TapBoard 2: Simple and Effective Touchpad-like Interaction on a Multi-Touch Surface Keyboard

Sunjun Kim and Geehyuk Lee
School of Computing, KAIST
Daejeon 305-701, Republic of Korea
{kuaa.net, geehyuk}@gmail.com

ABSTRACT

We introduce TapBoard 2, a touchpad-based keyboard that solves the problem of typing and pointing disambiguation. The pointing interaction design of TapBoard 2 is nearly identical to natural touchpad interaction, and its shared workspace naturally invites bimanual pointing interaction. To implement TapBoard 2, we developed a novel gesture representation scheme for a systematic design and gesture recognizer. A user evaluation showed that TapBoard 2 successfully supports collocated pointing and typing interaction. It was able to disambiguate typing and pointing actions with an accuracy of greater than 95%. In addition, the typing and pointing performance of TapBoard 2 were comparable to that of a separate keyboard and mouse. In particular, the bimanual pointing operations of TapBoard 2 are highly efficient and strongly favored by participants.

Author Keywords
TapBoard; pointing; typing; multi-touch; gesture; touch screen; touch pad; text entry system.

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User Interface

INTRODUCTION

The keyboard and mouse have been the de facto standard input devices for text entry and pointing. One common concern is the burden of frequently switching between devices and the extensive space that they require [4]. To address these concerns, many studies have tried to unify the keyboard and mouse control spaces [3,4,7,18,19]. One simple approach is to integrate an isometric joystick into a keyboard [13]. However, an isometric joystick is not as efficient as isotonic pointing devices [17].

Another approach to unifying keyboard and mouse control spaces is to add a finger-tracking function to a keyboard. FingerMouse [11], AirMouse [12], and FlowMouse [20] all utilize a down-looking camera and set pointing space above the typing space. For "on-the-keyboard" interaction, DGTS [7], Touch & Type [4], Moky [3], and FlickBoard [18] embed touch sensors on the keyboard and enable touchpad-like interactions. In these cases, typing and pointing share the same input space (the keyboard surface), and a method of disambiguating strokes for typing and pointing is employed. The disambiguating methods vary: a separate mode button [3,4,20], special hand shape [11,12], or software classifier [7,18] have been studied. However, the current sensor on a physical keyboard is not sufficiently fast and accurate for fluent cursor manipulation. In addition, common concerns exist regarding irregular touch surfaces [7] and limited touch sensor sensitivity because of thick keyboard buttons [4].

Yet another approach to unifying keyboard and mouse spaces is to use a large touch surface. A representative work in this field is that of Westerman [19], which is based on a FingerWorks TouchStream product. This touchpad-based approach provides a precise and smooth pointing operation compared to pointing on a physical keyboard. It uses a two-finger drag and tap gesture for mouse operations. However, a two-finger tap occasionally conflicts with consecutive overlapping taps during fast typing [19]. Multi-touch gestures on a flat keyboard are also used for different purposes besides pointing. These include menu invocation [5,8], alternative character insertion [1,5], and caret manipulation [6,9]. Moving distance [1,6,8], dwell time [9], and multi-touch [5,6,9] are often used to discriminate keystrokes and gestures.

In this study, we introduce TapBoard 2, which solves the problem of typing and pointing disambiguation by the TapBoard concept of Kim et al. [9]. They showed that people usually type on a touchscreen keyboard by tapping and that a tapping-only soft keyboard is as fast and accurate as an ordinary soft keyboard. These results may also be valid with respect to a touchpad-based keyboard and motivated us to implement a tapping-only soft keyboard on a touchpad. One result is that we can use dragging gestures longer than a tap threshold (450 ms and 5 mm) for pointing, thereby solving the typing and pointing disambiguation problem without requiring a mode key or special gestures.

The purpose of the current study is to verify the feasibility and benefits of TapBoard 2. First, we show that the concept of TapBoard effectively solves the typing and pointing disambiguation problem. Second, we show that typing and

CHI'16, May 07-12, 2016, San Jose, CA, USA
© 2016 ACM. ISBN 978-1-4503-3362-7/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2858036.2858452

pointing performance is preserved after the two inputs share the same control space. Third, we display the benefit of a large keyboard-sized touch surface that invites natural bimanual touch operations. As an example, we show the advantage of dividing pointing and clicking operations between the two hands.

TAPBOARD 2

We implemented a TapBoard 2 prototype using a touchscreen tablet computer that embeds a fast (100 fps) and high-resolution (sub-millimeter) multi touch surface. The tablet always displays a keyboard pattern and functions as an external keyboard and mouse device for a host computer.

TapBoard Gesture Recognizer

As shown in Figure 1, a touch point exists in any one of the following three phases: *Touched*, *Dwelling*, and *Moving*. A touch point starts at the *Touched* phase upon finger touchdown. After a time threshold τ (450 ms [9]), it automatically transitions to the *Dwelling* phase. Whether in the *Touched* or *Dwelling* phase, when a touch point travels more than a distance threshold δ (5 mm [1,9]), it enters the *Moving* phase. At the *Moving* phase, touch point does not change its phase until finger lift-off. The gesture recognizer enumerates the number of touch points in each phase. We define a system state as having three consecutive numbers such as `213`, which denotes two *Touched* points, one *Dwelling* point, and three *Moving* points.

A state sequence then defines a TapBoard gesture. For example, `{000,100,000}` defines a tap gesture, `{000,100,001}` defines a drag gesture, and `{000,100,200,101,002}` defines a two-finger dragging gesture. The system declares that a gesture has been detected when the latest state history matches the state sequence of the gesture. For ease of illustration, consider a simple gesture keyboard that moves a text caret with a one-finger dragging gesture and scrolls the document with a two-finger dragging gesture. Gesture definitions of such a keyboard are shown in Table 1.

This historical record of gestures may prevent conflicts between gestures more effectively than can state-based gesture recognition. Suppose that a naïve gesture recognizer maps `001` (one-finger moving) as a caret-move gesture and `002` (two-finger moving) as a scroll gesture. After performing a scroll gesture, a user may then release two fingers separately within a fraction of a second. In this case, the recognizer may detect a caret-move gesture, which results in an unintended small caret movement. In our system, the state sequence of the caret-move gesture `{000,100,001}` does not match the state history of releasing two fingers `{002,001,000}` and, thus, this type of erratic caret movement cannot occur. This problem may be addressed using a timer or context variable. However, our approach enables a more systematic and compact gesture recognizer implementation.

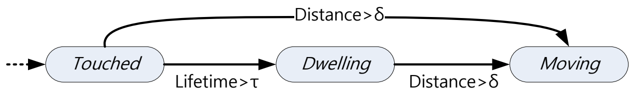

Figure 1. Three phases of a touch point. We set τ = 450 ms and δ = 5 mm for our implementation.

Gesture	Description
`{000,100,000}`	Keystroke
`{000,100,001}`	Caret move
`{000,100,200,101,002}`	Document scroll

Table 1. Simple gesture keyboard showing TapBoard gesture representation

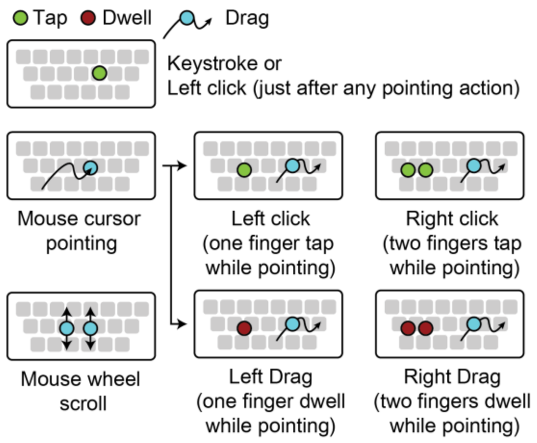
Tap
Dwell
Drag
Keystroke or
Left click (just after any pointing action)
Mouse cursor pointing
Left click (one finger tap while pointing)
Right click (two fingers tap while pointing)
Mouse wheel scroll
Left Drag (one finger dwell while pointing)
Right Drag (two fingers dwell while pointing)

Figure 2. Designed touchpad-like pointing interaction.

Gesture	Description
`{100,000},{200,100},{300,200}`	Keystroke
`{100,001},{100,010}`	Mouse cursor move
`{000,100,000},{010,110,010},{001,101,001}`	Mouse left-button click
`{000,100,200,100,000},{001,101,201,101,001},{010,110,210,110,010}`	Mouse right-button click
`{101,011},{110,011},{011,002},{110,020,011}`	Mouse left-button press and drag
`{201,111,021},{120,021},{021,012},{021,012,003}`	Mouse right-button press and drag
`{011,001},{010,000},{001,000}`	Mouse left and right release (if pressed)
`{200,101,002}`	Mouse-wheel scroll

Table 2. Interaction represented by TapBoard gesture definitions. Underlined items represent conflicting gestures (see text).

Touchpad-like Pointing Interaction on a Keyboard

Using the TapBoard gesture recognizer, we designed a touchpad-like pointing interaction (Figure 2) that can coexist with a tapping-only keyboard. Tapping functions as a keystroke except when it follows a pointing gesture. For our system, mapping a release-and-tap gesture to a left-click event was required because this is a general touchpad skill. However, this matches both keystroke `{100,000}` and left-button `{000,100,000}` gestures and, thus, represents a conflict. We treated this as an exception and have the system execute a left-button gesture instead of keystroke only for a tap within one second and 15 mm from the previously touched pointing gesture. The other mouse operations are defined without conflicts. One-finger dragging is used to control a mouse cursor. Two-finger dragging is used for mouse-wheel scrolling. During one-finger dragging, additional finger touches function as mouse-button clicks, a one-finger tap functions as a left click, and a two-finger tap functions as a right click. Dragging with a button pressed is supported similarly by including *Dwelling* touches. The other hand should preferably perform these mouse-button gestures. In addition, we include dwell+taps as mouse-button gestures to enable a click without a cursor move.

EVALUATION

To evaluate the proposed design, we conducted a within-subject user study to answer the following questions.

1) Does pointing interfere with typing or vice versa?
2) Does the combined pointing-typing design impair typing and pointing performance?
3) Does bimanual interaction offer any advantages regarding pointing performance?

Tasks and Metrics

We designed a task that requires frequent switching between pointing and typing (Figure 3). First, a start point appears at the screen center, and clicking it starts a 2D Fitts' law pointing task [16]. We used three distance (D = 100, 200, and 800 px) and three target diameter (W = 40, 60, and 80 px) levels. Index of difficulty (ID) was calculated using the Shannon formulation $ID = \log_2 (D/W+1)$. We designed IDs to be distributed evenly from 1.0 to 4.5. Clicking a target point ends the pointing task. An effective index of difficulty (IDe) was calculated based on [16]. Following the pointing task, a textbox appears that begins a typing task. A word preview appears during the pointing task to minimize the effect of cognitive load on task time. Words of 5-7 characters are randomly chosen from the Enron corpus [10]. Pressing the Enter key ends the typing task.

In our study, as the two tasks were repeated, we measured the following metrics. For transition cost metrics, we measured the interval between the start of the new task – in other word, end of the typing task – and the first mouse move event (i.e., *Typing to Pointing transition time* or *T2P*), and the interval between successful target selection and the first keystroke (i.e., *Pointing to Typing transition time or P2T*). For pointing performance metrics, we calculated $Throughput = MT$ (movement time) / IDe, and the number of failed target selections (*Targeting Error*). For typing performance metrics, we measured *Typing Speed* in word per minute (WPM) and *Total Error Rate* [15]. For interference metrics, we measured *Keyboard Error*, which counts the number of keystrokes during the Fitts' law test, and *Mouse error,* which counts the number of mouse move or click events during typing. Finally, we measured the total task completion time (*Total Time*).

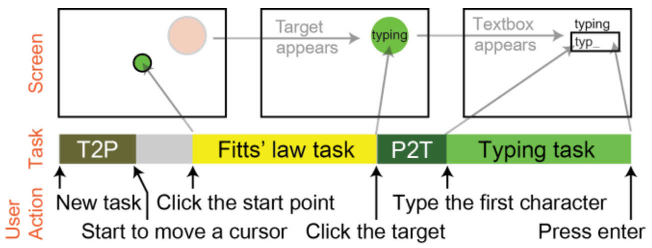

Figure 3. Task sequence consists of pointing and typing

Conditions and Apparatus

We established three *Conditions*: **Separated-Tap (ST)**, **Combined-Tap (CT)**, and **Combined-AddFinger (CA)**.

ST is the base condition that simulates a separate touchpad and keyboard configuration. It uses two tablet devices adjacently placed. One device simulates a soft keyboard and the other simulates a touchpad. The keyboard device interprets all touches as keystrokes, and the touchpad device interprets a drag gesture as a mouse move and a tap as a click gesture.

CT and **CA** are test conditions that use the proposed combined pointing-typing method. A single tablet device functions as both keyboard and touchpad. Both conditions share the same implementation. The sole difference concerns click-gesture choice. In our study, we instructed users to employ release-and-tap with **CT** and an additional finger tap with **CA** for the left-click gesture.

We implemented **ST**, **CT**, and **CA** on Microsoft Surface Pro 3 touchscreen tablets. To mimic touchpad-like appearance, we configure the tablets to show only a QWERTY layout with 18.2 x 18.6 mm keys, except the touchpad device for **ST** that displays only a black screen. The tablets generated keyboard and mouse events, and an Arduino Leonardo board interpreted these events to legacy USB keyboard and mouse packets. A separate laptop conducted the experiment tasks driven by the USB packets. The measured latency between the tablets and laptop were 50 ms on average. In our system, a keystroke highlights the corresponding soft key and generates a short beep sound (800 Hz tone, 10 ms). In **CT**, a small circle indicating the "clickable" area for the release-and-tap gesture is shown on the keyboard. Mouse acceleration, keyboard size, and device orientation were identical across conditions and participants.

Procedure

We recruited 18 paid participants (nine males and nine females, average age of 21.5) who were familiar with QWERTY keyboards. All participants were right-handed and operated a mouse with that dominant hand. A "block" in our study consists of all three conditions and 27 trial runs of pointing and typing tasks for each condition. Each participant completed three blocks. Thus, 243 trials (3 x 3 x 27) in total were conducted. We counterbalanced the order of conditions across the participants. The first block was a practice block, and we analyzed the data from the last two blocks.

Results

Metrics	ST	CT	CA
T2P Time (ms)	497	473	469
Targeting Error (%)	2.6	1.9	2.2
Throughput (bits/s)	3.1	2.9	**3.9**
Keyboard Error (%)	**0.7**	**1.2**	3.9
P2T Time (ms)	579	**490**	640
Typing Speed (WPM)	39.6	37.6	37.4
Total Error Rate (%)	5.6	6.9	7
Mouse Error (%)	**0.6**	3.4	2.9
* Mouse Click Error (%)	0.4	0.3	0.2
Total Task Time (ms)	4944	4868	4761

Table 3. The average results of blocks 2 and 3 (Boldface values indicate the best performance groups.)

Table 3 shows the average results of the second and third blocks ordered by the task flow illustrated in Figure 3. The study was a 2 (*Block*) × 3 (*Condition*) within-subject design. We performed a repeated measures ANOVA. *Condition* exhibited significant main effects on *Throughput*, *Keyboard Error*, *P2T*, *Typing Speed* and *Mouse Error*. *Block* exhibited significant main effects on *T2P* and *Throughput*. There were significant *Condition* * *Block* interaction on *T2P* and *P2T*. For the detailed results, refer to Appendix 1.

As a post hoc test, we performed pairwise Tukey's HSD tests between *Conditions* with $\alpha = 0.05$. Boldfaced values in Table 3 indicate the best performance groups. We summarize the major findings below.

- For *T2P*, the pairwise comparison failed to find a difference between conditions. For *P2T*, the fastest condition was **CT**, followed by **ST** and **CA**.
- For pointing performance, **CA** exhibited the highest *Throughput*. No difference was found in *Targeting Error*.
- For typing performance, the pairwise comparison failed to find a difference between conditions, although *Condition* exhibited a main effect on *Typing Speed*. No difference was found in *Total Error Rate*.
- For interference metrics, **CA** produced more *Keyboard Errors* than did **ST** and **CT**. **CT** and **CA** generated more *Mouse Errors* than did **ST**.

Subjective Ratings

We asked participants to rank the exposed conditions. We then collected comments about conditions. The average ranks were 1.22 (**CA**), 2.22 (**CT**), and 2.56 (**ST**). **CA** was the most favored (15 of 18 participants ranked it as first).

Afterwards, we provided instructions on mouse operations (Figure 2). Participants rated the ease of use on a seven-point Likert scale. Left click received a score of 6.78, left drag 6.00, right click 5.89, and right drag 5.16.

Discussion

ST is better in disambiguation and typing performance. However, frequent device switching caused participant fatigue. **CT** improves transition time and is comparable to **ST** regarding pointing and typing performance. Participants appreciated short hand movements, but timeout-based click interaction produced strain. Bimanual interaction generated the best *Throughput* but the slowest *P2T* times for **CA**. Forcing a left finger to release to complete a mouse clicks may reduce *P2T* time in **CA**, whereas other conditions enabled participants to prepare left-hand keys. **CA** also exhibits the worst *Keyboard Error*. Raw log analysis showed that participants often release two fingers simultaneously for left-button click gestures, which records as `{001,101,100,000}` and fails to match predefined gestures. Participants then typed a word, which generated keystrokes before successful target selection. Nevertheless, participants mostly favored **CA** for its combined workspace and continuous bimanual target selection method.

Both **CT** and **CA** exhibited more *Mouse Errors* than did **ST**, but 88% of errors were move events. Participants sometimes move their fingers away from a key in order to cancel a keystroke and then swipe on the backspace key to remove an entire word. **ST** ignores such dragging movements, whereas **CT** and **CA** interpreted them as mouse-move events. Filtering out these mouse-move errors eliminated *Mouse Error* differences (Table 3 *).

The cursor-moving gesture requires some slack (5 mm) or dwelling (450 ms). Most participants initiated a cursor move by means of a shaking gesture [2], in which case the spatial slack becomes less pronounced. However, some participants complained that the system was unresponsive during subtle cursor manipulation. Here, the moved distance falls within the slack threshold and dwelling activates the cursor. We may address this limitation in the future by incorporating a probabilistic approach [14].

CONCLUSION

We developed TapBoard 2, a pointing-typing keyboard based on a systematic gesture recognizer. In our study, the system distinguished typing and pointing with over 95% accuracy. Moreover, typing and pointing performance with the combined interface was comparable to that of the base condition **ST**. Finally, because of bimanual interaction, **CA** outperformed other conditions in pointing *Throughput*, and participants strongly favored this.

A future study will extend the application of our design. TapBoard 2 functionality can be easily embedded in many touch-sensing platforms such as touchscreen tablets, tabletops, laser projection keyboards, flat keyboards such as Microsoft Surface Touch Cover, and glass keyboards. This design can also be applied to physical keyboards as soon as they are equipped with a fast and reliable touch-sensing technology.

ACKNOWLEDGMENTS
This work was supported by Hancom Inc. Its intellectual property rights reserved by KAIST and Hancom.

REFERENCES

1. Ahmed Sabbir Arif, Michel Pahud, Ken Hinckley, and Bill Buxton. 2014. Experimental study of stroke shortcuts for a touchscreen keyboard with gesture-redundant keys removed. In *Proceedings of GI '14*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 43-50.
2. Robert Ball and Chris North, Analysis of user behavior on high-resolution tiled displays. 2005. *Human-Computer Interaction-INTERACT* 2005.Vol 3585 of the series Lecture Notes in Computer Science: 350-363. http://doi.org/10.1007/11555261_30
3. Enyoung Cho, Moky: invisible touchpad keyboard. 2015. Retrieved Sep. 7, 2015 from https://www.indiegogo.com/projects/moky-invisible-touchpad-keyboard
4. W. Fallot-Burghardt, M. Fjeld, C. Speirs, S. Ziegenspeck, H. Krueger, and T. Läubli. 2006. Touch&Type: a novel pointing device for notebook computers. In *Proceedings of NordiCHI '06*. 465-468. http://doi.acm.org/10.1145/1182475.1182538
5. Leah Findlater, Ben Lee, and Jacob Wobbrock. 2012. Beyond QWERTY: augmenting touch screen keyboards with multi-touch gestures for non-alphanumeric input. In *Proceedings of CHI '12*. ACM, 2679-2682. http://dx.doi.org/10.1145/2207676.2208660
6. Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. Gestures and widgets: performance in text editing on multi-touch capable mobile devices. In *Proceedings of CHI '13*. ACM, 2785-2794. http://dx.doi.org/10.1145/2470654.2481385
7. Iman Habib, Niklas Berggren, Erik Rehn, Gustav Josefsson, Andreas Kunz, and Morten Fjeld. 2009. Dgts: Combined typing and pointing. *Human-Computer Interaction-INTERACT* 2009, Vol. 5727 of the series Lecture Notes in Computer Science: 232-235. http://doi.org/10.1007/978-3-642-03658-3_30
8. Poika Isokoski. 2004. Performance of menu-augmented soft keyboards. *In Proceedings of CHI '04*. ACM, 423-430. http://dx.doi.org/10.1145/985692.985746
9. Sunjun Kim, Jeongmin Son, Geehyuk Lee, Hwan Kim, and Woohun Lee. 2013. TapBoard: making a touch screen keyboard more touchable. In *Proceedings of CHI '13*. ACM, 553-562. http://doi.acm.org/10.1145/2470654.2470733
10. Bryan Klimt and Yiming Yang. 2004. The enron corpus: A new dataset for email classification research. *Machine learning:* ECML. Springer Berlin Heidelberg, 217-226. http://doi.org/10.1007/978-3-540-30115-8_22
11. Thomas A. Mysliwiec, 1994. Fingermouse: A freehand computer pointing interface. In *Proceeding of International Conference on Automatic Face and Gesture Recognition,* 372-277. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.4906
12. Michael Ortega and Laurence Nigay. 2009. AirMouse: Finger gesture for 2D and 3D interaction. *Human-Computer Interaction–INTERACT* 2009. Springer Berlin Heidelberg, 214-227. http://doi.org/10.1007/978-3-642-03658-3_28
13. Joseph D. Rutledge and Ted Selker. 1990. Force-to-motion functions for pointing. In *Proceedings of INTERACT '90*. 701-706.
14. Julia Schwarz, Scott Hudson, Jennifer Mankoff, and Andrew D. Wilson. 2010. A framework for robust and flexible handling of inputs with uncertainty. In *Proceedings of UIST '10*. ACM, New York, NY, USA, 47-56. http://dx.doi.org/10.1145/1866029.1866039
15. R. William Soukoreff and I. Scott MacKenzie. 2003. Metrics for text entry research: an evaluation of MSD and KSPC, and a new unified error metric. In *Proceedings of CHI '03*, ACM, 113-120. http://doi.acm.org/10.1145/642611.642632
16. William Soukoreff and I. Scott MacKenzie. 2004. Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. *International journal of human-computer studies,* 61, 6 (December 2004): 751-789. http://dx.doi.org/10.1016/j.ijhcs.2004.09.001
17. Christian Sutter and Martina Ziefle, 2005. Interacting with notebook input devices: An analysis of motor performances and user's expertise. *Human Factors,* 47: 169-187. http://doi.org/10.1518/0018720053653893
18. Ying-Chao Tung, Ta Yang Cheng, Neng-Hao Yu, Chiuan Wang, and Mike Y. Chen. 2015. FlickBoard: Enabling Trackpad Interaction with Automatic Mode Switching on a Capacitive-sensing Keyboard. In *Proceedings of CHI '15*. ACM, 1847-1850. http://doi.acm.org/10.1145/2702123.2702582
19. Wayne Westerman. 1999. *Hand tracking, finger identification, and chordic manipulation on a multi-touch surface*. Doctoral dissertation, University of Delaware.
20. Andrew D. Wilson, and Edward Cutrell. 2005. FlowMouse: a computer vision-based pointing and gesture input device, *Human-Computer Interaction-INTERACT*. Springer Berlin Heidelberg, 565-578. http://doi.org/10.1007/11555261_46

APPENDIX

We present the per block results, the repeated measures (RM) ANOVA results, and the post hoc test results, which we omitted in Table 3, in the following tables.

	Block 2			Block 3		
Metrics	**ST**	**CT**	**CA**	**ST**	**CT**	**CA**
T2P Time (ms)	510	493	472	483	**454**	**465**
Targeting Error (%)	2.7	1.6	1.4	2.5	2.1	2.9
Throughput (bits/s)	3.08	2.88	**3.79**	3.12	2.98	**4.00**
Keyboard Error (%)	**1.0**	**1.0**	4.1	**0.4**	**1.4**	3.7
P2T Time (ms)	591	**489**	645	567	**492**	634
Typing Speed (WPM)	39.7	37.7	36.5	39.5	37.5	38.2
Total Error Rate (%)	5.3	6.7	6.7	5.8	7.1	7.2
Mouse Error (%)	**0.4**	3.7	3.3	**0.8**	3.1	2.5
* Mouse Click Error (%)	0.4	0.4	0.2	0.4	0.2	0.2
Total Task Time (ms)	4961	4846	4807	4926	4889	4715

Table 4. Complete results from blocks 2 and 3

	Condition		Block		Interaction	
	F(2,16), α=.05		F(1,17), α=.05		F(2,16), α=.05	
Metrics	***F*-val**	***p*-val**	***F*-val**	***p*-val**	***F*-val**	***p*-val**
T2P Time (ms)	0.72	.500	15.2	**< .001**	3.72	**.047**
Targeting Error (%)	.516	.607	.938	.346	.991	.393
Throughput (bits/s)	74.3	**< .001**	12.3	**.002**	1.53	.247
Keyboard Error (%)	8.54	**.003**	.321	.578	.634	.543
P2T Time (ms)	25.0	**< .001**	.783	.388	25.0	**< .001**
Typing Speed (WPM)	8.75	**.003**	.511	.484	1.73	.208
Total Error Rate (%)	2.52	.112	.994	.333	0.01	.989
Mouse Error (%)	5.63	**.014**	.747	.395	.974	.399
* Mouse Click Error (%)	1.60	.233	.191	.668	.056	.946
Total Task Time (ms)	1.70	.214	.232	.636	.579	.572

Table 5. RM-ANOVA results

Metrics	Block 2	Block 3
T2P Time (ms)	ST = CT = CA	ST = CT = CA
Throughput (bits/s)	CA > ST = CT	
Keyboard Error (%)	CA > ST = CT	
P2T Time (ms)	ST = CA > CT	CA=ST, ST=CT CA > CT
Typing Speed (WPM)	ST = CT = CA	
Mouse Error (%)	CT = CA > ST	

Table 6. Conditions sorted by Tukey's HSD test results. The insignificant metrics according to RM-ANOVA results are excluded. We performed block-wise analyses for T2P and P2T, which exhibited *Condition*Block* interactions according to RM-ANOVA results.