

Adaptive interface for text input on large-scale interactive surfaces

Johannes Hirche
Luleå University of Technology
johannes.hirche@ltu.se

Peter Bomark, Mikael Bauer, Pawel Solyga
Natural User Interface Europe AB
{pb|mb|ps}@natural-ui.com

Abstract

In this paper we present a novel approach to text input on large interactive surfaces using a combination of strategies to resolve the inherent difficulties with text input on such a device. Instead of using a conventional full size QWERTY based layout, the idea is to use a very limited set of buttons that, by using word prediction and hints, would only require minimal finger movement. The input mechanism is somewhat related to input methods employed when using keyboards with a limited size and amount of keys, commonly found in phones and other 10 digit keyboards. Given that the main motivation to this approach was not the limited size but rather to overcome the difficult task of locating keys with fingers on a flat and featureless surface, making touch-typing very difficult and requiring frequent visual monitoring of the finger position, we opted to enhance the interaction with easily made gestures, a layout that adapts to the hand anatomy of the user, and easy control over the text prediction.

1. Introduction

Text input and editing plays a fundamental role when interacting with computers and other electronic devices. To this day the keyboard design based on mechanical typewriters is ubiquitous, predominantly using a QWERTY[8] layout, the only other ANSI registered layout created by Dvorak[4], which was designed with the physiology of the hand and letter frequency of the (english) language in mind, failed to displace the QWERTY layout. There have been numerous efforts to introduce new text input devices, [9, 2, 7, 1] to name just a few, with little success at replacing the established keyboard other than in niche areas like wearable computing or solutions for handicapped people, where it is awkward or impossible to type using both hands.

Alternative forms of text input like handwriting recognition, which would be technically possible on an interactive surface, suffer from speed limitations at typically 20 wpm, or in the case of speech recognition from correctness

and most prominently in the case of a multi-user interactive surface from privacy and ambiguity with respect to the currently speaking person as well[10].

1.1. Typing on touch interfaces

One of the major drawbacks when using a keyboard on a touch interface is that the user does not get any tactile feedback from the surface, as one would get with physical buttons or keys. This often results in an increased amount of typing errors, a need for the user to get more visual feedback by looking at what the hands are doing, and a general sense of uncertainty when typing. Touch typing is hard to accomplish as well because of the lack of tactile feedback. In case of an on-screen full QWERTY keyboard the hands occlude the lower part of the keyboard and may even cause ghost inputs with the palms.

A part of the problem is that normal keyboard layouts requires a pretty large movement distance for the fingers when typing. Excessive use of gestures and drawing, as frequently used in combination with a stylus, can cause problems when using bare hands because of friction, especially when pressure sensitivity is used as well [3]. By decreasing the distance needed to move the fingers the effects of a lack of tactile feedback could be reduced.

As stated by [6], the familiarity of the user with the input technique is of great importance in order to gain acceptance, on the other hand we wanted to avoid a full QWERTY keyboard layout. An input layout that has become almost ubiquitous is found in mobile phones that use a 12-key keypad either using *Multitap* or dictionary-based predictive disambiguation methods. Even though this kind of text entry was developed because of the space limitation of handheld devices, the predictive nature combined with the small number of buttons necessary makes it appealing to be used on bigger devices as well. When using this method it is possible to use a very limited amount of buttons, thereby minimizing the amount of distance a finger has to be moved when typing. On a regular basis the user will only need to use 8-10 buttons, effectively one button per finger, thereby eliminating the distance the fingers have to be moved completely.

To further enhance the input interface an adaptive layout scheme for the buttons can be used, so the keyboard adaptively positions itself and places the buttons according to the users hands and finger. This means that, as far as possible, the user will always get a keyboard that is specifically designed for their individual hand anatomy. According to Fitt's law[5], this should reduce the time it takes to hit the right key as compared to a QWERTY based keyboard, since the distance to the target is very short and the target area is moderately large.

By using this method the user does not need to concentrate on the finger placement or hitting the right buttons, as they will be positioned directly below the user's fingers. This will result in a better comfort when working with text input on a touch screen. The user should not have to pay attention to the writing technique and instead put the focus on the text being written. The ultimate goal is to achieve a sense of flow when using the keyboard, a feeling that the text is almost writing itself.

When using any type of keyboard or input interface on a vertical screen, the orientation is known automatically as it's very unlikely that the user will be standing upside down using the screen. However, when interacting with a horizontal screen the orientation of the user must be taken into account since the user can approach the table from any possible angle. A common solution is to snap the rotation of the virtual keyboard to 90 degrees rotation, so that it will always be aligned to one of the table's border. Our implementation solves this issue automatically by using the adaptive layout scheme. This basically means that when the user requests a keyboard object, the alignment of the user's fingers is analyzed and the keyboard will always spawn with the correct rotation and size.

2. Implementation

2.1. Layout

The input area displays a set of keys when the user places his hands on it, keys are positioned under the users fingers adapting to the anatomy of the user. Each key represents three or four characters in the alphabet. The character layout used for now is an alphabetical one, see Figure 1, leveraging on familiarity rather than using a character order based on a frequency analysis optimised for typing speed. The input area blocks input under the users hands to avoid palm-check like events as seen in Figure 2.

On screen keyboards usually suffer visibility problems when arms and hands block line of sight to the screen, creating a peck-and-hunt like behaviour when typing if the layout is not well known. By having visual character aid positioned above the area where the keys are, we avoid this kind

of behaviour, allowing arms and hands to remain in place, ready to type the next letter.

As fingers cannot rest on the surface without causing inputs, the area below the input buttons is disabled as well, providing a resting space for fingers.

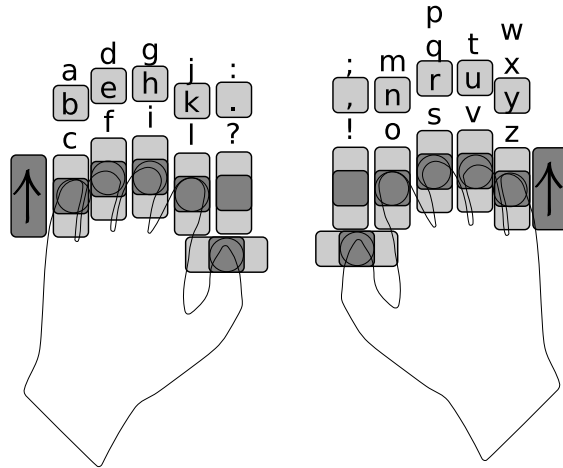


Figure 1. Placing hands on the typing area.

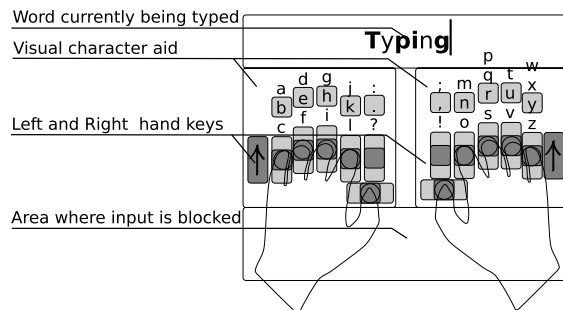


Figure 2. Block layout of the keyboard

2.2. Typing process

Keyboard instances are invoked from the text-area itself, and are visually connected to a corresponding text area. Since collaborative use is one of the biggest benefits of an interactive table or wall, multiple simultaneous users are supported by connecting keyboards to text areas, rather than having one keyboard and selecting which text area to type in with a cursor.

The basic idea is very simple, to type a word simply press the buttons containing the correct keys in sequence. Word prediction algorithms interpret the key combinations

and suggest a word that contains those keys. Typing speed is improved by adding the ability to lock any key to a character while typing by simply sliding that finger up or down before releasing the button. A locked key is always interpreted as that particular character, a non-locked key may be interpreted as any of the characters on that particular key.

If there are several possible words in the dictionary that the sequence of keys could represent, a scrolling list in which the correct word can be chosen appears. If the desired word is not in the dictionary, there is an option in the list to enter the new word into the dictionary. When this is activated, the prediction engine is turned off and characters are written as usual.

Shift keys allows typing of capital letters, a single tap on shift changes the next letter to capital, two taps on shift enables caps lock. If caps lock is enabled a single tap disables it. The double tap could be done by pressing the two different shift keys simultaneously.

By pressing space two times in a row you execute a line-break and a confirmation, the intended way of doing this is by hitting both space buttons at the same time, but two taps on either button also works. By entering a whitespace or a punctuation mark, the currently entered word is considered completed and the prediction will be reset and start a new word.

Backspace is executed by swiping one of the space keys to the left. To erase the whole word currently being typed simply press all ten fingers down at the same time. Holding all ten fingers down will reposition the keys of the keyboard.

By holding down a button for approximately one second, the corresponding number will be printed. For example, holding down the button below the left pinky, the number 1 would be printed.

If a shift key is pressed and held down, the numerical and special characters keys are shown on the opposite hand from which the shift key is being pressed. The cross use of the shift keys avoids problems when using the same finger to press shift and type certain characters. The layout of the numerical and special characters keyboard is shown in Figure 3.

2.3. Sample Implementation

For the prototype implementation we used an in-house developed framework for multiplatform multitouch application development. Figure 4 shows the sample implementation in use.

The framework is a C++ library which uses OpenGL as its rendering backend. It comes with its own input detection and tracking engine, providing multitouch support over several screens. This provides for very fast run-time speeds while at the same time offering an easy to use framework for developing applications with multitouch support.

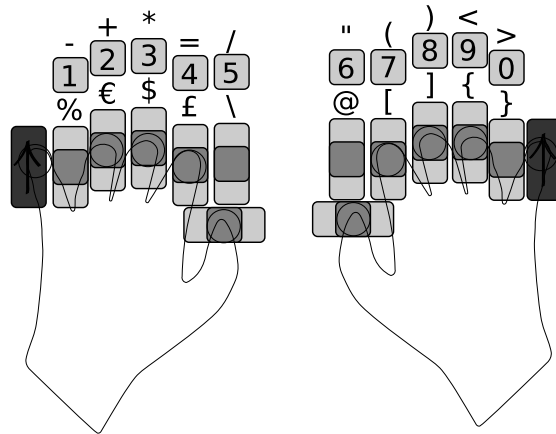


Figure 3. Numerical and special characters layout.

The keyboard widget described in this paper is a part of the framework's graphical user interface and therefore it is supposed to work in any situation where text input is required, be it entering web addresses, sending emails or writing whitepapers. It can be seen as an alternative to the normal QWERTY based keyboard.

By developing the keyboard widget as part of the framework's graphical user interface, it can be connected to any input box in any application running in the framework. The core methods for word prediction and hinting can of course be used in any application, even if they don't exist in the framework.

The keyboard widget uses some of the GUI widgets already found in the framework, such as buttons and input event handling.

For our sample implementation, a very basic application was created. It was merely acting as a big input box, waiting for keyboard events and rendering the input as a string. The application gets associated with the keyboard widget and receives the events through the framework's event handling when keys are hit.

The keyboard widget itself was done using a stretched textured OpenGL quad as the background input area. The size of this quad was determined depending on the position of the user's fingers. On top of the background quad we render the available buttons and the key layout, along with a brief description text. A gradient stripe is rendered starting from the top of the keyboard and ending where the next word will be put in the input box, to make it easier for the user to visualize the outcome of her actions. A scrollable list of predicted words is shown in the top middle, with the

most likely word highlighted.



Figure 4. Screenshot of keyboard prototype.

3. Future Work and Discussion

The character-to-key layout chosen was motivated by the demand to create a typing environment that is familiar to many users because of the use of mobile phones. Other character combinations may be more efficient and allow for higher typing speed. Finger movement when typing words known to the dictionary is rather minimal already, the sequence needed to type may include many cases of multiple presses of the same finger, and does not take into account which finger is used for frequent characters. More user experiments may be necessary to find an optimal layout. Due to the dictionary based prediction, the typing speed is strongly influenced by the size and content of the dictionary, as well as the language used and context. Punctuation characters and numbers pose a problem as they can't be predicted. Using context sensitive buttons and modifier buttons they become easily accessible.

4. Conclusion

Using the flexibility of a touchscreen we were able to create an adaptive text input interface, requiring only little non-vertical finger movement. Using text prediction methods does not require training like chorded keyboards but allows instant text entry on the precondition that the user is at least somewhat familiar with predictive disambiguation methods, commonly found in mobile phones and other handheld devices. Using visual hints and clues even a novice user should be able to enter text in a satisfactory manner. The style of entering text could be described as

more word-based typing and word-based editing as in contrast to conventional keyboards that are more cursor-based. Word based typing fits well with the use of word prediction and automatic completion, editing and handling text on a word by word basis is well suited in case of several separate text areas having focus at the same time.

References

- [1] *Frogpad*. <http://www.frogpad.com>.
- [2] *Veyboard*. <http://www.veyboard.nl>.
- [3] W. Buxton, R. Hill, and P. Rowley. Issues and techniques in touch-sensitive tablet input. *SIGGRAPH Comput. Graph.*, 19(3):215–224, 1985.
- [4] A. Dvorak, N. L. Merrick, W. L. Dealey, and G. C. Ford. *Typewriting Behaviour*. American Book Company, 1936.
- [5] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.
- [6] U. Hinrichs, M. Hancock, C. Collins, and S. Carpendale. Examination of text-entry methods for tabletop displays. In *IEEE International Workshop on Horizontal Interactive Human-Computer Systems (Tabletop'07)*, pages 105–112, 2007.
- [7] Infogrip. *BAT Keyboard*. <http://www.infogrip.com>.
- [8] C. L. Sholes. US patent: 207,559, 1878.
- [9] M. Willner. *AlphaGrip*. <http://www.alphagrips.com>.
- [10] S. Zhai, M. Hunter, and B. A. Smith. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 119–128, New York, NY, USA, 2000. ACM.