A Special Virtual Keyboard for Disabled Computer Users

Shiu Fung Lau and Henry C. B. Chan

***Abstract*—In this paper, we present a special virtual keyboard for computer users with disabilities who have problems typing. In particular, the keys are placed based on user typing habit and determined by a genetic algorithm (GA). When a disabled user types on a QWERTY keyboard, the typing cost is high, because the keys are not optimally arranged for disabled users, who can only type with one finger. With the proposed keyboard, the keys can be customised and arranged to minimise the typing cost (e.g., certain keys should be placed close to one another). Our analysis indicates that the proposed keyboard can outperform the traditional keyword by about 40 per cent. Hence the special virtual keyboard can facilitate disabled users to type more effectively and efficiently.**

***Index Terms*—assistive technology, keyboard, human computer interface.**

I. Introduction

TODAY, computers and smartphones play an important role in people's daily lives. People type messages to communicate with others through WhatsApp, WeChat and LINE every day. The keyboard is the typical device for interacting with computers. In this paper, we study a new keyboard design for people with disabilities. Basically, the keys on a keyboard are re-organised in a new way based on a genetic algorithm (GA). The objective is to minimise the typing cost, so as to facilitate people with disabilities to type more efficiently. Examples will be provided, and an analysis of the algorithm will be explained in detail.

Disabled people suffering from muscular dystrophy have difficulties in typing. In other words, typing can be a difficult task for them. They can only move their hands a little bit, and they tire easily because of their weak muscles. Some can use only one hand, or only one finger [1][2]. People without disabilities can use the traditional QWERTY keyboard and easily type with 10 fingers. For people without disabilities, pressing a key takes less than a second. Typing a word takes a few seconds. Typing 40 words takes one minute on average [3]. However, for those who are disabled, typing tasks are not easy.

Some researchers have suggested that disabled people use a brainwave-based keyboard. For example, P300 is a brain-computer interface (BCI) system based on a matrix of letters, numbers and symbols with visual feedback/stimuli. For the matrix, the columns and rows flash continuously for input purposes. Basically, a user should focus on the flashing item that he/she wants to choose. According to [4] and [5], the typing speed of the P300 system is 1-4 words per minute[4][5]. Hence the input speed is slow. Furthermore, it is very expensive, so most disabled users would not be able to afford to use it.

Inspired by [6] and [7], we present a GA-based keyboard in this paper. Compared with the previous works [6] [7], we focus on designing a customised virtual keyboard for disabled people, who can only use one finger to type. Furthermore, more performance analysis is conducted. In general, the proposed keyboard can also be used with a computer mouse, and other sensors or tools as an input device [8].

The rest of the paper is organised as below. Section II defines the typing problems and typing cost. Section III presents the design of the proposed keyboard. Section IV compares the performance of the proposed keyboard and standard QWERTY keyboard. Section V presents the implementation of the special virtual keyboard. Section VI concludes this paper.

II. Current problem

When people type, they need to move their fingers to the corresponding keys and then press the keys. For our performance analysis, we define typing cost as the sum of moving cost and pressing cost. The moving cost reflects the cost of moving a finger from one key to another. The pressing cost reflects the cost of pressing a key after moving the finger to the top of the key. Two estimations are made, to compare the typing cost for abled users, with the typing cost for those who have a disability.

A. *Estimation of typing cost for people without disabilities*

This estimation is based on three assumptions to calculate the typing cost for people without a disability.

1) Moving cost is 1 per key move.
2) Pressing cost is 1 per key press.
3) People without a disability can use both hands (i.e., 10 fingers) to type with standard finger placement.

Using standard finger placement, both hands are placed on the keyboard. Left-hand fingers (i.e., little, ring, middle and index fingers) are placed on the keys "A", "S", "D" and "F". Right-hand fingers (i.e., index, middle and ring fingers) are placed on the keys "J", "K" and "L" (see Table I)[9].

For example, for "A", the left little finger is placed on top of "A", so the moving cost is 0. For another example, "Q", as the finger needs to move from "A" to "Q" and the keys are next to one another, the moving cost is 1 (see Table II).

Table II summarises the moving cost for people without disabilities typing with standard finger placement. The maximum cost is 2. When users type on the same key, the moving cost is 0 and the pressing cost is 1.

For example, the cost of typing the word "APPLE" can be estimated by the following six steps:

Manuscript received January 08, 2017; revised February 02, 2017. This work was supported by the Department of Computing, The Hong Kong Polytechnic University.

Shiu Fung Lau and Henry C. B. Chan are with the Department of Computing, The Hong Kong Polytechnic University. E-mail: (shiu-fung.lau@connect.polyu.hk, cshchan@comp.polyu.edu.hk)

TABLE I
FINGER PLACEMENT ON A STANDARD KEYBOARD

Letter	Responsible finger
Q, A, Z	Left little finger
W, S, X	Left ring finger
E, D, C	Left middle finger
R, F, V, T, G, B	Left index finger
Y, H, N, U, J, M	Right index finger
I, K	Right middle finger
O, L	Right ring finger
P	Right little finger

TABLE II
MOVING COST WHEN USING TEN FINGERS TO TYPE

Letters	Moving cost
A, S, D, F, J, K, L	0
Q, W, E, R, T, U, I, O, P, Z, X, C, V, N, M	1
Y, B	2

1) For “A”, the moving cost is 0 and the pressing cost is 1.
2) For “P”, the moving cost is 1 and the pressing cost is 1.
3) For the second “P”, the moving cost is 0 and the pressing cost is 1.
4) For “L”, the moving cost is 1 and the pressing cost is 1.
5) For “E”, the moving cost is 1 and the pressing cost is 1.
6) Summing up all the costs, the total cost is 8.

B. Estimation of typing cost for people with disabilities

Here, we consider the typing cost for disabled people. The cost estimation is based on four assumptions:

1) Moving cost is 1 per key move.
2) Pressing cost is 1 per key press.
3) The target disabled users can use only one finger to type.
4) The finger is placed on the first letter of the word before typing.

The finger moving cost can be estimated by key distance. When users type the same key, for example “ZZZ”, the finger remains on the same key, so the moving cost is 0. When users type nearby keys, for example “AS”, after typing “A”, the finger should move from “A” to “S” with a moving cost of 1. When moving from “A” to “D”, the moving cost is 2 and when moving from “A” to “F”, the moving cost is 3. The maximum cost is 9 e.g., from “A” to “P” and from “P” to “Q”.

For example, the cost to type the word “APPLE” can be determined by the following six steps:

1) For “A”, the moving cost is 0 (based on the aforementioned assumption) and the pressing cost is 1.
2) For “P” (i.e., from “A” to “P”), from “A” to “P”, there are the letters “S”, “D”, “F” ... “L” then “P”. The moving cost is 9 and the pressing cost is 1.
3) For the second “P”, the finger does not need to move, so the moving cost is 0 and the pressing cost is 1.
4) For “L” (i.e., from “P” to “L”), the letters are adjacent to one another, so the moving cost is 1 and the pressing cost is 1.
5) For “E” (i.e., from “L” to “E”), there are the letters “K”, “J”, “H” ... “D” then “E”. The moving cost is 7 and the pressing cost is 1.
6) Summing up all the costs, the total cost is 22.

For people without disabilities, the cost to type the word “APPLE” is 8, while the cost for target disabled people to type “APPLE” is 22, which is 2.75 times higher than the cost for abled people (i.e., typing with 10 fingers).

III. KEYBOARD DESIGN

The current QWERTY keyboard design can basically be modelled as a 4 times 10 rectangle shape. It is not a compact design. For general communication, a disabled user needs 26 letters (A to Z) and 10 numbers (0 to 9). There are a total 36 units. Therefore, we propose using a 6 times 6 square shape (i.e., more compact design) for disabled people.

One key problem is how to organise the letters and numbers. One possible method is to organise the keys in ascending order. However, this does not take into consideration the relationship between the letters (e.g., it is better to place certain letters close to one another). Note that our goal is to minimise the moving cost to facilitate disabled users to be able to type with only one finger.

Another simple method is to generate the keys randomly. With the use of a random number generation algorithm, we can easily build a 6 times 6 matrix. The following matrix (see Table III) is an example. However, it is quite confusing, in that numbers and letters are mixed together. It is difficult for users to distinguish letters from numbers. Therefore, it is better to separate numbers and letters (see Algorithm 1).

TABLE III
A MATRIX/KEYBOARD EXAMPLE

H	X	U	7	5	C
P	S	A	1	I	8
L	Q	N	Y	3	V
B	J	R	D	E	F
T	K	M	O	W	2
6	Z	G	9	0	4

In this paper, we study a GA-based keyboard. The same as with the current QWERTY keyboard design, the numbers 0-9 are placed in the top row. In the proposed design, only six keys can be placed in each row. Therefore, numbers 0-9 are placed in the top two rows, with two remaining spots. Assume that “Y” and “Z” are the least commonly used letters. Therefore, they are placed in the remaining spots in the second row. For the remaining letters, a genetic algorithm is applied to arrange the keys.

As one of the evolutionary algorithms, GA is commonly used for solving general optimisation problems. Here we study the use of GA to organise the keys (see Algorithm 2). The steps are discussed as follows.

Initialisation - First, a population of chromosomes is generated based on the random number generation algorithm and the preset matrix. Note that each chromosome should contain the basic elements: A to Z and 0 to 9. As an example,

Algorithm 1 Generating a matrix with random keys

```
procedure RandomMatrixPreset($matrix[6][6]$)
    $matrix[0][0] = 0$
    $matrix[0][1] = 1$
    ...
    $matrix[1][3] = 9$
    $Letter = [A, B, C...X, Y, Z]$
    for each $i \in Letter$ do
        while $True$ do
            if $matrix[random][random] == \emptyset$ then
                $matrix[random][random] \leftarrow i$
                $Break$
            end if
        end while
    end for
end procedure
```

Algorithm 2 Using a genetic algorithm to generate keys

```
procedure GAMatrix($matrix[6][6]$)
    $checkingTable[letter] = [A, B, C...X, Y, Z]$
    $c1 \leftarrow matrix[random]$
    $c2 \leftarrow matrix[random]$
    for $row \leftarrow 2 : 3$ do
        $GAmatrix[row] \leftarrow c1[row]$
        for $col \leftarrow 0 : 5$ do
            $letter \leftarrow c1[row][col]$
            $Checking(letter) \leftarrow True$
        end for
    end for
    for $row \leftarrow 4 : 5$ do
        for $col \leftarrow 0 : 5$ do
            $letter \leftarrow c2[row][col]$
            if $letter \notin checkingTable$ then
                $Checking(letter) \leftarrow True$
                $GAmatrix[row][col] \leftarrow c2[row][col]$
            end if
        end for
    end for
    for $row \leftarrow 4 : 5$ do
        for $col \leftarrow 0 : 5$ do
            if $GAmatrix[row][col] == \emptyset$ then
                $GAmatrix[row][col] \leftarrow Remain()$
            end if
        end for
    end for
    mutation($GAmatrix[6][6]$)
end procedure
```

the initial population size is set at 10. In other words, 10 initial chromosomes are generated (see Figure 1). Note that the initial population size can be adjusted.

Selection - Second, the cost of each chromosome is computed based on a fitness function. Then, two chromosomes are randomly selected from the pool. For example, the second chromosome (C1) and ninth chromosome (C2) are selected as shown in Figure 2.

Crossover - Third, the crossover operation is performed for the selected chromosomes. As row 1 and row 2 are preset, they will not be changed by the crossover operation. That

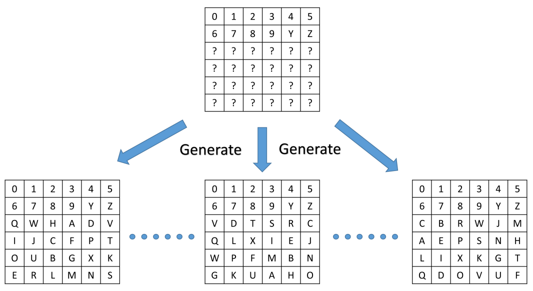

Fig. 1. Initialisation

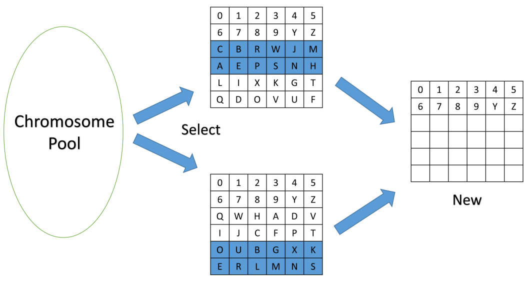

Fig. 2. Selection

means the crossover operation is performed for rows 3 to 6 only. In the example, row 3 to row 4 of C1 and row 5 to row 6 of C2 are selected to generate a new chromosome based on the crossover operation. However, there are some duplicate letters. That means, a direct crossover cannot be performed and a special arrangement should be made. To handle this issue, a checking table is maintained for recording the used letters or checking for duplication (see Figure 3). The letters in C1 are recorded in the table for checking the letters in C2. In the example, “B”, “E”, “R”, “M”, “N” and “S” are duplicated. They are not added to the new chromosome. Instead, “D”, “F”, “I”, “Q”, “T” and “V” remain and are randomly filled to the new chromosome (see Figure 4).

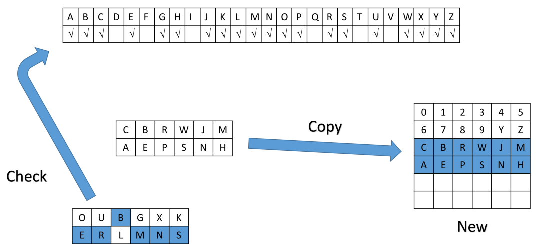

Fig. 3. Crossover step 1

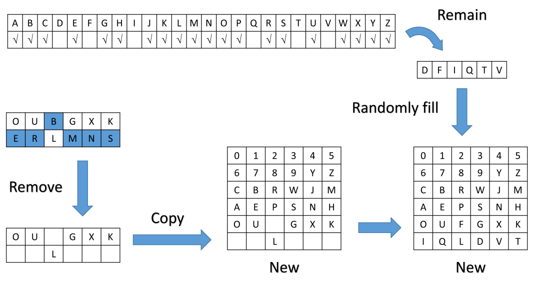

Fig. 4. Crossover step 2

Mutation - Then, mutation is applied to the chromosomes with a certain probability in order to enhance the results. Mutation is useful because it seeks to enhance genetic diversity by introducing changes. For example, in a chromosome, two letters are swapped with a probability of 0.1. In the example, letter “M” and letter “J” are swapped (see Figure 5).

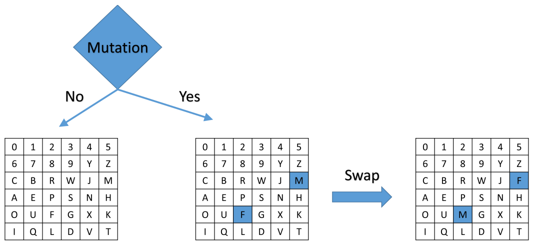

Fig. 5. Mutation

Termination - The generation process ends when the required number of chromosomes is reached. In the example, the process ends when 100 chromosomes are generated. The best one will be used for generating the keyboard.

IV. Performance analysis

In order to analyse the performance of the GA-based keyboard, it is necessary to adjust the parameters of the genetic algorithm [10]. In the following sub-sections A to E, the initial population size, mutation probability and termination size are adjusted to evaluate performance. The standard QWERTY keyboard is a good reference for comparison. To compare the performance between a GA-based keyboard and a standard QWERTY keyboard, Algorithm 3 is used for computing the typing cost of each keyboard. Basically, the moving cost between two keys is based on distance (i.e., as calculated by their coordinates). For example, if key 1 and key 2 have coordinates (1,2) and (3,4), respectively, the moving cost (i.e., from key 1 to key 2) is $\sqrt{(3-1)^2+(4-2)^2} = 2.8284$.

Algorithm 3 Calculating the moving cost between two keys

```
1: procedure NormalCost($text$)
2:     $score \leftarrow 0$
3:     for each $i \in text$ do
4:         $x1, y1 \leftarrow$ LookupStandardKeyboard $(i)$
5:         $x2, y2 \leftarrow$ LookupStandardKeyboard $(i)$
6:         $distance \leftarrow \sqrt{(x2-x1)^2+(y2-y1)^2}$
7:         $score \leftarrow score + distance$
8:     end for
9:     return $score$
10: end procedure
```

A. Tuning initial population

Figure 6 shows the performance improvement of the GA-based keyboard as compared with the normal (QWERTY) keyboard for different initial population sizes. The simulation results indicate that performance is not stable when the initial population size is small. When the initial population size is 10, the performance is unstable and the proposed GA-based keyboard outperforms the QWERTY keyboard by approximately 35 to 40 per cent. When the initial population size is set to 50, performance becomes more stable. With more than 1,000 chromosomes, performance becomes stable and the performance improvement of the GA-based keyboard is around 42 per cent. When the initial population size is set to 100, the performance becomes steadier and the performance improvement of the GA-based keyboard is about 42 per cent after 1,000 chromosomes. Therefore, it is suggested to set the initial population size to at least 100 chromosomes to achieve a better and more stable performance.

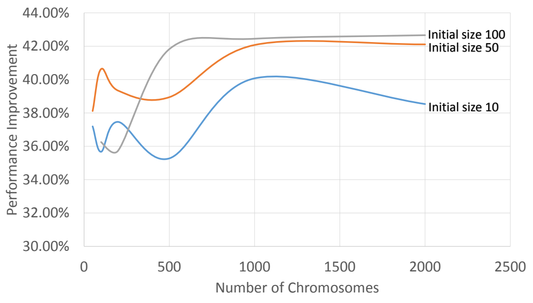

Fig. 6. GA initial population and performance

B. Tuning mutation probability

Figure 7 shows the performance improvement of the GA-based keyboard as compared with the normal (QWERTY) keyboard for different mutation probabilities. The simulation results indicate that the mutation probability has little effect on performance improvement. When the number of chromosomes is small, performance improvement is around 37 per cent. When the number of chromosomes is large, performance improvement becomes steady at around 42 per cent. Based on the simulation results, it is suggested to set the mutation probability to 5 per cent.

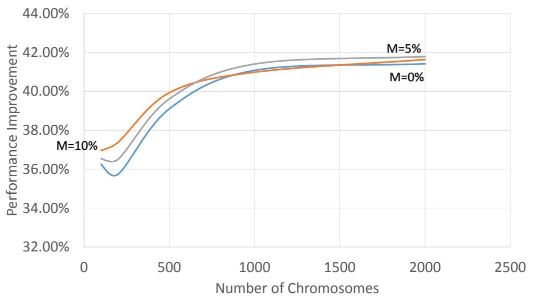

Fig. 7. GA mutation probability and performance

C. Tuning termination

Figure 8 shows the time cost of the simulations. With more than 200 chromosomes, the time cost increases more than double. To generate 1,000 chromosomes, it takes around three minutes. For generating 2,000 chromosomes, it takes close to seven minutes. According to sub-sections A and B, performance becomes steady when the program generates 1,000 chromosomes. Performance does not improve distinctly after that. However, the time cost and computer resources cost increase significantly. Therefore, it is suggested to terminate a simulation when 1,000 chromosomes are generated.

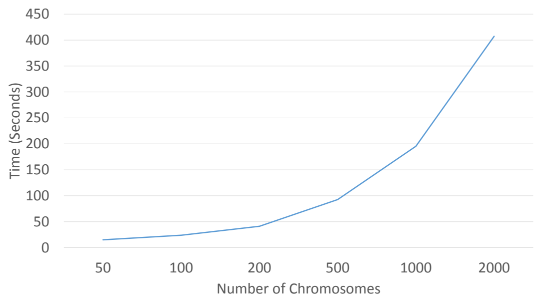

Fig. 8. GA running time cost

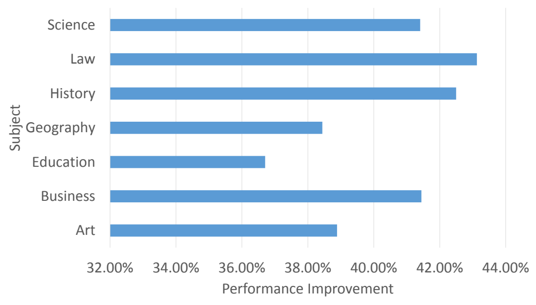

Fig. 10. Performance of the GA-based keyboard for seven subjects

D. Comparing typing cost for different subjects

The above analysis provides the preferred setting for the simulations: initial population size of 100 chromosomes, mutation probability of 5 per cent and termination size of 1,000 chromosomes. The following evaluation is based on this preferred setting.

In order to evaluate the performance of the GA-based keyboard for different subjects, various articles from the Internet were used. The article subjects are Art, Business, Education, Geography, History, Law and Science. Figure 9 shows the typing cost per word for each subject for people with disabilities using the standard QWERTY keyboard. The typing cost is around 15 to 16 except for Law, which is higher than 17. This is because legal terms are generally longer and more complicated than in other subjects. Therefore, the typing cost per word for Law is higher than for other subjects. For the same reason, Science has the second highest typing cost. It is found that Art has the lowest typing cost compared with other subjects.

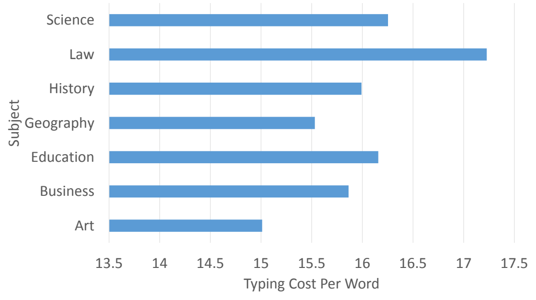

Fig. 9. Typing cost per word for seven subjects

Figure 10 shows the performance improvement (i.e., as compared with the typing cost for the normal keyboard). As mentioned, the costs for Law and Science are higher, because legal and scientific terms are generally longer and more complex. Note that moving cost is affected by word complexity. With the design of the GA-based keyboard, moving costs can be minimised. In general, the performance improvement is between 37 and 42 per cent as compared with a QWERTY keyboard. This shows that the proposed GA-based keyboard can enhance the standard QWERTY keyboard when typing long and complicated words. For example, the performance improvement for Law and History can reach more than 42 per cent. The performance improvement for Science and Business can reach more than 40 per cent. In summary, the proposed GA-based keyword can facilitate people with disabilities to type more effectively and efficiently.

Next, it is of interest to analyse the relationship between performance improvement and the word count. Another set of articles from the Internet was used for the evaluation. The number of words was between 250 and 2,000. Figure 11 shows that in general, there is no direct relationship between performance and word count. In other words, a long article (with more words) has little effect on the typing cost per word. However, it is found that performance improvement is better when the number of words is between 450 and 900, possibly because people tend to use simpler words for shorter articles. Also, performance decreases slightly when the number of words falls between 900 and 2,000.

In general, performance improvement is approximately 35 to 45 per cent. That means, when a disabled person uses a GA-based keyword for typing, the typing cost can be reduced by 40 per cent, compared with a normal keyword.

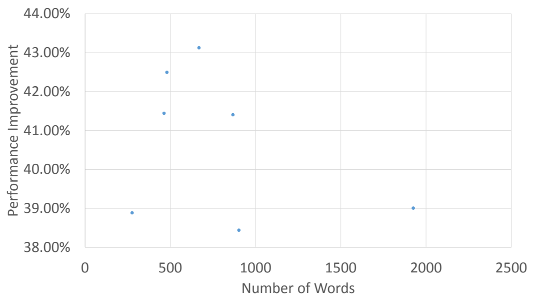

Fig. 11. Performance versus number of words

E. Comparison for different keyboard designs

Last but not least, the typing cost for different keyboard designs is evaluated based on a set of news articles obtained from the Internet. Each article, from different newspaper websites e.g. BBC, CNN, etc., has 200-250 words. Figure 12 shows that the typing cost for the standard keyboard is the highest. In comparison, keyboards with keys arranged in ascending or random order can provide some improvements. The GA-based keyboard provides the best performance. Its typing cost is about 40 per cent of the normal QWERTY keyboard. Hence people with disabilities can type more efficiently with the GA-based keyboard.

V. Implementation

For demonstration and evaluation purposes, a virtual keyboard prototype has been implemented based on the Model-View-Controller (MVC) model [11]. Model (M) is related to

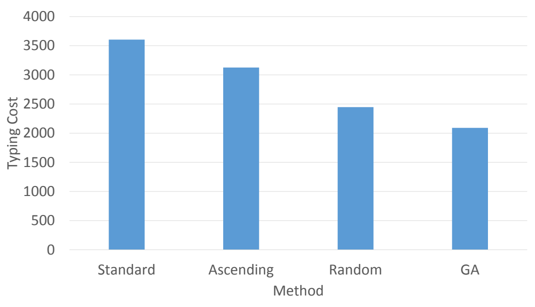

Fig. 12. Comparison of different keyboards (lower cost is better)

the keyboard algorithm as well as data management. View (V) is about keyboard interface (i.e., based on the design discussed above). Controller(C) is for handling a user request with the Leap Motion sensor [12].

First, it is necessary to construct the program structure. There are two threads. One is responsible for keyboard operation. Another one is responsible for Leap Motion communications. Both threads start at the beginning when the keyboard program is initiated. There is a controller in the MVC model program. It is necessary to provide a dedicated listener for the Leap Motion sensor. The listener is managed by the controller. It is added to the controller when the program is started. In addition, the listener is removed before the program ends.

The suggested keyboard size is 600 pixels in height and 600 pixels in width (see Figures 13 and 14). Each button is 70 pixels in height and 70 pixels in width. The gap between the buttons is 30 pixels. The button size is designed to be larger than the normal key size, so that it is easier for a disabled user to select the number or letter. The gap is to prevent the user from inadvertently clicking an incorrect button.

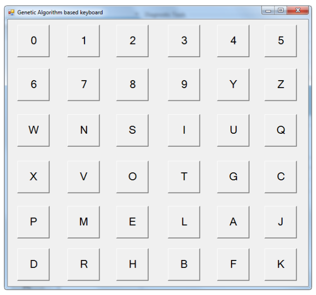

Fig. 13. Keyboard prototype

The proposed keyboard may also be used with other input tools for disabled users, some of whom may have other special tools for using computers. Scanning software is particularly useful for disabled users. With just one button, the software can scan the screen from left to right horizontally and then up and down vertically [13]. Users can select the position by pressing a button. Eye tracking software may also be used with the proposed keyboard. The larger button size is designed to allow disabled users to click the buttons more easily. Also, the larger gap between consecutive buttons seeks to prevent them from selecting an incorrect key.

VI. Conclusion

In summary, Table IV gives the suggested setting for the genetic algorithm (i.e., for generating the GA-based

Fig. 14. Testing the special virtual keyboard prototype

keyboard). The parameters are found based on the aforementioned performance analysis. It is suggested to set the initial population size to 100 chromosomes, mutation probability to 5 per cent and termination size to 1,000 chromosomes. A virtual keyboard prototype has been built with the Model-view-controller (MVC) model. The performance analysis indicates that the proposed keyboard can outperform the normal QWERTY keyboard by about 40 per cent in terms of typing cost.

TABLE IV
Suggested GA setting

Initial population	100 chromosomes
Mutation probability	5 percent
Termination size	1,000 chromosomes

References

[1] N. Shirahama, Y. Sakuragi, S. Watanabe, N. Nakaya, Y. Mori, and K. Miyamoto, "Development of input assistance application for mobile devices for physically disabled," in *2014 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2014, pp. 1–6.

[2] M. Salter and L. Cheshire, *Hand therapy: principles and practice*. Elsevier Health Sciences, 2000.

[3] T. R. Ostrach, "Typing speed: How fast is average: 4,000 typing scores statistically analyzed and interpreted," *Orlando, FL: Five Star Staffing*, 1997.

[4] J. J. Daly and J. R. Wolpaw, "Brain–computer interfaces in neurological rehabilitation," *The Lancet Neurology*, vol. 7, no. 11, pp. 1032–1043, 2008.

[5] F. Akram, M. K. Metwally, H.-S. Han, H.-J. Jeon, and T.-S. Kim, "A novel p300-based bci system for words typing," in *2013 International Winter Workshop on Brain-Computer Interface (BCI)*. IEEE, 2013, pp. 24–25.

[6] M. Raynal and N. Vigouroux, "Genetic algorithm to generate optimized soft keyboard," in *CHI05 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2005, pp. 1729–1732.

[7] C. R. Brewbaker, "Optimizing stylus keyboard layouts with a genetic algorithm: customization and internationalization," *Dept. of Computer Science, Iowa State University*, 2008.

[8] J. Guna, G. Jakus, M. Pogačnik, S. Tomažič, and J. Sodnik, "An analysis of the precision and reliability of the leap motion sensor and its suitability for static and dynamic tracking," *Sensors*, vol. 14, no. 2, pp. 3702–3720, 2014.

[9] Wikipedia, "Touch typing," Available at https://en.wikipedia.org/wiki/$Touch_{t}yping$.

[10] R. L. Haupt and S. E. Haupt, "The continuous genetic algorithm," *Practical Genetic Algorithms, Second Edition*, pp. 51–66, 2004.

[11] T. Reenskaug and J. O. Coplien, "The dci architecture: A new vision of object-oriented programming," *An article starting a new blog:(14pp) http://www.artima.com/articles/dcivision.html*, 2009.

[12] LeapMotion, "Leap motion product," Available at https://www.leapmotion.com/product/desktop.

[13] P. Biswas and P. Langdon, "A new input system for disabled users involving eye gaze tracker and scanning interface," *Journal of Assistive Technologies*, vol. 5, no. 2, pp. 58–66, 2011.