


Robust Eye-Based Dwell-Free Typing

Yi Liu ^a, Bu-Sung Lee^b, and Martin J. McKeown^c

^aNanyang Institute of Technology in Health and Medicine, Interdisciplinary Graduate School, Nanyang Technological University, Singapore; ^bSchool of Computer Engineering, Nanyang Technological University, Singapore; ^cDepartment of Medicine, Pacific Parkinson's Research Centre, University of British Columbia, Vancouver, Canada

ABSTRACT

For a subset of physically challenged people, assistive technologies, such as alternative form of text entry, can be of tremendous benefit. However, speed of text entry with current methods limits their more widespread adoption. Eye gaze technologies have potential for text entry, but still tend to be relatively slow. Recently dwell-free eye-typing systems have been proposed, but can be vulnerable to common text entry problems, such as selection of the wrong letters. In this article, a recognition approach for inferring the words which the user intends to type is proposed. The method is robust to missing letters and even when a neighboring letter on the keyboard is incorrectly selected. Simulation and experiment results suggest that our proposed approach has better accuracy and more resilience to common text entry errors than other currently proposed dwell-free systems.

1. Introduction

In modern society, access to, and communication via, the internet is important for adequate quality of life. However, some people with severe motor disabilities have difficulty in communicating using their hands for typing, such as people with amyotrophic lateral sclerosis (ALS) or locked-in syndrome, conditions with nearly 120,000 new cases diagnosed worldwide annually (Majaranta, Aoki, Donegan, Hansen, & Hansen, 2011). People with these conditions may not be able to use even speech, and control of the eyes may be the only means to communicate. If eye tracking can be used to achieve text entry, called eye typing, this could provide a reliable method of communication.

Research on eye tracking dates back to the late 18th century when people's saccadic eye movements were investigated in a reading task with the first eye tracker (Biswas & Langdon, 2015; Huey, 1908). For a long time, eye-tracking research was mainly focused on cognitive analysis (Just & Carpenter, 1976; Monty & Senders, 1976; Rayner, 1977, 1998), exploring what eye movements revealed about perceptual and cognitive processes (Jacob & Karn, 2003; Ohno, 2007; Underwood, 2005). More recently, with advances in both eye-tracking technology and computer systems, eye-tracking research has gradually focused on human-computer interactions. Eye movements can be an input source to control electronic devices and applications where eye trackers are used as an interaction device (Murata, 2006; Zander, Gaertner, Kothe, & Vilimek, 2010). This role of eye tracking started to become significant for people with physical impairment (Adjouadi, Sesin, Ayala, & Cabrerizo, 2004; Donegan et al., 2009; Kocejko, Bujnowski, & Wtorek, 2009; Su, Wang, & Chen,

2006) and helps them conduct a more efficient communication (Caligari, Godi, Guglielmetti, Franchignoni, & Nardone, 2013; Spataro, Ciriaco, Manno, & La Bella, 2014). Among communication techniques, eye typing is a very promising area (San Agustin, Skovsgaard, Hansen, & Hansen, 2009), and achieving text entry through the focus of the gaze can now provide a means of communication for people who are only capable of moving their eyes (Majaranta & Räihä, 2002).

Speed of text entry is a crucial limitation to current eye-typing systems (Hansen, Hansen, Nielsen, Johansen, & Stegmann, 2002; Kotani, Yamaguchi, Asao, & Horii, 2010; MacKenzie & Zhang, 2008; Räihä & Ovaska, 2012). The early eye-typing systems allowed users to type only two or three words per minute, which is impractical for communication. Most eye-typing systems employ a virtual keyboard on the screen, with the gaze point of the user on the virtual keyboard being monitored by an eye tracker. Based on the analysis of gaze behavior, the typing system estimates which letters the user is fixating on and infers whether the letter is actually what the user wants to type. With such an approach, there are two main reasons for slow text entry: (1) *Selection method*, users need to fixate their gaze on a key for a minimum interval (the dwell time—typically 200–600 ms (Jacob, 1995)), which limits the text entry speed; and (2) *Text entry method*, early eye-typing systems only supported complete, individual character input, i.e., if a user wanted to input “hello,” they had to select five separate, individual letters.

In order to increase the speed of text entry, researchers have developed a number of improvements. To increase the typing speed when selecting individual characters, the dwell time can be reduced or dynamically adjusted (Majaranta, Ahola, & Špakov, 2009), or the keyboard can be modified

from the standard QWERTY layout to more efficient forms (Hansen, Tørning, Johansen, Itoh, & Aoki, 2004; Sarcar, Panwar, & Chakraborty, 2013; Urbina & Huckauf, 2010; Ward & MacKay, 2002). However, the user might get rapidly tired when attempting to fixate for every letter of every word. A complementary approach is to predict the next letter(s) and/or words based on previous entries when typing (MacKenzie & Zhang, 2008; Urbina & Huckauf, 2010), enhancing overall communication rate. Although the speed of text entry is increased, it is still limited by dwell time which requires the user to fixate at the correct keys.

Recently, the concept of “dwell-free” typing has been proposed (Kristensson & Vertanen, 2012). Dwell-free typing allows the user to type a word by just gazing at letters sequentially rather than pausing at each individual letter for at least the dwell time. The basic idea is that such a system should be able to recognize the intended word based on the user’s gaze trace. Since each word consists of an ordered sequence of letters, the word can often be recognized based on the sequential letters detected and extracted from user’s gaze trace. For example, a gaze trace of “Hello” in the system as shown in Figure 1. The user will gaze at “H,” and then saccade through “H”→“E”→“L”→“O.” (A saccade is the type of eye movement where the gaze rapidly jumps from one point to another.) Note that during the typing period, the user does not need to select the individual letters with dwell time. Such an approach allows users to focus on the entire word while typing instead of the individual letters.

Kristensson and Vertanen (2012) investigated the potential of dwell-free eye typing with a QWERTY-based keyboard layout. They demonstrated that dwell-free eye typing is theoretically much faster than existing eye-based text entry techniques. However, their system assumed an ideal case in which the word that the user intends to type is completely known in advance and only accepts the input letter corresponding to the next letter in the word; if the next letter selected does not fit the known word, it intentionally ignores it. Therefore, words can be written by only accepting correct letters, even if the user looks at extra letters. While useful as a demonstration, clearly in practice it is impossible to know what the user wants to input in advance, and such noisy letter sequences with extra letters must be handled by practical dwell-free techniques in eye typing.

Pedrosa, Pimentel, Wright, and Truong (2015) proposed a filter-based dwell-free typing system for practical text entry, called Filteryedping, which is able to handle the “extra letters”

error. The system was used to find all possible words formed by filtering extra letters gazed at by the user. The possible 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 an average of 14.75WPM, comparing with AltTyping (10.77WPM), the fastest dwell-based eye-typing tool (Majaranta et al., 2009; Räihä & Ovaska, 2012). After 2 hr of practice, the fastest participant typed at 19.28WPM with Filteryedping. Thus, it appears that dwell-free systems have the potential to be much faster than the dwell-based systems in practice.

Although Filteryedping is able to handle the extra-letter error, it cannot handle missing-letter errors, i.e., the intended word must be a subset of the letter sequence gazed by the user. If the user omits gazing at one or two letters of the intended word, the system fails to recommend the intended word. In practice, with drift and imperfect calibration of the eye-tracking apparatus, as well as possible overshoot and/or undershoot of saccades, missing letters must also need to be accounted for.

In order to solve the practical issue, Liu, Zhang, Lee, Lee, and Chen (2015) proposed a dwell-free eye-typing system, GazeTry, with the Moving Window String Matching algorithm (MoWing) which is able to handle missing letters. Although the system out-performed previous systems in the missing-letter case and overall accuracy improved for expert users, it sacrifices its performance in the more common scenario of extra-letter errors, especially for novices. As the number of extra letters increases, its accuracy decreases dramatically.

Therefore, in this article, we propose an “intelligent” dwell-free approach which is robust to the common errors afflicting eye typing: extra letters, missing letters, and incorrect character selection that is a neighbor of the desired letter. Our experimental results from both simulations and field testing with users suggest that such an approach results in improved accuracy. We thus develop a dwell-free eye-typing system which achieves finger swipe-like typing.

The contributions of the current work are:

- (1) We investigate common text entry errors in practical eye-typing compared with traditional finger-typing systems.
- (2) We propose a state transition model recognition approach to infer the word that the user intends to type.
- (3) We store the word dictionary in a prefix tree structure to accelerate the word recommendation.

2. Dwell-Free Eye-Typing Systems

In an eye-typing system, there are five steps: calibration, detection, recognition, prediction, and construction. Compared with traditional finger-typing systems, calibration and recognition are two unique steps in an eye-typing system. Although some calibration-free gaze estimation systems have been proposed and perform well on saliency analysis (Alnajjar, Gevers, Valenti, & Ghebreab, 2013; Chen & Ji, 2011, 2015;



Figure 1. The gaze trace of “Hello.”

Sugano, Matsushita, & Sato, 2010; Wolfe & Eichmann, 1997), the reported accuracy is relatively low compared to commercial eye trackers; thus they are less suitable for eye typing. The calibration process maps the user's eye movement to the coordinates on the screen. This necessitates a personal calibration process with the eye tracker, since each person has different eye movement characteristics. The eye-tracking software models individual-specific eye movements in order to estimate gaze accurately. Since the calibration process is hardware dependent, researchers investigating eye-typing usually assume *a priori* that calibration is acceptable, or make it more reliable through periodic recalibration (e.g., every 10–15 min). Even so, small drifts in calibration can take place, so any practical pattern recognition system should be robust to this possibility (Zhang & Hornof, 2014).

The recognition step is the key step in an eye-typing system and is the focus of our work. Compared with a finger-typing system, there are additional challenges, e.g., inaccurate eye movement or gaze, as well as small calibration drift, and sometimes what the system detects is not even what the user intends to input. Therefore, the main goal of the recognition step is to recognize the correct intended words based on a noisy input from the user. This process does not rely on the context and semantic analysis which can be applied in subsequent prediction and construction steps.

Due to issues at calibration and detection steps, there are two further problems that make the intended word recognition challenging in a dwell-free system:

- *The Midas Touch problem* (Jacob, 1990): “everywhere you look, the command will be activated,” that means you cannot look anywhere without issuing a command. Things might be selected against the user's inclination (Jacob, 1991).
- *The low accuracy or drift problem*: the fovea, the part of the retina with highest visual acuity, is restricted to the central 5 degrees, and yet the accuracy of measured point of gaze is about 0.5 degrees. Also, even if a newly calibrated device is quite accurate, the calibration may start to drift (Majaranta & R  ih  , 2002).

In practice, the two problems usually occur together, leading to three types of errors: extra-letter errors, neighbor-letter errors, and missing-letter errors. An extra-letter error may be caused by The Midas Touch problem. When the user sequentially gazes at letters of a word, besides the correct letters, the user also gazes at extra letters accidentally which are also detected. The neighbor-letter error occurs when the user gazes at an adjacent letter rather than the intended letter. The missing-letter error is that the user does not gaze at the intended letter nor any neighbor, totally omitting the letter. A practical dwell-free typing system should be robust to these three types of error.

Previous approaches have assumed the word being typed is known *a priori* (Kristensson & Vertanen, 2012), have filtered extra letters, but failed to handle neighbor-letter and missing-letter errors (Pedrosa, Pimentel, & Truong, 2015; Pedrosa et al., 2015), or have significantly reduced accuracy in extra-letter errors (Liu et al., 2015).

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 parts of a word are missing, our approach is still able to recognize and recommend the intended word. Our simulation and experimental results show that our approach has better accuracy and higher resilience to text entry errors.

3. Methodology

The extra-letter error, neighbor-letter error, and missing-letter error are three main problems at the recognition step in the dwell-free typing system.

For the extra-letter error caused by *The Midas Touch problem*, if we do not rely on other devices (e.g., electroencephalogram (EEG) to detect special event-related potentials (Guan, Thulasidas, & Wu, 2004; Zander et al., 2010)), the only way to avoid this problem is using dwell time. However, if dwell time is used, the user has to fixate at a key for a minimum time to activate/select it, increasing the overall time taken to type a word. However, by analyzing the gaze trace, we can extract the letters that the user has gazed at, and then obtain the letter sequence. Based on the duration of gazed letters, we can find the letters that are dominant for the intended word. Dominant letters are letters where the user spends more time gazing at during the word entry period. Although there is no fixed dwell time for selecting an intended letter, the user can gaze at an intended letter for a relatively longer time (e.g., Figure 2 shows the heat map of the gaze while typing “hello”). The letter is called a dominant letter, because it could be a part of the intended word. Using the letter sequence and dominant letters, it is possible to find words in the dictionary with high probability of being correct despite the presence of possibly extraneous letters in the sequence.

For the neighbor-letter error and missing-letter error caused by *the drift problem*, our approach uses a neighbor weight. 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” as potential letters. While this can create many more candidate words, we use a common states mapping score (described below) to restrict the number of potential words.

3.1. Common States Mapping

In the eye-typing system, the actual eye movement is mapped to its relative position (coordinate x, y) on the virtual keyboard.

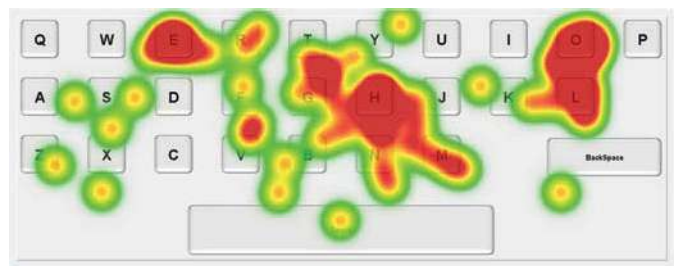


Figure 2. The gaze heat map of “hello.”

Through initial data processing, $\langle x, y, l \rangle$ tuples can be generated, where x and y indicate the location of the gaze and l indicates the nearest letter at the location. According to the saccade-fixation model (Salvucci, 1999) shown in Figure 3a, the saccade is the rapid eye movement from one location to another, and the fixation is stationary pointing over one target. Intuitively, the fixations indicate the intent of the user while typing words. Therefore, the tuples of fixations are extracted, in which the saccade-fixation detection algorithm (Kumar, 2007) is implemented in the article.

If tuples have the same nearest letter during a period of time, they belong to the same intended fixation location; thus we assign them into the same state. The state is interpreted by a tuple $\langle l, t \rangle$, where l indicates the nearest letter at the state and t indicates the time interval of the state. Thus, the state is consisting of two elements in the fixation location domain (l) and time domain (t), respectively. The process of typing a word is converted into the states transition shown in Figure 3b. Since the states directly indicate the user's typing, such as which letters the user gazes at sequentially, and how much time the user spends on each letter, these states are regarded as observable states, and the states transition is an *observable transition* which is taken as (possibly noisy) user input in the recognition step. We then need to recommend the potential words (including the intended word) through the input.

Every word in a dictionary consists of certain ordered letters. If we regard every letter inside as a state, the whole word is consisting of states with certain one-direction transition, and we define potential words as a *potential transition* while processing the input. Potential transitions are generated by words from the dictionary, so each word is corresponding to a potential transition, and each letter indicates a state in the potential transition, in which there is only the location domain, i.e., the letter itself, but no time domain like the observable transition. Consecutive same letters are represented by a state, e.g., "apple" has four states, a, p, l, e.

In the proposed method, drawing on the idea of the longest common subsequence method, our goal is to find the longest common states between *potential transition(s)* and *observable transition(s)* which reflect their mapping similarity. We take the set similarity as a simple example. Given two sets, A and

B , according to Sorensen–Dice coefficient, the two sets similarity is measured by $2|A \cap B|/(|A| + |B|)$, and the goal is to find the maximum $|A \cap B|$, which is the size of the common elements. Although set elements are non-ordered and non-repeated, the basic idea is similar.

We use a 2-D table to describe the method. The row represents the *observable transition*, and the column represents *potential transition* as shown in Table 1. There are three conditions as follows to fill in the table:

- If the locations of two transitions are the same, the corresponding cell is filled with value of the time domain of the observable transition.
- if the locations are adjacent, i.e., one is a neighbor (up, down, left, right) of the other, the cell is the value multiplied by a weight (Figure 4 shows preliminary results on neighbor weight in which the optimal weight is around 0.4).
- Otherwise, the cell is zero.

Finally, a m -by- n matrix is generated (Algorithm 1). One example is shown in Table 2, the observable transition is "SCXAR" with the corresponding time interval, and the time interval is processed depending on sampling frequency. The potential transition is a word, "car."

Instinctively, while the user is typing, each fixation is related to only one intended letter, but several fixations can

Algorithm 1. Matrix generation algorithm.

```

1: function BUILDMATRIX(observable, potential)
2:    $a_{ij}$  is the  $i$ th row,  $j$ th column cell of Matrix
3:    $S_i$  is the  $i$ th state of observable
4:    $S'_j$  is the  $j$ th state of potential
5:   if  $S_i.l = S'_j.l$  then
6:      $a_{ij} \leftarrow S_i.t$ 
7:   else if  $S_i.l \in \text{neighbor}(S'_j.l)$  then
8:      $a_{ij} \leftarrow S_i.t * \text{weight}$ 
9:   else
10:     $a_{ij} \leftarrow 0$ 
11:   end if
12:   return Matrix
13: end function

```

Table 1. The observable transition and potential transition.

	S_1	S_2	S_3	...	S_n
S'_1	a_{11}	a_{12}	a_{13}	...	a_{1n}
S'_2	a_{21}	a_{22}	a_{23}	...	a_{2n}
S'_3	a_{31}	a_{32}	a_{33}	...	a_{3n}
...
S'_m	a_{m1}	a_{m2}	a_{m3}	...	a_{mn}

Table 2. An example of the transition mapping.

	$\langle S, 10 \rangle$	$\langle C, 20 \rangle$	$\langle X, 10 \rangle$	$\langle A, 15 \rangle$	$\langle R, 15 \rangle$
C	0	20	2	0	0
A	2	0	0	15	0
R	0	0	0	0	15

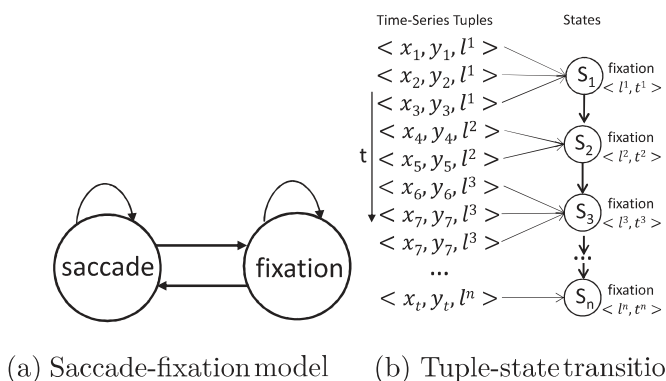


Figure 3. State transition.

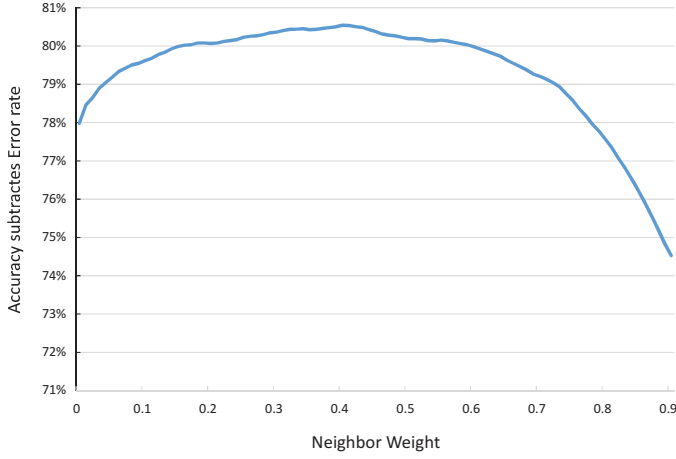


Figure 4. Preliminary results on neighbor weight.

be related to the same intended letter which is a cause of extra-letter errors. Based on this fact, we define the two-transition mapping as a many-to-one mapping shown in Figure 5, and the method in the article is called the longest common states mapping method (LCSMapping). As the name suggests, we are trying to find the most common states between the input stream (the observable transition) and each word (the potential transition). One state in the observable transition can only map to one state in the potential transition. Since the transition is ordered, if an observable transition S_1 is mapping S'_2 in potential transition, S_2 cannot map the S'_1 which is earlier than S'_2 . According to the above two constraints, the problem is modeled by the following mathematical formula (3), which is a dynamic programming problem. The solution is the longest common states, and we solve it using a recursive formula in Algorithm 2 (the detailed proof is shown in Appendix A).

$$\begin{aligned} \text{Maximize : } & \sum_{j=1}^n a_{ij}, i \in [1, 2, \dots, m] \\ \text{Subjectto : } & \forall a_{i_1 j_1}, a_{i_2 j_2}; \\ & \text{if } j_1 < j_2, \text{ then } i_1 \leq i_2 \end{aligned} \quad (1)$$

3.2. Potential Transition Scoring

After common states mapping, we extract the longest common states between *potential transition* and *observable transition*. Then, we need to infer the hidden transition which consists of the next letter of the word that the user intends to type. In our method, we use a variant of Sorensen–Dice coefficient in the following formula (2).

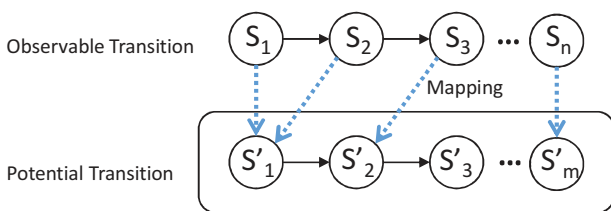


Figure 5. Transition mapping.

$$\text{Score} = \frac{|A_{(A \cap B)}|}{|A|} + \frac{|B_{(A \cap B)}|}{|B|}, \quad (2)$$

where $|A_{(A \cap B)}|$ indicates the time interval of the longest common states and $|A|$ indicates the overall time interval; thus $|A_{(A \cap B)}|/|A|$ is the proportion that the observable transition maps to the potential transition. $|B_{(A \cap B)}|$ indicates the number of states in potential transition that are mapped by the observable transition and $|B|$ indicates the number of states in potential transition, so $|B_{(A \cap B)}|/|B|$ is the proportion of potential transitions mapped by observable transitions. The score range is same as Sorensen–Dice coefficient, from 0 to 2.

3.3. Candidate Ranking

After deriving the potential transition scoring, the score of the potential transition created by each word in the dictionary can be computed. Since the score reflects the similarity of the input and each word, it is regarded as a ranking metric. The larger the score, the higher is the similarity of the input to the word. The *observable transition* is what the user inputs; thus *potential transition* is the word which the user probably intends to type. According to the score, we rank all words in the dictionary, and the higher the score of a word, the closer to the beginning of the candidate list it will be. 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 scroll. Note that in the current approach, we do not rely on the word occurrence frequency in the dictionary, and only use the score to sort the candidate words.

4. Data Structure

Individually mapping the potential transitions created by each word in the dictionary is consuming in both time and space, because every time we need to create a new matrix for each newly selected word from the dictionary for comparison. The comparison time is proportional to the size of the total number of characters in the dictionary. From Algorithm 2, if we add a new state after the last state of potential transition, the longest common states of original matrix does not change, and the longest common states of new matrix only relates to the last state, which is a first-order system. Therefore, it allows us to create an augmented matrix.

In this work, we apply a prefix tree structure (Trie) to maintain the words of the dictionary. As shown in Figure 6,

Algorithm 2. Longest common states mapping algorithm.

```

1: function LCState(Matrix)
2:    $f(1, j) = \sum_{k=1}^j a_{1k}$ 
3:    $f(i, 1) = \max(a_{i1}, a_{21}, \dots, a_{n1})$ 
4:    $m, n \leftarrow \text{Matrix}$ 
5:   if  $i, j > 1$  then
6:      $f(i, j) = \max(f(i, j-1) + a_{ij}, f(i-1, j))$ 
7:   end if
8:   return  $f(m, n)$ 
9: end function

```

each node contains a letter, its next node, and a flag that indicates whether it is able to form a word through the current node and previous nodes. From the tree, if words have the same prefix, we only operate on them once. In addition, for one branch of the tree, we only need to build one augmented matrix instead of building matrices for each word. From our preliminary experiment results, using the prefix tree structure cuts operation time by nearly half compared to brute force method mentioned earlier. Compared with the recent dwell-free approaches, Filterypeding (the execution time of FdpRevised is the same as FdpRevised's) and MoWing, the execution time of our approach using the tree structure is reduced by 11% and 46%, respectively (Figure 7).

5. Evaluation

In this section, we evaluate four standard dwell-free eye-typing approaches, the proposed LCSMapping approach, Filterypeding (Pedrosa et al., 2015), a revised version of Filterypeding (FdpRevised), and MoWing in GazeTry (Liu et al., 2015).

5.1. Filterypeding, FdpRevised, and MoWing Approaches

The approach in Filterypeding only relies on the sequential gazed letters. The basic method is to find all possible words which consist of subsequences of the input letter sequence. If the input sequence contains all letters of the intended word,

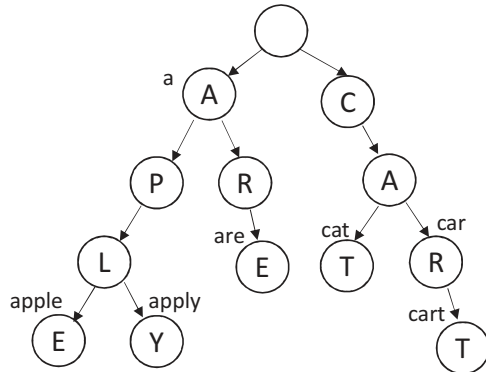


Figure 6. Transition mapping.

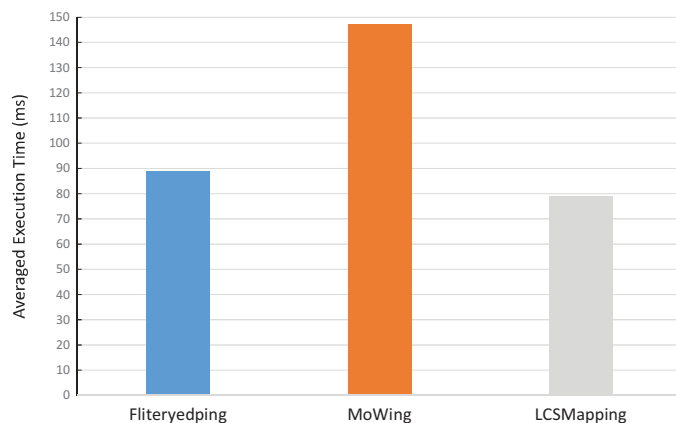


Figure 7. The comparison of execution time per word.

the word will be shortlisted by the system. Then, these possible words in the candidate list are sorted based on the length and frequency in dictionary using the following formula:

$$\text{score}(\text{word}) = \log(\text{freq}(\text{word})) + w * \text{length}(\text{word}), \quad (3)$$

where $\text{freq}(\text{word})$ is the number of times the word appears in the Corpus of Contemporary American English (COCA) (Davies, 2011), $\text{length}(\text{word})$ is the number of letter in the word, and w is a weight. The higher the score of a word, the closer to the beginning of the candidate list it will be. In their study, the optimal value of the weight w was found to be 1.09.

Since the original Filterypeding algorithm fails to solve the neighbor-letter error and missing-letter error, we simply revise the matching algorithm part in order to handle the neighbor-letter error (Algorithm 3). The revised one is FdpRevised. The ranking algorithm is the same as the original Filterypeding using formula (3).

The MoWing in GazeTry relies on a cost function which calculates the converting cost between the input letter sequence and each word, and then sorts the candidates according to the cost. In this approach, there are two moving windows, one is for the input letter sequence and the other is for the candidate word, and it calculates the converting cost between the two windows while keeping moving the two windows forward according to corresponding rules, and the summation of the each window converting cost is taken as the final cost. This approach is greatly affected by extra letters. Although it focuses on local optimization, as the number of extra letters increases, the total error rate also increases.

5.2. Experiments

Two experiments were carried out. In Experiment 1, a simulator was built to simulate the user input with different types of errors, i.e., extra-letter errors, neighbor-letter errors, and missing-letter errors. In Experiment 2, 10 actual users taking part in the experiment were instructed to type words into the system.

Algorithm 3. The FdpRevised's matching algorithm.

```

1: function MATCH(word, sequence)
2:    $i, j \leftarrow 0$ 
3:   while  $i \leq \text{word.length}$  do
4:     validity  $\leftarrow$  false
5:     while  $j \leq \text{sequence.length}$  do
6:       if isNeighbor(word[i], sequence[j]) then
7:         validity  $\leftarrow$  true
8:         break
9:       end if
10:       $j \leftarrow j + 1$ 
11:    end while
12:     $i \leftarrow i + 1$ 
13:  end while
14:  return validity
15: end function

```

Experiment 1: Simulation

We built a word generator which generates sets of words simulating different text entry errors to compare the resilience of the different algorithms. In the extra-letter error, the simulator created a random noisy input, e.g., irrelevant letters, into a word which simulated that the user gazed at extra letters accidentally. In addition, for the simulated input, the time interval of intended letter was around 10–50% longer than extra irrelevant letters, which was to simulate that the user spent relatively longer time on the intended letters. In the neighbor-letter error, the simulator included some neighbor of letters in a word, which simulated that the user gazed at a neighbor of an intended letter rather than the letter itself. In the missing-letter error, the input lacked some letters of a word, simulating that the user does not gaze at the intended locations nor neighbors. The simulation configuration was the same as Liu et al. (2015).

For each type of error, the simulator randomly selects 1000 words from the dictionary to generate 1000 letter sequences, and this operation was repeated 100 times. Thus, a total of 100,000 letter sequences were generated for each type of error. The set of simulated letter sequences were used as input to the four algorithms to establish the recommended words based on the letter sequences.

Figure 8 shows the top five rate results with the extra-letter error. The top five 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. The x -axis is the number of simulated extra letters, which indicates how many irrelevant letters the simulator generates into the word. From the results, when there was no extra letter (i.e., the number of the extra letter is 0) the top five rates of Filteredyping, MoWing, and LCSMapping were almost the same (approaching 100%); however, the top five rate of FdpRevised was only 62.8%. As the number of the extra letter increased, the top five rate of MoWing decreased dramatically. When the number of the extra letters was 1 or 2, the top five rate of Filteredyping was 0.2% higher than LCSMapping, likely because the LCSMapping algorithm had a bigger search space for possible words. If the score of the intended word was not ranked in the first five positions in the candidate list,

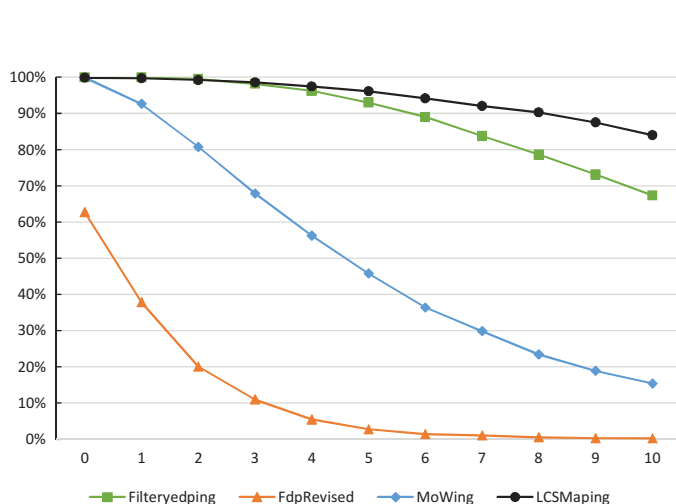


Figure 8. The top five rate in extra-letter error.

it is flagged as unsuccessful detection of intended word. When the number of the extra letters was larger than 2, LCSMapping gradually performed better than Filteredyping, with LCSMapping performing 20% higher with 10 extra letters. From our preliminary experiment of actual users, the typical number of extra letters is around 5, and LCSMapping accuracy at this point is around 4% higher than other algorithms. Statistical analysis showed significant differences between LCSMapping and Filteredyping ($p = 0.0174 < 0.05$). However, since FdpRevised allowed a word to consist of neighbors of the gazed letters, the search space was much larger than the original Filteredyping approach. When the number of the extra letters increased, the top five rate of FdpRevised exponentially decreased to nearly 0%.

To further understand the accuracy of the different approaches, we were also interested in the average position of the intended word in the candidate list. Figure 9 shows the comparison of the average position of the different method, in which the x -axis is the number of simulated extra letters and the y -axis is the average position. This result also supports the top five rate in Figure 8. When the number of the extra letters is 0–4, Filteredyping and LCSMapping have the similar results, i.e., the average position is around 1 which means the intended words are always listed in the first position. As the number of extra letters increases, the average positions of Filteredyping gradually increase and exceed that of LCSMapping, indicating deteriorating performance. The average position of FdpRevised is high even when there is a single extra-letter error indicating clearly that it is unable to cope with the extra-letter error. In the case of MoWing, the average positions increase as the number of extra letter is increasing again showing its inability to cope with the extra-letter error problem.

Figure 10 shows the performance of four algorithms when there are both neighbor-letter and missing-letter errors. In Figure 10a, the x -axis is the number of letter which is replaced by its neighbor. In Figure 10b, the x -axis is the number of missing letter in the word. Due to the word length limitation, only parts of the word can be changed to neighbors or missing; otherwise it is not meaningful if the whole word information is lost. Thus, from our preliminary investigation,

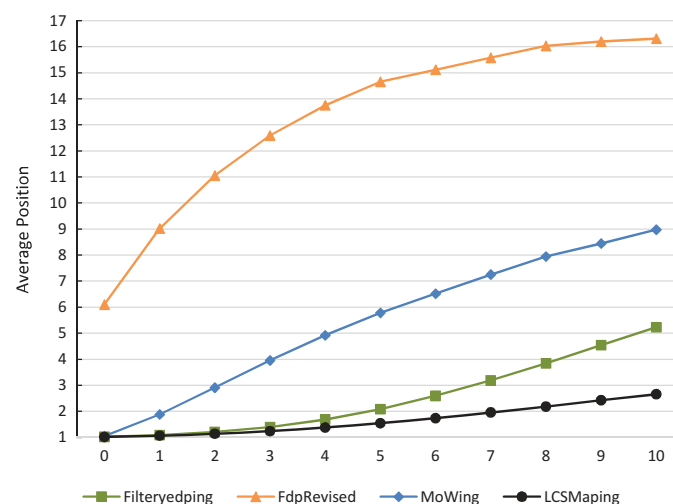


Figure 9. Average position of words in the candidate list.

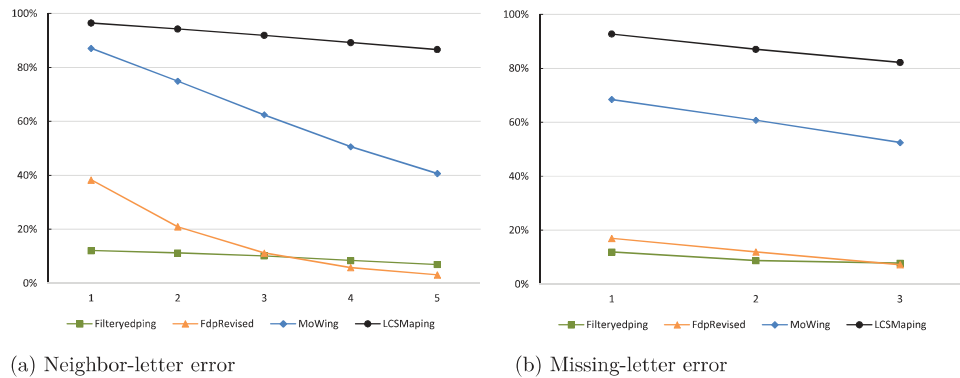


Figure 10. The top five rate in two errors.

we set the neighbor-letter error from 1 to 5, which means that, at most, five letters of the word were changed to their neighbors. The missing-letter error was set to 1 to 3, which means at most three letters of the word can be missing.

The results show that LCSMapping is able to handle both neighbor-letter and missing-letter errors, in which the top five rate is still higher than 80% even though the neighbor-letter error is 5 and missing-letter error is 3, which is better than MoWing (40.7% and 52.5%, respectively). However, although the neighbor-letter and missing-letter errors are 1, the top five rate of Filteryedping is around 12%. Thus, Filteryedping is not suitable for handling text entry with these types of errors. Although the top five rate of FdpRevised (37.85%) is much higher than Filteryedping (11.89%), its performance decreases dramatically once the error increases due to its poor search-space control.

The results of simulation show that LCSMapping outperforms other approaches in both neighbor-letter and missing-letter errors without sacrificing performance in extra-letter errors. The Filteryedping approach had a good performance in the extra-letter error case, while in both neighbor-letter and missing-letter errors cases it had very poor performance. Although the FdpRevised approach improves the accuracy in the neighbor-letter error case, its overall performance is relatively low due to its poor search-space control. While the MoWing approach is able to solve the problem of both neighbor-letter and missing-letter errors cases, it sacrifices the accuracy in the extra-letter error case, which, in fact, is the most common error case in practical eye-typing environment.

Experiment 2: Actual User

We evaluated the performance of the proposed approach via actual user experiments. The interface was presented on a Dell P2314 H 23-inch LED-lit monitor with a resolution of 1920×1080 at a frame rate of 60 Hz and were run on a Dell-PC with a Xeon(R) processor at 3.5GHz and 16GB DDR-RAM under 64-bit Windows *T M* 7. We used a low-cost eye tracker (TheEyeTribe¹) put under the monitor shown in Figure 11. The eye tracker had a sampling rate of 60 Hz. Nine-point calibration was used in the study. The tracked gaze coordinates data were mapped onto the mouse cursor using moving average smoothing algorithms provided by the device. A chin rest, used to help participants understand that doing eye typing they only need to

move their eyes and also help to keep calibration, was placed 50 cm in front of the monitor. The participants did not need to recalibrate the tracker unless they were uncomfortable with the accuracy of the eye tracker, or there is a big drift.

We recruited 10 volunteers (seven males, three females; 23–35 years) from the local university. All of them had 4 to 8 hr per day computer usage and had normal vision. Three of them had rich prior experience with eye tracking, and seven of them had no such eye-tracking experience before.

All participants had at least half an hour training to control the cursor using eye movements and type letters using gaze. In the experiment, there were five sessions, but from results the users did not demonstrate significant different performance among the five sessions. Between sessions, the participants had a 5-min break and then underwent recalibration. In each session, each participant needed to type 200 words using the proposed eye-typing system. These words were randomly selected one by one from the dictionary which has 5000 commonly used words in the COCA (Davies, 2011).

In the experiment, a random word was selected and displayed at the top of the screen. The user was required to type the word using eye movement and gaze. Since the proposed system is a dwell-free typing system, they did not have to spend

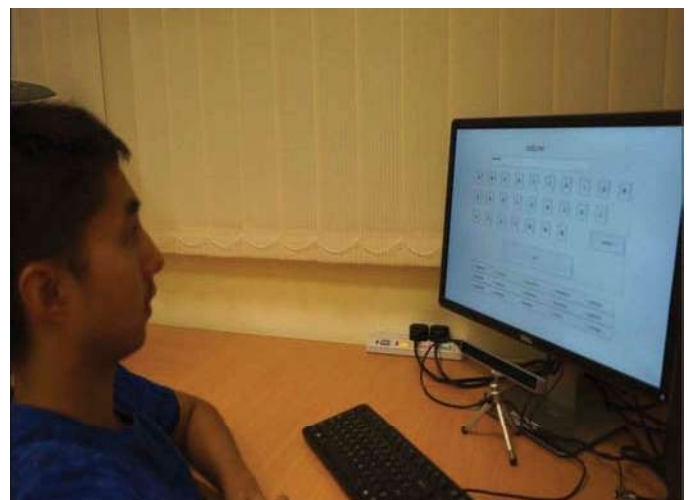


Figure 11. Participant performing experiments.

¹<http://theeyetribe.com/>.

dwelling time typing individual letters of the word. Like the finger-swipe-based typing, they only needed to “swipe” the letters of the word using gaze. Upon completing the word entry, they had to gaze at the “SPACE” key to indicate the end of entry. The system then displayed the candidate words at the bottom of the screen. At the bottom of the screen, there were four rows of the top five ranked words recommended by each of the four algorithms—Filteryeding, FdpRevised, MoWing, LCSMapping, respectively. The system simultaneously showed up the top five words at the four lists. Since we are evaluating the performance of four algorithms in typing errors rather than the typing performance of the participants, they did not need to choose the word from candidate words, but they were encouraged to look at the recommended results, and kept typing the next given random word. Then, a new random word was displayed at the top of the screen.

Figure 12a and b demonstrate the top five rate and the error rate of the 10 participants. 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. As can be seen, the performance of the three experts was better than seven novices. Figure 13 shows the average top five rate with error bars, where the results of MoWing and Filteryeding were almost same. This result fully supports our preliminary observations that the extra-letter error is the most common error, with new users trying to input all letters of the intended words resulting in more extra letters in their input. Therefore, although the MoWing approach improves its performance in neighbor-letter and missing-letter errors cases, its overall performance does not improve with sacrificing the performance in the most common, extra-letter error case. Among the seven novices, the top five rate of LCSMapping was around 20% higher than Filteryeding and MoWing, and the error rate of LCSMapping was only half of Filteryeding and MoWing. From the results of the three experts, the top five rate of LCSMapping was around 20% higher than Filteryeding, and the error rate of LCSMapping was nearly 0%, but the error rate of Filteryeding was higher than 10%.

6. Discussion

Through running experiments with simulation and actual users, the proposed LCSMapping is a robust approach in the eye-typing system that is able to handle extra letters,

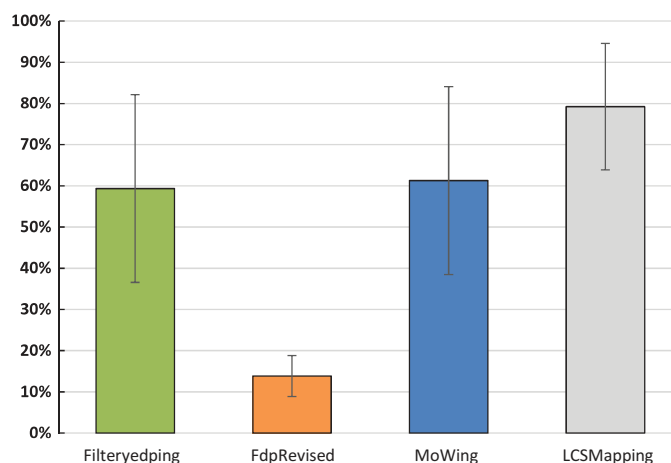


Figure 13. The average top five rate with error bars.

neighboring letters, and even missing letters, in terms of both accuracy metrics and error metrics. In the simulation study, we developed a word error generator to simulate the different types of errors as well as the number of such errors. This is essential for us to determine the performance of the different algorithms. It helps us collect an amount of data that could be difficult to obtain with actual users, because in eye-tracking area, we commonly experience difficulties in finding users for our eye-tracking studies (Eraslan, Yesilada, & Harper, 2016). Therefore, the simulation experiment is able to not only provide a theoretical testing of the approach, but also allow us to control specific (error) environment so that we optimize the system before testing on actual users. It is important for the eye-typing system design or even other HCI systems in some cases, for example, finding actual users is difficult, requiring repeated practical experiment is expensive, and quickly testing prototype is necessary.

During our study, we identified several steps in a typical eye-typing system, i.e., calibration, detection, recognition, prediction, and construction. While calibration and recognition are unique steps in the eye-typing system compared with a traditional finger typing system. Specifying the unique parts helps us define the clear purpose/scope of developing the system. The quality of calibration is hardware dependent, e.g., the accuracy of the eye tracker. In general, almost all eye-typing work assumed the perfect calibration, i.e., the user most probably fixates on

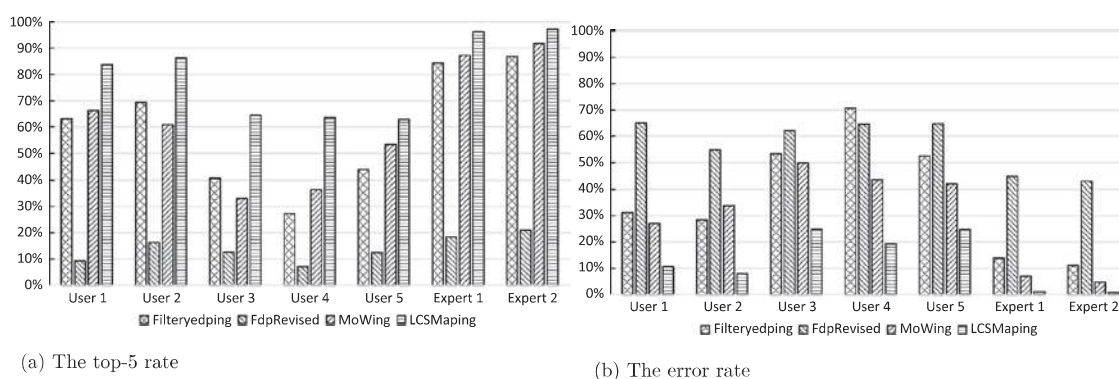


Figure 12. The results of actual users.

the correct letter directly. In order to guarantee that, usually the high-quality eye trackers are applied, or users are required to recalibrate frequently. However, the high-quality eye trackers usually are very expensive, which could be one main reason limiting the prevalence of eye-tracking applications requiring the high tracking accuracy. Nowadays, as human-computer interaction tends to be seamless and natural, repeating the calibration process would be burdensome, even though the eye-tracking devices are non-invasive for users. Therefore, reducing the calibration is a challenging issue that cannot be ignored when developing the eye typing. Although some automatic recalibration works have been proposed (Vadillo, Street, Beesley, & Shanks, 2015; Zhang & Hornof, 2014), they are not suitable for the real-time eye-typing system, and still contain the error range. Therefore, handling poor calibration or calibration drift is necessary in the rest of steps, and our system addresses this problem in the recognition step. The detection step exists in any typing system, which is basically that the system receives the “operation command” from the human with some initial processing.

In our study, we distinguished recognition and prediction, even though sometimes the two terms are interchangeable in typing systems. In this context, recognition is to process noisy inputs of the user, while prediction is a further process relying on syntactic/semantic analysis. This work is focusing on the recognition step instead of prediction. The purpose of our work is to efficiently recognize the potential words with high probabilities only based on what the user inputs, i.e., a robust approach handling noisy input. In recognition approaches, the key algorithm is to define the similarity/dissimilarity between candidate words in the dictionary and input of the user. In general, they are categorized into shape-based matching approaches and string-based matching approaches.

Shape-based approaches compare the shape of the path covered by the eye gaze with shapes stored in a word list (Hoppe, Löchtefeld, & Daiber, 2013; Kristensson & Zhai, 2004; Wobbrock, Myers, & Kembel, 2003; Zhai & Kristensson, 2003), and these approaches have been widely used in finger-based typing systems with the touch screen, such as Swype² and SwiftKey³ entry system for smart phones. The touch screen has enabled users to enter text by just dragging their fingers across the screen keyboard. While in eye-typing system, the shapes of the eye-gaze paths are always sharp (not smooth as dragging by fingers), and the shape-based approaches are less helpful. Pedrosa et al. (2015b) investigated the shape-based approach in which the error rate is very high in the eye-typing system, and it is not suitable for practical eye typing.

While string-based approaches usually apply string matching techniques finding the similarity/dissimilarity of candidate words and converted letter sequence of input stream of the user. Edit distance and longest common subsequence are widely used as the similarity metrics, and more comprehensive surveys can be found in (Navarro, 2001). However, the existing string similarity metrics are location independent (or keyboard layout independent), for example in the edit distance method, the distance between “a” and “s” is the same as the distance between “a” and “p,” but as we know, they cannot be regarded as the same distance according to

the QWERTY layout. In practice, the location information is very important when low accuracy or calibration draft exists, and the neighboring letters of gaze fixations (or gazed letters) should be also taken into account. In addition, another important factor in the eye typing we mentioned before is the time duration of gazed letters. The duration factor is able to help us identify the dominant letters, which improve the recognition accuracy and speed up the system. Therefore, the proposed LCSMapping is a string-based matching approach with location and time duration information, i.e., space and time dimensions, which are two intuitive factors of eye typing.

Although our work is not focusing on prediction and construction steps, we still discuss the potential research that can be integrated with our system. Our system is able to allow the user to focus on the entire word entry instead of individual letters. Furthermore, with syntactic/semantic analysis and even typing behaviors of the user, it is possible to allow the user to focus on the entire sentence, especially daily sentences. The recommended words in our system can also be a “coarse” input feeding forward into the prediction engine, and then the better qualified words would be recommended. In eye-typing systems, the real-time prediction is seldom used, because it would disturb the attention of the user, and then the user usually types all letters of the intended word at once (Trnka, McCaw, Yarrington, McCoy, & Pennington, 2008). In traditional finger-typing systems, input controller (finger) and visual feedback (eye) receiver are different. While using eye movement as the input controller, the visual feedback cannot be received in parallel, and then the audio feedback can be a good alternative (Majaranta, MacKenzie, Aula, & Rähä, 2003). Although candidate words are ranked highly with their corresponding probabilities, the users still need to select the intended word that requires cognitive effort and takes some time. Thus, if the intended words are always ranked in the first position, it could be more portable for practical typing. In our experiment, using LCSMapping, 87.6% intended words of the three expert users were ranked in the first position, compared with Filteredyping (61.4%), FdpRevised (2.5%), and MoWing (76.5%).

We summarize the differences of four algorithms in Table 3. On processing the gazed letters of a user, Filteredyping and FdpRevised only extract the ordered letters that the user has gazed at. MoWing and LCSMapping extract time duration of gazed letters, besides the letter sequence. The gaze duration provided us with the dominant letters in the sequence of letters entered, thus enabling a better decision-making by our algorithm. As for the similarity calculation (string matching), Filteredyping finds the possible words consisting of the letter sequence. The intended word must be a subset of the sequence of letters entered, which eliminates many possible words. Thus, it is not able to handle missing-letter errors as well as neighbor-letter errors. FdpRevised takes letter sequence’s neighbor into account making its search space much larger, which makes it fail in controlling the size of candidate words. Therefore, although the ranking method in Filteredyping and FdpRevised is reasonable, it still cannot recommend words efficiently. While MoWing uses a cost function to calculate

²<http://www.swype.com/>.

³<http://swiftkey.com/en/>.

Table 3. The summary of four algorithms.

	Only gazed letter sequence		Gazed letter sequence with corresponding time duration	
	Filteryedping	FdpRevised	MoWing	LCSM Mapping
Letter sequence string matching	Find words consisting of letters of the sequence	Find words consisting of letters of the sequence, or neighbors of these letters	Calculate the cost from the sequence to words	Find the longest common states mapping between sequence and words
Candidate ranking	$\log_{10}(\text{freq}(\text{word})) + w \star \text{length}(\text{word})$		Sorting by the cost	Sorting by the mapping score
Types of error solved	Extra-letter error	Neighbor-letter error (little)	Extra-letter error (little) Neighbor-letter error missing-letter error	Extra-letter error Neighbor-letter error missing-letter error

the converting cost of the entered letters and the words in the dictionary, which controls the size of the candidate-word space, this method searches for the local optimal. As the number of extra letters increases, the performance deteriorates gradually. LCSMapping finds the longest mapping states between the letter sequence and each word, and this method is able to reach the global optimal. From the simulation and actual users experiments, Filteryedping is only able to handle the extra-letter error, and FdpRevised has the very limited ability to handle the neighbor-letter error, while MoWing is able to handle the small number of the extra-letter error. The proposed LCSMapping is resilient to all of the three common errors.

In the experiment, the users were not required to select the intended word, but they were encouraged to look at the candidate list (top five words). The purpose of our work is to understand common text entry errors using gaze and improve accuracy of the work, rather than the entry speed, because the speed is not only depending on the proposed approach, but also other factors, e.g., user proficiency, UI design, and even the eye tracker. It is promising that in the future, thorough experiments are carried to control these factors and also investigate the usability of the system. From the experiment results, novices show relatively poor performance where even using our proposed algorithm, LCSMapping, the top five rate is less than 70%. From their experiment records and feedbacks, there are two main reasons, the first one is that they are not good at controlling the cursor using gaze and the other one is that they forget some letters of the word while gazing sequentially. It can be a guide of subsequent UI and experiment design, such as replacing the cursor, and allowing the users to memorize the given word carefully first before starting to type the word.

7. Conclusion and Future Work

In order to increase the communication ability of physically challenged people, a number of eye-based typing systems have been proposed, but most have been plagued by slow text entry speed. Recently, dwell-free systems have been proposed, but they still are vulnerable to common text entry errors. The user may gaze at extra irrelevant letters accidentally, and the Filteryedping (Pedrosa et al., 2015) method has overcome this problem by filtering out the extra letters. However, in practice, the low accuracy or drift problem caused by inaccurate calibration is another critical issue. The user might only gaze at an adjacent letter (neighbor-letter error), and even might not gaze at all letters of the word (missing-letter error). Then the MoWing (Liu et al., 2015) method has been proposed to improve its performance in neighbor-letter and

missing-letter errors cases while sacrificing some degree of performance in extra-letter error case. However, due to The Midas Touch problem, the extra-letter error is the most common error encountered in eye typing, and the overall performance of MoWing is not superior in practice.

Therefore, in this article, we have proposed a robust dwell-free recognition approach, LCSMapping, to handle the neighbor-letter and missing-letter errors without sacrificing the performance in the extra-letter error case. Unlike simple filtering and string matching methods, the proposed approach takes the letter location and gaze duration into account besides the gazed letters, which has made dwell-free typing even more accurate and more resilient to text entry error, much akin to finger-swipe typing, even using a low-cost eye-tracking device. Furthermore, implementing the tree structure to process the recommendation, the operation time of LCSMapping is reduced by 11% and 46%, respectively, compared with Filteryedping (FdpRevised) and MoWing.

However, limitations still exist. First, although the LCSMapping algorithm can handle slight calibration drift well, it is unable to handle free head motion. Unfortunately, current most low-cost eye trackers do not allow natural head movements. Second, although we apply the low-cost eye tracker with a relatively simple setup, the eye-tracker configuration is still a form of extra burden. Therefore, it can be a good future work to bring the merit of our work to eye-tracking systems using the single normal camera (Webcam). Since the Webcam is a standard configuration of PC or even mobile devices, it would be promising to achieve eye typing using such more flexible devices instead of extra eye trackers. In addition, another interesting future work is to integrate our eye-typing system with some popular network applications, e.g., WhatsApp, WeChat, to help people communicate with the outside world.

Due to the policy and location constraints, in the experiment, we cannot recruit the target population, i.e., people with motor disabilities. In order to make the system more practical, in the future, some target people would be invited to participate in our study and explore new demands/solutions.

ORCID

Yi Liu  <http://orcid.org/0000-0001-7575-2299>

References

- Adjouadi, M., Sesin, A., Ayala, M., & Cabrerizo, M. (2004). *Remote eye gaze tracking system as a computer interface for persons with severe motor disability*. Springer Berlin Heidelberg.

- Alnajar, F., Gevers, T., Valenti, R., & Ghebreab, S. (2013). Calibration-free gaze estimation using human gaze patterns. In *2013 IEEE International Conference on Computer Vision (ICCV)* (pp. 137–144). Sydney, NSW: IEEE.
- Biswas, P., & Langdon, P. (2015). Multimodal intelligent eye-gaze tracking system. *International Journal of Human-Computer Interaction*, 31(4), 277–294.
- Caligari, M., Godi, M., Guglielmetti, S., Franchignoni, F., & Nardone, A. (2013). Eye tracking communication devices in amyotrophic lateral sclerosis: Impact on disability and quality of life. *Amyotrophic Lateral Sclerosis and Frontotemporal Degeneration*, 14(7–8), 546–552.
- Chen, J., & Ji, Q. (2011). Probabilistic gaze estimation without active personal calibration. In *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 609–616). Providence, RI: IEEE.
- Chen, J., & Ji, Q. (2015). A probabilistic approach to online eye gaze tracking without explicit personal calibration. In *IEEE Transactions on Image Processing*, 24(3), 1076–1086.
- Davies, M. (2011). *Word frequency data from the corpus of contemporary American English (coca)*. Retrieved from <http://www.wordfrequency.info/>
- Donegan, M., Morris, J. D., Corno, F., Signorile, I., Chiò, A., Pasian, V., & Holmqvist, E. (2009). Understanding users and their needs. *Universal Access in the Information Society*, 8(4), 259–275.
- Eraslan, S., Yesilada, Y., & Harper, S. (2016). Eye tracking scanpath analysis on web pages: How many users? In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications* (pp. 103–110). Charleston, SC: ACM.
- Guan, C., Thulasidas, M., & Wu, J. (2004). High performance p300 speller for brain-computer interface. In *2004 IEEE International Workshop on Biomedical Circuits and Systems* (pp. S3–S5). Singapore: IEEE.
- Hansen, D. W., Hansen, J. P., Nielsen, M., Johansen, A. S., & Stegmann, M. B. (2002). Eye typing using Markov and active appearance models. In *Proceedings of the Sixth IEEE Workshop on Applications of Computer Vision, 2002 (WACV 2002)* (pp. 132–136). Orlando, FL: IEEE.
- Hansen, J. P., Tørning, K., Johansen, A. S., Itoh, K., & Aoki, H. (2004). Gaze typing compared with input by head and hand. In *Proceedings of the 2004 Symposium on Eye Tracking Research & Applications* (pp. 131–138). San Antonio, TX.
- Hoppe, S., Löchtefeld, M., & Daiber, F. (2013). Eype—using eye-traces for eye-typing. In *Workshop on Grand Challenges in Text Entry (chi 2013)*. Paris, France.
- Huey, E. B. (1908). *The psychology and pedagogy of reading*. Bristol, UK: Thoemmes Press.
- Jacob, R., & Karn, K. S. (2003). Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind*, 2(3), 4.
- Jacob, R. J. (1990). 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). Seattle, WA.
- Jacob, R. J. (1991). The use of eye movements in human-computer interaction techniques: What you look at is what you get. *ACM Transactions on Information Systems (TOIS)*, 9(2), 152–169.
- Jacob, R. J. (1995). Eye tracking in advanced interface design. In W. Barfield & T. A. Furness (Eds.), *Virtual environments and advanced interface design* (pp. 258–288). Oxford, UK: Oxford University Press.
- Just, M. A., & Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology*, 8(4), 441–480.
- Kocejko, T., Bujnowski, A., & Wtorek, J. (2009). Eye-mouse for disabled. In Z. S. Hippe & J. L. Kulikowski (Eds.), *Human-computer systems interaction* (pp. 109–122). Springer Berlin Heidelberg.
- Kotani, K., Yamaguchi, Y., Asao, T., & Horii, K. (2010). Design of eye-typing interface using saccadic latency of eye movement. *International Journal of Human-Computer Interaction*, 26(4), 361–376.
- Kristensson, P. O., & Vertanen, K. (2012). 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). Santa Barbara, CA.
- Kristensson, P.-O., & Zhai, S. (2004). Shark 2: A large vocabulary short-hand writing system for pen-based computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (pp. 43–52). Santa Fe, NM: ACM.
- Kumar, M. (2007). *Gaze-enhanced user interface design* (Unpublished doctoral dissertation). Stanford University, Stanford, CA.
- Liu, Y., Zhang, C., Lee, C., Lee, B.-S., & Chen, A. Q. (2015). Gazetry: Swipe text typing using gaze. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction* (pp. 192–196). Melbourne, Australia.
- MacKenzie, I. S., & Zhang, X. (2008). 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). Savannah, GA.
- Majoranta, P., Ahola, U.-K., & Špakov, O. (2009). Fast gaze typing with an adjustable dwell time. In *Proceedings of the Sigchi Conference on Human Factors in Computing Systems* (pp. 357–360). Boston, MA.
- Majoranta, P., Aoki, H., Donegan, M., Hansen, D. W., & Hansen, J. P. (2011). *Gaze interaction and applications of eye tracking: Advances in assistive technologies* (1st ed.). Hershey, PA: Information Science Reference - Imprint of IGI Publishing.
- Majoranta, P., MacKenzie, I. S., Aula, A., & Riihã, K.-J. (2003). Auditory and visual feedback during eye typing. In *Chi'03 Extended Abstracts on Human Factors in Computing Systems* (pp. 766–767). Fort Lauderdale, FL.
- Majoranta, P., & Riihã, K.-J. (2002). Twenty years of eye typing: Systems and design issues. In *Proceedings of the 2002 Symposium on Eye Tracking Research & Applications* (pp. 15–22). New York, NY: ACM. doi:10.1145/507072.507076
- Monty, R. A., & Senders, J. W. (1976). *Eye movements and psychological processes* (Tech. Rep.). DTIC Document. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Murata, A. (2006). Eye-gaze input versus mouse: Cursor control as a function of age. *International Journal of Human-Computer Interaction*, 21(1), 1–14.
- Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1), 31–88.
- Ohno, T. (2007). Eyeprint: Using passive eye trace from reading to enhance document access and comprehension. *International Journal of Human-Computer Interaction*, 23(1–2), 71–94.
- Pedrosa, D., Pimentel, M. D. G., & Truong, K. N. (2015). Filteredyping: 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). Seoul, Korea: ACM.
- Pedrosa, D., Pimentel, M. D. G., Wright, A., & Truong, K. N. (2015). Filteredyping: Design challenges and user performance of dwell-free eye typing. *ACM Transactions on Accessible Computing (TACCESS)*, 6(1), 3.
- Riihã, K.-J., & Ovaska, S. (2012). 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). Austin, TX.
- Rayner, K. (1977). Visual attention in reading: Eye movements reflect cognitive processes. *Memory & Cognition*, 5(4), 443–448.
- Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3), 372–422.
- Salvucci, D. D. (1999). Inferring intent in eye-based interfaces: Tracing eye movements with process models. In *Proceedings of the Sigchi Conference on Human Factors in Computing Systems* (pp. 254–261). Pittsburgh, PA.
- San Agustin, J., Skovsgaard, H., Hansen, J. P., & Hansen, D. W. (2009). Low-cost gaze interaction: Ready to deliver the promises. In *Chi'09 Extended Abstracts on Human Factors in Computing Systems* (pp. 4453–4458). Boston, MA.
- Sarcar, S., Panwar, P., & Chakraborty, T. (2013). 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). Bangalore, India.
- Spataro, R., Ciriaco, M., Manno, C., & La Bella, V. (2014). The eye-tracking computer device for communication in amyotrophic lateral sclerosis. *Acta Neurologica Scandinavica*, 130(1), 40–45.
- Su, M.-C., Wang, K.-C., & Chen, G.-D. (2006). An eye tracking system and its application in aids for people with severe disabilities. *Biomedical Engineering: Applications, Basis and Communications*, 18(6), 319–327.
- Sugano, Y., Matsushita, Y., & Sato, Y. (2010). Calibration-free gaze sensing using saliency maps. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2667–2674). San Francisco, CA: IEEE.
- Trnka, K., McCaw, J., Yarrington, D., McCoy, K. F., & Pennington, C. (2008). Word prediction and communication rate in AAC. In *Telehealth and Assistive Technologies (Telehealth/AT)* (pp. 19–24). Baltimore, MD.
- Underwood, G. (2005). *Cognitive processes in eye guidance*. New York: Oxford University Press.
- Urbina, M. H., & Huckauf, A. (2010). 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). Austin, TX.
- Vadillo, M. A., Street, C. N., Beesley, T., & Shanks, D. R. (2015). A simple algorithm for the offline recalibration of eye-tracking data through best-fitting linear transformation. *Behavior Research Methods*, 47(4), 1365–1376.
- Ward, D. J., & MacKay, D. J. (2002). Fast hands-free writing by gaze direction. arXiv preprint cs/0204030.
- Wobbrock, J. O., Myers, B. A., & Kembel, J. A. (2003). Edgewrite: A stylus-based text entry method designed for high accuracy and stability of motion. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (pp. 61–70). Vancouver, British Columbia.
- Wolfe, B., & Eichmann, D. (1997). A neural network approach to tracking eye position. *International Journal of Human-Computer Interaction*, 9(1), 59–79.
- Zander, T. O., Gaertner, M., Kothe, C., & Vilimek, R. (2010). Combining eye gaze input with a brain-computer interface for touchless human-computer interaction. *International Journal of Human-Computer Interaction*, 27(1), 38–51.
- Zhai, S., & Kristensson, P.-O. (2003). Shorthand writing on stylus keyboard. In *Proceedings of the Sigchi Conference on Human Factors in Computing Systems* (pp. 97–104). Fort Lauderdale, FL.
- Zhang, Y., & Hornof, A. J. (2014). Easy post-hoc spatial recalibration of eye tracking data. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (pp. 95–98). Safety Harbor, FL.

About the Authors

Yi Liu is a PhD student of computer science at Interdisciplinary Graduate School, Nanyang Technological University. His research spans brain-computer interaction, human-computer interaction, eye tracking, and computer vision. He obtained his BEng from the School of Software, Harbin Institute of Technology, China.

Bu-Sung Lee is currently an Associate Professor with the School of Computer Science and Engineering, Nanyang Technological University. He held a joint position as Director (Research) HP Labs, Singapore, from 2010 till 2012. His major research interests are in the area of mobile and

pervasive network, distributed systems, and cloud/grid computing technology.

Martin J. McKeown is a Professor of Medicine and ECE (Adjunct), Director of the Pacific Parkinson's Research Centre at UBC, and holds the PPRI/UBC Chair in Parkinson's Research. In addition to seeing patients with Movement Disorders, he is developing new analytical methods to assess brain imaging data and investigating novel treatments for Parkinson's Disease.

Appendix A

We define $f(i, j)$ as the longest common states of first i states in *observable transition* and first j states in *potential transition*.

First given a 1-by- n matrix, in which the *potential transition* contains only one state, the longest common states in formula (1) is sum of this row:

$$f(1, j) = \sum_{k=1}^j a_{1k}, j \in [1, 2, \dots, n] \quad (A1)$$

Then given a m -by-1 matrix, in which the *observable transition* contains only one state, the longest common states in formula (1) is the row with the maximum value:

$$f(i, 1) = \max(a_{i1}, a_{21}, \dots, a_{n1}), i \in [1, 2, \dots, m] \quad (A2)$$

Given a m -by- n matrix ($m, n > 1$), we verify the following Equation (A3):

$$f(i, j) = \max(f(i, j-1) + a_{ij}, f(i-1, j)) \quad (A3)$$

We use the mathematical induction proof as follows.

- (1) When $m = 2$ and $n = 2$, $f(2, 2)$ has three combinations with constraints in formula (1), $a_{11} + a_{12}$, $a_{11} + a_{22}$, $a_{21} + a_{22}$, and $f(2, 2) = \max(a_{11} + a_{12}, a_{11} + a_{22}, a_{21} + a_{22})$, in which $a_{11} + a_{12} = f(1, 2)$ (Equation (A1)). According to Equation (A2), $\max(a_{11}, a_{21}) = f(2, 1)$, so $\max(a_{11} + a_{22}, a_{21} + a_{22}) = f(2, 1) + a_{22}$, and then $f(2, 2) = \max(f(2, 1) + a_{22}, f(1, 2))$ satisfying Equation (A3).
- (2) When $m = 2$ and $n = 3$, $f(2, 3) = \max(a_{11} + a_{12} + a_{13}, a_{11} + a_{12} + a_{23}, a_{11} + a_{22} + a_{23}, a_{21} + a_{22} + a_{23})$, in which $a_{11} + a_{12} + a_{13} = f(1, 3)$ (Equation (A1)), and $\max(a_{11} + a_{12}, a_{11} + a_{22}, a_{21} + a_{22}) = f(2, 2)$ at the first step. So $f(2, 3) = \max(f(2, 2) + a_{23}, f(1, 3))$ satisfying Equation (A3).
- (3) When $m = 3$ and $n = 2$, $f(3, 2) = \max(a_{11} + a_{12}, a_{11} + a_{22}, a_{11} + a_{32}, a_{21} + a_{22}, a_{21} + a_{32}, a_{31} + a_{32})$, in which $\max(a_{11}, a_{21}, a_{31}) = f(3, 1)$ (Equation (A2)), so $\max(a_{11} + a_{32}, a_{21} + a_{32}, a_{31} + a_{32}) = f(3, 1) + a_{32}$. According the proof at the first step, $\max(a_{11} + a_{12}, a_{11} + a_{22}, a_{21} + a_{22}) = f(2, 2)$, so $f(3, 2) = \max(f(3, 1) + a_{32}, f(2, 2))$ satisfying Equation (A3).
- (4) When $m = 3$ and $n = 3$, $f(3, 3) = \max(C_{32} + a_{33}, C_{23})$, in which C_{ij} indicates the possible combinations with constraints at the first i rows and j columns. According to the second and third steps, $\max(C_{32}) = f(3, 2)$, $\max(C_{23}) = f(2, 3)$, so $f(3, 3) = \max(f(3, 2) + a_{33}, f(2, 3))$ satisfying Equation (A3).
- (5) Given $f(m, n-1)$, $f(m-1, n)$ satisfying Equation (A3), there are $f(m, n-1) = \max(C_{m, n-1})$, $f(m-1, n) = \max(C_{m-1, n})$. Due to $f(m, n) = \max(C_{m, n-1} + a_{mn}, C_{m-1, n})$, then $f(m, n) = \max(f(m, n-1) + a_{mn}, f(m-1, n))$ satisfying Equation (A3).