A Robust Recognition Approach in Eye-Based Dwell-Free Typing

Yi Liu[1,2], Bu-Sung Lee[2], Martin J. McKeown[3], and Chonho Lee[4]

[1]Nanyang Institute of Technology in Health & Medicine, Interdisciplinary Graduate School
[2]School of Computer Engineering, Nanyang Technological University, Singapore
[3]Pacific Parkinson's Research Centre, Department of Medicine, University of British Columbia, Vancouver, BC, Canada
[4]School of Computing, National University of Singapore, Singapore
{*yliu028,ebslee*}*@ntu.edu.sg, martin.mckeown@ubc.ca, chonho@gmail.com*

***Abstract*—Alternative form of text entry is a tremendous benefit for a subset of physically challenged people, spurning the development of many assistive technologies. However, speed of text entry with these methods is still a critical problem. Eye gaze technologies have potential for text entry, but still tend to relatively slow. Recently dwell-free eye-typing systems have been proposed, but can be vulnerable to some common text entry problems, such as selection of the wrong character. In this paper, we propose a recognition approach for inferring the words which the user intends to type. The method is robust to missing letters and even when a neighboring character on the keyboard is incorrectly selected. Simulation and experiments results suggest that our proposed approach has better accuracy and more resilience to text entry errors than currently proposed dwell-free systems.**

***Keywords*—Assistive technologies, Eye-typing, Dwell-free systems**

I. INTRODUCTION

In modern society, access to and communicating via the internet is fundamental 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 [1]. 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 new reliable method of communication.

However, speed of text entry is a crucial limitation to current eye-typing systems. The early eye-typing systems allowed users to type only two or three words per minute, which is impractical for communication. In general, there are two main reasons for the relatively slow text entry: 1). *Selection method*, users need to fixate their gaze on a key for an interval (the dwell time – typically 200-600 ms), which limits the text-entry speed. 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 characters.

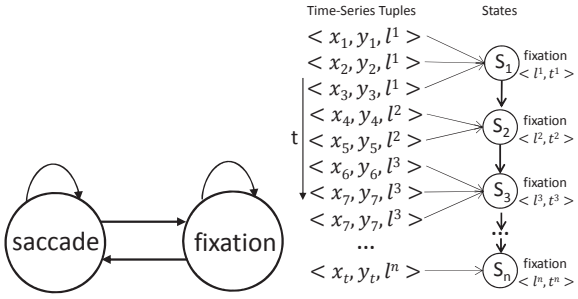
In order to increase the speed of text entry, researchers have developed a number of improvements. To increase typing speed when selecting individual characters, the dwell time can be reduced or dynamically adjusted [2], or the keyboard can be modified from the standard QWERTY layout to more efficient forms [3], [4]. However, the user frequently gets 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 [4], [5], enhancing overall communication rate. Although the speed of text entry is increasing, 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 [6]. Dwell-free typing allows the user to type an intended 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 be often be recognized based on the sequential letters detected and extracted from user's gaze trace. 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 [6] investigated the potential of dwell-free eye typing with a QWERTY-based keyboard layout. In their study, 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 known in advance, and only accepts the input letter corresponding to the next letter in the word; it intentionally ignores a letter if it is not the next letter in the word. Therefore, words can be written by only accepting correct letters, even if the user looks at extra keys. 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 and Pimentel [7] proposed a filter-based dwell-free typing system for practical text entry, 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 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

978-1-4673-9088-0/15/$31.00 ©2015 IEEE



(a) Saccade-fixation model (b) Tuple-state transition

Fig. 1: State Transition

typing tool [2], [8]. After two hours of practice, the fastest participant typed at 19.28WPM with Filteryedping. Thus it appears that dwell-free systems are much faster than the dwell-based systems in practice.

Although Filteryedping is able to handle the extra letters error, it cannot handle missing letters 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.

Therefore, in this paper we propose an "intelligent" dwell-free approach which is robust to many common errors afflicting eye-typing: missing letters, and incorrect character selection that is a neighbor of the desired letter. Our experimental results from simulation and field testing with users suggest that such an approach results in improved accuracy.

II. METHODOLOGY

A. 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. Through initial data processing, $< x, y, l >$ 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 [9] shown in figure 1a, 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 [10] is implemented in the paper.

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 $< l, t >$, 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 2. 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 the user input in the recognition step. We then need to recommend the potential words (including the intended word) through the input.

TABLE I

	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}

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 the certain one-direction transition, and we define a potential word 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 our 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 coefficent, 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 at Table I. There are three conditions as follows to fill in the table (Algorithm 1):

- 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 (from preliminary results, the optimal weight is 0.2).
- Otherwise, the cell is zero.

Instinctively, when the user is typing, each fixation is related to only one intended letter, and several fixations can be related to the same intended letter. Based on this fact, we define the two transitions mapping as a many-to-one mapping shown in figure 2, and the method in the paper is called the longest common states mapping method (LCSMapping). One

Algorithm 1 Matrix Generation Algorithm

function BUILDMATRIX($observable, potential$)
  a_{ij} is the i^{th} row, j^{th} column cell of $Matrix$
  S_i is the i^{th} state of $observable$
  $S_j^{'}$ is the j^{th} state of $potential$
  if $S_i.l = S_j^{'}.l$ **then**
    $a_{ij} \leftarrow S_i.t$
  else if $S_i.l \in neighbor(S_j^{'}.l)$ **then**
    $a_{ij} \leftarrow S_i.t * weight$
  else
    $a_{ij} \leftarrow 0$
  end if
  return $Matrix$
end function

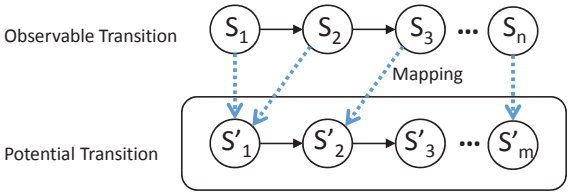


Fig. 2: Transition Mapping

state in the observable transition can only map one state in the potential transition. Since the transition is ordered, if in observable transition $S1$ is mapping $S_2^{'}$ in potential transition, $S2$ cannot map the $S_1^{'}$ which is earlier than $S_2^{'}$. According to the above two constraints, the problem is modelled by the following mathematical formula 1, which is a dynamic programming problem. The solution is the longest common states, and we solve it using a recursive formula in algorithm 2.

$$\begin{aligned} Maximize: &\ \sum_{j}^{n} a_{ij}, i \in [1,2,...m] \\ Suject\ to: &\ \forall a_{i_1 j_1}, a_{i_2 j_2}; \\ &\ if\ \ j_1 < j_2, then\ \ i_1 \leq i_2 \end{aligned} \tag{1}$$

B. 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 consist of sequential 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.

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

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 mapping to potential transition. $|B_{(A \cap B)}|$ indicates the number of states in potential transition that are mapped by observable transition, and $|B|$ indicates the number of states in potential transition, so $|B_{(A \cap B)}|/|B|$ is the proportion that potential transition mapped by observable transition. The score range is same as Sorensen-Dice coefficient, from 0 to 2.

Algorithm 2 Longest Common States Algorithm

function LCSTATE($Matrix$)
  $f(1,j) = \sum_{k=1}^{j} a_{1k}$
  $f(i,1) = \max(a_{11}, a_{21}, ..., a_{i1})$
  $m, n \leftarrow Matrix$
  if $i, j > 1$ **then**
    $f(i,j) = \max(f(i,j-1) + a_{ij}, f(i-1,j))$
  end if
  return $f(m,n)$
end function

C. 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 the 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 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 score to sort the candidate words.

III. EVALUATION

In this section, we evaluate three approaches, the proposed LCSMapping approach, Filteryedping [7], and a revised version of Filteryedping (FdpResived) which tolerates adjacent letters.

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, seven actual users taking part in the experiment were instructed to type words into the system.

A. Experiment 1: Simulation

We built word generators with different types of text entry errors to compare the resilience of the various algorithms. In the extra-letter error, the simulator creates a random noisy input, e.g. irrelevant letters, into a word which simulates that the user gazes at extra letters accidentally. In addition, for the simulated input, the time interval of intended letter is around 10% to 50% longer than extra irrelevant letters, which is to simulate that the user spends relatively longer time on the intended letters. In the neighbor-letter error, the simulator generates the input consisting some neighbor of letters in a word, which simulates that the user gazes at a neighbor of an intended letter rather than the letter itself. In the missing-letter error, the input is lacking some letters of a word, simulating

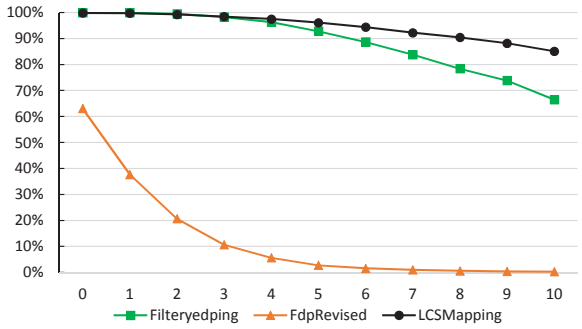
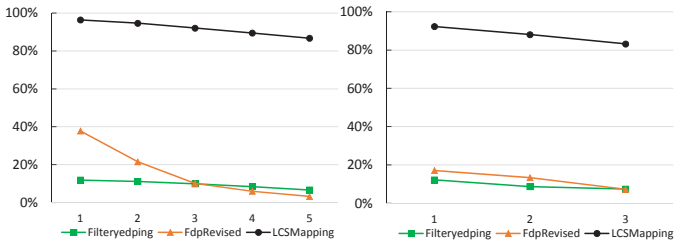


Fig. 3: The top-5 rate in extra-letter error



(a) Neighbor-letter error (b) Missing-letter error

Fig. 4: The top-5 rate in two errors

Fig. 5: Participant performing experiments

that the user does not gaze at the intended locations nor neighbors.

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, a total of 100 thousand letter sequences were generated for each type of error. Then the simulated letter sequences were input to the three algorithms to establish the recommended the words based on the letter sequences.

Figure 3 shows the top-5 rate results with the extra-letter error. 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. The X axis is the number of simulated extra letter, which indicates how many irrelevant letters the simulator generates into the word. From the results, when the number of the extra letter is 0 to 2, the top-5 rate of Filteryedping is 0.2% higher than LCSMapping, likely because the LCSMapping algorithm has a bigger search space for possible words. If the score of the intended word is not ranked in the first five positions in the candidate list an error will occur. When the number of the extra letters is larger than 2, LCSMapping performs better than Filteryedping, with LCSMapping performing 20% higher with 10 extra letters. From our preliminary experiment of actual users, the average of the extra letters is typically $\tilde{5}$, and LCSMapping accuracy is still around 4% higher. The statistical analysis shows significant difference between LCSMapping and Filteryepding ($p = 0.0174 < 0.05$). However, since FdpRevised allows a word to consist of neighbors of the gazed letters, the search space is much greater than the original Filteryedping approach. When the number of the extra letters increases, the top-5 rate of FdpRevised decreases to nearly 0% exponentially.

Figure 4 shows the performance of three algorithms in both neighbor-letter and missing-letter errors. In figure 4a, the X axis is the number of letter which is replaced by its neighbor. In figure 4b, 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, we set the neighbor-letter error is from 1 to 5, which means at most five letters of the word are changed to their neighbors. The missing-letter error is set to 1 to 3, which means at most three letters of the word can be missing.

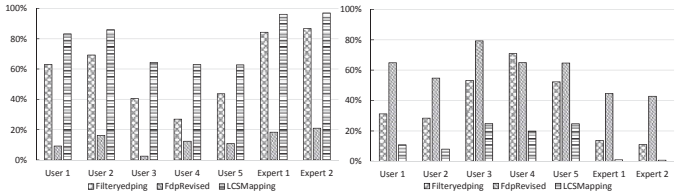
The results show that LCSMapping is able to handle both neighbor-letter and missing-letter errors, in which the top-5 rate is still higher than 80% even though the neighbor-letter error is 5 and missing-letter error is 3. However, although the neighbor-letter missing-letter errors are 1, Filteryedping achieves around 12% top-5 rate which is likely impractical for handling typing errors of the user. Although the top-5 rate of FdpRevised (37.85%) is much higher Filteryepding(11.89%), its performance decreases dramatically once the error increases due to its poor search-space control.

The results of simulation show that LCSMapping is good at handling not only the extra-letter errors, but also neighbor-letter and missing-letter errors.

B. 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 used a low-cost eye tracker (TheEyeTribe[1]) put under the monitor shown in figure 5. The eye tracker works at a sampling rate of 60Hz. Nine-point calibration was used in the study. The tracked gaze coordinates data were mapped onto the mouse cursor with using moving average smoothing algorithms provided by the device. The participants do not

[1]http://theeyetribe.com/



(a) The top-5 rate (b) The error rate

Fig. 6: The results of actual users

need to recalibrate unless they are uncomfortable with the accuracy of the eye tracker, and there is a big drift.

We recruited seven volunteers (6 male, 1 female; 23-35 years) from the local university. All of them had 4 to 8 hours per day computer usage, and had normal vision. Two of them had rich prior experience with eye tracking, and five of them had no such eye-tracking experience before.

All participants had at least half-hour training to control the cursor using eye movement, and type characters using gaze. In the experiment, there were five sessions, but from results the users did not demonstrate significant different performance the five sessions. Between the two sessions, the participants had five-minute break and then took the recalibration. In each session, each participant needed 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 in the Corpus of Contemporary American English (COCA) [11].

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 our system is a dwell-free typing system, they did not have to spend dwell 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. Since we evaluate the performance of three algorithms, the system simultaneously showed the top 5 words recommended by three algorithms based on the same input. The participants kept typing the given random words, and did not need to choose the word from candidate words in the experiment. Then a new random word was displayed at the top of the screen.

Figure 6a and figure 6b show the top-5 rate and error rate of the seven 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 two experts is better than five novices. Among the five novices, the top-5 rate of LCSMapping is around 20% higher than Filteryedping, and the error rate of LCSMapping is only half of Filteryedping. From the results of the two experts, the top-5 rate of LCSMapping is around 20% higher than Filteryedping, and the error rate of LCSMapping is nearly 0%, but the error rate of Filteryedping is higher than 10%.

The results of actual users also show that in practical typing LCSMapping has better accuracy and higher resilience.

IV. Conclusion

In order to increase the communication ability of physically challenged people, a lot of eye-based typing systems have been proposed, but these have been plagued by slow text entry speed. Recently, dwell-free systems have been proposed, but they still are vulnerable to some common practical text entry errors. Due to the Midas touch problem, the user may gaze at extra irrelevant letters accidentally. In practice the low accuracy or drift problem caused by inaccurate calibration will also cause problems. 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). Therefore, in this paper, we have proposed a robust dwell-free recognition approach to handle the neighbor-letter and missing-letter errors. Our proposed approach has made dwell-free typing have better accuracy and more resilient to text entry error, much akin to finger-swipe typing.

ACKNOWLEDGMENT

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

References

[1] Paivi Majaranta, Hirotaka Aoki, Mick Donegan, Dan Witzner Hansen, and John Paulin Hansen. *Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 1st edition, 2011.

[2] Päivi Majaranta, Ulla-Kaija Ahola, and Oleg Špakov. Fast gaze typing with an adjustable dwell time. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 357–360. ACM, 2009.

[3] Sayan Sarcar, Prateek Panwar, and Tuhin Chakraborty. Eyek: an efficient dwell-free eye gaze-based text entry system. In *Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction*, pages 215–220. ACM, 2013.

[4] Mario H Urbina and Anke Huckauf. Alternatives to single character entry and dwell time selection on eye typing. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, pages 315–322. ACM, 2010.

[5] I Scott MacKenzie and Xuang Zhang. Eye typing using word and letter prediction and a fixation algorithm. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, pages 55–58. ACM, 2008.

[6] Per Ola Kristensson and Keith Vertanen. The potential of dwell-free eye-typing for fast assistive gaze communication. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 241–244. ACM, 2012.

[7] Diogo Pedrosa, Maria Da Graça Pimentel, Amy Wright, and Khai N Truong. Filteryedping: Design challenges and user performance of dwell-free eye typing. *ACM Transactions on Accessible Computing (TACCESS)*, 6(1):3, 2015.

[8] Kari-Jouko Räihä and Saila Ovaska. 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*, pages 3001–3010. ACM, 2012.

[9] Dario D Salvucci. Inferring intent in eye-based interfaces: tracing eye movements with process models. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 254–261. ACM, 1999.

[10] Manu Kumar. *Gaze-enhanced user interface design*. PhD thesis, Stanford University, 2007.

[11] Mar Davies. Word frequency data from the corpus of contemporary american english (coca), 2011.