MotionInput: Gestural Text Entry in the Air

Shuo Qiu	Kyle Rego	Lei Zhang	Feifei Zhong	Michael Zhong
M.S. in CS '13	SEAS '13	M.S. in CS '13	M.S. in CS '13	SEAS '13
sq2144	kar2150	lz2343	fz2185	mqz2001
@columbia.edu	@columbia.edu	@columbia.edu	@columbia.edu	@columbia.edu

ABSTRACT

Gestural keyboards have become a popular alternative to character-by-character entry on touch screen devices today. The obvious limitation to text entry via a touch screen though is that it is fixed to the device. For this project, we aimed to examine methodologies to input text with hand gestures in the air. This paper outlines a means to extend the implementation of gestural text entry on a touch screen and map such an input into gestural inputs in the air. In our implementation, we provide a simple means for the user to, in one continuous motion, enter text by selecting a cluster of letters. Given the selected cluster of letters, the user can then select a list of possible words. Our approach has several drawbacks, mainly that it required the use of a reference visual interface and the text-entry is tediously slow. However, we found that our system is easy to learn and it does successfully input text via hand gestures through the air.

Keywords

Text Entry, Gestures

INTRODUCTION

Over the last decade there has been a surge of commercially available gestural keyboards from products such as Swype, ShapeWriter, SwiftKey, TouchPal, and T9 Trace [1]. These gestural keyboards in experimental settings typically far outperform manual character-entry methods in speed (as measured by words per minute) and are on par in terms of error rate. For a more complicated version of gestural keyboards, that is with bimanual gesture keyboards on a mobile device, the experiment reported superior text-entry speed, and it also noted that users were able to easily adapt to a gestural keyboard approach. Gestural keyboards have become steadily more popular in touch screens and have been successful commercially.

With all the advantages a gestural keyboard contains, the gestural keyboard implementation also has limitations that form the basis for our research. One issue is simply that the gestural keyboard is fixed to a touchscreen, whether it be a smartphone or a touchpad. A second issue is that all gestural keyboards presently rely on a reference frame, usually indicated by having an image of a keyboard on the touchscreen. This requires active attention using both touch and sight and can be susceptible to the fat finger problem, an issue that arises when the user's finger occludes the user from seeing where the finger is actually touching on the screen [2]. This can be detrimental because of the gestural keyboard's reliance on a reference image and visual input.

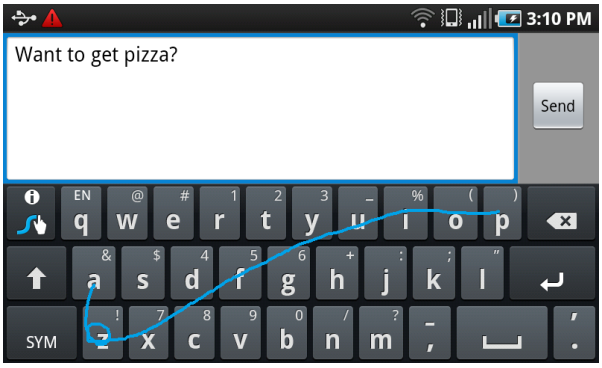

Figure 1. Swype, a commercial gestural keyboard, displayed in action. The word pizza is being entered using a single gesture.

For our project, we strove to alleviate these problems found with a touchscreen-based gestural keyboard. Our project, MotionInput, is a gestural text entry system that operates by making hand gestures, primarily by tracing a finger. Specifically we display an octagon where each edge contains a group of letters and a word is spelled out by moving one's finger towards each of the respective letter groups, as illustrated below.

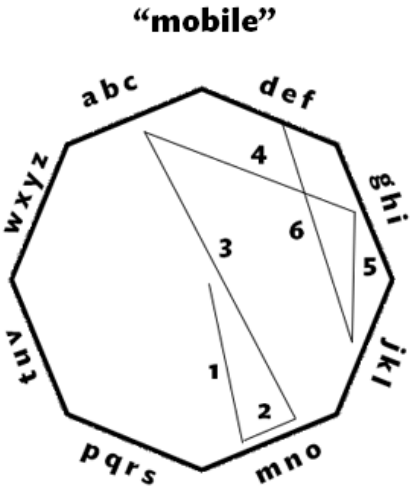

Figure 2. The word "mobile" is being spelled out here by tracing one's finger towards the appropriate letter clusters.

Once the letter group is traced out, using the T9 predictive word algorithm [3,4], if there are three or more possible words, the algorithm will pick the three most popular words, which the user can then pick from by raising one, two, or three fingers.

We were unable to implement a reliable system for MotionInput that excluded the use of visual input—we ended up using a dynamically updating visual image for referencing. However, MotionInput, in our user study, was found to be very easy to learn and by not being confined to a touchscreen, does not encounter the fat finger occlusion problem.

RELATED WORK

There has been considerable research put into text-entry the past two decades. However, to our knowledge, there is no existing per-word gesture text-entry system implemented through the air. In addition, according to developers working for Leap Motion, there presently have been no text-entry systems that have used the Leap device, a controller yet to be shipped that can detect hand and finger movements.

Gestural keyboards on a touchscreen are related to the work we propose. The fastest gestural keyboards on a touchscreen enter text far more quickly than MotionInput. These systems though are not implemented in the air, are more confined in terms of what is an acceptable movement, and have a different character layout on the keyboard [1,5].

Hex, a gestural text-entry system implemented by Williamson and Murray-Smith (2005) has a similar character layout to MotionInput and a similar text-entry method [6]. However, Hex, pictured below, too is confined to touchscreens.

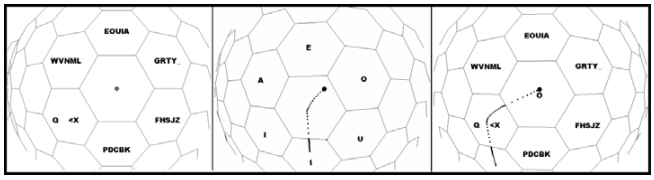

Figure 3. The character layout of the text-entry system Hex.

One of the earliest text-entry systems not confined to a touchscreen was Roeber, Bacus, and Tomasi's research in 2003. For their work, they produced a keyboard that could be projected onto a flat surface and then typed into manually [7]. Such a system was novel at the time, but is different from our work in that it enters text on a per-character basis rather than a per-word basis and it still requires a flat surface.

More recently in 2008, Castellucci and MacKenzie produced a text entry system called Unigest. Unigest does input characters in a three-dimensional space by using a Nintendo Wii controller to produce characters. Each character in the alphabet is mapped to a certain set of gestures [8]. Unigest differs from MotionInput in that, although it does enter text in three dimensions and does succeed in achieving our end goal of entering text without any visual references, it enters words on a per-character basis and does not enter in a word using a single elongated gesture. Unigest provides more accurate notation to mark the beginning and end of a word by use of pressing the Wii controller's "A" button. Unlike our project, this feature requires the input device to be in the user's hand while entering text.

There have been many gesture recognizers that can detect hand movement in the air. One of many examples is the $1 Recognizer that can detect a variety of gestures on a limited scale with up to 99 percent accuracy [9]. These gesture recognizers have a fixed number of patterns though and to our knowledge have not transferred over into the realm of text-entry.

IMPLEMENTATION

The inner workings of MotionInput can be divided up into three parts: detecting the gesture of the hand position and finger movements as well as mapping the finger position to the corresponding letter cluster, drawing the finger movements onto the visual reference, and then selecting the possible words associated with the given set of letter clusters.

Gesture Detection

For gesture detection, we used a Leap Motion controller, which was built specifically for hand and finger detection (rather than full-body detection, like the Kinect). Presently the Leap Motion is not commercially available and we acquired a Leap Motion controller by applying through its developer's program. The latest software version we used was 0.7.3. To interact with the Leap Motion device, we used its Python API available through its Software Development Kit. Using the Leap Motion's built-in API calls, we were able to acquire data on the finger positions as well as the number of fingers at any given time.

For our implementation, we relied exclusively on finger gestures for MotionInput. Having one finger out produced the pointer which selected the first candidate character cluster. If the user wished to remove a letter, the user would extend out five fingers. Once the desired word was spelt out, extending four fingers would signal the end of the word. Then given the most likely words, the user would then tap one, two, or three fingers to select the desired word.

Drawing and Animation

For the visual output, we used standard tkinter GUI toolkit taken from Python's libraries. The canvas is about 400x400 pixels and in the middle is an octagon with each line of the octagon representing a different letter cluster that could be chosen. The visual interface's boundaries are indicated by two circle around the octagon. When a user moves the finger through a line of the octagon into the button area, MotionInput would know which button is chose. Each time

MotionInput receives a new input, it then uses the T9 algorithm (outlined below) to get the candidate list of words and display them on the top of the window. Each time MotionInput would display at most three candidate words and users can use gestures to see the rest of the words.

We mapped the Leap Motion's finger coordinates to the canvas, using relative positioning and computing the angle of the change in direction in order to identify the position of the pointer at any point, which is then corresponded to the right letter cluster.

Below is an image of the final visual output produced after a complete run-through of one word of the MotionInput. The word "am" is being spelt out in the below image.

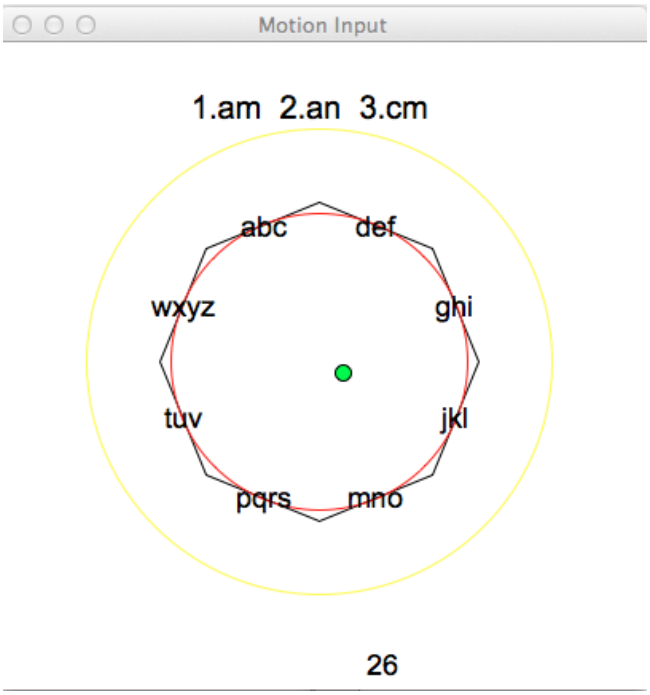

Figure 4. The MotionInput visual interface.

Text Input Algorithm

MotionInput uses a simplified version of the proprietary T9 algorithm, which was developed by Tegic Communications (now a part of Nuance Communications), and can predict the most probable words given a number-to-letter-cluster mapping. The number-to-letter-cluster mapping used by the T9 algorithm is the exact same as the number-to-letter-cluster used on a standard keypad of a telephone. We adopted the same mapping for our text input algorithm. Using the simplified T9 algorithm provided in open source by GitHub user npezolano, we read a dictionary file—which can be manually altered—and stored every word into the search tree. When MotionInput gets the new input, it would search the words in the tree using regular expression and then provide the word list based on frequency. Right now our algorithm runs at near-instantaneous speeds for words of seven or fewer characters, but slows down considerably for words longer than that. Given that our end goal was focused more on examining the usability and intuitiveness of entering text in the air, and less so with performance and glitches, while this bothered us, we were satisfied with the simplified T9 algorithm's word-finding prowess.

USER STUDY

For our user study, we had two main goals in mind. The first was to test the usability and intuitiveness of entering text in the air through hand and finger gestures. The second was to evaluate the performance of our work. Our end goal was to survey some of the implications of entering text through a per-word, single-gesture text system.

Participants

We recruited 10 participants (4 females) passing by the second floor of Lerner Hall at Columbia University, at random, between the ages of 19 to 25. Of the 10, 9 used their right hand to enter in text. Each participant used his or her dominant hand to enter in text.

Apparatus

Each user interacted with the Leap Motion controller, with a brief demonstration on how to use the controller. The visual output was displayed on a 13" MacBook Pro.

Procedures

We tested each user with two different interfaces—our primary, aforementioned interface, and a different interface where the user indicates with one, two, three, or four fingers the appropriate character of the given letter cluster. For each user, we alternated which interface the user used first. Users were given two minutes to practice with both interfaces beforehand.

For each interface we provided a script of words for the user to enter in through MotionInput or the second system mentioned in the previous paragraph, allowing the user to type in as many words as possible within a three-minute window. The script was read out loud; immediately after a user finished entering in a word, we provided them the next word.

Following the end of the test, users were asked to fill out a questionnaire concerning the usability and issues with entering text through MotionInput.

Results

Tests were conducted for both interfaces. We investigated how quickly and how accurately a user could input text. With regards to accuracy, on average, using MotionInput, the user erred 16 percent of the time. The second interface was considerably more buggy, produced considerably more errors, and users reported was less enjoyable to use.

With respect to speed, using MotionInput, the average user entered in text at 8.4 words per minute, with the best user entering text at 11 words a minute. The second system was considerably slower. On average, the user was able to enter in 4.1 words a minute, with the top user entering in 7 wpm.

Performance wise, the system performed poorly in terms of speed and accuracy relative to touchscreen gestural text-entry systems, and in our questionnaire, users overwhelmingly said the most important improvement needed for MotionInput was to make text-entry faster. Users reported finding our system easy and comfortable to use. When asked about the comfort of the system and the ease of learning the system, on a 1-5 scale, users reported scores of 4.2 and 4.8 respectively.

DISCUSSION

Our study results showed that our system was learnable and comfortable to use. Performance wise, relative to other commercial text-entry systems like Swype, ShapeWriter, or TouchPal, as well as research text-entry systems such as Hex, MotionInput was found to be considerably slower. In some instances, MotionInput was slower by triple the word speed relative to the fastest text-entry systems. For the error rate, while closer to other text-entry systems than its speed, MotionInput still performed worse.

Our work can be viewed as a recognition that users can readily pick up this implementation of in-the-air gestural text entry, but as it is presently defined and implemented, is not a practical product.

FUTURE WORK

There are many ways we can develop on our initial prototype of entering text through a three-dimensional field. Purely implementation-based, there are a few nagging bugs that we need to correct in our system—for example, presently with words more than seven characters long, it takes too long for the T9 algorithm to produce an output. Conceptually, one regret with this project was that we were unable to provide a reliable implementation that did not rely on a visual output in order to guide hand gestures. One way to make our system more mobile would be eliminate this dependency on a visual reference.

Furthermore, while users were able to adapt to our system easily, the character layout is not as intuitive as, for example, a keyboard layout. The words per minute measurement is not particularly low—which is acceptable for our purposes of seeing whether users could quickly grasp using an in-the-air text entry system, but makes the system almost obsolete for practical purposes. Including some machine learning techniques and incorporating the use of shape-detection and pattern-recognition techniques for gestures, rather than relying on a reference point, would certainly be worth further investigation.

Our system is also a fairly crude implementation—that is, it is restricted just to the English alphabet and does not support the use of any symbols. We could investigate autocomplete features, autocorrect functionality, including punctuation, and/or correcting errors in words. In short, there are many paths we can take that build off of our initial prototype.

CONCLUSION

This paper outlines what we believe to be the first per-word gesture-based text entry system produced. We conducted a user study to test MotionInput's performance. We found text entry with MotionInput to be considerably slower than commercial text-entry systems such as Swype or SwiftKey. Our interface though was very easy to grasp and provided more flexibility for movement—and again, was not confined to a surface—than a touchscreen interface.

REFERENCES

1. Bi, X., Chelba, C., Ouyang, T., Partridge, K., & Zhai, S. (2012). Bimanual gesture keyboard. In Proceedings of the 25th annual ACM symposium on User interface software and technology (pp. 137-146). ACM.
2. Vogel, D., & Baudisch, P. (2007). Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 657-666). ACM.
3. Definition of T9: What is T9 Predictive Text? http://cellphones.about.com/od/phoneglossary/g/t9predictivetext.htm
4. Python T9 Implementation https://github.com/npezolano/Python-T9-implementation
5. Kristensson, P. O., & Zhai, S. (2004). SHARK 2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology* (pp. 43-52). ACM.
6. Williamson, J., & Murray-Smith, R. (2005). Hex: Dynamics and probabilistic text entry. In *Switching and Learning in Feedback Systems* (pp. 333-342). Springer Berlin Heidelberg.
7. Roeber, H., Bacus, J., & Tomasi, C. (2003). Typing in thin air: the canesta projection keyboard-a new method of interaction with electronic devices. In *CHI'03 extended abstracts on Human factors in computing systems*(pp. 712-713). ACM.
8. Castellucci, S. J., & MacKenzie, I. S. (2008). Unigest: text entry using three degrees of motion. In *CHI'08 Extended Abstracts on Human Factors in Computing Systems* (pp. 3549-3554). ACM.
9. Wobbrock, J. O., Wilson, A. D., & Li, Y. (2007, October). 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* (pp. 159-168). ACM.