GazeTry: Swipe Text Typing Using Gaze

Yi Liu, Chi Zhang
Interdisciplinary Graduate School
Nanyang Technological University, Singapore
{yliu028, czhang024}@e.ntu.edu.sg

Chonho Lee, Bu-Sung Lee, Alex Qiang Chen
School of Computer Engineering
Nanyang Technological University, Singapore
{leechonho, ebslee, alexchen}@ntu.edu.sg

ABSTRACT
Over the last decades, eye gaze has become an alternative form of text entry by some physically challenged people. Recently a dwell-free system has been proposed, which has been proven to be much faster compared to other existing dwell-free systems. However, it is vulnerable to some common text entry problems. In the paper, we propose GazeTry, a dwell-free gaze-based text entry system which allows people to type a word by gazing sequentially at the letters of the word. Simulation and experiments results show that our proposed new dwell-free system, GazeTry with the Moving Window String Matching (MoWing) algorithm has better accuracy and more resilience to text entry errors.

Author Keywords
Eye-typing, Assistive technologies, Dwell-free

ACM Classification Keywords
H5.2. [Information interfaces and presentation]: User Interfaces-*Input devices and strategies*.

INTRODUCTION
The ability to express quickly and efficiently in precise language is fundamental for quality of life. However, some physically challenged people have difficulty in communicating using our "common" method, e.g. Motor neuro disease (MND), such as Amyotrophic lateral sclerosis (ALS) or locked-in Syndrome, nearly 120,000 cases diagnosed worldwide each year (Majaranta et al., 2011). Therefore, control of the eyes may be the only way for affected people. As technology develops, it is possible to enable people to communicate using eye tracking devices. If we can make use of eye tracking to achieve text entry, it would be able to provide people with motor disabilities a new reliable method of communication.

In general, there are two main reasons of relatively slow text entry, 1). *Selection method*, users need to fixate a key with the gaze for a while, and the duration is called dwell time, i.e. if a user wants to select a letter, they have to fixate at the letter on the keyboard for some time (typically 200-600ms). The dwell time selection method limits the speed. 2). *Text entry method*, early eye-typing systems only support individual character input, i.e. if a user wants to input "hello", they have to select five characters which limits the maximum entry speed.

In order to increase the speed of text entry, researchers have worked on improving the above two methods (MacKenzie and Zhang, 2008; Majaranta et al., 2009; Urbina and Huckauf, 2010; Sarcar et al., 2013). Although the speed of text entry is increased, it is still limited by dwell time in which users need to fixate at the correct keys. Also, the user could get tired in a shorter time span when attempting to fixate every key entry. Therefore, it is necessary to innovate and develop a new text entry system. Moreover, if the system allows users to focus on the entire word typing instead of individual letter typing, the speed of text entry would improve.

In the dwell-free typing field, Kristensson and Vertanen (2012) investigated the potential of dwell-free typing, and implemented the typing prototype which demonstrated dwell-free typing is promising and much faster than existing dwell-based typing. However, their software assumes an ideal typing environment in which it only accepts a letter corresponding to the intended letter. They do not solve any of three text entry errors, extra-letter error, neighbor-letter error, and missing-letter error. The extra-letter error is caused by The Midas Touch problem (Jacob, 1990). When the user sequentially gazes at letters of a word, besides correct letters, the user also gaze at extra letters accidentally which are also activated. The neighbor-letter error is that the user gazes at an adjacent letter (neighbor) rather than the intended letter. The missing-letter error is that the user does not gaze at the intended letter nor any neighbor. The three types of error should be handled by a practical typing system.

Pedrosa and Pimentel (2015) proposed a filter-based dwell-free typing system for practical text entry, Filteryedping, which allowed to handle the extra letters error. The system was used to find all potential words formed by filtering extra letters gazed by the user. The words are subsets of the letter sequence, which are sorted by their length and occurrence frequency in the dictionary. In their experiment, their approach enabled participants reach 19.28WPM, comparing with AltTyping (10.77WPM), the fastest dwell-based eye typing tool (Raiha and Ovaska, 2012). It turns out that the dwell-free system is much faster than dwell-based systems in practice.

However, although Filteryedping is able to handle the extra letters error, it cannot handle other errors, i.e. if the user does not gaze one or two letters of the intended

OzCHI '15 , December 07 - 10 2015, Melbourne, VIC, Australia
Copyright © 2015 ACM 978-1-4503-3673-4/15/12... $15.00
http://dx.doi.org/10.1145/2838739.2838804.

word, the system fails to recommend the intended word. In practice, it is a common case in the eye tracking area, in which calibration is not perfect, and will drift gradually. Due to overshoot or undershoot of the saccade, the user cannot always perfectly gaze at every letter of a word. In addition, if the typing system requires users to gaze at all letters of a word, it is also an extra burden.

Therefore, we propose an approach to solve the practical problems, in which users do not need to gaze at all letters of an intended word. Although the user does not gaze at the intended letter but the neighbor of the letter, or even one or two letters of a word are missing, our approach is still able to recognize and recommend the intended word. Our experimental results of simulation and actual users show that our approach has better accuracy and higher resilience to text entry errors.

GAZETRY SYSTEM

The extra-letter error, neighbor-letter error, and missing-letter error are three main problems in the dwell-free typing system. For the extra-letter error, the common way is dwell time. However, if using dwell time, the user has to fixate at a key for a while to activate it, it will affect the time taken to type a word. In fact, there are some patterns while the user types a word, e.g. gaze trace, and gaze duration. By analysing the gaze trace, we can get the letters over the trace. Based on the duration of gazed letters, we can find the letters that are dominant for the intended word. Dominant letters are letters that the user spend more time gazing at during the word entry period. Although there is no dwell time for selecting an intended letter, instinctively the user could gaze at an intended letter for a relatively longer time. Using the letter sequence and dominant letters, it can find possible words in the dictionary with high probability of correctness despite of extra irrelevant letters in the sequence.

For the neighbor-letter and missing-letter errors, we solve them using a cost function. Once a letter is detected, its neighbors are also regarded as possible input. For example, when "S" is detected, we also consider its neighbors, "A", "W", "D", "X" are potential letters. But it could bring more candidate words, and possibly it is difficult to find the intended word. In our approach, the neighbor or missing letters will be assigned a cost, and then we use the cost to control the size of possible words. Figure 1 shows the paradigm of our system. The system consists of three modules: letter sequence generation, string matching and candidate ranking.

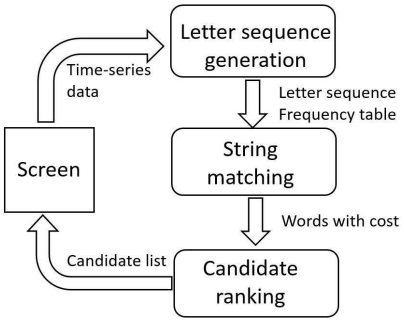

Figure 1. The paradigm of our system.

Letter Sequence Generation Module

This module is to transform keyboard coordinates to letter sequences in our system. Each gaze has a coordinate x, y on the keyboard. Given the coordinates of those letters on the keyboard, each gaze coordinate is corresponding to a nearest letter. The border coordinate of all letters is predefined as shown in Figure 2, and the red dotted box is the effective area of the key. When the coordinate does not has the nearest letter, i.e. the gaze is outside the screen "keyboard" area, we label its nearest letter as "null".

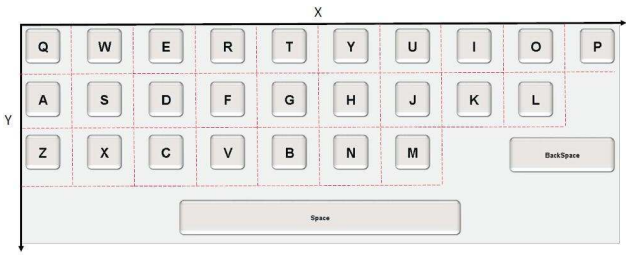

Figure 2. Keyboard coordinate.

The first step is to find the nearest letter for each time point, in which the coordinate and nearest letters are one-to-one mapping. However, raw time-series sequence contains some non-alphabet characters, e.g. "null", "space", and "backspace". Then the second step is to filter these characters, and obtain processed time-series letter sequence. Next, consecutive same letters will be merged to create the frequency table. It is also a time-series letter sequence, and the order of different letters is still maintained, though same consecutive letters are merged. The fourth step is to remove outliers caused by eye involuntary movement to obtain a more compressed letter frequency table.

After this module, the letter sequence and corresponding letter frequency table are generated. The letter sequence reflects the gaze trace, i.e. how the user gazes at the letters sequentially. The letter frequency table reflects the gaze duration of the letters, i.e. how long time the user gazes at the individual letters. If the frequency of a letter is large, it is a dominant letter as we mentioned before, and it could be a part of the intended word with a high probability.

String Matching Module

The string matching module is to find possible words with the high similarity (low dissimilarity) to the letter sequence. Inspired by the approximate string matching method (Hall and Dowling, 1980), we define the dissimilarity of two strings is measured by the cost of converting one string to the other. Then we propose the **Mo**ving **W**indow String Match**ing** (MoWing) algorithm to calculate the cost of converting the letter sequence to a word (Algorithm 1).

In the algorithm, there are two windows: the letter sequence window (*LSW*, the size is 1), and the word window (*WW*, the size is 2, which means looking one step forward). The letters of the windows are to be matched, in which there is one letter in the sequence window (*SP*), and two letters in word window (the first position as *WP1*, the second position as *WP2*). In order to calculate

total cost of converting the letter sequence to the word, we divide it into augmented cost of converting *LSW* to the *WW*.

Algorithm 1 The Moving Window String Matching Algorithm

```
Initialization :
i ← −1; j, cost ← 0
WP1 ← empty; WP2 ← word[0]
repeat
    SP ← seq[j]
    if SP = WP1 or WP2 then
        cost ← cost + 0
    else if SP ∈ neighbor(WP1 or WP2) then
        cost ← cost + neighborCost
    else
        cost ← cost + errorCost
    end if
    if SP = WP1 and WP2 then
        i ← i + 1
    else if SP ≠ WP1 then
        if SP = WP2 and SP ∈ neighbor(WP2) then
            i ← i + 1
        end if
    end if
    if i ≠ −1 and i ≠ word.length − 1 then
        WP1 ← word[i] WP2 ← word[i + 1]
    end if
    j ← j + 1
until j = seq.length
wordCost ← (word.length − i + 1) /word.length
totalCost ← cost/seq.length + wordCost
```

In the approximate string matching method, the cost of converting a letter to the same letter is 0, and the cost of converting to a different letter is 1. But in our method we consider two different cost of converting a letter (*L1*) to a different letter (*L2*). If *L2* is an adjacent letter (neighbor) of *L1*, the converting cost is smaller than 1 (from the preliminary study, the optimal one is 0.2), and we call the cost as a neighbor cost. Otherwise, the cost is 1, which is called error cost. In addition, we stipulate the two windows converting cost as the following:

1. If *SP* is the same as either *WP1* or *WP2*, the converting cost from *LSW* to *WW* is no cost;
2. If *SP* is a neighbor of either *WP1* or *WP2*, the converting cost is neighbor cost;
3. If *SP* does not satisfy above two, the converting cost is error cost.

Another important part in the algorithm is moving the two windows. In the typing system, due to the extra-letter error, the letter sequence could be much longer than a word, and the words might be a subset of the sequence. Thus, we are trying to convert the letter sequence to the word, and the *LSW* is keeping moving forward at every loop. For the *WW*, the moving condition is that all possible letters in the letter sequence have been converted into the current *WW*, as the following:

1. If *SP* is same as both *WP1* and *WP2*;
2. If *SP* is not same as *WP1*, but it is same as *WP2* or a neighbor of *WP2*.

If *LSW* has been the end of the letter sequence but *WW* has not been the end, there is a cost (word cost) for the converting process.

Candidate Ranking Module

After the string matching module, we can get converting cost from the letter sequence to each word in the dictionary. Since the converting cost reflects the dissimilarity of the letter sequence and each word, it is regarded as the ranking metric. The lower the converting cost, the higher is the similarity of the sequence of letters to the word. The letter sequence is what the user input, thus the word is probably what the user want. According to the converting cost, we rank all words in the dictionary, and those words which are at the top positions have high priority to be intended words. The goal of the text entry system is that the intended word is being at the first position or at least the first page, and the user does not need to use page up or page down. In the approach, we do not rely on the word occurrence frequency in the dictionary, and only use the converting cost to sort candidate words.

EVALUATION

In this section, we will evaluate three algorithms, MoWing in GazeTry, Filteryedping (Pedrosa et al., 2015), and a revised version of Filteryedping (FdpResived).

Experiment

Experiment 1: Simulation

We built a word generator with different type of text entry error to compare the resilience of the algorithms. For the zero-error, the letter sequence is the intended word. For the extra-letter error, the letter sequence contains all letters of the intended word and other extra random letters. For the neighbor-letter error, the letter sequence contains most letters of the intended word, a neighbor of rest letters, and other extra random letters. For the missing-letter error, the letter sequence contains most letters of the intended word, and other extra random letters. An example is shown in Table 1, '*' means random letters from 1 to 5 letters.

Error Types	Simulated Letter Sequence
Zero error	NATIONAL
Extra-error	*N*A*T*I*O*N*A*L*
Neighbor-letter error	*N*S*T*I*O*N*A*L*
Missing-letter error	*N*T*I*O*N*A*L*

Table 1. Letter Sequences in Four Errors.

For each type of error, the simulator randomly selects one thousand words from the dictionary to generate one thousand letter sequences, and this operation is repeated 100 times. Thus, totally 100 thousand letter sequences were generated for each type of error. Then the simulated letter sequences were input to the three algorithms to recommend the words based on the letter sequences.

Figure 3 shows the top-5 rate and the error rate results. The top-5 rate is the percentage of the intended words existing at the first five positions of the list of recommended words, which reflects the accuracy of a typing system. If the intended words show up at the first five positions (the first page), the user does not need to use page up and page down to search the words, which

will reduce searching time. The error rate is the percentage of the intended words being out of top 30 in the candidate set, which reflects the robustness of the typing system.

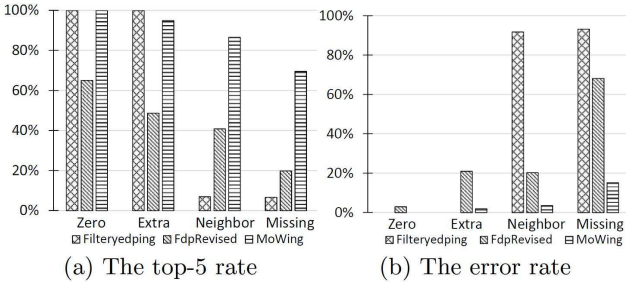

Figure 3. The results of simulation.

In Figure 3, although Filteryedping is good at handling zero-error and extra-letter error, for neighbor-letter error and missing-letter error, its top-5 rate dramatically drops to 7% and 6.7% respectively. Our MoWing algorithm is able to handle all types of error. Despite for the missing-letter error, MoWing achieved 69.6% top-5 rate and 15.2% error rate. For the extra-letter error, the top-5 rate of Filteryedping (100%) is slightly higher than MoWing (94.5%). This is because the MoWing algorithm has a bigger search space for possible words. Thus, there is a possibility that the "extra" letters could create a situation where words with the additional letters have a lower cost function than the intended word.

Experiment 2: Actual User

The interface is presented on a Dell P2314H 23-inch LED-lit monitor with a resolution of 1920 × 1080 at a frame rate of 60Hz, and are run on a Dell-PC with a Xeon(R) processor at 3.5GHz and 16GB DDR-RAM under 64-bit window 7. We use low-cost eye tracker (TheEyeTribe) attached to the monitor. The eye tracker works at a sampling rate of 60Hz. Nine-point calibration was used in the study, and the tracked gaze coordinates data were mapped onto the mouse cursor. The participants do not need to recalibrate unless they are uncomfortable with the accuracy of the calibration.

The two participants had at least 3-hour training to control the cursor using eye movement, and type characters using gaze. In the experiment, there were five sessions. In each session, each participant needs to type 200 words using our eye typing system. These words were randomly selected one by one from the dictionary which has 5000 common used words (Davies, 2011). In the experiment, a random word is selected and displayed at the top of the screen. The user is required to type the word using eye movement and gaze. Like the finger swipe-based typing, they only need to "swipe" the letters of the word using gaze. Upon completing the word entry, they have to gaze at the "SPACE" key to indicate the end of entry. The system will then display the candidate words at the bottom of the screen. Since we evaluate the performance of three algorithms, the system will simultaneously show top 5 words recommended by three algorithms based on the same input. The participants keep typing the given random words, and do not need to choose the word from candidate words. Then a new random word will be displayed at the top of the screen.

Figure 4 and Figure 5 show the results of two participants. The results indicate that the accuracy (top-5 rate) of the MoWing algorithm is slightly higher than Filteryedping in the real eye typing. From participant 1 results, the highest top-5 rate of MoWing is 91.5% while Filteryedping's is 86%. From the result of the second participant, the highest top-5 rate of MoWing is 97.5% while Filteryedping's is 93.5%. The results also indicate that the error rate of MoWing is lower than Filteryedping. For the first one, the lowest error rate of MoWing is 3% while Filteryedping's is 9.5%. For the second one, the lowest error rate of MoWing is 2% while Filteryedping's is 6.5%.

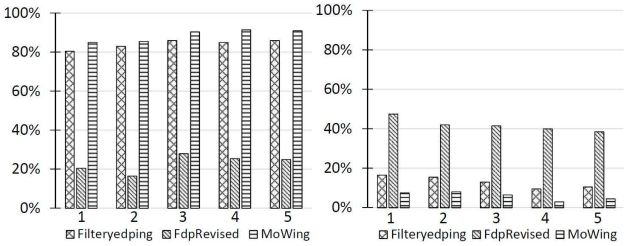

Figure 4. The results of participant 1.

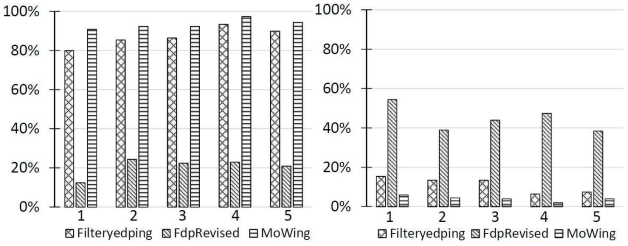

Figure 5. The results of participant 2.

Comparing with simulation results, the results of actual users indicate less difference of the top-5 rate and error rate between MoWing and Filteryedping. That is because two users always try to gaze at every letters of the intended words so that MoWing does not show enough performance in the neighbor-letter error and missing-letter error cases. Although the number of users is limited, the tentative experiment can be guides for the future work.

CONCLUSION

In order to increase the communication ability of physically challenged people, we have proposed a dwell-free eye typing system, GazeTry, which allows users to enter words using gaze. Comparing with existing eye-typing works, our system does not require people to spend a duration fixating a key to input the letter. With GazeTry, the user is able to simply type a word by gazing sequentially at the letters of the word. The experimental results show that comparing with the recent dwell-free typing system, Filteryedping, our approach has better accuracy and higher resilience to text entry errors which would improve the effectiveness of communication.

ACKNOWLEDGMENTS

This work is a collaboration with the Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY).

REFERENCES

Davies, M. (2011). Word frequency data from the Corpus of Contemporary American English (COCA).

Hall, P. A., & Dowling, G. R. (1980). Approximate string matching. ACM computing surveys (CSUR), 12(4), 381-402.

Jacob, R. J. (1990, March). What you look at is what you get: eye movement-based interaction techniques. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 11-18). ACM.

Kristensson, P. O., & Vertanen, K. (2012, March). The potential of dwell-free eye-typing for fast assistive gaze communication. In Proceedings of the Symposium on Eye Tracking Research and Applications (pp. 241-244). ACM.

MacKenzie, I. S., & Zhang, X. (2008, March). Eye typing using word and letter prediction and a fixation algorithm. In Proceedings of the 2008 symposium on Eye tracking research & applications (pp. 55-58). ACM.

Majaranta, P., Ahola, U. K., & Špakov, O. (2009, April). Fast gaze typing with an adjustable dwell time. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 357-360). ACM.

Majaranta, P. (Ed.). (2011). Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies: Advances in Assistive Technologies. IGI Global.

Pedrosa, D., Pimentel, M. D. G., Wright, A., & Truong, K. N. (2015). Filteryedping: Design challenges and user performance of dwell-free eye typing. ACM Transactions on Accessible Computing (TACCESS), 6(1), 3.

Pedrosa, D., Pimentel, M. D. G., & Truong, K. N. (2015, April). Filteryedping: A Dwell-Free Eye Typing Technique. In Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (pp. 303-306). ACM.

Räihä, K. J., & Ovaska, S. (2012, May). An exploratory study of eye typing fundamentals: dwell time, text entry rate, errors, and workload. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 3001-3010). ACM.

Sarcar, S., Panwar, P., & Chakraborty, T. (2013, September). EyeK: an efficient dwell-free eye gaze-based text entry system. In Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction (pp. 215-220). ACM.

Urbina, M. H., & Huckauf, A. (2010, March). Alternatives to single character entry and dwell time selection on eye typing. In Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications (pp.315-322).ACM.