GestAKey: Touch Interaction on Individual Keycaps

Yilei Shi, Haimo Zhang, Hasitha Rajapakse, Nuwan Tharaka Perera, Tomás Vega Gálvez, Suranga Nanayakkara
Augmented Human Lab, Singapore University of Technology and Design
{yilei, haimo, hasitha, tharaka, tomas, suranga}@ahlab.org

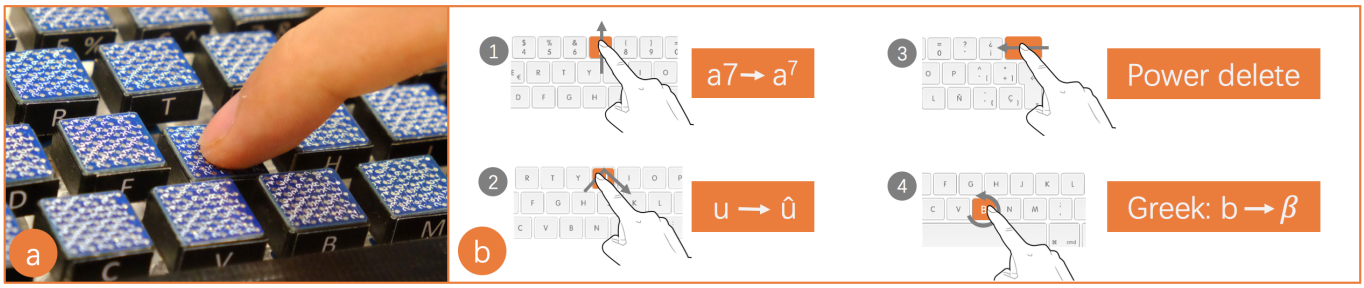

Figure 1. GestAKey, enabling novel keyboard interactions by detecting micro-gestures on individual keycaps. (a) Custom-made capacitive sensing keycaps mounted on a regular keyboard. (b) Microgestures on keycaps open up new possibilities: b1) Input superscript letters with swipe up; b2) Input accented letters by drawing the accent, b3) Delete contents on the left or right with swipe left/right, b4) Input Greek letters by circling.

ABSTRACT

Conventionally, keys on a physical keyboard have only two states: "released" and "pressed". As such, various techniques, such as hotkeys, were designed to enhance keyboard expressiveness. Realizing that users inevitably perform touch actions during keystrokes, we propose GestAKey, leveraging the location and motion of touch on individual keycaps to augment the functionalities of existing keystrokes. With a log study, we collected touch data for both normal usage (typing and hotkeys) and while performing touch gestures (location and motion), which were analyzed to assess the viability of augmenting keystrokes with simultaneous gestures. A controlled experiment was conducted to compare GestAKey with existing keyboard interaction techniques, in terms of efficiency and learnability. The results show that GestAKey has comparable performance with hotkeys. We further discuss some insights on integrating such touch modality into existing keyboard interaction, and demonstrate several usage scenarios.

Author Keywords

Keyboard Interaction; Touch Interaction; Multifunctional Keystroke.

ACM Classification Keywords

H.5.2 User Interface: Input devices and strategies

CHI 2018, April 21–26, 2018, Montréal, QC, Canada.

http://dx.doi.org/10.1145/3173574.3174170

INTRODUCTION

Although the keyboard remains the dominant input device for text entry on computers, various applications require other types of input, such as issuing commands, calling for more expressive interactions on a keyboard. However, the keyboard is restricted by its limited number of keys. To enhance the expressiveness of a keyboard, users are often required to learn different interaction techniques (e.g., hotkeys) or use other input modalities (e.g., pressure or touch).

A widely used technique to expand the keyboard input space is "chording" (i.e., pressing several keys simultaneously), such as hotkeys. Although found to be more efficient than using a mouse [21], hotkeys are underutilized by most people [27, 21] due to the difficulty of learning and [15] executing [24] them. Another approach to enrich the expressiveness of a keyboard is to introduce new input modalities. Most research in this direction excludes normal interaction with the keyboard while interacting with new modalities [28, 29, 18], treating the keyboard as a mere platform to host the sensors and actuators of added modalities. Only a few studies synthesize the new modalities with existing keyboard interactions [9, 38], such that the output is decided by the combination of both interaction modalities. For example, in [38], both the key pressed and the user's hand posture decide the command. However, these techniques require users to push a key from different sides [9], or press a key in uncommon ways [38], interrupting their normal interactions with a keyboard.

The aim of this research is to enhance the expressiveness of keyboard interaction, in a way that is easy to learn and perform, and is integrated into the existing modality of a keyboard. Realizing that users inevitably perform touch actions during keystrokes, we created GestAKey, a technique to enhance expressiveness of a keystroke with touch information

on individual keycaps. For example, pressing "S" while swiping up can take a screenshot and save it to the desktop. Pressing "S" while swiping down can take a screenshot and save it to the clipboard. Pressing "B" while drawing a circle could input a special character β". GestAKey is a new keyboard interaction method with the following features: (1) Allows users to maintain the hand posture they currently use for typing to minimize the interference with existing interaction; (2) Creates new interaction mnemonics that help users remember the mapping between the functions and their corresponding operations. (3) Combines the new modality with existing keyboard interactions (i.e., keystrokes).

This paper contributes in the following three aspects:

- A novel interaction technique that leverages touch information while performing keystrokes, and the hardware prototype of a touch-sensitive keycap for such interaction.
- A log study that explored the possible touch gestures of different fingers within the area of individual keycaps. A controlled experiment that compared the efficiency and learnability of GestAKey with hotkeys and long pressing.
- Demonstrations of several usage scenarios of the GestAKey technology.

RELATED WORK

Our work related to research in two areas: gesture interaction on soft keyboards and augmented physical keyboards.

Gesture Interactions on Soft Keyboards

Soft keyboards have been widely explored for their potential in gestural interaction, beyond the mere emulation of their physical counterparts. In [16], a marking-menu was used to augment a stylus-based soft keyboard. After a user lands the stylus on a key, the character is entered and a marking-menu would pop up. Users could then directly input another letter by moving the stylus on to the selection on the marking-menu. Instead of separating taps to input single letters, shape-writing has been leveraged to input words on a soft keyboard [35, 19, 36]. To input a whole word with shape-writing, a user performs a continuous gesture that passes through all the characters used to spell that word, which turned out to be fast and easy to learn for most common words. Based on the gesture-typing technique, Alvina et al.[5] designed a soft keyboard that mapped input gesture features to a continuous output parameter space, such as the RGB color of the text. Other research has used gestures on soft keyboards to input special symbols [14], replace functionalities of special keys (i.e., Space, Backspace, Shift and Enter) [22, 7], input alternative characters [3, 2], or even issue specific commands on mobile devices [4].

Although some research on soft keyboards shared the feature of typing and gesturing on a key with our work, most of them focused on text entry [16, 35, 14, 7, 2], and only a few explored the potential of issuing commands [4]. Our work differentiates from these previous works on soft keyboards in two ways: (1) We explore gestural interaction within individual physical keycaps, whose edges provide distinct haptic feedback to support eyes-free gestural interaction; (2) The concurrent execution of gesture and actual keystrokes is explored in our study, which is not possible on a touch-only interaction paradigm. As powerful shape mnemonics [25, 6, 34], gestures can augment existing functions of keystrokes. They showed desirable learnability and intuitiveness in our evaluation.

Augmented Physical Keyboards

Various methods have been applied to augment the physical keyboard. One strategy is to leverage the existing input modality of the keyboard without introducing any other modifications. For example, GestKeyboard [37] recognized motion gestures on the whole keyboard according to the sequence of pressing multiple adjacent keys; Wobbrock et al. [32] used 4 keys to mimic the writing of Roman letters for high learnability; and Jannotti [17] presented a technique to mimic the shape of characters on a numeric keypad. Sequential key pressing has also been used to issue commands [8]. Most of these works have limited gesture resolution [37] or inferior input throughput [32, 17].

An alternative strategy is to integrate sensors and actuators into a physical keyboard to support other interaction modalities. SmartPad [26], equipped with an array of electrode sensors, was able to detect finger position hovering above the keypad. A vision-based method [31] to detect pinch gestures above a keyboard also exists. Taylor et al. [28] equipped the keyboard with a low-resolution infrared proximity sensor matrix, enabling the detection of 3D hand gesture above the keyboard. Fallot-Burghardt [12] combined the conventional keyboard with an extended touchpad, making the whole key area touch sensitive. Tung et al. [29] realized a similar function by covering the keyboard with a capacitive sensing grid and allowed users to switch between typing and pointing modes with a foot pedal. On the new MacBook Pro, TouchBar [1] is a strip-shaped area on a physical keyboard where users can perform touch interactions. It enables users to perform simple tasks such as changing the volume or brightness. Although the interactions listed above introduced new input modalities on a keyboard, they did not synthesize these new interactions with the existing interaction of a keyboard.

In contrast, Métamorphe [9] used actuated keys to augment the hotkey feature. Users can laterally push the key in different directions to trigger various commands. Finger-aware shortcut [38] realized multifunctional keystrokes by using hand posture detection. These two works augmented the hotkey feature, but users still had to learn and remember arbitrary mappings between actions and system responses.

Other modalities have also been used to augment physical keyboards. Pressure sensing keyboard [11] augmented the function of keystrokes with force sensors. Touch-display keyboard [10] realized multifunctional keystrokes by projecting icons representing different functions onto individual keycaps, which gave users a visual guidance and opened novel input design possibilities on every single key.

Inspired by these previous works, GestAKey explores the option of augmenting individual keycaps with an even higher resolution modality of input - 2D touch gestures.

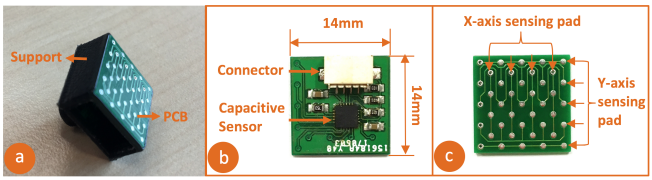

Figure 2. Structure of a single GestAKey keycap: (a) Each keycap consists of two parts: 3D-printed support and PCB. (b) Bottom layer: capacitive sensing IC and connector; (c) Top layer: sensing pad matrix.

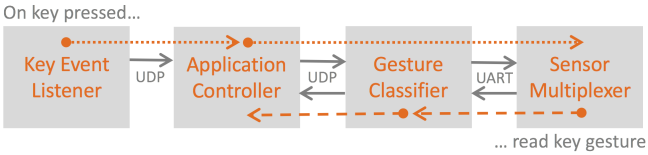

Figure 3. Data flow in the software system. Gray arrows between the components indicate the actual communication protocols used for data transmission. Red lines indicate two directions of data flow: keystroke events (orange dotted line), and gesture data (orange dashed line).

GESTAKEY

GestAKey Concept

A key on a physical keyboard is a normally-open switch having only two states: ON and OFF. As such, with single key presses, the input space is limited by the number of keys on a keyboard. Based on the fact that a user's fingers inevitably touch the keycaps as they perform keystrokes, the key concept of GestAKey is to leverage the keycap surface as a touchpad to enable touch interactions during a keystroke. Our work integrates keystroke with touch gesture in the two following ways. First, by integrating touch gestures on individual keycaps, we integrate touch interaction together with keystrokes. Second, we leverage the press and release events of a key as a way to delimit an intended touch gesture, which eliminates the false positives of gesture detection. With GestAKey, users are allowed to perform diverse touch gestures between the pressing and releasing of a key. By integrating touch gesture information and key events, the functions of a keystroke could be augmented in various ways. Our goal is to provide a new interaction technique that can benefit users in the following aspects:

- **Subtle Interaction**: The area of a keycap is used as a touchpad where users can perform subtle touch gestures without changing their natural hand posture. Such subtle movements of finger are easy to perform while typing.
- **Iconic Gesture**: Compared with keyboard shortcuts, spatial and iconic gestures have cognitive advantages and help users to recall commands [6].
- **Large Input Space**: Hotkeys are restricted by the combination of modifier keys that can be used. In contrast, a touchpad usually provides more diverse input based on the gestures users perform.

GestAKey consists of two subsystems: 1) The hardware that reads touch data from keycaps, and 2) The software that listens for key events, recognizes gestures, and triggers the corresponding output.

Hardware Design

The hardware system has three components: 1) 3D-printed keycaps equipped with capacitive sensing matrices (5x4 points) and custom-made PCBs (14x14mm) (Figure 2); 2) Multiplexers connecting to capacitive sensing keycaps; 3) Microcontroller Unit (MCU) that reads data from the capacitive sensing keycaps via the multiplexers and transmits to the computer. The top layer of the PCB has a 5x4 matrix of capacitive sensing pad and the bottom layer has a capacitive sensing IC (MPR121) as well as a connector to the multiplexer. When a key is pressed, the MCU receives the corresponding keycode from the operating system, reads touch data from the capacitive sensing matrix of the respective keycap, and sends it back to the software system. Given that MPR121 can only be configured to 4 different I2C addresses, we used multiplexers to read from 27 different keys.

Software System

The software system consists of four main modules:

- **Key Event Listener**: a C program that listens for OS-level key presses/releases.
- **App Controller**: a Node.js server that orchestrates data flow and triggers functions.
- **Gesture Recognizer**: a Python script that classifies gestures.
- **Sensor Multiplexer**: an Arduino program that reads data from multiple capacitive touch matrices.

In the current implementation, there are two directions of data flow - for keystroke events (orange dotted line) and gesture data (orange dashed line) respectively (Figure 3).

Keystroke Data Flow: The Key Event Listener listens for key press/release events and sends them to the App Controller, which in turn relays them to the Sensor Multiplexer. The MCU then multiplexes into the touch sensing matrix of the corresponding keycap and extracts raw data.

Gesture Data Flow: The Sensor Multiplexer sends touch data from the selected keycap to the Gesture Classifier. Classified gestures are then sent to the App Controller, where application-specific functions are triggered using Apple's JavaScript for Automation framework.

Gesture Design and Recognition

To design the gestures, we observed behaviors during text entry and hotkey activities and made two key observations. First, users would usually strike on a certain sub-area within a keycap, leaving the rest of the keycap rarely touched, specifically on the edges and corners. Second, a finger would rarely slip during a keystroke. Inspired by these observations, we defined static and dynamic gestures that leverage the position and motion of a touch (Figure 4). For static gestures, users need to touch a specific area of the keycap while pressing it. We defined 9 positions for static touch on a keycap: north (N), south (S), east (E), west (W), northwest (NW), northeast (NE), southeast (SE), southwest (SW) and middle (M).

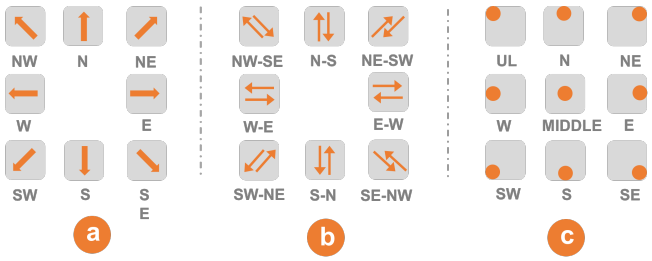

Figure 4. Log study gesture list: (a) 8 single directional dynamic gestures; (b) 8 compound complex dynamic gestures; (c) 9 static gestures.

For dynamic gestures, users needed to perform a directional motion gesture while performing a keystroke. We defined 8 single-directional gestures and 8 reciprocal-directional compound gestures. Single gestures required users to swipe at a specific direction, which includes N, S, E, W, NW, NE, SE and SW. Compound gestures required users to swipe at a specific direction first, then, without lifting the finger, swipe back to the original position.

Our gesture set imposed several challenges for recognition. The first was deciding what touch data represent users' intended gestures. Our approach was to delimit a gesture sample using the key-press and key-release events, between which touch points on that key were extracted as valid gesture data. The reason for this treatment was that in practical applications, the finger might be already touching the keycap before the keystroke, making it difficult to decide the start of actual gesture data. Delimiting gesture data by key events would not only make it easier for gesture segmentation, but also allow the user to perceive the start and end of the gesture through the tactile feedback of the keys on the physical keyboards.

Another challenge in using our gesture set was in classifying gestures based on location, orientation, and scale sensitivity. This controlled whether gestures of the same shape but with different location, orientation, or size should be considered as the same gesture by the classifier. For our static gestures, the gesture class should be scale and location sensitive; otherwise, the classifier would not distinguish between the different locations of touch, and might wrongly accept larger gestures that involve motion. For dynamic gestures, the gesture classes should be orientation sensitive but location and scale invariant; otherwise, the classifier would not be able to distinguish between different directions of the gestures. The implementation of the gesture recognizing framework was based on the $1 gesture recognizer [33]. The original $1 recognizer is orientation and scale invariant by default, and its successor, Protractor [23], is inherently scale-invariant. Thus, these recognizers cannot be used as-is. We modified the $1 preprocessing pipeline to unify the gesture classification procedure for all our gestures (Figure 5).

In the original $1 recognizer, three core transformations, "Rotate", "Scale", and "Center", removes rotation, scale, and location sensitivity after each stage, but provides no mechanism to selectively control sensitivity of each type of transformation. To achieve this mechanism to support our gesture set

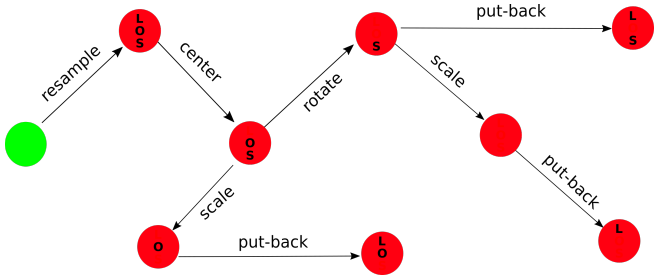

Figure 5. The gesture preprocessing pipeline: an incoming sample (the green node) undergoes different paths, creating preprocessed samples for gesture classification with different transformation sensitivities. Presence/absence of the letters on the red nodes denotes the transformation sensitivity/invariance at the respective preprocessing stage: L - location sensitive, O - orientation sensitive, S - scale sensitive.

with heterogeneous transformation sensitivities, we moved the "Center" stage before the optional and independent "Rotate" and "Scale" stages, and introduced the "Put-back" stage to restore location sensitivity after removing orientation and scale sensitivity. As illustrated in Figure 5, a sample undergoes different paths in the preprocessing pipeline, which are used for similarity measure calculation against gesture classes with the same transformation sensitivities. For example, a preprocessed gesture sample after the "Resample → Center → Scale" stages of the pipeline are used to compare with gesture classes defined to be position and scale invariant, but rotation sensitive (e.g., a top-to-bottom straight line), whose template underwent the same preprocessing when added to the class (see details in [33]). A different version of the same sample after the "Resample → Center → Rotate → Scale → Put-back" stages are used for classes that are rotation and scale invariant but location sensitive (e.g., a static touch on the top-right corner of the keycap). In addition, we employed proportional scaling instead of the non-proportional scaling in the original $1 recognizer, to increase robustness for one-dimensional gestures in our gesture set.

LOG STUDY

A log study was conducted to investigate feasibility of the GestAKey technology in two aspects: 1) whether our prototype system has sufficient spatial and temporal resolution to support different touch actions, and if so, 2) whether the users are able to reliably perform various touch gestures. For the log study, we identified two existing activities on a keyboard, text entry and hotkey. Additionally, we hypothesize that gesturing on individual keycaps could also be a viable interaction technique. Altogether, we conducted the log study in two sessions, the first collecting touch data for representative text entry and hotkey tasks using a keyboard, and the second collecting sample data for the gesture set we defined in the previous section.

Participants and Apparatus

We recruited 12 participants (8 male, 4 female, mean age = 26.3, SD = 2.3) for the log study. All participants were right-handed and experienced with physical keyboard operations, averaging a daily computer usage of around 8 hours (mean

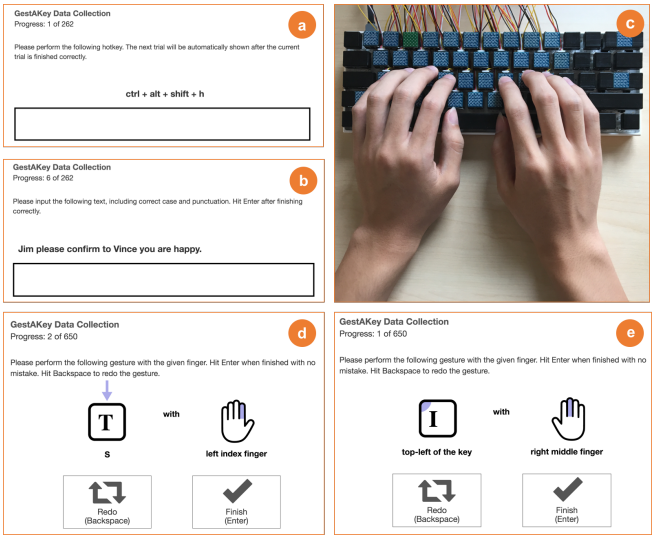

Figure 6. GestAKey apparatus and log study application user interface: (a) UI for hotkey log study; (b) UI for text entry log study; (c) GestAKey prototype; (d) UI for dynamic gesture log study. The purple arrow indicates the gesture to be performed and the purple finger indicates the finger to be used; (e) UI for static gesture log study. The purple dot indicates the static gesture to be performed.

= 8.0, SD = 1.6). To minimize variance caused by different typing styles (e.g., touch typing, hunt-and-peck, etc.), we selected participants that used standard touch-typing in their daily lives.

We modified a mechanical keyboard with GestAKey technology to conduct the log study. The keyboard had a 60% QWERTY layout and was chosen due to its simple design that facilitated our modifications. Keycaps of the 26 alphabetical keys were replaced by custom-made capacitive-sensing keycaps (Figure 6 (c)). To avoid interference with the PCB on the top of the keycaps, key labels were front-printed (Figure 1(a)), which retained the visual accessibility as in normal keyboard usage. The log study was run as a web app hosted on a local Node.js server. Both OS-level keyboard events and touch data from the keycaps were sent to the server through sockets and stored in a local MongoDB database. The whole system ran on a MacBook Pro laptop computer with 3GHz Intel Core i7 processor, 8GB of memory, and running the macOS Sierra operating system. The user interface for the tasks in the experiment was shown on a 23-inch display in a full-screen tab in the Google Chrome browser.

Study Design, Task and Stimuli

The log study consisted of two sessions. The first session collected data about the participants' existing text entry and hotkey actions (Figure 6 (a), (b)), and the second session collected sample data for intended gestural interactions that happened concurrently with a keystroke (Figure 6 (d) (e)).

The first session randomly interleaved two kinds of tasks: text entry and hotkey. For text entry tasks, participants were required to correctly input a sentence with touch-typing. As shown in Figure 6 (a), a text box was provided for user text entry. The expected sentence was always visible above this textbox. The user-entered text was color-coded for the correctness of the input: green for correct letters, and red for incorrect letters. The text corpus for the text entry tasks were chosen from the Enron Mobile corpus [30], using its subset of 80 sentences with representative character bigram frequencies. Participants had to input the correct sentence, including punctuation and letter casing, followed by pressing the "Enter" key, to proceed to the next trial. The participants were allowed to perform text entry and edit in normal ways provided by the operating system ("Delete", "Backspace", "Home", "End", arrow keys, etc.), except for use of mouse. For hotkey trials, a stimulus showed a character key and at least one modifier keys above the textbox (Figure 6 (b)). If a participant performed the correct hotkey combination, the study automatically advanced to the next trial. Otherwise, the presented hotkey combination would turn red to signify an incorrect response. All 7 combinations of three modifier keys, "Ctrl", "Opt" (macOS equivalence of the Windows "Alt" key) and "Shift" were used with each of the 26 alphabetical keys. The total number of trials in the first session of log study was therefore: $26\,\text{keys} \times 7\,\text{modifier combinations} + 80\,\text{sentences} = 262$. It typically took ~30 minutes to finish the first session of the log study.

In the second session, participants were required to perform various touch gestures on a keycap while pressing it. The gesture set defined in the earlier section was used. As shown in Figure 6 (d) and (e), the gestures to be tested were indicated by purple arrows above the key or dots. The purple finger was the one that had to be used to perform the gesture. For this session, stimuli were given as illustrations, not indicating the actual shape of the gestures expected of the participants. Participants had to perform a gesture that they thought matched the illustration. Participants had to confirm the completion of a gesture input by pressing the "Enter" key, which saved the gesture sample and advanced to the next trial. Otherwise, the participants could press the "Backspace" key to perform the gesture again. In total, there were $26\,\text{keys} \times (9\,\text{static positions} + 8\,\text{single direction gestures} + 8\,\text{compound gestures}) = 650$ trials in this session. It typically took ~40 minutes to finish the second session of the log study.

The two sessions of the log study were run consecutively, with a 3-minute break in between the sessions. Except for 2 participants who attended to personal matters, all other participants were ready to start the second session before the break finished, indicating that fatigue was minimal.

Results

Touch Position

1 participant's data didn't show touch actions, so we didn't analyse it. The reason may be that he pressed the keys with fingernails, making the touch event hard to detect with capacitive sensing. For each trial in the first session of the log study (text entry and hotkey), touch locations were aggregated for each key over all participants and shown as heatmaps (Figure 7). It was observed that while touch locations on certain keys were consistently concentrated within a small sub-area,

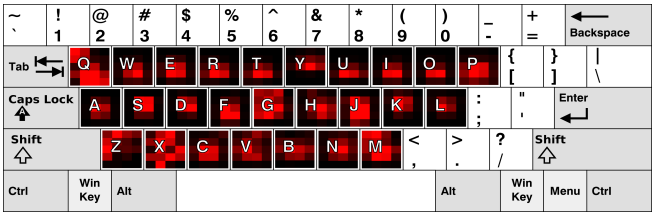

Figure 7. The heatmap of the touch point while typing and performing hotkey, overlaid on each key

touch locations on other keys were subject to higher variances. For keys in the home positions of touch typing (i.e., "A", "S", "D", "F", "J", "k", "L"), the touch locations reflected the natural posture when the fingers rested on these keys. The touch locations on the "A", "S", "D", "F" keys matched the curvature formed by the fingertips when they were placed on the flat surface. Similarly, the "J", "K", "L" keys reflected similar but symmetric touch locations for the right hand. For other keys, the touch locations reflected the way they were typed with the touch-typing technique. For example, in touch typing, the "Z" key is supposed to be pressed by the little finger, which idly rests on "A". As seen in its heatmap, the touch locations on the "Z" key mostly concentrated on the top-left corner, which might reflect that participants moved their fingers minimally from "A" to "Z", and landed on the part of the "Z" key that was nearest to "A".

For keys that required large movement of the fingers from their resting positions, the touch locations either had large dispersion (e.g., "Q", designated to the left little finger, which rests on the "A" key in touch typing), or were very concentrated (e.g., "Y", designated to the right index finger, which rests on the "J" key in touch typing). Although it is not the scope of this research to understand why certain keys are touched in certain ways, we speculate that the difference between the touch patterns on "Q" and "Y" could be due to the fact that while there are various ways to extend the left little finger to reach "Q" from "A". The usual way of extending the right index finger from "J" to "Y" is a to do a full extension of the finger, hitting a very concentrated area on "Y". Overall, the per-key heatmaps reveal that it is feasible to utilize touch locations on individual keycaps to enhance the expressiveness of keystrokes. Specifically, for keys with concentrated touch areas, such as "A", "S", "D", "F", "W", "E", "R", "T", "K", "L", "U", "I", "O", it might be easier to distinguish between a normal keystroke and a "bezel" keystroke, which is performed which touching the edges of these keycaps.

Gesture Distinguishability

Gesture samples from the second session of the log study enabled us to perform a preliminary evaluation of the quality of our gesture set and provides implications on the practical use of these gestures.We conducted an exhaustive leave-one-out cross-validation on all gesture samples collected in the log study[1]. Due to time constraints of the user study, only one sample per gesture per key per user was collected, resulting in a challenging scenario for the cross-validation.

[1]https://goo.gl/vxm8wm

By further requiring that the number of touch points in each gesture sample be at least 4 for static gestures, and 8 for dynamic gestures (compared to the minimum resampling count of 32 recommended in [33]), 3087 gesture samples participated the cross-validation. On average, there were $3087 \div 25\,\text{classes} \div 26\,\text{keys} = 4.75$ tests that could be performed for each gesture class on each key, which would be distributed into a row of 25 cells in the confusion matrix for that key. Therefore, it makes little sense to investigate confusion matrix for individual keys. Instead, the confusion matrix for the full set of 25 static and dynamic gestures aggregated over all keys and users is presented in Figure 8 (left half). The row labels of the confusion matrices are input gesture classes, and column labels are output gesture classes produced by the recognizer. The cell with index (i, j) represents the percentage of occurrences where the i^{th} gesture is classified as the j^{th} gesture. To facilitate comprehension of the data, the percentages are transformed as horizontal bars that fill from left to right. It can be seen that while static gestures and dynamic gestures are perfectly distinguished from each other, there still exists much ambiguity within these categories, indicated by the non-zero elements that do not lie on the diagonal of the matrix.

For static keys, it appeared that the "center" static touch was difficult to differentiate from other static touch. In addition to the intuition that the "center" touch should not be used as a gesture, we should also be skeptical of the use of a special non-gesture class to distinguish between gesture and normal usage of the keyboard. As revealed in the heatmap, there was a high variance in touch locations in text entry and hotkey activities on the keyboard. The difficulty of differentiating a specifically instructed "center" touch from other gestures indicates that a touch generated from normal usage of the keyboard is likely to be equally ambiguous. For the scope of this research, we focused on how well the gestures could be differentiated, and future research would need to address how to reliably differentiate between non-gestures and gestures on individual keycaps.

For dynamic gestures, an interesting observation was that single-directional gestures were harder to distinguish from two-directional gestures that start with the same direction compared to the other classes. This may have been caused by users' subconscious motion back to the center of the keycap when finishing a gesture during the log study. Despite the low accuracies exhibited in the confusion matrix, the higher percentage value of the elements on the diagonal compared to other individual cells in that row indicates that there is reason to believe that the accuracy would improve with increased spatial and temporal resolution of the sensing hardware, as well as more samples used for cross-validation.

As we do not expect all 25 gestures to be used to overload the keys with special functions (compared with the 7 combinations of the "Ctrl", "Shift", "Alt" modifiers that overload a key, and the 10 combinations of hand form and finger in [38]), we investigated whether reducing the number of classes would improve accuracy. From the initial 9 positions/directions of static and dynamic gestures, we removed

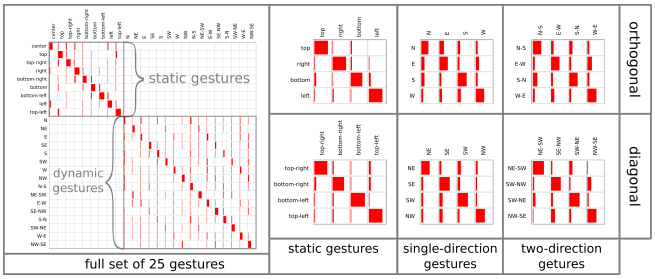

Figure 8. Confusion matrices for the cross validation. Left: confusion matrix for all 25 gestures; Right: 6 possible subset of gestures derived from the full set.

every other gesture class, leaving either the diagonal or the orthogonal positions/directions. For each reduced set of dynamic gestures, we further used subsets that were either one-directional or two-directional. The respective confusion matrices are shown in Figure 8 (right half). 6 confusion matrices are organized by direction ("orthogonal/diagonal") in rows, and type of gestures ("static/single-direction/two-direction") in columns.

Gesture Performance

We focused on accuracy and speed of gesture performance as the two most important indices to decide the feasibility of augmenting each key. The accuracy was calculated as the proportion of correctly (i.e., true positive) classified gestures performed on each keycap, over all participants and all gesture classes, based on the cross-validation result. The speed was calculated as the average duration of these gestures, on each keycap, over all participants and all gesture classes.

The more accurate and faster gestures are performed on a keycap, the more feasible it is to augment that keycap. To visually suggest the keys for augmentation, Figure 9 shows the speed and accuracy for each keycap, for all types of gesture and for each type of gestures individually. The X-axes shows the gesture duration in milliseconds, and the Y-axes shows accuracy in terms of percentage of true positive classification results. Each data point is visualized using the corresponding character of the keycap, and is color-coded according to the finger (for both hands) assigned to that keycap: red for index finger, green for middle finger, blue for ring finger, and purple for little finger. As the top-left corner represents perfect accuracy and minimum gesture articulation time, the proximity of keys to the top-left corner indicates their feasibility for augmentation. It can be seen from the figure that keycaps corresponding to the index finger are most appropriate for augmentation, most notably the keys "H", "Y", "T", "B", "G", and "N", which are not on the "home-columns" of the index fingers. Interestingly, keycaps on the "home-column" of the index fingers are not as feasible.

Discussion and Limitations

The log study exposes several challenges in supporting gestural interactions on such small areas on keycaps. First, with the number of individual touch-sensitive keycaps and the simple electronics we used, scaling up in terms of spatial or temporal resolution is a non-trivial task. Nonetheless, performant device would increase the resolution of gesture samples, which

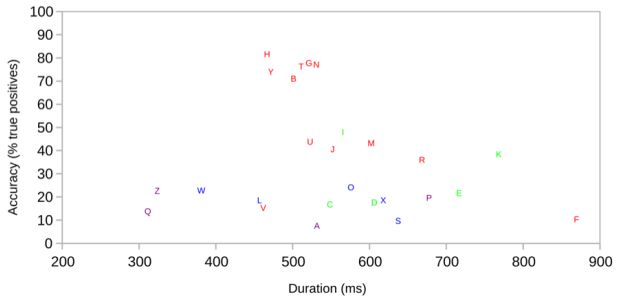

Figure 9. Speed and accuracy of gesture performance for individual keycaps. The letters are color-coded based on the corresponding fingers in touch-typing: red - index finger; green - middle finger; blue - ring finger; purple - little finger.

in turn improves accuracy for gesture classification. Second, given the differences in touch behaviors on different keycaps, it is desirable to employ per-key gesture classification. At the current stage of the prototype system, there are not enough gesture samples to support this approach, and future investigation could explore such possibilities. Third, instead of using a designed gesture set, a gesture elicitation study could be done to investigate gestures that are comfortable and meaningful to users.

Based on the confusion matrix (Figure 8) and per-key speed and accuracy (Figure 9), we recommend two approaches to improve gesture performance on a chosen key: 1) reduce the number of gesture classes; and 2) consider allowing different fingers to perform gestures on a keycap.

Our experiment had limitations. In the data analysis, it was shown that reducing number of classes improved the classification accuracy, but further improvement of the sensing hardware and tuning of the algorithm is still required, as the highest recognition accuracy among all the possible reduced gesture sets is mere 85.2% for the "bottom-left" static gesture and 67.9% for the "NE-SW" dynamic gesture. Due to concerns of user fatigue, length of the study, and the high dimensional nature of the dataset, we were unable to collect enough data to make statistically robust inquiries about the feasibility of GestAKey on each keycap. Nonetheless, we have carried out a preliminary analysis to determine which keys are suitable for augmentation with the GestAKey technique.

CONTROLLED EXPERIMENT

We hypothesize that compared to overloading a key with modifiers, which is the approach of hotkeys, overloading a key with gestures is more flexible, and can therefore lead to more intuitive and learnable input methods. To verify this hypothesis, we conducted a controlled experiment where the performance of GestAKey was compared with two traditional input techniques, hotkey and long press. We selected the task of text input for accented characters for two reasons: 1) It highlights the importance of self-explanatory shape mnemonics in real-world scenarios [20], in contrast to previous research where arbitrary action-to-command mapping was used [38], or only low-level motor performance was assessed [9]; 2) All participants were equally unfamiliar with

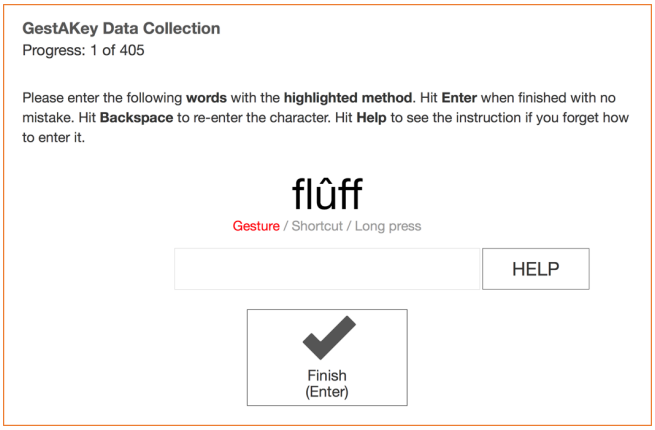

Figure 10. The user interface for controlled experiment.

each input technique, ensuring fair comparison in terms of expertise, and allowing for observation of learning effects. We investigated the speed, accuracy, error rate, and learnability of each technique.

Participants and Apparatus

12 participants (5 female, mean age = 26, SD = 2.1) were recruited for the experiment. All participants were touch-typists and right-handed. The same apparatus used in the log study was used for the controlled experiment. To mitigate the effect of low gesture classification accuracy exposed in the log study, we implemented a simple heuristic-based gesture classifier for the small set of inputs used in the study. It reached an accuracy of 90% with single directional gesture and compound directional gesture.

Task and Stimuli

The task of the controlled experiment was to input words with accented letters. For every trial, participants were asked to input a word with accented vowels using one of three input techniques: hotkeys, long press or gesturing on individual keycaps.

We chose commonly used accented vowels as the stimuli of the experiment. All five vowels, (i.e. "a", "e", "i", "o" and "u") were used to create the experiment trials. To mimic the real text entry scenario, we chose to add different accents to the vowels of English words, which the participants would find familiar and provided the feeling of typing a whole word. For fairness, all 5 words contained exactly one vowel and the vowel was near the middle of the word. The five words were: *clack, whelm, hicks, fords, fluff*. Based on these 5 words, we applied three accents, which were " ´ ", " ` ", " ^ ", on the vowels and created three new words for each one (e.g., "cláck", "clàck", "clâck").

With the gesture technique, the user performed three gestures corresponding to the three shapes of the accent, "NE", "SE", and "N-S". With the hotkey technique, the user first performed a dead key[2], followed by the respective character to add accent to. For example, "Opt + e" followed by "o" would produce "ó". With the long press technique, the user pressed and holded a character until a pop-up list appeared, with accented versions of the character. The user then selected the index of the desired accented character using numeric input. The latter two techniques are built-in mechanisms used in common macOS computers.

[2]https://goo.gl/NGQ8EV

Procedure

The same user interface used for the log study was used for the controlled experiment(Figure 10). Similar to the text entry tasks in the log study, the target word was shown above the textbox. Participants were allowed to use the "Backspace" key to edit the input, until it correctly matched the stimulus and the"Enter" key was pressed to proceed to the next trial. In addition, a cheat-sheet was accessible by holding the "Space" key. The cheat-sheet showed how to input each of the accents for each technique.

Design

The experiment was designed as a within-subject, repeated measures, full factorial experiment. We were interested in the factor of techniques and the learning effect. Therefore, we counterbalanced the order of techniques within each block. A participant would use one technique to finish all trials with different words and accents, each repeated three times, in random order, before moving on to the next technique. Three blocks with identical sequence of techniques were administered, and trials in each technique were randomized. In total, the design was as follows:

$$5\,\text{Letters} \times 3\,\text{Accents} \times 3\,\text{Techniques} \times 3\,\text{Repetitions} \times 3\,\text{Blocks} = 405\,\text{Trials}$$

Typically it took about 40 to 50 minutes to finish the controlled experiment.

Measures

We defined four measures to reflect the efficiency and learnability while performing the trials. The first two were completion time of the whole trial, and number of times "Delete" was pressed. These two conventional measures reflect the speed and accuracy of task performance. The third measure was the time it took to input the accented character alone. This measure was different from the completion time of the whole word, as it reflected the actuation speed using each technique, by disregarding the time of cognitive process without physical actions, such as recalling the mapping between gestures and the shape of the accent. The last measure was the number of times the "Space" key was pressed during the trial. This was a measure of learnability and directly indicates how well the participant has learnt each technique.

Results

Completion Time

The overall average completion time of whole words was M_{word} = 5.42s (SD_{word}= 1.35s) and for single accented characters was M_{word} = 1.06s (SD_{word}= 0.52s). The three techniques had the following mean completion time of whole

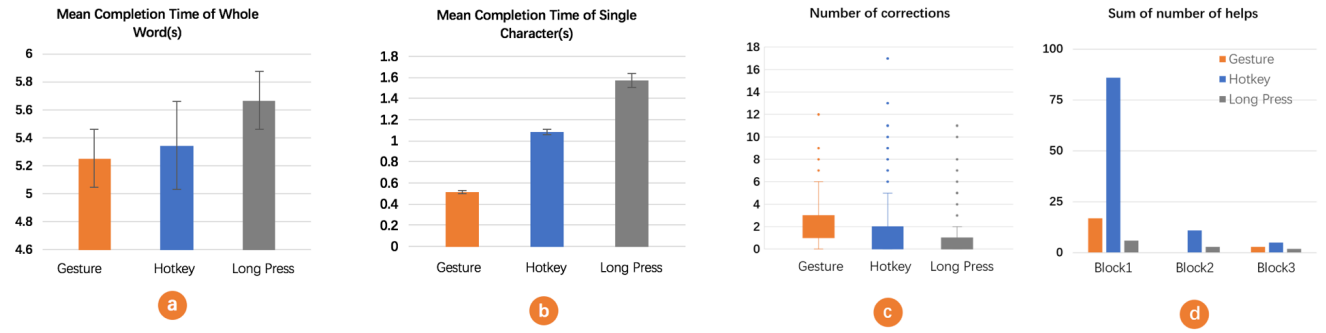

Figure 11. Experiment results: (a) Completion time of whole word with 95% confidence interval; (b) Completion time of single accented character with 95% confidence interval; (c) Box plot to show the distribution of corrections for each technique; (d) Total number of helps for each technique in each block.

word: $M_{Long\ press}$ = 5.67s ($SD_{Long\ press}$= 0.98s), M_{Hotkey} = 5.35s (SD_{Hotkey} = 1.72s) and $M_{Gesture}$ =5.25s ($SD_{Gesture}$ = 1.25s). For single accented characters, the three techniques had the respective mean completion time of $M_{Long\ press}$ = 1.57s ($SD_{Long\ press}$ = 0.41s), M_{Hotkey} = 1.08s (SD_{Hotkey} =0.22s) and $M_{Gesture}$ = 0.51s ($SD_{Gesture}$ = 0.17s).

It was found that block ($F_{2, 22}$= 56.83, $p < 0.01$) had significant main effects on the completion time of a whole word. TukeyHSD post-hoc tests found that: (1) Block 1 was significantly slower than block 2 ($p < 0.01$) and block 3 ($p < 0.01$). (2) Block 3 was significantly faster than block 2 ($p = 0.02$). No significant difference was found between techniques ($p = 0.14$). There was a significant interaction effect between block and technique ($F_{4, 44} = 3.95$, $p < 0.01$). From Figure 12(a) we could see that users learned keyboard shortcut fastest, although it was the slowest technique during the initial use.

For the completion time of a single character, both block ($F_{2, 22}$= 5.70, $p < 0.01$) and technique ($F_{2, 22}$= 132.91, $p < 0.01$) had significant main effects. TukeyHSD post-hoc tests found that: (1) Block 1 was significantly slower than block 2 and block 3 ($p < 0.01$). But there was no significant difference between block 2 and block 3. (2) Gesture was 571ms faster than hotkey, and 1056ms faster than long press (all significant: $p < 0.01$). Hotkey was also significantly faster than long press ($p < 0.01$). There was no significant interaction effect between block and technique ($F_{4, 44}$= 1.33, $p > 0.1$). From Figure 12(b) we could see that the learning effect of different techniques was not obvious. Users could reach the expert performance quickly when they use GestAKey.

Error Rate

The average number of corrections per trial for each technique was: $M_{Long\ press}$ = 0.27 ($SD_{Long\ press}$ = 0.16), M_{Hotkey} =0.50 (SD_{Hotkey} = 0.31), $M_{Gesture}$ = 0.75 ($SD_{Gesture}$ = 0.33). The average number of corrections per trial for each blocks was: M_{Block1} = 0.63 (SD_{Block1} = 0.41), M_{Block2} = 0.46 (SD_{Block2} = 0.30), M_{Block3} = 0.42 (SD_{Block3} = 0.25).

Both technique ($F_{2, 22}$ = 29.26, $p < 0.01$) and block ($F_{2, 22}$ = 6.06, $p < 0.01$) had significant main effects on error rate. TukeyHSD post-hoc analysis found that long press had 0.48 corrections less than gesture, and 0.25 corrections less than hotkey (all significant: $p < 0.01$). Block3 had 0.21 correction less than Block1 (significant: $p < 0.01$). Block 2 had 0.17 corrections less than Block 1 (significant: $p < 0.01$).

Number of Help Requests

Both technique ($F_{2,22}$ = 28.16, $p < 0.01$) and block ($F_{2,22}$ = 36.16, $p < 0.01$) had significant main effects on how many time participants hit the spacebar for help. TukeyHSD post-hoc analysis found that Hotkey caused most space bar hitting, 0.041 more than gesture ($p < 0.01$) and there was no significant difference between long press and gesture ($p = 0.80$), Block3 and Block2 ($p = 0.90$).

Discussion and Limitations

The results showed that although long press was the most accurate and easiest-to-learn, it was the slowest technique to input accented words. For the other two techniques, gesturing on a keycap had a comparable overall speed with hotkeys. However, gesturing on keycaps had better performance in terms of input speed for a single accented character compared to hotkeys. In addition, it was also easier to remember gestures. These two findings suggest a potential improvement on the other two techniques. The possible reasons of higher error rate of the gesture technique could be: (1) although we chose rarely used shortcuts, experienced users, such as touch-typists, could still leverage their familiarity with the keyboard. However, gesturing on individual keycaps was new

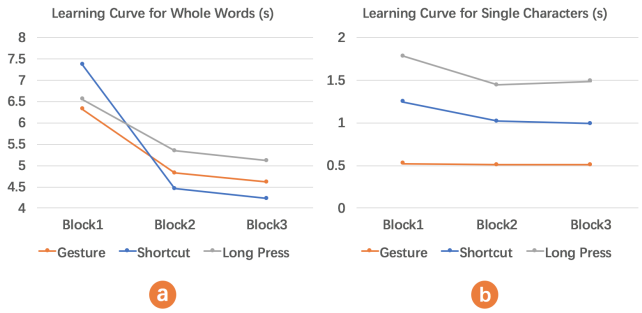

Figure 12. Completion time of (a) whole words and (b) single characters in each block .

Figure 13. GestAKey enables users to input three kinds of text formatting with gestures: (1) Superscript and subscript, (2) Accented and (3) Greek letters.

to them. (2) while keystrokes can be reliably detected, gesture recognition still suffers from low accuracy. In summary, despite the temporary low accuracy limited by the current implementation, GestAKey's balance between efficiency and learnability gives it potential to open up novel interactions.

Our experiment had limitations. Firstly, to highlight the importance of intuitive shape mnemonics, we chose accented letters as our stimuli, resulting in an unbalanced key selection. Secondly, there exists simpler hotkeys than those we used. More studies need to be done to compare GestAKey with simple hotkeys. Finally, there were only 3 different accents used in our study. In the future, we can investigate more gestures, which may reveal results we didn't discover this time.

APPLICATIONS

We implemented several applications to demonstrate the possibilities of the GestAKey: Personalized Shortcuts, Augmented Text Entry, and Enriched Gaming Experience.

Personalized Shortcuts

GestAKey can assign more meaningful and memorable functionalities to a single key, which can be associated to the position or direction of gestures. We implemented a demo personalized shortcut application that supported a number of GestAKey interactions. For example, the application could be personalized such that pressing "M" and swiping up displayed information about the current music track. Pressing and swiping down on"M" downloaded it. Pressing "Y" and swiping up launches a browser and navigate to YouTube. Circular motions on the "V" key increased or decreased the volume, which conventionally requires more than one key to control.

Augmented Text Entry

We implemented an intuitive stylized text entry plugin in Microsoft Word. It allowed the user to input stylized text while entering the text. No explicit mode switch was required for different styles, nor a mouse needed to interact with the GUI. It leveraged the shape of gestures as mnemonics to facilitate memorization. To input an accented letter, users could press a key and perform the corresponding gesture, such as inputting "á" by swiping to the top-right while performing a keystroke on the "a" key. In a rich-text editor like Microsoft Word, by pressing a letter and swiping up or down, users could input that letter in superscript/subscript format, in a simple gesture that did not distract users from the normal typing actions. A a set of potential gestures to enable styling text using GestAKey is shown in Figure 13.

Enriched Gaming Experience

GestAKey can also enrich the gaming experience by enabling continuous parameter control. For example, when playing a game, a running action is usually triggered by a specific key. With GestAKey, players can accelerate the movement of the character by swiping the fingertip towards the top of the keycap, or decelerate by swiping the fingertip towards the bottom edge. To control the distance of a jumping action, players can touch different areas of the keycap when performing the keystroke for the jump action.

LIMITATIONS AND FUTURE WORK

In this paper, GestAKey technology was only evaluated for text entry tasks. In our future investigations, we will evaluate the usability of GestAKey in other specific application scenarios.

Our current scope only concerns a single-touch gesture on a single keycap. It is interesting to explore multi-touch or multi-key interactions using the GestAKey technology. For example, two fingers can perform the same gesture on two keys to further increase the interaction space of GestAKey. Another future direction is to expand the GestAKey technology to cover other types of keys, such as the spacebar, which has a large area suitable for touch interaction, such as leveraging the spacebar as a small touchpad. Finally, the GestAKey system can be a tool to investigate users' typing behaviors. Our log study could be extended to a full investigation that identifies touch patterns associated with keyboard usage, such as typing, so as to extend our general knowledge on how we type, which complements the inspirational work of [13].

CONCLUSION

In this work, we introduced GestAKey, a technique that enabled multifunctional keystrokes on a touch-sensitive keycap, opening up new design opportunities for interactions on a keyboard. We developed a custom-made keycap and built the software and hardware systems to detect touch on individual keycaps. We conducted a log study and verified the feasibility of gestural interaction on keycaps. A controlled experiment showed that although subject to system imperfections, our technique still had a comparable overall performance. Applications using the GestAKey technology should leverage the flexibility of gestural input and tightly couple it with existing key functions. With this approach, we believe GestAKey technology will enable the creation of intuitive, learnable, and efficient interaction experiences on keyboards.

ACKNOWLEDGEMENT

This work is partially funded by the first Authors's PhD Scholarship (President's Graduate Fellowship, Singapore University of Technology and Design). Also, we thank all study participants for their time and valuable feedback.

REFERENCES

1. How to use the touchbar on your macbook pro. `https://support.apple.com/en-sg/HT207055`. Accessed: 2017-12-18.
2. ios 11: How the one-handed and quicktype keyboards work.

https://www.macworld.com/article/3226394/ios/ios-11-how-the-one-handed-and-quicktype-keyboards-work.html. Accessed: 2017-12-18.

3. Messageease keyboard for touch screen devices. http://www.exideas.com/ME/index.php. Accessed: 2017-12-18.
4. Alvina, J., Griggio, C. F., Bi, X., and Mackay, W. E. 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*, ACM (2017), 17–28.
5. Alvina, J., Malloch, J., and Mackay, W. E. Expressive Keyboards: Enriching Gesture-Typing on Mobile Devices. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16* (2016), 583–593.
6. Appert, C., and Zhai, S. Using Strokes as Command Shortcuts : Cognitive Benefits and Toolkit Support. *Proceedings of the 27th international conference on Human factors in computing systems (CHI'09) 2009* (2009), 2289–2298.
7. Arif, A. S., Pahud, M., Hinckley, K., and Buxton, B. Experimental Study of Stroke Shortcuts for a Touchscreen Keyboard with Gesture-Redundant Keys Removed. *Proceedings of the 40st Graphic Interface Conference (GI'14)* (2014), 43–50.
8. Au, O. K.-C., Yeung, K. Y.-W., and Cheung, J. K. DownChord and UpChord. *Proceedings of the Fourth International Symposium on Chinese CHI - ChineseCHI2016* (2016), 1–10.
9. Bailly, G., Pietrzak, T., Deber, J., and Wigdor, D. J. Métamorphe: augmenting hotkey usage with actuated keys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 563–572.
10. Block, F., Gellersen, H., and Villar, N. Touch-display keyboards. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10* (2010), 1145.
11. Dietz, P. H., Eidelson, B., Westhues, J., and Bathiche, S. A practical pressure sensitive computer keyboard. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, ACM (2009), 55–58.
12. Fallot-Burghardt, W., Fjeld, M., Speirs, C., Ziegenspeck, S., Krueger, H., and Läubli, T. Touch&Type: a novel pointing device for notebook computers. *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction*, October (2006), 465–468.
13. Feit, A. M., Weir, D., and Oulasvirta, A. How we type: Movement strategies and performance in everyday typing. In *Proceedings of the 2016 chi conference on human factors in computing systems*, ACM (2016), 4262–4273.
14. Findlater, L., Lee, B., and Wobbrock, J. Beyond QWERTY: augmenting touch screen keyboards with multi-touch gestures for non-alphanumeric input. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12* (2012), 2679–2682.
15. Grossman, T., Dragicevic, P., and Balakrishnan, R. Strategies for accelerating on-line learning of hotkeys. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07* (2007), 1591–1600.
16. Isokoski, P. Performance of Menu-Augmented Soft Keyboards. *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04 6*, 1 (2004), 423–430.
17. Jannotti, J. Iconic text entry using a numeric keypad. *Unpublished. http://www. jannotti. com/papers/iconic-uist02. pdf* (2002).
18. Kato, J., Sakamoto, D., and Igarashi, T. Surfboard: keyboard with microphone as a low-cost interactive surface. In *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology*, ACM (2010), 387–388.
19. Kristensson, P.-O., and Zhai, S. SHARK: A Large Vocabulary Shorthand Writing System for Pen-Based Computers. *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04 6*, 2 (2004), 43–52.
20. Kurtenbach, G., Moran, T. P., and Buxton, W. Contextual animation of gestural commands. In *Computer Graphics Forum*, vol. 13, Wiley Online Library (1994), 305–314.
21. Lane, D. M., Napier, H. A., Peres, S. C., and Sandor, A. 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.
22. Li, F. C. Y., Guy, R. T., Yatani, K., and Truong, K. N. The 1line keyboard: A QWERTY layout in a single line. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11* (2011), 461–470.
23. Li, Y. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, ACM (New York, NY, USA, 2010), 2169–2172.
24. McLoone, H., Hinckley, K., and Cutrell, E. Bimanual Interaction on the Microsoft Office Keyboard. *Human-Computer Interaction INTERACT'03* (2003), 49–56.
25. Morrel-Samuels, P. Clarifying the distinction between lexical and gestural commands. *International Journal of Man-Machine Studies 32*, 5 (1990), 581–590.

26. Rekimoto, J., Oba, H., and Ishizawa, T. Smartpad: a finger-sensing keypad for mobile interaction. In *CHI'03 Extended Abstracts on Human Factors in Computing Systems*, ACM (2003), 850–851.
27. Tak, S., Westendorp, P., and van Rooij, I. Satisficing and the use of keyboard shortcuts: Being good enough is enough? *Interacting with computers 25*, 5 (2013), 404–416.
28. Taylor, S., Keskin, C., Hilliges, O., Izadi, S., and Helmes, J. Type-hover-swipe in 96 bytes. *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14* (2014), 1695–1704.
29. Tung, Y.-C., Cheng, T. Y., Yu, N.-H., Wang, C., and Chen, M. Y. Flickboard: enabling trackpad interaction with automatic mode switching on a capacitive-sensing keyboard. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM (2015), 1847–1850.
30. Vertanen, K., and Kristensson, P. O. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, ACM (2011), 295–298.
31. Wilson, A. D. Robust computer vision-based detection of pinching for one and two-handed gesture input. *Proceedings of the 19th annual ACM symposium on User interface software and technology - UIST '06* (2006), 255.
32. Wobbrock, J. O., Myers, B. A., and Rothrock, B. Few-key text entry revisited: mnemonic gestures on four keys. *Proceedings of ACM CHI 2006 Conference on Human Factors in Computing Systems 1* (2006), 489–492.
33. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, ACM (New York, NY, USA, 2007), 159–168.
34. Zhai, S. Foundational Issues in Touch-Surface Stroke Gesture Design An Integrative Review. *Foundations and Trends® in HumanComputer Interaction 5*, 2 (2012), 97–205.
35. Zhai, S., and Kristensson, P.-O. Shorthand writing on stylus keyboard. *Proceedings of the conference on Human factors in computing systems - CHI '03*, 5 (2003), 97.
36. Zhai, S., and Kristensson, P. O. The word-gesture keyboard. *Communications of the ACM 55*, 9 (2012), 91.
37. Zhang, H., and Li, Y. Gestkeyboard: enabling gesture-based interaction on ordinary physical keyboard. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, ACM (2014), 1675–1684.
38. Zheng, J., and Vogel, D. Finger-Aware Shortcuts. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16* (2016), 4274–4285.