ that have the same sign. Each iteration will reduce distance between $c_{\min}$ and $c_{\max}$ by half. The iterations will stop when the distance between $c_{\min}$ and $c_{\max}$ is below a certain threshold. In this thesis the threshold is set to 0.001 for both a and c estimates.

Figure B.3 and B.4 shows examples of iterations when fitting c and a.

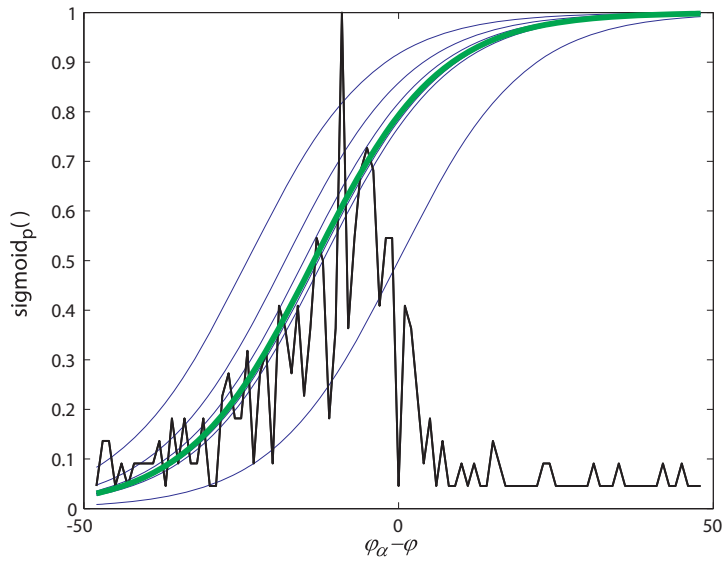


Figure B.3: Fitting of c for $a = 1$, $\varphi' = -3.757$ and $\beta = 0.5$. The green line shows the final solution.

After the single sigmoid function is fitted to the left part of the data, a circular sigmoid product function is fitted to all the data. a_1 and c_1 are set to the values of a and c from the sigmoid function. The estimates of a_2 and c_2 are done the same way.

Figure B.5 and B.6 shows examples of iterations when fitting c_2 and a_2.

Smoothing parameters It is important to have a smooth continuous function. Each of the four parameters to the circular sigmoid product function, have been estimated for each of the n bins. The n times four estimates are replaced by four functions of the input feature. This is done by linear regression with a cubic basis. It is very important that $a_1 > 0$ and $a_2 < 0$. Otherwise the circular sigmoid product function will be close to 0 or 1 for all values of x. A

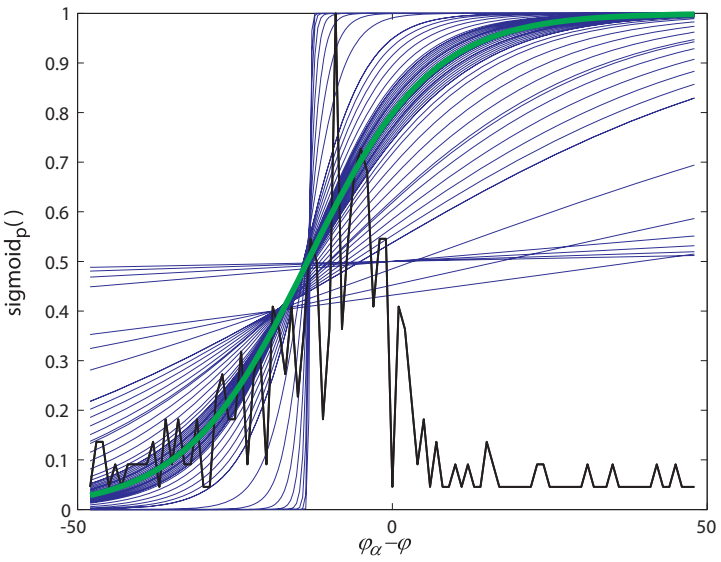


Figure B.4: Fitting of a for $\varphi' = -3.757$ and $\beta = 0.5$. The green line shows the final solution.

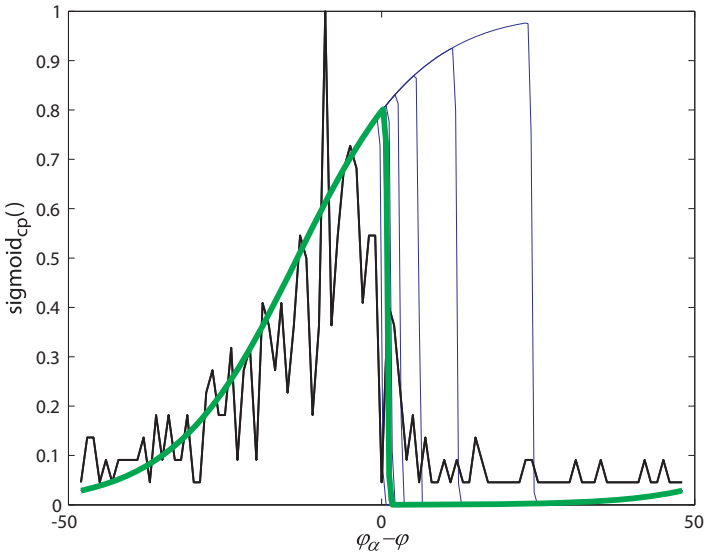


Figure B.5: Fitting of c_2 for $a_2 = -10$, $\varphi' = -3.757$ and $\beta = 0.5$. The green line shows the final solution.

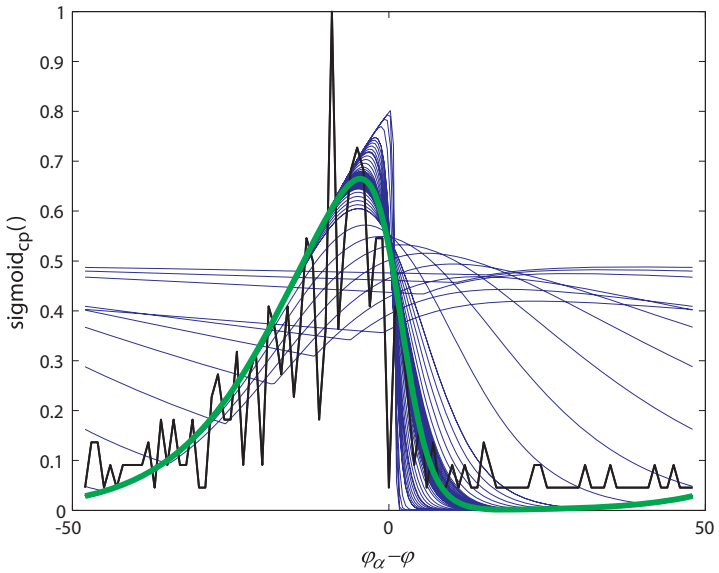


Figure B.6: Fitting of a_2 for $\varphi' = -3.757$ and $\beta = 0.5$. The green line shows the final solution.

way to ensure this is to substitute a_1 and a_2 with the A_1 and A_2 defined as:

$$A_1 = \log_{10}(a_1) \tag{B.7}$$
$$A_2 = \log_{10}(-a_2) \tag{B.8}$$
$$a_1 = 10^{A_1} \tag{B.9}$$
$$a_2 = -10^{A_2} \tag{B.10}$$

Figure B.7 shows an example of the fitted parameters and there estimates.

The error between observations and the fitted sigmoid product function is defined as the sum of squared errors for each row, multiplied by the number of observations in each row, divided by the total number of observations.

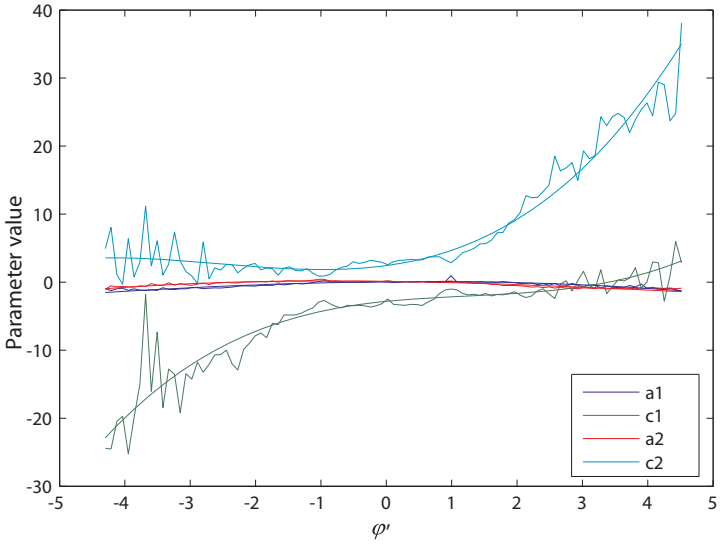


Figure B.7: Parameters for all n values of φ'. The lines show the estimates made by linear regression.

Appendix C

Statistical analysis of TUP iPod evaluations

This section explains the statistical models that have been used on the data from the TUP iPod evaluations.

C.1 Building a model

The main goals are to find the learning rate for TUP, the effect of using TUP different places and the effect of the participants' prior experience. The learning rate is expected to follow the power law of learning from equation 3.5. In this analysis the `Session` number will be used as the unit of n. The text entry sessions were completed many different places. It is very likely that it will have an influence on the text entry speed. Mixed linear models are used to be able to model the random effects of the different participants.

The data is loaded into R. To be able to use linear regression to find the learning rate, the logarithmic values of text entry speed `WPM` and session number `Session` are added to the data. Participants in the two evaluations share the same ID numbers. To avoid confusion a new factor `EvalSub` is added to the data. It is the concatenation of `Evaluation` and `Subject`.

Figure C.1 shows the speed of each phrase for both evaluations.

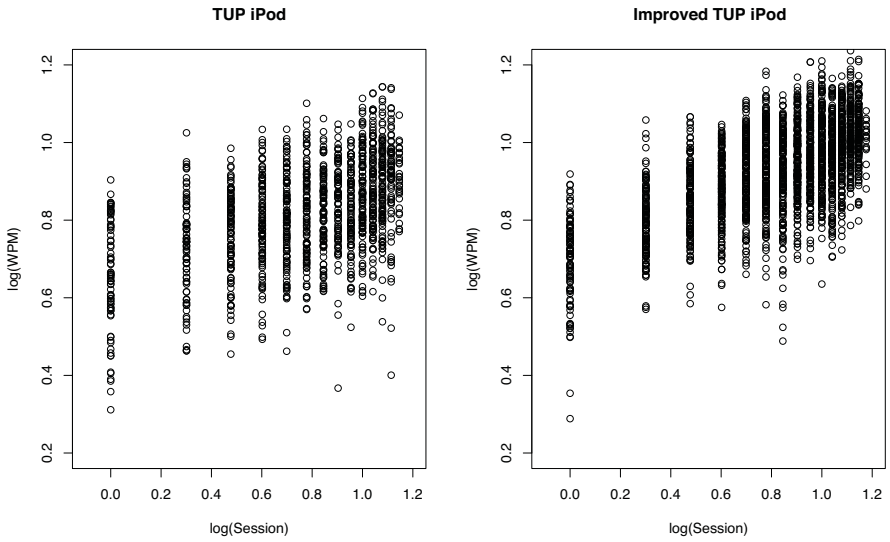


Figure C.1: Loglog plot of text entry speed for each entered phrase

C.1.1 Investigating variance structure

The fist part of the analysis is to investigate different variance structures for the observations. The observations are fitted to a different slopes model. Two different variance structures are tested.

Model 1 (C.1) use a simple variance structure (C.2) where the participants is supposed to be the only random effect.

$$\begin{aligned}\text{logWPM}_i = \alpha(\text{Evaluation}_i) + \beta(\text{Evaluation}_i) \cdot \text{logSession}_i + \\ a(\text{EvalSub}_i) + \epsilon_i\end{aligned} \tag{C.1}$$

$$a(k) \sim N(0, \sigma_a^2), k = 1...4152, \epsilon_i \sim N(0, \sigma^2) \tag{C.2}$$

Model 2 (C.3) adds an additional random coefficient to the variance structure (C.4) to allow the participants to have individual learning rates.

$$\begin{gathered}\text{logWPM}_i = \alpha(\text{Evaluation}_i) + \beta(\text{Evaluation}_i) \cdot \text{logSession}_i + \\ a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{gathered} \tag{C.3}$$

$$(a(k), b(k)) \sim N(0, \begin{pmatrix}\sigma_a^2 & \sigma_{ab} \\ \sigma_{ab} & \sigma_b^2\end{pmatrix}), k = 1...4152, \epsilon_i \sim N(0, \sigma^2) \tag{C.4}$$

Table C.1 and C.2 show the variances for the models.

Model 1	$-2l_{\text{REML}} = -10283.93$
σ_a^2	0.00752
σ^2	0.00474

Table C.1: Variance for model 1

Model 2	$-2l_{\text{REML}} = -10333.21$
σ_a^2	0.00917
σ_b^2	0.00122
σ_{ab}	-0.435
σ^2	0.00464

Table C.2: Variance for model 2

To test if the reduction in variance structure from model 2 to model 1, a restricted/residual likelihood ratio test is used. The test size is:

$$G_{\text{Model 2}\rightarrow\text{Model 1}} = -2l_{\text{RELM}}^{(\text{Model1})} - -2l_{\text{RELM}}^{(\text{Model2})} = 49.28 \tag{C.5}$$

This is compared to a χ_2^2 distribution. Two degrees of freedom is used because the reduced model have removed two variance components. The p value is $1.99 \cdot 10^{-11}$, so the reduction is very significant. It is therefore decided to use model 2 as a starting point for investigating the fixed effects.

C.1.2 Investigating fixed effects

The ANOVA table for Model 2 is shown in table C.3.

The interaction between `Evaluation` and `logSession` is insignificant, so it is removed from the model. Model 3 in equation C.6 shows the updated model. The ANOVA table is shown in table C.4.

Effect	numDF	denDF	F-value	p-value
Evaluation	1	22	9.4960	0.0055
logSession	1	4126	857.9701	<.0001
Evaluation:logSession	1	4126	1.6102	0.2045

Table C.3: Type 1 ANOVA table for model 2

$$\begin{aligned}\text{logWPM}_i &= \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i \\ &+a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.6}$$

Effect	numDF	denDF	F-value	p-value
Evaluation	1	22	9.3750	0.0057
logSession	1	4127	830.1892	<.0001

Table C.4: Type 1 ANOVA table for model 3

Next step is to include `Place`, `Light` and `Motivation` in the model. They are included one at a time in model 4 (C.7), 5 (C.8) and 6 (C.9). The ANOVA tables can be found in table C.5, C.6 and C.7.

$$\begin{aligned}\text{logWPM}_i &= \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i + \gamma(\text{Place}_i)+ \\ &a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.7}$$

Effect	numDF	denDF	F-value	p-value
Evaluation	1	22	8.6836	0.0075
logSession	1	4118	832.3985	<.0001
Place	9	4118	3.4308	0.0003

Table C.5: Type 1 ANOVA table for model 4

$$\begin{aligned}\text{logWPM}_i &= \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i + \gamma(\text{Place}_i)+ \\ &\lambda \cdot \text{Light}_i + a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.8}$$

$$\begin{aligned}\text{logWPM}_i &= \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i + \gamma(\text{Place}_i)+ \\ &\lambda \cdot \text{Motivation}_i + a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.9}$$

Only `Place` and `Motivation` are significant. A similar analysis is made to find the effect of the participants' prior experience. The factors tested are `Text.Entry.EXP`, `iPod.EXP`, `Tech.EXP` and `English`. None of these factors have a significant effect.

Effect	numDF	denDF	F-value	p-value
Evaluation	1	21	11.5982	0.0027
logSession	1	3977	777.9725	<.0001
Place	9	3977	3.5347	0.0002
Light	1	3977	2.4448	0.1180

Table C.6: Type 1 ANOVA table for model 5

Effect	numDF	denDF	F-value	p-value
Evaluation	1	21	11.1750	0.0031
logSession	1	3971	794.8257	<.0001
Place	9	3971	3.5244	0.0002
Motivation	1	3971	7.1319	0.0076

Table C.7: Type 1 ANOVA table for model 6

C.1.3 Levels of fixed effects

The final model (equation C.9) is used for finding the levels of the factors.

The model can be transformed to:

$$\begin{aligned} \mathrm{WPM}_i = 10^{\alpha(\mathrm{Evaluation}_i)} \cdot \mathrm{Session}_i^{\beta} \cdot 10^{\gamma(\mathrm{Place}_i)} \cdot 10^{\lambda \mathrm{Motivation}_i} \\ \cdot 10^{a(\mathrm{EvalSub}_i)} \cdot \mathrm{Session}_i^{b(\mathrm{EvalSub}_i)} \cdot 10^{\epsilon_i} \end{aligned} \tag{C.10}$$

C.1.3.1 Places

The levels of the Place factor γ are shown in table C.8.

The average effect of the `Place` factor is:

$$10^{\gamma(\overline{\mathrm{Place}_i})} = 1.0174 \tag{C.11}$$

C.1.3.2 Motivation

The motivation coefficient λ is found to be 0.00256. The lower and upper confidence intervals are 0.00068 and 0.00443. The mean value of the `Motivation` factor is 5.83. The average effect of the `Motivation` factor is:

γ(Place)	Count	Estimate	Lower.CI	Upper.CI
Unknown	179	-	-	-
$\text{Bed}_{a,b}$	439	0.0142	-0.0098	0.0383
$\text{Couch}_{c,d}$	929	0.0116	-0.0119	0.0351
Garden	15	-0.0294	-0.0724	0.0135
Home	1210	0.0059	-0.0178	0.0288
Lecture	11	0.0298	-0.0200	0.0796
Livingroom	195	0.0026	-0.0224	0.0277
$\text{Office}_{e,f}$	837	0.0123	-0.0109	0.0355
$\text{StandWalk}_{a,c,e}$	26	-0.0383	-0.0742	-0.0023
$\text{Transport}_{b,d,f}$	311	-0.0088	-0.0332	0.0157
Average		0.0075	-0.0169	0.0319

Table C.8: Estimates and 0.95 confidence intervals for the levels of the `Place` factor. The count off phrases written in different places is also shown. The subscripts indicate that these levels are significantly different using a 5 % level and Tukey-Kramer corrections of p-values. All other levels are not significantly different.

$$10^{\overline{\lambda\text{Motivation}_i}} = 1.0349 \tag{C.12}$$

C.1.3.3 Learning rates

After the average levels for `Place` and `Motivation` have been estimated the learning rates can be found. The ANOVA table for model 2 showed that there was no interaction between `Evaluation` and `logSession`. This means that the learning rate β is the same for both evaluations. The estimate of β is 0.2415. The lower and upper confidence intervals are 0.2249 and 0.2582.

The levels of the `Evaluation` factor are shown in table C.9. The average effects of `Place` and `Motivation` are included in the estimate. There is a significant difference between the evaluations.

α(Evaluation)	Estimate	Lower.CI	Upper.CI
Evaluation 1	0.6162	0.5548	0.6777
Evaluation 2	0.7259	0.6666	0.7852

Table C.9: Estimates and 0.95 confidence intervals for the levels of the Evaluation factor. The average effect of Place and Motivation are included in the estimate.

The final estimate of the learning rate for evaluation 1 is shown in equation C.13 and in equation C.14 for evaluation 2.

$$\mathrm{WPM}_n = 4.13n^{0.2415} \tag{C.13}$$

$$\mathrm{WPM}_n = 5.32n^{0.2415} \tag{C.14}$$

The plots in figure C.2 show the learning rates and the observations.

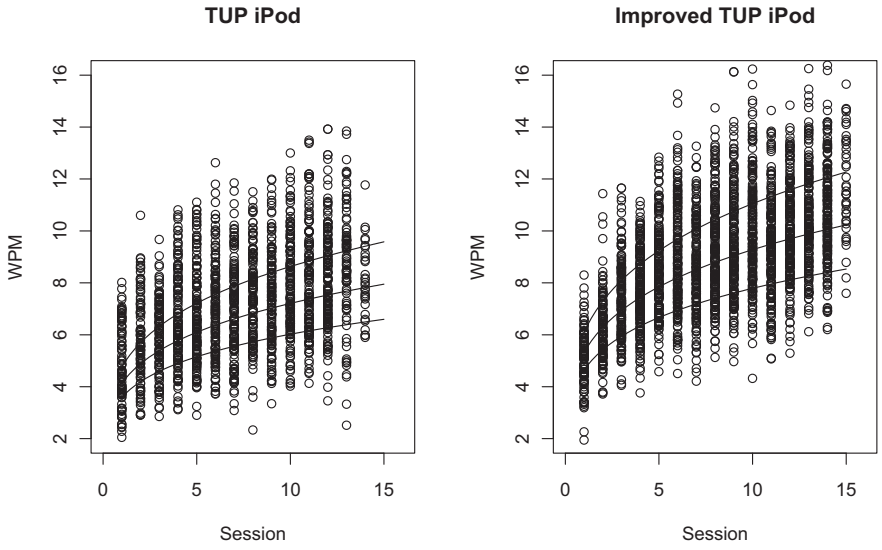


Figure C.2: Plots of text entry speed for each entered phrase. The three lines is the estimated learning rates and the upper and lower confidence intervals.

C.1.4 Investigating the influence of error rates on text entry speed

To be able to model the influence of error rates, model 6 is extended with the corrected error rate `CorErrRate` and uncorrected error rate `UncErrRate` factors. They are both included as coefficients. The new model is shown in equation C.15.

$$\begin{aligned}\text{logWPM}_i = \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i + \gamma(\text{Place}_i) + \\ \lambda \cdot \text{Motivation}_i + \mu \cdot \text{UncErrRate}_i + \nu \cdot \text{CorErrRate}_i + \\ a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.15}$$

Effect	numDF	denDF	F-value	p-value
Evaluation	1	21	6.371	0.0197
logSession	1	3969	823.869	<.0001
Place	9	3969	6.494	<.0001
Motivation	1	3969	13.312	0.0003
CorErrRate	1	3969	3217.996	<.0001
UncErrRate	1	3969	3.311	0.0689

Table C.10: Type 1 ANOVA table for model 7

`UncErrRate` is not significant. It is therefore removed in model 8 (equation C.16).

$$\begin{aligned}\text{logWPM}_i = \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i + \gamma(\text{Place}_i) + \\ \lambda \cdot \text{Motivation}_i + \nu \cdot \text{CorErrRate}_i + \\ a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.16}$$

Effect	numDF	denDF	F-value	p-value
Evaluation	1	21	6.340	2e-02
logSession	1	3970	835.251	<.0001
Place	9	3970	6.490	<.0001
Motivation	1	3970	13.312	3e-04
CorErrRate	1	3970	3215.952	<.0001

Table C.11: Type 1 ANOVA table for model 8

A new model (equation C.17) is created with the possibility of interaction between `CorErrRate` and `Evaluation`. The ANOVA table (table C.12) shows that there is no such interaction.

$$\begin{aligned}\text{logWPM}_i = \alpha(\text{Evaluation}_i) + \beta \cdot \text{logSession}_i + \gamma(\text{Place}_i) + \\ \lambda \cdot \text{Motivation}_i + \nu(\text{Evaluation}_i) \cdot \text{CorErrRate}_i + \\ a(\text{EvalSub}_i) + b(\text{EvalSub}_i) \cdot \text{logSession}_i + \epsilon_i\end{aligned} \tag{C.17}$$

The model 8 can be transformed to:

$$\begin{aligned}\text{WPM}_i = 10^{\alpha(\text{Evaluation}_i)} \cdot \text{Session}_i^{\beta} \cdot 10^{\gamma(\text{Place}_i)} \cdot 10^{\lambda \text{Motivation}_i} \\ \cdot 10^{\nu \text{CorErrRate}_i} \cdot 10^{a(\text{EvalSub}_i)} \cdot \text{Session}_i^{b(\text{EvalSub}_i)} \cdot 10^{\epsilon_i}\end{aligned} \tag{C.18}$$

Effect	numDF	denDF	F-value	p-value
Evaluation	1	21	6.338	0.0200
logSession	1	3969	838.679	<.0001
Place	9	3969	6.490	<.0001
Motivation	1	3969	13.315	0.0003
CorErrRate	1	3969	3216.064	<.0001
Evaluation:CorErrRate	1	3969	1.204	0.2725

Table C.12: Type 1 ANOVA table for model 9

The estimate of ν is -0.8217, with lower and upper confidence intervals of -0.8501 and -0.7933.

$$(10^{\nu})^{\mathrm{CorErrRate}_i} = 0.1508^{\mathrm{CorErrRate}_i}. \tag{C.19}$$

Table C.13 shows some examples of the expected reduction of the text entry speed for different error rates.

CorErrRate	Reduction in WPM
0.00	0.0%
0.05	9.0%
0.10	17.2%
0.15	24.7%
0.20	31.5%
0.25	37.7%

Table C.13: Effect of corrected error rate on text entry speed

C.1.5 Estimate of the error free learning rate

To be able to find the effect of the corrected error rate, the learning rates is estimated for error free performance. The estimate of β is 0.2184. The lower and upper confidence intervals are 0.2022 and 0.2346.

The levels of the `Evaluation` factor are shown in table C.14. The average effects of `Place` and `Motivation` are included in the estimate. There is a significant difference between the evaluations.

α(Evaluation)	Estimate	Lower.CI	Upper.CI
Evaluation 1	0.6988	0.6392	0.7584
Evaluation 2	0.7666	0.7096	0.8236

Table C.14: Estimate 0.95 confidence interval for the levels of the Evaluation factor. The average effects of Place and Motivation are included in the estimate.

The estimate of the learning rate for evaluation 1 is shown in equation C.20 and in equation C.21 for evaluation 2. These estimates assume that the corrected error rate is 0.

$$\mathrm{WPM}_n = 5.00n^{0.2184} \tag{C.20}$$

$$\mathrm{WPM}_n = 5.84n^{0.2184} \tag{C.21}$$

Appendix D

Creating a text message corpus

To be able to understand how text messages are related to each other it is necessary to use a large corpus of text messages. The corpus should not consist of single text messages, but of series of sent and received messages from a number of persons. This it needed to be able to find patterns in the language. The number of text message series and the length of each series should be large enough to get statistically significant results.

D.1 Methodology

It was decided to collect messages from 20 users over four weeks. Only users that expected to sent and receive at least 100 messages were used. This group is not representative for the entire population. Users that write few messages will not benefit much from adaptive language models. Therefore the focus has been on users that wrote more messages.

D.1.1 Technical setup

There are many ways to collect text messages. The most optimal solution is to cooperate with a mobile service provider. They can connect all messages without any user intervention. This gives a very precise data set, and is not dependent on the users' phones. The main drawback of this method is that you need close collaboration with a service provider, and that all the users need to have the same service provider. Another solution is to let the users collect the text messages and send them to the evaluator. Most new mobile phones can export all sent and received messages to a computer. This approach eliminates the need for a mobile service provider, but has a number of drawbacks. The different phones use different message formats, so it requires a lot of post processing to get the messages into a common format. Some phones can not export the time stamp of sent messages. This will reduce the quality of the data set. Finally the method requires a lot of user intervention. The users have to be experienced with their mobile phones and computers.

The last method was used for the collecting of text messages.

D.1.2 Finding users

Because the chosen method requires some technical knowledge, the users need to have more experienced than average users of mobile phones. The users were recruited through an article on the website `http://mobil.nu/`. It is a Danish website with news about mobile phones. The publishing group behind the website does also make a monthly magazine about mobile phones. The audience of the website is expected to be tech-savvy persons with a great knowledge of mobile phones.

The users were rewarded with a gift certificate on 200,- DKK. The publishing group sponsored three one-year subscriptions to their magazine. There was a draw between all the users for the three subscriptions.

D.1.3 Ethical issues

Text messages are used to communicate all kinds of information, including very personal messages. It was a great concern whether it was possible to find enough users that would share their messages. To be able to find the persons, a strong focus was placed on the privacy and security of the users. The participants were

allowed to delete messages before they were sent to the evaluator. They were guaranteed that no messages or part of messages would be published without their prior permission. The users received a Microsoft Excel document to collect their messages in. The document contained a macro that would anonymise all phone numbers by replacing them with their md5 hash value.

D.2 Collecting the messages

The text messages were collected in March and April 2008. 24 people responded on the article on the website. 23 wanted to participate while one was requesting further information. All the participants received an email where they were asked to make a test transfer of messages to the computer before they started. Participants less than 18 years old were asked to get their parents permission to participate. The parents should return a mail containing their permission. Only 12 people responded on the letter. 10 still wanted to participate and two could not transfer messages to the computer.

After four weeks a new email was sent to all participants except the two that could not transfer messages. 12 people returned their messages. Some participants that responded on the first mail did no returned anything, but other participants that did not respond to the first mail returned their messages. Some participants experienced computer crashes or problems with their phones that prevented them from participating.

Two users had trouble with their phones, and got an extended deadline for sending in the messages.

D.3 Demographics and text message statistics

An overview of the 12 participants is given in table D.1. There were 10 male and two female participants. The participants were between 15 and 33 years old, with an average of 23 years. The phones used by the participants are all advanced models. The HTC S710 has both an ITU-T and miniature QWERTY keyboard. The rest of the phones only have an ITU-T keyboard for text entry. The demographics of the participants fit well with the typical tech savvy user.

In total the participants sent 10,799 messages and received 13,571 messages. The average length of the messages was around 60 characters per message.

Participants who sent many messages tend to write shorter messages than participants who sent fewer messages. Only two participants chose to delete messages before they were sent to the evaluator.

D.4 Post processing of text messages

The text messages were returned in many different formats, dependent on the phone and software used. Table D.2 shows the formats of the returned messages. In the Nokia-Excel and SE-Excel format, all phone numbers were already replaced with the md5 hash value. For the other formats it was done after the messages had been imported into Excel. All the messages from all participants were combined in one Excel document.

The Excel document was imported into Matlab by a script. The script loads all the data and converts it into formats optimized for Matlab. The text messages are stored in a struct. The format of the struct is explained in table D.3.

User	Sex	Age	Phone	Sent	Recv	Sent size	Recv size	Contacts	Deleted
1	M	25	Nokia E50	193	440	121	122	128	0
2	M	19	SE W910i	2,383	2,641	71	83	133	0
3	M	18	SE K800i	463	489	49	39	33	0
4	M	15	Nokia N95	679	819	28	45	34	0
5	M	32	Nokia N95 8GB	102	143	100	94	79	0
6	M	31	HTC S710	527	325	126	139	144	0
7	F	33	Nokia E65	202	195	70	53	21	0
8	M	18	SE K850i	1,874	1,912	35	33	102	0
9	M	19	Nokia E51	1,824	2,067	45	51	58	1
10	M	21	Nokia 8600	1,123	1,091	89	70	65	0
11	F	19	SE K850i	1,347	3,114	54	44	180	0
12	M	24	Nokia N95	82	335	73	85	59	2
Total				10,799	13,571	61	59	1,036	3

Table D.1: Overview of users and messages in the text message corpus. Sent size and received size is the average length of the sent and received text messages. Contacts is the number of unique contacts that each participants have received or sent messages to. Deleted is the number of messages that were deleted by the users before the messages were sent to the evaluator.

User	UID	Phone	Message format
1	3	Nokia E50	Nokia-Excel
2	5	SE W910i	SE-Excel
3	6	SE K800i	SE-table
4	7	Nokia N95	Nokia-Excel
5	9	Nokia N95 8GB	Nokia-Excel
6	13	HTC S710	HTC
7	15	Nokia E65	Nokia-Excel
8	17	SE K850i	SE-txt
9	18	Nokia E51	Nokia-Excel
10	20	Nokia 8600	Vmessage
11	21	SE K850i	SE-Excel
12	22	Nokia N95	Nokia-Excel

Table D.2: Participant that supplied messages to the text message corpus. User is the number that is used to identify the users in the thesis. UID is an internal identification number that each participant got. Only half of the participants returned messages for the final corpus.

Name	Description
`id`	Id of each message.
`uid`	Id of the participant that received or sent the message (from table D.1).
`recv`	Type of message. 1 for received messages, 0 for sent.
`time`	Timestamp in matlab date number format. -1 for unknown.
`person`	Id of the sender or receiver of the messages. In format XXYYYY. XX equals `uid`, YYYY is a unique id of each sender and receiver.
`msg`	The text of each message.

Table D.3: Format of the text message corpus in matlab

Bibliography

Apple. Fifth generation ipod (late 2006) - technical specifications. Apple Website, November 2006. URL `http://support.apple.com/specs/ipod/iPod_Fifth_Generation_Late_2006.html`.

Timothy Bell, Ian H. Witten, and John G. Cleary. Modeling for text compression. *ACM Comput. Surv.*, 21(4):557–591, 1989. ISSN 0360-0300.

Jerome R. Bellegarda. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93–108, 2004.

T. Bellman and I. S. MacKenzie. A probabilistic character layout strategy for mobile text entry. In *Proc. Graphics Interface 98*, pages 168–176. Canadian Inf. Process. Soc, 1998.

Martin Böcker, Bruno von Niman, and Karl Ivar Larsson. Increasing text-entry usability in mobile devices for languages used in europe. *interactions*, 13(5): 30–35, 2006. ISSN 1072-5520.

T. Boren and J. Ramey. Thinking aloud: reconciling theory and practice. *IEEE Transactions on Professional Communication*, 43(3):261–278, 2000. ISSN 03611434.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992a.

Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. An estimate of an upper bound for the entropy of english. *Comput. Linguist.*, 18(1):31–40, 1992b. ISSN 0891-2017.

Lee Butts and Andy Cockburn. An evaluation of mobile phone text input methods. In *Third Australasian conference on User interfaces*, pages 55–59. Australian Computer Society, Inc., 2002. ISBN 0-909925-85-2.

Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7): 396–410, 1980. ISSN 0001-0782.

Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983. ISBN 0898592437.

Steven J. Castellucci and I. Scott MacKenzie. Graffiti vs. unistrokes: an empirical comparison. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 305–308, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1.

Edward Clarkson, James Clawson, Kent Lyons, and Thad Starner. An empirical study of typing rates on mini-qwerty keyboards. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1288–1291, New York, NY, USA, 2005. ACM. ISBN 1-59593-002-7.

P.R Clarkson and A.J. Robinson. Language model adaptation using mixtures and an exponentiallydecaying cache. In *IEEE International Conference on Acoustics, Speech, and Signal Processing.*, volume 2, pages 799–802, 1997.

James Clawson, Kent Lyons, Edward Clarkson, and Thad Starner. Mobile text entry: An empirical study and analysis of mini-qwerty keyboards. Submitted to the Transaction on Computer Human Interaction Journal, 2006.

James Clawson, Kent Lyons, Alex Rudnick, Jr. Robert A. Iannucci, and Thad Starner. Automatic whiteout++: correcting mini-qwerty typing errors using keypress timing. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 573–582, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1.

J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984. ISSN 00962244.

J.G. Cleary, W.J. Teahan, and I.H. Witten. Unbounded length contexts for ppm. *Proceedings DCC '95 Data Compression Conference*, pages 52–61, 1995. ISSN 10680314.

T. Cover and R. King. A convergent gambling estimate of the entropy of english. *IEEE Transactions on Information Theory*, 24(4):413–421, 1978. ISSN 00189448.

Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991. ISBN 0471062596.

John J. Darragh, Ian H. Witten, and Mark L. James. The reactive keyboard: A predictive typing aid. *Computer*, 23(11):41–49, 1990. ISSN 0018-9162.

S. Deligne and F. Bimbot. Language modeling by variable length sequences: Theoretical formulation and evaluation of multigrams. In *Proc. ICASSP '95*, pages 169–172, Detroit, MI, 1995.

D.B. Devoe. Alternatives to handprinting in the manual entry of data. *IEEE Transactions on Human Factors in Electronics*, HFE-8(1):21–32, 1967. ISSN 0096249x.

Mark D. Dunlop. Watch-top text-entry: Can phone-style predictive text entry work with only 5 buttons? In *Mobile Human-Computer Interaction - MobileHCI 2004: 6th International Symposium, Glasgow, UK, September 13 - 16, 2004. Proceedings*, pages 342–346. Springer-Verlag, 2004. ISBN 3-540-23086-6.

Mark D. Dunlop, Andrew Glen, Sunil Motaparti, and Sanjay Patel. Adaptex: contextually adaptive text entry for mobiles. In *MobileHCI '06: Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, pages 265–265, New York, NY, USA, 2006. ACM. ISBN 1-59593-390-5. doi: http://doi.acm.org.globalproxy.cvt.dk/10.1145/1152215.1152277.

Eatoni Ergonomics Inc. Reviewer and user identified problems with t9 and ezi text. Technical report, Eatoni Ergonomics Inc., 2001. URL `http://eatoni.com/research/t9-problems.pdf`. Collection of materials about problems with T9 and eZi Text.

P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954. ISSN 0022-1015.

Jianfeng Gao and Min Zhang. Improving language model size reduction using better pruning criteria. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 176–182, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

David Goldberg and Cate Richardson. Touch-typing with a stylus. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 80–87, New York, NY, USA, 1993. ACM. ISBN 0-89791-575-5.

Jun Gong and Peter Tarasewich. Alphabetically constrained keypad designs for text entry on mobile devices. In *CHI '05: Proceedings of the SIGCHI*

conference on Human factors in computing systems, pages 211–220, New York, NY, USA, 2005. ACM. ISBN 1-58113-998-5.

I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3/4):237–264, 1953. ISSN 00063444.

Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. Language modeling for soft keyboards. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 194–195, New York, NY, USA, 2002. ACM. ISBN 1-58113-459-2.

Rebecca E. Grinter and Margery A. Eldridge. y do tngrs luv 2 txt msg? In *ECSCW'01: Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work*, pages 219–238, Norwell, MA, USA, 2001. Kluwer Academic Publishers. ISBN 0-7923-7162-3.

GSM Association. Strong global demand for mms and mobile email. GSM Association Press Releases 2007-48, GSM Association, 2006. URL `http://www.gsmworld.com/news/press_2006/press06_61.shtml`.

GSM Association. Global mobile communication is 20 years old. GSM Association Press Releases 2006-61, GSM Association, 2007. URL `http://www.gsmworld.com/news/press_2007/press07_48.shtml`.

Howard Gutowitz. Barriers to adoption of dictionary-based text-entry methods: A field study. In *10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2003.

Lester Haines. Teen breaks sms world speed record. Website: The Register, November 2006. URL `http://www.theregister.co.uk/2006/11/13/worlds_fastest_sms/`.

Trevor Hastie, Jerome Friedman, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

Eve Hoggan, Stephen A. Brewster, and Jody Johnston. Investigating the effectiveness of tactile feedback for mobile touchscreens. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1573–1582, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1.

Poika Isokoski and Timo Linden. Effect of foreign language on text transcription performance: Finns writing english. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 109–112, New York, NY, USA, 2004. ACM. ISBN 1-58113-857-1.

Poika Isokoski and I. Scott MacKenzie. Combined model for text entry rate development. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 752–753, New York, NY, USA, 2003. ACM. ISBN 1-58113-637-4.

TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Arrangement of digits, letters and symbols on telephones and other devices that can be used for gaining access to a telephone network. Technical Report E.161, INTERNATIONAL TELECOMMUNICATION UNION, February 2001. URL `http://www.itu.int/rec/T-REC-E.161/en`.

Christina James and Michael Longé. Bringing text input beyond the desktop. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 49–50, New York, NY, USA, 2000. ACM. ISBN 1-58113-248-4.

Christina L. James and Kelly M. Reischel. Text input for mobile devices: comparing model prediction to actual performance. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–371. ACM Press, 2001. ISBN 1-58113-327-8.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, first edition, 2000. ISBN 013122798X.

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(3):400–401, 1987. ISSN 00963518.

R. Kneser. Statistical language modeling using a variable context length. In *Proc. ICSLP '96*, volume 1, pages 494–497, Philadelphia, PA, 1996.

Hedy Kober, Eugene Skepner, Terry Jones, Derek Smith, and Scott MacKenzie. Letterwise: Prefix-based disambiguation for mobile text input. In *The 14th Annual ACM Symposium on User Interface Software and Technology*, 2001.

R. Kuhn and R. De Mori. A cache-based natural language model for speech recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(6):570–583, 1990. ISSN 0162-8828.

Karen Kukich. Technique for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, 1992. ISSN 0360-0300.

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, Mar 1951.

Ditte Laursen. *Det mobile samtalerum - om unges kommunikations- og samværsformer via mobiltelefonen*. PhD thesis, Syddansk Universitet, 2006.

Ted Laverack and Bruno von Niman. Character repertoires, orderings and assignments to the 12-key telephone keypad (for european languages and other languages used in europe). Specification ES 202 130, version 2.1.2, European Telecommunications Standards Institute (ETSI), 2007.

Kent Lyons, Thad Starner, Daniel Plaisted, James Fusia, Amanda Lyons, Aaron Drew, and E. W. Looney. Twiddler typing: one-handed chording text entry for mobile phones. In *Proceedings of the 2004 conference on Human factors in computing systems*, pages 671–678. ACM Press, 2004. ISBN 1-58113-702-8.

I. Scott MacKenzie. Kspc (keystrokes per character) as a characteristic of text entry techniques. In *Mobile HCI '02: Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*, pages 195–210, London, UK, 2002a. Springer-Verlag. ISBN 3-540-44189-1.

I. Scott MacKenzie. Mobile text entry using three keys. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 27–34. ACM Press, 2002b. ISBN 1-58113-616-1.

I. Scott MacKenzie and R. William Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17: 147–198, 2002. ISSN 0737-0024.

I. Scott MacKenzie and R. William Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 754–755, New York, NY, USA, 2003. ACM. ISBN 1-58113-637-4.

I. Scott MacKenzie and Shawn X. Zhang. The design and evaluation of a high-performance soft keyboard. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 25–31, New York, NY, USA, 1999. ACM. ISBN 0-201-48559-1.

I. Scott MacKenzie, Javier Chen, and Aleks Oniszczak. Unipad: single stroke text entry with language-based acceleration. In *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction*, pages 78–85, New York, NY, USA, 2006. ACM. ISBN 1-59593-325-5.

Juha Marila and Sami Ronkainen. Time-out in user interface: the case of mobile text input. *Personal Ubiquitous Computing*, 8(2):110–116, 2004. ISSN 1617-4909.

Edgar Matias, I. Scott MacKenzie, and William Buxton. Half-qwerty: a one-handed keyboard facilitating skill transfer from qwerty. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 88–94, New York, NY, USA, 1993. ACM. ISBN 0-89791-575-5.

Edgar Matias, I. Scott MacKenzie, and William Buxton. Half-qwerty: typing with one hand using your two-handed skills. In *CHI '94: Conference companion on Human factors in computing systems*, pages 51–52, New York, NY, USA, 1994. ACM. ISBN 0-89791-651-4.

T. Niesler and P. Woodland. A variable-length category-based n-gram language model. In *Proc. ICASSP '96*, pages 164–167, Atlanta, GA, 1996.

Donald A. Norman. *The Design of Everyday Things*. MIT Press, September 1998. ISBN 0465067107.

Joshua Oreman. *Podzilla2 Programmer's Reference*, 2007a. URL `http://opensvn.csie.org/courtc/tools/podzilla2/API.tex`.

Joshua Oreman. *TTK, An iPod GUI Library*, 2007b. URL `http://opensvn.csie.org/courtc/tools/ttk/API/API.tex`.

Antti Oulasvirta, Sakari Tamminen, Virpi Roto, and Jaana Kuorelahti. Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile hci. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 919–928, New York, NY, USA, 2005. ACM. ISBN 1-58113-998-5.

Oxford University Computing Services. The british national corpus. Website `http://www.natcorp.ox.ac.uk/`, 2001. The British National Corpus is a 100 million word collection of samples of written and spoken language from a wide range of sources, designed to represent a wide cross-section of current British English, both spoken and written.

Andriy Pavlovych and Wolfgang Stuerzlinger. Less-tap: A fast and easy-to-learn text input technique for phones. In *Proceedings of Graphics Interface*, pages 97–104. Canadian Information Processing Society, 2003.

Morten Proschowsky. Usability evaluation of tup. CICT Working Paper 107, Center for Information and Communication Technology, Technical University of Denmark, December 2005a.

Morten Proschowsky. Interaction design of text input for mobile devices. Master's thesis, Technical University of Denmark, May 2005b.

Marianne Rathje and Ole Ravnholt. R tjat å sms 1 trusl mod skriftsprågd? Nyt fra Sprognævnet, June 2002. URL `http://www.dsn.dk/nfs/2002-2.htm`. Newsletter in Danish.

Helena Roeber, John Bacus, and Carlo Tomasi. Typing in thin air: the canesta projection keyboard - a new method of interaction with electronic devices. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 712–713, New York, NY, USA, 2003. ACM. ISBN 1-58113-637-4.

Hokyoung Ryu and Katrina Cruz. Letterease: Improving text entry on a handheld device via letter reassignment. In *OZCHI '05: Proceedings of the 19th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction*, pages 1–10, Narrabundah, Australia, Australia, 2005. Computer-Human Interaction Special Interest Group (CHISIG) of Australia. ISBN 1-59593-222-4.

Walter Schneider and Richard M. Shiffrin. Controlled and automatic human information processing: I. detection, search, and attention. *Psychological Review*, 84(1):1–66, January 1977.

L. F. Shampine, Jr. R. C. Allen, and S. Pruess. *Fundamentals of numerical computing*. John Wiley & Sons, Inc., New York, NY, USA, 1997. ISBN 0-471-16363-5.

Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(3):379–423, July 1948. Continued 27(4):623-656, October 1948.

Claude E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, 30:50–65, 1951.

Helen Sharp, Yvonne Rogers, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 2007. ISBN 978-0-470-01866-8.

Miika Silfverberg, I. Scott MacKenzie, and Panu Korhonen. Predicting text entry speed on mobile phones. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 9–16. ACM Press, 2000. ISBN 1-58113-216-6.

R. William Soukoreff and I. Scott MacKenzie. Recent developments in text-entry error rate measurement. In *Extended abstracts of the 2004 conference on Human factors and computing systems*, pages 1425–1428. ACM Press, 2004. ISBN 1-58113-703-6.

R. William Soukoreff and I. Scott MacKenzie. Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 319–320, New York, NY, USA, 2001. ACM. ISBN 1-58113-340-5.

R. William Soukoreff and I. Scott MacKenzie. Metrics for text entry research: an evaluation of msd and kspc, and a new unified error metric. In *Proceedings of the conference on Human factors in computing systems*, pages 113–120. ACM Press, 2003. ISBN 1-58113-630-7.

Tom Stocky, Alexander Faaborg, and Henry Lieberman. A commonsense approach to predictive text entry. In *Extended abstracts of the 2004 conference*

on Human factors and computing systems, pages 1163–1166. ACM Press, 2004. ISBN 1-58113-703-6.

Andreas Stolcke. Entropy-based pruning of backoff language models. In *In Proc. DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274, 1998.

Kumiko Tanaka-Ishii, Yusuke Inutsuka, and Masato Takeichi. Entering text with a four-button device. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

Peter Tarasewich. Evaluation of thumbwheel text entry methods. In *CHI '03 extended abstracts on Human factors in computing systems*, pages 756–757. ACM Press, 2003. ISBN 1-58113-637-4.

W.J. Teahan and J.G. Cleary. The entropy of english using ppm-based models. *Proceedings of Data Compression Conference - DCC '96*, pages 53–62, 1996. ISSN 10680314.

Tonio Wandmacher, Jean-Yves Antoine, Franck Poirier, and Jean-Paul Départe. Sibylle, an assistive communication system adapting to the context and its user. *ACM Trans. Access. Comput.*, 1(1):1–30, 2008. ISSN 1936-7228.

John Williamson. *Continuous Uncertain Interaction*. PhD thesis, UNIVERSITY OF GLASGOW, 2006.

John Williamson and Roderick Murray-Smith. *Dynamics and probabilistic text entry*, volume Volume 3355/2005 of *Lecture Notes in Computer Science*, pages 333–342. Springer Berlin / Heidelberg, 2005.

Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 37:1085–1094, 1991.

Jacob O. Wobbrock and Brad A. Myers. Analyzing the input stream for character- level errors in unconstrained text entry evaluations. *ACM Transactions on Computer-Human Interaction*, 13(4):458–489, 2006. ISSN 1073-0516.

Jacob O. Wobbrock, Brad A. Myers, and John A. Kembel. Edgewrite: a stylus-based text entry method designed for high accuracy and stability of motion. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 61–70, New York, NY, USA, 2003. ACM. ISBN 1-58113-636-6.

Shumin Zhai and Barton A Smith. Alphabetically biased virtual keyboards are easier to use: layout does matter. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 321–322, New York, NY, USA, 2001. ACM. ISBN 1-58113-340-5.

Shumin Zhai, Michael Hunter, and Barton A. Smith. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 119–128. ACM Press, 2000. ISBN 1-58113-212-3.