KeySlide: Using Directional Gestures on Keyboard for Cursor Positioning

Arihant Parsoya
Indian Institute of Technology Bombay
Mumbai, India
parsoyaarihant@gmail.com

Venkatesh Rajamanickam
Indian Institute of Technology Bombay
Mumbai, India
venkatra@iitb.ac.in

ABSTRACT

KeySlide is a novel interaction technique to position the text cursor in text editors by refashioning standard physical keyboard used in desktop and laptop devices. The technique enables users to position the cursor at the desired location on text documents using natural and intuitive finger-sliding gestures over the keys of the keyboard. The natural mapping of the gesture to the action eliminates the learnability of the action, and minimises the disruption caused due to switching between the keyboard and the mouse. The design, implementation and evaluation of KeySlide is presented. While the theoretical simulation study found that KeySlide is 52% faster than standard keyboard shortcuts for cursor positioning, a comparison user study consisting of 18 participants, found that text editing performance using KeySlide is 19% slower than using keyboard shortcuts. The discrepancy is discussed.

Author Keywords

Cursor positioning; Text Editor; Gestural Input

CCS Concepts

•**Human-centered computing** → **Human computer interaction (HCI);** *Haptic devices;* User studies; Please use the 2012 Classifiers and see this link to embed them in the text: https://dl.acm.org/ccs/ccs_flat.cfm

INTRODUCTION

Keyboard was the sole input mechanism in early computer applications. Newer input mechanisms such as the mouse gave rise to complex GUI driven applications with vastly expanded features and capabilities. Specifically in the case of text editing, the applications grew increasingly sophisticated as it was now possible to embed a big range of functions that can be invoked from menus and tool boxes through the mouse. The downside to this bounty of features was the interruption to the workflow that the user had to endure from switching between the keyboard and the mouse. The interruption is particularly unwelcome in text editing applications where the user is primarily engaged with the keyboard.

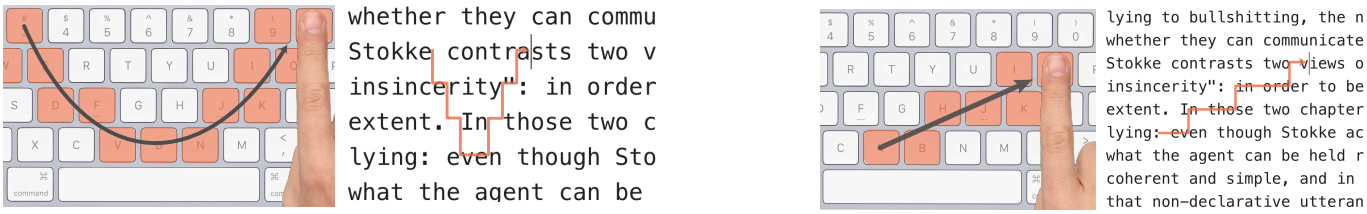
To overcome this, expert users rely on keyboard shortcuts. These shortcuts eliminate the homing time between the keyboard and the mouse but they remain underutilized [8]. One reason for this is the large learning curve it takes to learn new shortcuts [23]. Another method to overcome the disruption is to use gestures. Studies show that gestures can be effectively memorized and provide high accuracy [7, 13, 18].

In this paper, we present KeySlide, a novel interaction technique to position the text cursor [39] using gestural interactions on a standard physical keyboard used in desktop and laptop devices. The basic idea is simple: we refashion the keyboard as a trackpad, and use intuitive and natural finger sliding over the keyboard – we call this directional gestures – to position the cursor in a text editor. Users will be able to transfer the skills they are familiar with in using a mouse or trackpad directly to using this technique. The cursor is positioned at the desired location by sliding a finger over the keys on the keyboard. The speed of the cursor is proportional to the speed at which the gesture is performed, giving control over the behaviour of the cursor. The user can switch between the normal function of text input through keyboard and KeySlide by pressing a mode key on the keyboard. This gives seamless switching between their normal editing task and cursor positioning.

Previous techniques have used similar gestural input on a standard keyboard to trigger commands [43] such as cut, copy and paste. However, the use of directional gestures for executing continuous actions has not been explored. Using analog keyboard for gestural input introduces new challenges due to the low resolution data that is obtained from the keyboard. This makes the prediction of the direction of gesture a challenging task, and limits the accuracy of the position the cursor. Performing gestures on a physical keyboard also introduces challenges in the user experience and comfort of performing the gestures. These challenges are discussed in detail in this paper.

To understand the feasibility of our proposed technique KeySlide over keyboard shortcuts, we conducted two comparison studies between keyboard shortcuts which are already part of most text editors and KeySlide. In our first study (theoretical), we ran simulations to know the maximum efficiency

India HCI '19, Nov 01–03, 2019, Hyderabad, Telangana, India
ACM xxx-x-xxxx-xxxx-x/xx/xx. . . $15.00
DOI: http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx



(a) (b)

Figure 1: Different modes of KeySlide technique. *short* mode (a) allows for positioning the cursor in unit steps by sliding the finger on the keyboard. *long* mode (b) moves the cursor in the direction of the gesture.

that can be achieved through both the techniques. In the second study, we did a user study (experimental), we compare the theoretical results with how real users use the technique in real text editing tasks.

We begin with the discussion of related work done in keyboard gestures and cursor positioning. We then describe our design ideas of using keyboard as text-cursor positioning device. Next, we describe our technique and challenges associated with using an standard keyboard for directional gestures. Next, we discuss the two comparison studies we conducted and the results obtained from them. We conclude by discussing key insights and future work.

RELATED WORK

Gesture based interactions have been studied extensively in HCI [22]. However, gestures have been used rarely for cursor positioning [14]. Our work relates to two areas: keyboard gestures and cursor positioning.

Keyboard Gestures

GestKeyboard [43] is a keyboard based interaction technique which is performed on an ordinary physical keyboard. Both directional and symbolic gestures are performed on the keyboard. The keypress data obtained from the keyboard is used to detect the type of gesture the user is preforming. The gesture can be mapped to different commands the user want to trigger. For example, slide left command can be used to delete selected text in the text editor. The commands used in GestKeyboard are designed for discrete actions. GestKeyboard is not designed for continuous manipulation such as positioning of the cursor.

GestAKey [33] augments the function of the analog keyboard by detecting the location and motion of touch on the keycap. The user can perform the gesture on individual keycap which is used to trigger commands. The gestures detected by GestAKey is limited because of the low area of keycap over which the gesture is performed.

Soft keyboards have been widely used for gesture interaction as it provides with high resolution input data which is suitable for gesture classification. Previous research on gestural input has been used to trigger commands [1] and aid in text entry [11, 16, 5]. Multiple studies [12, 2, 36, 20] has shown how text editing and cursor navigation can aid from gestural interaction.

Cursor positioning

Pointing Stick. Pointing stick [40] allows for controlling the position of the cursor using a small joystick placed on the keyboard. The pressure applied on the joystick is used to control the speed of the cursor. Pointing stick is usually placed at the centre of the keyboard giving quick assess for cursor positioning.

Code editing. Special ([6]) methods have been used for code editing as the code is logically structured as a syntax tree [38]. Having structured code enable for automatic syntax corrections and completions [31, 28, 3], which help reduce the work of programmers to edit the code for corrections.

Most text editors provide keyboard shortcuts to execute commands in the editor. Many keyboard shortcuts are also provided for cursor positioning. VIM [41] is a text editor which is entirely based on keyboard based interactions. It contains *visual* mode which is used for positioning and manipulation of selected texts. VIM is used by programmers for faster text editing as it provides support for custom shortcuts and personalizing. However, these shortcuts have a longer learning curve and are software specific. Approaches to help improve command selection using keyboard shortcuts have also been explored [18, 10].

Gaze. Several gaze based pointing has been used to point the cursor [26, 29, 27, 34, 17, 37]. Gaze only provides the direction of users interest but not his intentions, hence multiple methods have been used to allow the user to trigger commands such as blinking or head gestures [29]. EyePoint [25] uses combination of gaze and keyboard triggers for pointing and selection. Gaze based pointing suffers from inaccuracies in pointing and jitter which makes it impossible to have pixel level accuracy with gaze alone [15].

Speech. Speech recognition has been used to position the cursor based on the command the user gives through voice [42, 24]. Two main approaches in speech based cursor control are target-based and direction-based [24]. Target based approach positions the cursor based on the appropriate command. For example, the command "move cursor to the top" will position the cursor to the top of the screen. Direction based approach positions the cursor based on the direction which the user intends to move the cursor. For example, the command "move 3 words right" will position the cursor by three words in the right direction. Speech based cursor control generally have low performance than mouse inputs.

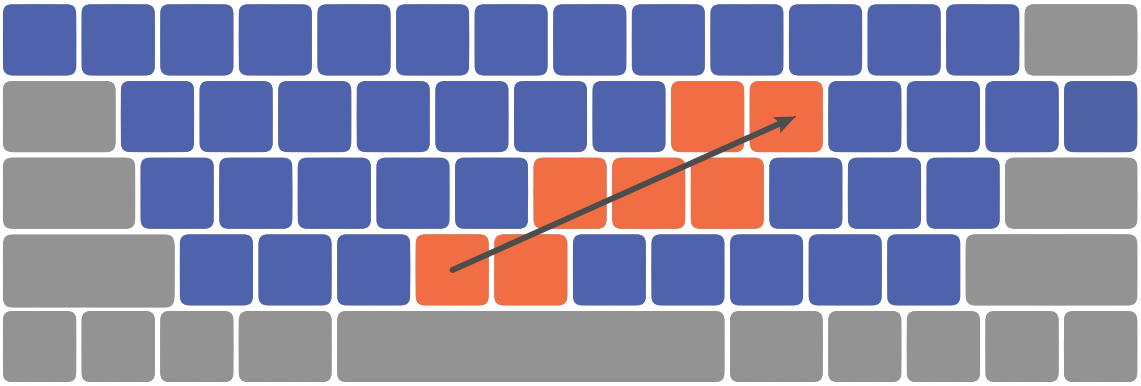


Figure 2: When KeySlide is activated, the keys highlighted in blue and orange can be used for cursor positioning. As the user performs the gesture on the keyboard, the keypress is detected (orange) and used for estimating the direction of the users gesture.

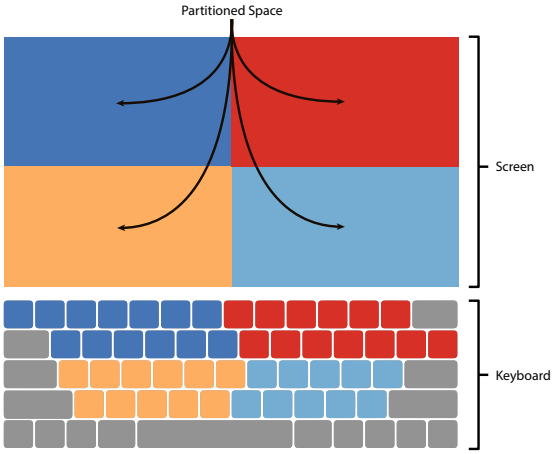


Figure 3: Positioning the cursor to different screen location by partitioning the screen space.

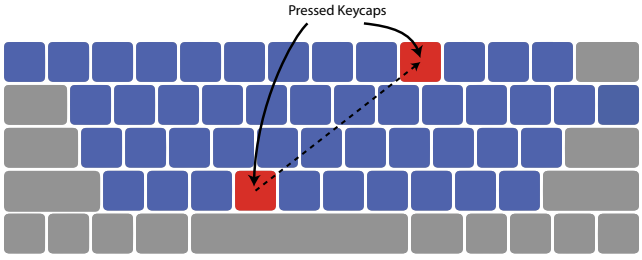


Figure 4: Inputting direction by pressing two keys on the keyboard. The dotted line represent the direction obtained from the keyboard data.

DESIGN SPACE

Using keyboard for spatial input is particularly interesting because the information obtained from the keyboard is discrete as the keycaps are spaciously placed as suppose to mouse or trackpad where continuous data is obtained. Hence, to support spatial input via keyboard, we came up with multiple design ideas to enable cursor positioning.

Cursor Direction. To position the cursor at a given target location, the user need to initiate the motion of the cursor in the direction of target location. To specify the direction, the user can press two keys on the keyboard (Figure 4), the angle between the pressed keys will determine the direction where the cursor needs to be moved. Once the direction of cursor motion is determined, cursor will move in that direction. The velocity of cursor movement can be constant or non-linear. The direction of motion can also be specified by performing a slide gesture on the keyboard (Figure 2). The direction of the gesture can be estimated by the keys which were pressed and their position on the keyboard. This technique has been used in KeySlide. The keys on the keyboard can also be mapped to move the cursor only in axial (vertically and horizontally) directions. VIM [41] uses similar cursor navigation technique, in "visual" mode of VIM, the cursor movement is mapped to the keys K (upwards), J (downwards), L (right) and H (left). In case of VIM, users requires to memorize the keys and direction mapping to use the navigation. We believe, the VIM navigation can be made more intuitive by mapping the keys based on the direction on keyboard.

Cursor Distance. As the direction of cursor movement is determined, the distance by which the cursor translates can be of variable distance. For example, when the cursor moves horizontally, it can be translated by fixed distance every time the user triggers the cursor motion. Or, the cursor can translate the beginning/end of the next word.

Area Partition. By partitioning the screen area into grids, the screen area can be mapped to the keyboard (Figure 3). The user can select the part of the screen he wants to focus by pressing the corresponding keys on the keyboard. The selected

space can be partitioned further and can be selected by the user until there cannot be further partitions.

KeySlide TECHNIQUE

Homing time Previous studies [35] show that using mouse as cursor positioning device requires an average of 0.8 second homing time - the time the user spends in switching between the keyboard and the cursor positioning device [35]. Using the keyboard as cursor positioning device will help in reducing the homing time since the user doesn't have to re-position hands in the process.

Directional Gestures Gestures have shown to have cognitive advantages in learning and recall over keyboard shortcuts [4]. In KeySlide, a swipe gesture is performed in the direction where the user intends to position the cursor. By changing the direction of gesture, the user can give continous input of direction to the editor. KeySlide requires little or no memorization as users are only required to memorize a single swipe gesture. Users who are accustomed to using trackpad or tablet devices will be easily be able to transfer their learning of swipe gesture to KeySlide.

Tactile feedback Performing gesture on keyboard provides tactile stimuli through the contact of keycaps and its edges with the fingers. In the *short* mode of KeySlide (which will be discusses in next section), the cursor is moved by one step whenever two consecutive keycaps are pressed during the gesture. This provides a correlation between the cursor movement and tactile feedback which the user experiences while performing the gesture. Studies have shown [32, 19, 21] that combined stimuli improves learning and memorization which will make KeySlide easier to memorize and perform.

KeySlide Description

The technique is guided by the gesture the user performs on the keyboard. When the user wants to position the cursor to a different position, he slides his fingers across the keyboard in the direction which he wants the cursor to position. To perform the gesture, the user first needs to switch the mode of text editor from *typing* mode to *positioning* mode where he can perform the gesture. To switch to the *positioning* mode, the user needs to press a key on the keyboard (CAPS LOCK, in our case). Switching to *positioning* mode stops the keyboard to perform its default function of typing and is used to record the keys pressed when the user performs the gesture.

When the user performs a gesture, the keys pressed during the gesture are recorded along with the time at which that key was pressed. The direction of the gesture is estimated by the coordinates of keys pressed. Since the keyboard keys are spaciously placed, the resolution of spatial data obtained from the keyboard is low. This limits the accuracy of detecting the direction in which the user intends to position the cursor. Hence, to accommodate for the low precision, we have developed two modes for the KeySlide technique.

Short Mode

Short mode allows for the positioning of the cursor in axial (vertical and horizontal) directions in unit steps. This mode helps the user to have precise control over the cursor position. As the user performs the gesture on the keyboard, pressing of two consecutive keys on the keyboard positions the cursor by unit distance in the editor. Depending on the direction of gesture, the cursor positions horizontally or vertically. As the user slides his fingers across the keyboard (Figure 1a), the cursor will follow the trajectory of the gesture.

Due to the limited number of keys available on the keyboard, the cursor can only be positioned for limited distance using a single gesture. The maximum horizontal distance the cursor can be positioned in the horizontal direction is 12 columns since a maximum of 13 keys are present consecutively in the horizontal direction. Also, the cursor can only be positioned vertically by 3 rows since the keyboard layout has 4 rows available for the gesture.

Long Mode

In this mode, the cursor can be positioned by variable distance by performing the gesture in the intended direction. As the user performs the gesture, the cursor continuously moves in the direction of gesture during and after the gesture is performed (Figure 1b). To stop the cursor at a particular location, the user can press any key on the keyboard or exit from *long* mode by releasing the key which activates the *long* mode. The cursor moves with a constant speed in the direction of the gesture. The speed of cursor movement is determined by the speed of the gesture performed by the user.

Strategy

As mentioned in the last section, *short* mode constraints the distance by which the cursor can be positioned. And due to low accuracy of recognizing direction in *long* mode, a combination of both modes needs to be used in order to position the cursor. When the user performs KeySlide in *long* mode, the cursor positions close to the target where he intends to position the cursor. Then, the user needs to use the *short* mode to precisely reach the target location. We conducted experiments to determine how close the cursor can reach the target location using the *long* mode. The experiment is detailed further in this paper.

IMPLEMENTATION

Formulation

As the user performs a KeySlide gesture, a series of data is obtained which contains the coordinate of key pressed and the time at which they were pressed. The coordinate of keys pressed is denoted as p_i and time at which the key was pressed as t_i. To estimate the slope M at which the gesture is performed, a linear regression is done over the keycap positions p_i.

The speed of the gesture S was modelled as a function of average time difference T between keypress:

$T = (t_{i+1} - t_i)/N$ for $i = 1$ to N

Heuristically, we found that using S as an exponential function of T gave better control on the speed of the cursor. Hence, we user $S(T) = a * T^b$ to calculate speed of the cursor and set the parameters a and b through pilot user tests.

The initial position of cursor is C_0 (a two dimensional vector) and the time at which the gesture starts is t_1. The position of

(a) Logitech K100 "full-size" keyboard

(b) Apple's Magic MLA22HN/A "compact" keyboard

Figure 5: Two types of keyboards for performing gestures

cursor C as a function of time t can be given by the following equations:

$$C(t) = C_0 + M * S(T) * (t - t_1)$$

The cursor coordinates C_t are converted to rows and columns of the text editor to set the position in text editor coordinates.

KeySlide technique is implemented as a package in Atom Text editor. Every keyboard has modifier keys which are used to trigger commands. Developers can use these modifier keys to trigger commands in their applications. KeySlide require the user to use the entire keyboard, hence the modifier key to trigger this interaction should not already be used by the editor by default. In Atom and in other text editors, most modifier keys are occupied by default. Hence, for our purpose, we used CAPS LOCK to trigger the interaction. Utilizing CAPS LOCK key as a modifier key does not hinder the function of CAPS LOCK and also provides a natural position for the user to access the key quickly. Since CAPS LOCK is located near the little finger while typing, we hoped that it will reduce the time the user takes to switch between *typing* mode and *positioning* mode.

The key CAPS LOCK was used to activate the *short* of technique. The key combination of CAPS LOCK + SHIFT was used to access the *long* mode of technique.

STUDY DESIGN

We did two comparison studies of KeySlide with existing keyboard shortcuts for cursor positioning which are provided in most text editors by default. Both keyboard shortcuts and KeySlide are keyboard based cursor positioning techniques and the user will not spend significant homing time in using these techniques. Keyboard shortcuts are already known to perform better than mouse-based navigation [23]. Hence, we use keyboard shortcuts as a benchmark to how well KeySlide performs for cursor positioning.

Parameter	Action (move the curor)
UP	one row in upward direction
DOWN	one row in downward direction
LEFT	one column left
RIGHT	one column right
WORD_RIGHT	end of the word
WORD_LEFT	beginning of the word
LINE_LEFT	beginning of line
LINE_RIGHT	end of line
WINDOW_UP	top of screen
WINDOW_DOWN	bottom of screen

Table 1: Table containing the actions which can be performed using keyboard shortcuts

As the user performs gestures on keyboards, the user experiences friction from the keys as the keys needs to be pressed into the keyboard. This introduces discomfort and hindrance in the speed at which the gesture can be performed. A previous study by Haimo [43] on keyboard gestures shows that user prefers compact keyboards (Figure 5b) over full-sized keyboards (Figure 5a) because of the low travel distance of keycaps on keyboard. Hence, for this study we used compact keyboard. In the rest of this paper, refer to keyboard shortcuts and KeySlide interactions with the term gestures.

SIMULATION STUDY

Since the accuracy of direction in KeySlide technique varies across different directions, we need to obtain data for different angles and distances to establish how well KeySlide works on an average positioning task. It would not be possible to experiment with thousands of tasks on real users. Hence, we ran simulations which would help us understand the behavior of KeySlide for all possible directions and distances.

Methodology

We created our own text editor program using Python which would allow us to run gestures virtually on the text editor. A text document containing 100 columns and 120 rows was created in our editor. These dimensions of document mimic that of a 13 inch MacBook Pro text editor with font size 20. [1]

The experiment was run for 10,000 simulation tasks. In each run of the simulation, the initial position of the cursor was positioned randomly in $100 * 120$ grid. Along with the cursor position, a target position was declared at random where the cursor had to be positioned through the given technique. Each technique was given a set of actions using which the cursor had to be positioned to the target location. Actions for keyboard shortcuts are listed in Table 1.

A greedy algorithm (Algorithm 1) was used to position the cursor using the least number of actions possible. At every step, the action which could position the cursor closest to the target location was performed. The list of executed actions and cursor position after each action was stored for further analysis.

[1]The source code for the simulation, https://github.com/info-design-lab/KeySlide/tree/master/Simulation

Parameter	KeySlide		Keyboard Shortcuts
	Short Mode	Long Mode	
Gestures	2.76 ±4.29	2.17 ±0.73	28.49 ±18.94
Time (s)	0.069 ±0.812	2.667 ±1.751	5.709 ±3.764

Table 2: Summary of results obtained from simulation study.

Algorithm 1 Simulation Algorithm

```
C ← randomly assigned cursor location
T ← randomly assigned target cursor location
while C! = T do
    A ← list of actions for the technique
    a ← action from A which takes C closest to T
    C ← perform action a
```

While simulating the action of KeySlide in *long* mode, we algorithmically detected which keys will be pressed on the keyboard based on the direction of the target location. We fixed the starting keys for different gestures at the corners of the keyboard to simulate the longest gesture possible. Based on the direction of the target location, the keys which will be pressed were determined and the slope for the cursor positioning was calculated from the coordinates of keys pressed (Algorithm 2).

Algorithm 2 Simulating KeySlide Gesture

```
C_0 ← initial cursor location
T ← target cursor location
if distance(C,T) <= 5 then                    ▷ short mode
    A ← [UP, DOWN, LEFT, RIGHT]
    a ← action from A which takes C closest to T
    C_1 ← perform action a on C_0
else                                          ▷ long mode
    M_0 ← slope of line from C_0 to T
    K ← list of keycaps pressed on keyboard when
  gesture of slope M is performed
    M_1 ← slope estimated using positions of K
    C_1 ← projection of T on line starting from
  C_0 with slope M_1
```

In the case of KeySlide, the gesture in the *long* mode was performed whenever the distance of the target location from cursor location is greater than 5 to mimic the strategy of KeySlide. When the distance of the target location is less than or equal to 5, the *short* mode was used to position the cursor.

To calculate the time required for task execution, we multiplied the number of gestures with the average time the user takes to perform a keypress. Based on existing research [9], the average time for keypress of an average skilled typist is 0.2 seconds. Since the time taken by KeySlide in *long* mode is independent of gestures, the time *long* mode was calculated based on the distance of target location from the cursor location.

Results

We based our analysis on time and number of gestures required to reach the target location. Table 2 summarizes the results

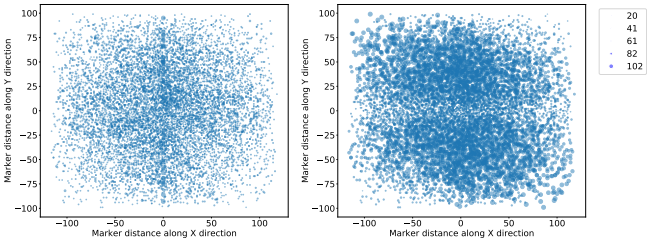


Figure 6: Average number of gestures executed to complete a task by KeySlide (left) and keyboard shortcuts (right). The horizontal and vertical axis represent the distance of target location from the initial cursor location.

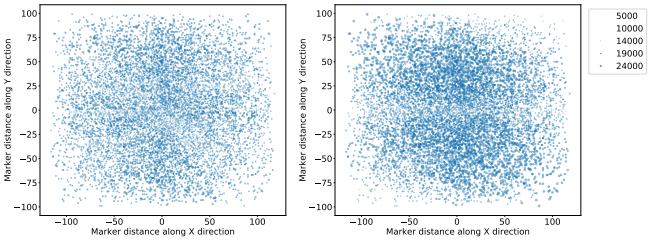


Figure 7: Average time to complete a task by KeySlide (left) and keyboard shortcuts (right). The horizontal and vertical axis represent the distance of target location from the initial cursor location. The radius of each circle is proportional to the time in milliseconds.

obtained from the experiment. The important results are that keyboard shortcut requires ($mean = 5.709, SD = 3.764$) and KeySlide requires ($mean = 2.736, SD = 1.833$) seconds to complete a task (Table 2). The total time to complete all 10,000 tasks was $27,358ms$ and $57,076ms$ by KeySlide and keyboard shortcuts respectively.

To understand how many gestures the user needs to perform to complete the task, we measured the number of gestures in both *long* and *short* mode. Results show that KeySlide requires ($mean = 2.16, SD = 0.74$) and ($mean = 2.75, SD = 4.27$) gestures in *long* and *short* modes. On the other hand, keyboard shortcuts require ($mean = 28.49, SD = 18.94$) gestures to complete the task.

Finally, we wanted to understand how close the cursor can reach the target location using KeySlide, we measured the offset distance of the cursor from target location after the KeySlide in *long* mode is performed. Results show that after the cursor is ($mean = 24.9, SD = 20.2$) and ($mean = 6.56, SD = 11.40$) units (in editor coordinates) away from the target location, after the first and second gestures.

Discussion

The simulation study shows that KeySlide is 53% faster than keyboard shortcuts. The number of gestures required for keyboard shortcuts is around 7 times more than the gestures required for KeySlide. Although the two cannot be compared directly because KeySlide involves gestural input and key-

board shortcuts require key presses, it gives an idea about the cognitive effort which is required to use the two techniques. Keyboard shortcuts require a higher number of gestures and planning which leads to higher cognitive effort compared to KeySlide.

The aspect ratio of keyboard is very high which makes the gesture distance in vertical direction lower than in horizontal direction. This leads to high errors in the detection of gesture direction. Also, as the user performs a vertical gesture, keys are aligned in zigzag order which makes the detection of direction harder. This would explain the high number of gestures required in the vertical direction (Figure 6).

Consistent with our expectations, the time taken to reach the target in KeySlide increases with distance. We also observed (Figure 7) that as the angle gets close to the vertical, the time to reach the target increases. This is attributed to higher number of gestures in the vertical direction (Figure 6). We observed that for angles closer than 30° to the vertical, the time increases drastically (Figure 7).

Both KeySlide and keyboard shortcuts require less time and gestures when the target location is close to the horizontal. We believe KeySlide has higher accuracy in the horizontal direction because the keycaps on the keyboard are consecutively aligned horizontally which increases the accuracy of slope detection in the horizontal. The keyboard shortcut can make use of WORD_RIGHT and WORD_LEFT actions to jump through the words in the horizontal direction which makes positioning in the horizontal direction quicker than vertical.

Contrary to our expectations, KeySlide in *long* mode requires (mean=2.17, SD=0.73) gestures which shows that the overall accuracy of direction is low. Due to the low accuracy of direction, the offset distance between the target and cursor position increases as the distance between them increases. We believe this would have a negative effect on the user experience as the user performs KeySlide because the cursor does not move accurately in the intended direction.

Noticeably, this study ignores the homing time and the time the user takes to mentally prepare themselves to execute the task. Simulation study also ignores any errors which the user might make in real text editing situations. For example, the user might execute the wrong gesture and correct it later while executing the task. We will address these issues in the next study.

USER STUDY

The goal of the second experiment was to determine whether similar results from the simulation could be achieved in real use case scenario. In case of simulation, homing time and time for mentally preparing to execute the task was ignored. We believe both these times will be different for both the techniques and will influence the overall efficiency of using these technique.

The study followed a single experimental design with one independent variable, positioning technique (KeySlide, keyboard shortcuts) The dependent variable was the task efficiency. By efficiency, we mean the time users require to complete the text editing task. Each participant was given a document which required modifications such as spelling corrections, text addition and deletions. A marker was displayed on the word where the text editing was required to be made. The positioning technique was randomly assigned to the participants.

Parameter	KeySlide		Keyboard Shortcuts
	Short Mode	Long Mode	
Gestures	523.1 ±226.9	95.2 ±36.6	991.2 ±197.8

Table 3: Number of gestures performed during the experiment

We created a package in Atom text editor to implement our interaction technique [2]. The experiment was done on Atom Text editor on a 27 inch iMac running MacOS Mojave. Apple Magic keyboard was used to perform gestures for both techniques. The font size of the editor was 20 pixels and the background of the text was coloured white. No visual modifications was made to the cursor.

A total of 18 individuals were recruited for the study (mean age = 23.18, SD=2.22, range= 21-28) who are experienced text editor users (mean experience in years = 10.75, SD=5.31, range= 4-19) and had no prior experience in using both keyboard shortcuts which were used during the experiment and KeySlide. The study was conducted in the premises of IIT Bombay.

Methodology

The experiment was conducted in a single sessions which lasted for 40 minutes. Each participant was randomly assigned one of the two techniques. The experiment was divided into two parts. In the first part, the participants were informed about the objective of the study and different stages of experiment. Next, the technique was explained to the participants and they were asked to practice 50 cursor positioning tasks in which the cursor had to be positioned at random locations on the screen. Visible markers were placed on the editor where the cursor had to be positioned. All the participants were given same positioning location and order. If required, the users were given extra time to practice the technique. The first part of the experiment lasted for 20 minutes.

In the second part of the experiment, the users were given a text document which required editing tasks such as text insertion or deletions. A total of 30 editing tasks were given to the users. A marker was placed on the word which required editing. An example of text editing task would be to "replace the word 'apple' by 'mango'". All the participants were given same editing task in the same sequence and word locations. The position of the words that require editing were randomly dispersed throughout the screen.

Results

We based our results on two parameters: time and number of gestures. The total time to complete all the task for keyboard

[2]The source code for the package, https://github.com/info-design-lab/KeySlide/tree/master/KeySlide%20(Atom%20Package)

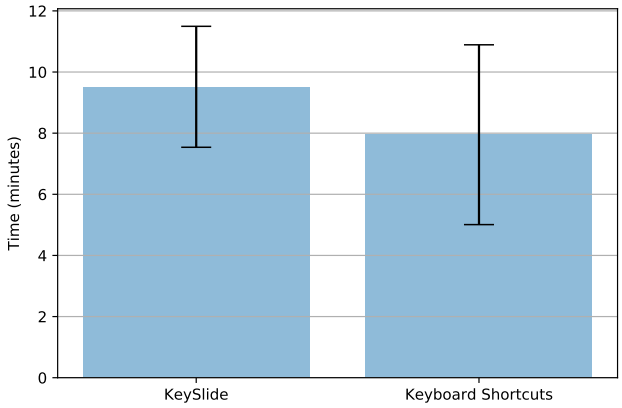


Figure 8: Time taken to complete all the tasks in the session by both KeySlide and keyboard shortcuts. The errors bar represent standard deviation.

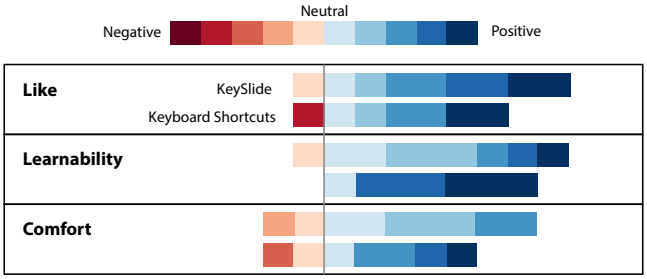


Figure 9: Responses to the questionnaire submitted by the participants.

shortcuts was (mean in minutes = 7.949, SD = 2.940, n=9) and for KeySlide was (mean in minutes = 9.516, SD = 1.978, n=9) (Figure 8). Total number of gestures performed to complete all the task in case of keyboard shortcuts was (mean = 1090.4, SD=179.4, n=9) and in case of KeySlide was (mean = 523.1, SD=226.9, n=9) and (mean = 95.2, SD=36.6, n=9) for *short* and long modes respectively (Table 3).

Questionnaire. We asked all the participants to rate different aspects of the two techniques. The rating scale varied from 1 (very negative) to 10 (very positive). Their results are shown in Figure 9. In addition to Figure 9, additional questions were asked to the participants based on the technique that was assigned to them. Participants using keyboard shortcuts were able to memorize almost all the shortcuts (mean=9.14, SD=1.21) out of 10. Surprisingly, participants using KeySlide rated the predictability of cursor movement as (mean=7.22, SD=1.30) on a scale of 1 (very unpredictable) to 10 (very predictable).

Discussion

The results of the study show that keyboard shortcuts are 19% faster than KeySlide. During the experiment, we found that KeySlide participants had difficulty in adapting the directional gestures on the keyboard. Few participants performed very short gestures stating that small gestures "require less effort" and they "perform small gesture on mobile". We believe this this due to the users legacy bias [30] of using trackpad and touch screens which require less effort to perform gestures.

Across all participants we observed that they used the following strategy for cursor positioning. First, move the cursor in vertical direction such that it is aligned in the line of target word. Second, move the cursor horizontally to reach the target word. In case of KeySlide, few participants reported that they were using this strategy because the accuracy of the cursor was poor in vertical direction. Hence, they first positioned the cursor vertically to later move horizontally which they found to be more accurate.

Participants using keyboard shortcuts reported that they wanted to have more keyboard shortcuts to move multiple lines vertically, similar to using WORD_RIGHT and WORD_LEFT in horizontal directions.

KeySlide

Across all participants using KeySlide, we observed that the cursor was overshooting or undershooting the target position because the speed of the cursor. We believe that this problem was due to the speed of the cursor. Although, the users were aware that the speed of the cursor was determined by the speed of the gesture, none of the users actively controlled the speed of their gestures to increase/decrease the speed of cursor.

Most participants reported that vertical gestures were inaccurate and restrictive. Due to limited number of keys which can be pressed in vertical direction, the users did not have freedom to perform long gestures as they can in horizontal direction. We also found that the anatomy of fingers play a role in performing vertical gestures. As the user performs gesture in vertically upward direction, the nails of the fingers can get stuck at the keycaps if the angle between the keyboard surface and fingers is steep.

Some participants preferred performing gestures for short distances. Few of them reported that they were performing short gestures because they are used to performing short gestures in mobile and tablet devices. Due to large area of the keycaps compare to the area of the fingers, the keyboard is not equipped to detect short gestures. This problem was particularly prominent in short mode when the users repeatedly performed the short gestures multiple times. For example, when the user performed gesture $G \rightarrow H$ to move the cursor right by one column. Performing the same gesture $G \rightarrow H$ again prevents the leads unexpected behaviour of the cursor because the program registers the keypresses as $G \rightarrow H \rightarrow G \rightarrow H$, where the sequence $H \rightarrow G$ moves the left by one column. Hence, performing short gestures led to unintended behaviour of the cursor.

Summary of observations:

- Users wanted to have the ability to adjust the speed of the cursor similar to mouse.

- Some users performed very small gestures on the keyboard which resulted in poor estimation of gesture direction. Users

stated that they are used to performing small gestures on mobile and tablet devices.

- Some users reported that this technique requires more effort to perform the gestures as they need to physically move their hand across the keyboard.
- During the experiment, we observed that users forget to switch between the key modes while performing the gesture.
- Few participants were concerned that their keyboard might wear out or get spoilt by performing the KeySlide gesture repeatedly.
- During training phase, one of the user assumed that the entire screen was mapped to the keyboard and expected the cursor to move like a trackpad.
- We observed that most users were unaccustomed to a the large 27 inch screen used in the test, especially for text editing tasks. Given such a large display they took some time to locate the cursor visually on the screen before using the gesture or shortcut to acquire it. Although the same display was used for both conditions, we feel the unfamiliar size introduced some unwanted hindrance in the task.
- The wide range of reactions from participants indicate that the protocol has to be refined further and the practice sessions lengthened.

CONCLUSION

In this paper, we presented KeySlide, a novel interaction technique to position the cursor by using directional gestures on the keyboard. The technique does not hinder the typing experience and allows for seamless switching between typing and cursor positioning. We demonstrated how speed and direction of gestures can be estimated from keypress data. We conducted two comparison studies between KeySlide and keyboard shortcuts which are generally implemented in most text editors. In our first study (simulation), we found that KeySlide is on an average 53% faster than keyboard shortcuts under ideal conditions and requires less number of gestures to be performed than keyboard shortcuts. In our second study (user study), we found that keyboard shortcuts are 19% faster than KeySlide. We finally discussed key insights from the experiment design and some practical aspects of using KeySlide.

In future work, we want to conduct longer user tests consisting of multiple sessions to understand different learning rate and how people can transfer gestural interaction from other devices to keyboards. We also want to investigate the effect of tactile feedback on keyboard gestures. We believe, through prolonged practice of KeySlide, the tactile feedback from the keypress can aid in the overall efficiency and accuracy of the technique.

ACKNOWLEDGEMENTS

We thank Anirudha Joshi and Girish Dalvi for their valuable inputs to the experimental design of the evaluation of KeySlide. Bruno Fruchard and Amit Jena for their feedback on the paper. We are greatly indebted to the participants of the user study.

REFERENCES

[1] Jessalyn Alvina, Carla F. Griggio, Xiaojun Bi, and Wendy E. Mackay. 2017. CommandBoard: Creating a General-Purpose Command Gesture Input Space for Soft Keyboard. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 17–28. DOI: http://dx.doi.org/10.1145/3126594.3126639

[2] Toshiyuki Ando, Toshiya Isomoto, Buntarou Shizuki, and Shin Takahashi. 2018. Press & Tilt: One-handed Text Selection and Command Execution on Smartphone. In *Proceedings of the 30th Australian Conference on Computer-Human Interaction (OzCHI '18)*. ACM, New York, NY, USA, 401–405. DOI: http://dx.doi.org/10.1145/3292147.3292178

[3] Brad A. Myers Andrew J. Ko. 2012. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *CHI '06 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, 387–396.

[4] Caroline Appert and Shumin Zhai. 2009. Using Strokes As Command Shortcuts: Cognitive Benefits and Toolkit Support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 2289–2298. DOI: http://dx.doi.org/10.1145/1518701.1519052

[5] 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 Graphics Interface 2014 (GI '14)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 43–50. http://dl.acm.org/citation.cfm?id=2619648.2619657

[6] Grace Lu Ravi Chugh Brian Hempel, Justin Lubin. 2018. Deuce: a lightweight user interface for structured editing. In *ICSE '18 Proceedings of the 40th International Conference on Software Engineering*. Society for Computer Simulation International, 654–664.

[7] Olivier Chapuis Bruno Fruchard, Eric Lecolinet. 2017. MarkPad: Augmenting Touchpads for Command Selection. In *CHI '17 Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM Press, 5630–5642. https://doi.org/10.1145/3025453.3025486

[8] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. 2014. Supporting Novice to Expert Transitions in User Interfaces. *ACM Comput. Surv.* 47, 2, Article 31 (Nov. 2014), 36 pages. DOI: http://dx.doi.org/10.1145/2659796

[9] D.B. Devoe. 1967. Alternatives to Handprinting in the Manual Entry of Data. In *IEEE Transactions on Human Factors in Electronics*. IEEE, 21 – 32.

[10] Sylvain Malacria Fanny Chevalier Emmanouil Giannisakis, Gilles Bailly. 2017. IconHK: Using Toolbar button Icons to Communicate Keyboard Shortcuts. In *CHI '17 Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM Press, 4715–4726.

[11] 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 the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2679–2682. DOI:http://dx.doi.org/10.1145/2207676.2208660

[12] Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. Gestures and Widgets: Performance in Text Editing on Multi-touch Capable Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2785–2794. DOI: http://dx.doi.org/10.1145/2470654.2481385

[13] William Buxton Gordon Kurtenbach. 1993. The limits of expert performance using hierarchic marking menus. In *CHI '93 Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. ACM Press, 482–487. https://doi.org/10.1145/169059.169426

[14] Horatiu-Stefan Grif and Cornel Cristian Farcas. 2016. Mouse Cursor Control System Based on Hand Gesture. *Procedia Technology* 22 (2016), 657 – 661. DOI: http://dx.doi.org/https://doi.org/10.1016/j.protcy.2016.01.137 9th International Conference Interdisciplinarity in Engineering, INTER-ENG 2015, 8-9 October 2015, Tirgu Mures, Romania.

[15] Kumiko Tanaka-Ishii I. Scott MacKenzie. 2007. *Text Entry Systems: Mobility, Accessibility, Universality*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[16] Poika Isokoski. 2004. Performance of Menu-augmented Soft Keyboards. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 423–430. DOI: http://dx.doi.org/10.1145/985692.985746

[17] Robert J. K. Jacob. 1991. The use of eye movements in human-computer interaction techniques: what you look at is what you get. In *ACM Transactions on Information Systems (TOIS)*. ACM Press, 152–169. https://doi.org/10.1145/123078.128728

[18] Carl Gutwin Andrea Bunt Joey Scarr, Andy Cockburn. 2012. Improving command selection with CommandMaps. In *CHI '12 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, 257–266.

[19] William P. Jones and Susan T. Dumais. 1986. The Spatial Metaphor for User Interfaces: Experimental Tests of Reference by Location Versus Name. *ACM Trans. Inf. Syst.* 4, 1 (Jan. 1986), 42–63. DOI: http://dx.doi.org/10.1145/5401.5405

[20] Sunjun Kim and Geehyuk Lee. 2016. TapBoard 2: Simple and Effective Touchpad-like Interaction on a Multi-Touch Surface Keyboard. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 5163–5168. DOI: http://dx.doi.org/10.1145/2858036.2858452

[21] Yuki Kubo, Buntarou Shizuki, and Jiro Tanaka. 2016. B2B-Swipe: Swipe Gesture for Rectangular Smartwatches from a Bezel to a Bezel. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3852–3856. DOI: http://dx.doi.org/10.1145/2858036.2858216

[22] Gordon Kurtenbach and William Buxton. 1993. The Limits of Expert Performance Using Hierarchic Marking Menus. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 482–487. DOI: http://dx.doi.org/10.1145/169059.169426

[23] David M Lane, H Albert Napier, S Camille Peres, and Aniko Sandor. 2005. Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts. *International Journal of Human-Computer Interaction* 18, 2 (2005), 133–144.

[24] Andrew Sears Jeremy Lozier Liwei Dai, Rich Goldman. 2004. Speech-based cursor control: a study of grid-based solutions. In *Assets '04 Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*. ACM Press, 94 – 101. https://doi.org/10.1145/1029014.1028648

[25] Terry Winograd Manu Kumar, Andreas Paepcke. 2007. EyePoint: practical pointing and selection using gaze and keyboard. In *CHI '07 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, 421–430.

[26] Giovanni Spagnoli Marco Porta, Alice Ravarelli. 2010. ceCursor, a contextual eye cursor for general pointing in windows environments. In *ETRA '10 Proceedings of the 2010 Symposium on Eye-Tracking Research Applications*. ACM Press, 331–337. https://doi.org/10.1145/1743666.1743741

[27] Garth Shoemaker Michael Ashmore, Andrew T. Duchowski. 2005. Efficient eye pointing with a fisheye lens. In *GI '05 Proceedings of Graphics Interface 2005*. Canadian Human-Computer Communications Society School of Computer Science, 203–210.

[28] Amjad Altadmri Michael KÃűlling, Neil C. C. Brown. 2015. Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. In *WiPSCE '15 Proceedings of the Workshop in Primary and Secondary Computing Education*. ACM Press, 29–38.

[29] PÃd'ivi Majaranta Oleg Åăpakov. 2012. Enhanced gaze interaction using simple head gestures. In *UbiComp '12 Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM Press, 705–710. https://doi.org/10.1145/2370216.2370369

[30] Jaime Ruiz and Daniel Vogel. 2015. Soft-Constraints to Reduce Legacy and Performance Bias to Elicit Whole-body Gestures with Low Arm Fatigue. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3347–3350. DOI: http://dx.doi.org/10.1145/2702123.2702583

[31] Hans Vangheluwe Sagar Sen, Benoit Baudry. 2010. Towards Domain-specific Model Editors with Automatic Model Completion. In *Simulation*. Society for Computer Simulation International, 109–126.

[32] Joey Scarr, Andy Cockburn, and Carl Gutwin. 2013. Supporting and Exploiting Spatial Memory in User Interfaces. *Found. Trends Hum.-Comput. Interact.* 6, 1 (Dec. 2013), 1–84. DOI: http://dx.doi.org/10.1561/1100000046

[33] Yilei Shi, Haimo Zhang, Hasitha Rajapakse, Nuwan Tharaka Perera, Tomás Vega Gálvez, and Suranga Nanayakkara. 2018. GestAKey: Touch Interaction on Individual Keycaps. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 596, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3174170

[34] Steven Ihde Shumin Zhai, Carlos Morimoto. 1999. Manual and gaze input cascaded (MAGIC) pointing. In *CHI '99 Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM Press, 246–253.

[35] Allen Newell Stuart K. Card, Thomas P. Moran. 1980. The keystroke-level model for user performance time with interactive systems. In *Communications of the ACM*. ACM Press, 396–410.

[36] Kenji Suzuki, Kazumasa Okabe, Ryuuki Sakamoto, and Daisuke Sakamoto. 2016. Fix and Slide: Caret Navigation with Movable Background. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*. ACM, New York, NY, USA, 478–482. DOI: http://dx.doi.org/10.1145/2935334.2935357

[37] John Paulin Hansen Vijay Rajanna. 2018. Gaze typing in virtual reality: impact of keyboard design, selection method, and motion. In *ETRA '18 Proceedings of the 2018 ACM Symposium on Eye Tracking Research Applications*. ACM Press. https://doi.org/10.1145/3204493.3204541

[38] Wikipedia. 2019a. Abstract syntax tree. (2019). https://en.wikipedia.org/wiki/Abstract_syntax_tree.

[39] Wikipedia. 2019b. Cursor (User Interface). (2019). https://en.wikipedia.org/wiki/Cursor_(user_interface).

[40] Wikipedia. 2019c. Pointing Stick. (2019). https://en.wikipedia.org/wiki/Pointing_stick.

[41] Wikipedia. 2019d. Vim (text editor). (2019). https://en.wikipedia.org/wiki/Vim_(text_editor).

[42] Shin Takahashi Yoshiyuki Mihara, Etsuya Shibayama. 2005. The migratory cursor: accurate speech-based cursor movement by moving multiple ghost cursors using non-verbal vocalizations. In *Assets '05 Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*. ACM Press, 76–83. https://doi.org/10.1145/1090785.1090801

[43] Haimo Zhang and Yang Li. 2014. GestKeyboard: Enabling Gesture-based Interaction on Ordinary Physical Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1675–1684. DOI:http://dx.doi.org/10.1145/2556288.2557362