Collaborative Smart Virtual Keyboard with Word Predicting Function

Chau Thai Truong[1], Duy-Hung Nguyen-Huynh[1], Minh-Triet Tran[1], and Anh-Duc Duong[2]

[1] University of Science, VNU-HCM, Vietnam
[2] University of Information Technology, VNU-HCM, Vietnam
{0912034,0912202}@student.hcmus.edu.vn, tmtriet@fit.hcmus.edu.vn, ducda@uit.edu.vn

Abstract. The authors propose a table-top with virtual keyboards for multi-users to work in a collaborative environment. The proposed system has two main modules: a system for virtual keyboards with touch event detection from depth data of a Kinect and a word predicting module based on the idea of Hidden Markov Model and Trie data structure. The system can replace physical keyboards, improve the accuracy of a virtual keyboard, and increase the typing speed of users. Our experimental results show that our system archives an accuracy of 94.416% with the virtual keyboard, saves 11-22% of keystrokes, and corrects 89.02% of typing mistakes.

Keywords: table top, virtual keyboard, word prediction, 3D interaction.

1 Introduction

Human-computer interaction (HCI) plays an important role in the evolution of computing society. Researches in the field of interaction between users and computers aim to enhance the comfort, ergonomics, and portability as well as to save time for users. To efficiently assist users in various activities in daily life, HCI not only provides useful utilities for single users to interact with computing systems but also collaboration environment for multiple users to work together.

Text-based data entry is one of the most common tasks in most applications. Thus different types of keyboards have been developed to enhance the usefulness and comfort for users. Inspired by the augmenting interactive table system with mice and keyboards of Hartmann [6], we propose a collaborative smart virtual keyboard system with word predicting function. Our proposed system uses a regular projector to project over an arbitrary relatively-flat surface the images of multiple virtual keyboards and a Kinect to capture depth information for touch event detection. Multiple users can now work together on a single large area, e.g. a desk, using only virtual devices. Furthermore, the layout, language, and size of any virtual keyboards can be visually customized to save extra cost for real physical devices to meet users' various needs.

Besides, the word predicting function is designed to support users to increase the typing speed and correct typing mistakes quickly. We use a dictionary stored in a prefix tree (Trie [12]) to save memory and to increase processing speed. The dictionary contains both words and their frequencies in the same category, e.g. sports, education, politics, etc. There are two parts of this function: predicting the words that a user intends to type and correcting words that a user mistyped. The operations of these parts are based on word frequencies in a trained dictionary and the prefix characters that a user has just typed. Experimental results show that our virtual keyboard system achieves the accuracy of 94.42% in keystrokes detection, the word prediction can save 10-20% keystrokes, and the word correction can eliminate up to 80-90% mistakes when users type text data with the same topic as that of the used dictionary.

The content of the paper is as follows. In Section 2, the authors briefly review the approaches in HCI, especially in developing table top and virtual keyboards. Our proposed system and experimental results are presented in Section 3 and 4 respectively. Conclusions are discussed in Section 5.

2 Related Work

Interactive surfaces of different materials, technologies, and sizes have become popular means of interaction between users with computing devices and systems. While touch screens are suitable for small and medium sized devices, such as tablets, mobile devices, ATM machines, etc., tabletops are more applicable for economic and large sized interactive surfaces.

Different approaches have been proposed to develop various models of tabletops, such as using laser to detect and localize touch events [1], using single or multiple traditional cameras to detect hands and fingers' actions with or without markers/gloves [2], using sensors of multi-touch surfaces to perform interaction [10]. With the appearance of depth-sensing technology and depth cameras such as Kinects, vision-based methods for tabletop interaction can take a further step with extra useful information of depth data. Wilson uses depth data as touch sensor and can determine interaction points by using a depth camera [3]. In this paper, we follow this new trend to develop our smart virtual keyboard with depth data captured from a Kinect.

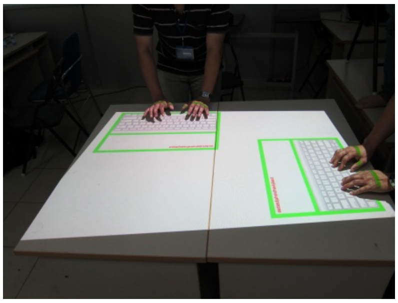
Among with tabletops, applications that support multi-user work or collaboration are also a topic of concern. A multi-user web browser system is proposed to support multiple people to search and to watch the same webpages simultaneously [8]. Klinkhammer et. al. develop a system that enables many people to share information, data while working on the same interactive surface [9]. WeSearch system is proposed to enable a group of up to four members to use a web browser at the same time on a tabletop [7]. Especially, Hartmann et. al. propose eight interaction methods that are used in a working desk that supports interactions with real keyboards and mice [6]. This motivates our inspiration to develop our system. However, we take a further step. Our system does not require any physical input devices, e.g. keyboards, mice. Users interact through virtual devices projected by a projector over any relatively-flat surface and touch events are detected from depth data captured from a Kinect. Therefore, keyboard layouts and languages can be customized easily.

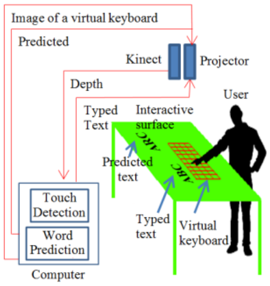
Beside applications of tabletops and supports for collaboration, the efficiency in typing on virtual keyboards is also a practical demand. Findlater et. al. propose a method to evaluate an adaptive personalized virtual keyboard layout to improve typing on a touch-screen [4]. Wigdor et. al. examine the hand patterns in touch-typing on a flat surface to suggest a new design for touch screen keyboards [5]. In this paper, our approach is to perform word prediction and correction using word frequencies in a dictionary containing words in the same category. This approach has many advantages. Different dictionaries can be built by training large data from online articles of the same topic from the Internet. The frequencies of words can be combined with the neighborhood keys to correct mistyped words.

3 Proposed System

This section shows the main components and operations of our smart virtual keyboard with word prediction and correction. The overview of our system is presented in Sec. 3.1. The operations of Kinect are described in Sec.3.2. Finally, Sec.3.3 shows the operations of word prediction function including the data structure to store a dictionary and our proposed methods to predict words from prefixes.

3.1 Overview of the System


(a) Overview



(b) Main components

Fig. 1. Overview and architecture of the system

Figure 1 illustrates the overview and main components of our proposed system. A user interacts with a virtual keyboard displayed on an non-touch interactive surface. In our system, the interactive surface and the Kinect are in fixed positions during an interaction process. A projector is used to project images of multiple virtual keyboards, texts, and multimedia information on demand. A single computer is used to processed all keystrokes from multiple users and other functions. These functions are divided into two subcomponents: touch detection and word prediction.

Touch detection: A Kinect device is used to continuously capture depth images on the interactive surface. The computer receives depth information to detect touched points on virtual keyboards and generates appropriate keyboard events. Finally, the projector shows the image of virtual keyboards and typed texts. The role of Kinect in this component is described in detail in Sec.3.2.

Word prediction: this subcomponent is designed to enhance the accuracy of touch detection and to increase typing speed. When a prefix is typed, the system suggests appropriate words to the user. The use of data structure to store a dictionary and our proposed methods to predict words are presented in Sec.3.3.

3.2 Touch Event Detection with a Kinect

In this part, we present the use of a Kinect in touch detection. First, a background image is trained to estimate the main interaction plane. It should be noticed that the interaction plane is not required to be perfectly flat. Any fixed, relatively flat surface can be used as the interaction plane.

Then, when users start using the system, query images of depth data are computed periodically to detect touched points. However, when computing the background image and query images, due to the instability of Kinect depth data, the depth information in an image should be denoised on consecutive frames by a median filter.

Background estimation: A background image contains only depth data of the interactive surface. Therefore, the computation of the background image must be done on many continuous frames at the beginning of an interaction process. Let N_1 be the number of depth frames used in training the background, $I(x, y)$ be the value of a pixel (x, y) in an image I, I_B be the background image and I_{B_1}, I_{B_2}, …, $I_{B_{N_1}}$ be the frames that are used to compute I_B. We estimate the background image with the following formula: $I_B(x,y) = \text{median}\{I_{B_i}(x,y); i \in [1, N_1]\}$

Computing query images: Suppose the process starts at the s-th frame. Let N_2 be the number of consecutive depth frames used to compute a query image. The i-th query image (i >0), denoted by I_Q is computed from the frame $s + (i-1) \cdot N_2$ to the frame $s + i \cdot N_2 - 1$ by the following formula:

$$I_Q(x,y) = \text{median}\{I_{Q_i}(x,y); i \in [s + (i-1) \cdot N_2, s + i \cdot N_2 - 1]\}$$

Finding the points that are near the interactive surface: Depending on its real distance D_P to Kinect, a point P in a depth image is classified into one of the three classes: inside, near, and far point. The depth value at a pixel is the distance from Kinect to the plane that contains the pixel and is perpendicular to the viewing direction of Kinect. Let D' be the distance from Kinect to the plane containing P. $D' > D_P$ means that P is inside the interactive surface. This is certainly a noise data because Kinect and the surface are assumed to be stable. Therefore, if $D' > D_P$, we reset $D' = D_P$ to eliminate noise. Finally, we have $D' \leq D_P$, $\forall P \in I_Q$.

Let $\Delta_P = D_P - D'$, $\Delta_{\min} = \min\{\Delta_P \mid \forall P \in I_Q\}$ and $\Delta_{\max} = \max\{\Delta_P \mid \forall P \in I_Q\}$.

We define the set of near points $C = \{P \in I_Q \mid \Delta_{min} \leq \Delta_P \leq \Delta_{max}\}$. With the same value of $\Delta_{\max}$, the smaller value of $\Delta_{\min}$ causes more noise data between $\Delta_{\min}$ and $\Delta_{\max}$. Otherwise, the larger value of $\Delta_{\min}$ makes the touch events generated earlier. With the same value of $\Delta_{\min}$, if $\Delta_{\max}$ is larger, the noise between $\Delta_{\min}$ and $\Delta_{\max}$ is higher. Otherwise, the time interval of a touch event is shorter and a touch event is more likely to be missed. The experiment to choose $\Delta_{\min}$ and $\Delta_{\max}$ is presented in Sec.4.1.

Touch detection: For a set T containing connected near points, if $|T| > \min_{\text{Area}}$, a touch event is triggered at the position (x_{touch}, y_{touch}) where:

$$x_{touch} = \frac{\sum_{P \in T} P.x}{|T|} \quad \text{and} \quad y_{touch} = \frac{\sum_{P \in T} P.y}{|T|}$$

3.3 Word Prediction and Correction

When a user types a prefix, i.e. a sequence of initial characters of a word, the system suggests a number of words with highest probabilities. If the user accepts, he or she can save time to type that word. This feature is also used to correct typing mistakes. These mistakes can be typing mistakes of users and touch event mistakes caused by the system. When a prefix is typed, the system corrects and suggests new words based on the neighborhood characters of the typed characters. Using this feature, users can save time for both typing and correction.

Both functions use dictionaries to perform. The dictionary contains frequencies of words in the same topic with the typing text. Together with the dictionary, user profiles can also be utilized to enhance the accuracy of these functions. Therefore, the more often a user uses the system, the more efficient the system becomes.

Prefix tree (Trie): Trie data structure is first proposed by Fredkin [12]. Trie is a tree $G = (V, E)$ with a root r, a vertex set V, and an directed edge set $E = (V, V)$. Each edge $(u, v) \in E$ contains a character $c_{(u,v)}$. For a node x and its children nodes y_i and y_j, we have $c_{(x,y_i)} \neq c_{(x,y_j)}$ for $y_i \neq y_j$. Figure 2 illustrates an example of a trie.

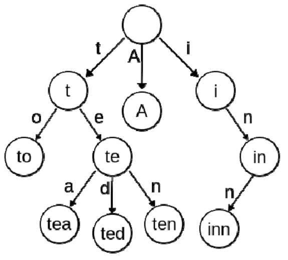


Fig. 2. Trie built from strings “A”, “to”, “tea”, “ted”, “ten”, “i”, “in” and “inn”

The efficiency of using Trie in dictionary: Let S be the set of words in a dictionary, len(x) be the length of string x and $\overline{|u_l|}$ be the average length of all words in the dictionary, we consider the following operations.

- Storing a set of string $T = \{v_i\}$ with the same prefix u.

Assume that we need one memory unit to store a character, the number of memory units that can be saved when using Trie to store T is as follow:

Total number of characters in T – number of memory unit to store using Trie

$$= \left(\sum_{v_i \in T} \left(len(v_i) \right) \right) - \left(len(u) + \sum_{v_i \in T} \left(len(v_i) - len(u) \right) \right)$$
$$= (|T| - 1).\, len(u)$$

- Finding a prefix $v = v_1 v_2 \ldots v_k$ in S

When not using Trie, the system iterates through every word or binary searches the wordlist to find v. The complexity of these operations are $O\left(|S|.\overline{|u_i|}\right)$ and $O\left(\log_2 |S|\,.\,\overline{|u_i|}\right)$ respectively. When using Trie, the system starts from root r and visits child nodes until v is found or cannot visit more. The complexity is then $O\left(\overline{|u_i|}\right)$. Therefore, using Trie can reduce searching time of a prefix.

- Adding a new word $v = v_1 v_2 \ldots v_k$ to S

When not using Trie, the complexity to insert v at position i in alphabetical order is $O\left(|S|\overline{|u_i|}\right)$. When using Trie, the system finds a prefix of v. Then, it iterates through every remaining position to add new child nodes. The complexity of this is $O\left(\overline{|u_i|}\right)$. Therefore, the adding time is faster.

In conclusion, the use of Trie to store dictionaries has three advantages: smaller capacity, faster time to search and add words. That is the reason of our choice to use Trie to perform main functions for word prediction and correction in our system.

Hidden Markov Model (HMM) [11] has the following elements:

- A finite number, denoted by N, of states in the model. At each clock time, t, a new state is entered based upon a transition probability distribution which depends on the previous state.
- After each transition is made, an observation output symbol is produced according to a probability distribution which depends on the current state. This probability distribution is held fixed for the state regardless of when and how the state is entered. There are thus N such observation probability distributions which, of course, represent random variables or stochastic processes.

All of those elements are formally defined in the following:

T = length of the observation sequence
N = number of states in the model
M = number of observations
$Q = \{q_1, q_2, \ldots, q_N\}$, states
$V = \{V_1, V_2, \ldots, V_M\}$, observations
$A = \{a_{ij}\}$, $a_{ij} = \Pr(q_j \text{ at } t+1 \mid q_i \text{ at } t)$, state transition probability distribution
$B = \{b_j(k)\}$, $b_j(k) = \Pr(v_k \text{ at } t \mid q_j \text{ at } t)$, observation probability distribution in state j
$\pi = \{\pi_i\}$, $\pi_i = \Pr(q_i \text{ at } t = 1)$, initial state distribution

Prediction based on frequencies of prefixes: When a user types a prefix s, a complete word u_i that has the prefix s and has the highest probability is suggested. Let sum_s be the frequency of s. The probability of u_i is defined as follow:

$$P(u_i|s) = \frac{sum_{u_i}}{sum_s}$$

Prediction based on finger position and keyboard layout: Let $s = s_1s_2...s_m$ be the typed prefix, the system suggests the list of prefixes $u = u_1u_2...u_m$ that has the highest probability. The suggestions of prefixes u depends on two criteria:

- The frequency of u in a dictionary.
- The neighborhood characters of s_i (for $i \le m$) is the set $H(s_i)$ of characters b such that the keys of s_i and b are identical or share a common part in key borders. For example, in a standard US keyboard, the keys E, R, F, C, X ,S are neighborhoods of D. These keys b have the nearest distances from their centers to the center of key s_i. Therefore, we choose this set as a criterion. Let $P_t(s_i, b)$ be the probability that a user actually wants to type b instead of s_i. We assign $P_t(s_i, b) = r\%$ when $s_i \neq b$. $P_t(s_i, s_i)$ changes depending on r and $|H(s_i)|$.

Let $\sigma(u, i) = u_1u_2...u_i$. The application of hidden Markov model is as follows:

- The states are the prefixes $\sigma(u, i)$ $(i \le m)$ that exist in the dictionary and u_i is a neighborhood of s_i.
- The observations of the current state $\sigma(u, i)$ $(i < \text{m})$ is the set $H(s_{i+1})$.
- Transition probability distribution A: if $i < j$, $\sigma(u, i)$ is a prefix of $\sigma(u, j)$. We have:
$$A_{\sigma(u,i),\sigma(u,j)} = \frac{sum_{\sigma(u,j)}}{sum_{\sigma(u,i)}}$$
- Observation probability distribution B: given the state $u' = \sigma(u, i)$ $(i < m)$ and $\text{H}(s_{i+1})$ are the observations. We have: $B_{u',c} = P_t(s_{i+1}, c)$ where $c \in H(s_{i+1})$
- Initial state distribution π_u: Let P be the set of all prefix in the dictionary, we have:
$$\pi_u = \frac{sum_u}{\sum_{t \in P} sum_t}$$
- The probability of an observation series $u_1u_2...u_m$ is:

  + For $m = 1$, we have $P(u_1) = P\left(\emptyset \rightarrow u_1\right) * P_t(s_1, u_1) = A_{\emptyset, u_1} * B_{\emptyset, u_1}$
  $= \pi_{u_1} * P_t(s_1, u_1)$ where $\emptyset$ is the empty string.

  + For $m > 1$, we have:
$$P(u_1u_2 \ldots u_m) = \prod_{i=1}^{m-1} P\left(\sigma(u, i-1) \rightarrow \sigma(u, i)\right) * P\left(\sigma(u, i-1) \rightarrow u_i\right)$$
$$= \prod_{i=1}^{m} A_{\sigma(u,i-1),\sigma(u,i)} * B_{\sigma(u,i-1),u_i} = \prod_{i=1}^{m} \frac{sum_{\sigma(u,i)}}{sum_{\sigma(u,i-1)}} * P_t(s_i, u_i)$$

4 Experimental Results

In this section we present four experiments. The experiment in Sec.4.1 determines the optimum values of $\Delta_{\min}$ and $\Delta_{\max}$ for touch detection. Sec.4.2 measures the accuracy of our virtual keyboard with the chosen values of $\Delta_{\min}$ and $\Delta_{\max}$ in Sec.4.1. The experiments in Sec.4.3 and Sec.4.4 are to evaluate the efficiency of word prediction based on word frequencies and the accuracy of word correction based on both word frequencies and neighborhood characters. These experiments are performed on the system using CPU core i7 2.2Ghz, 6GB RAM.

4.1 Finding Optimal Values of Δ_{min} and Δ_{max}

This experiment is to find the optimum values of Δ_{min} and Δ_{max} to achieve the highest precision in touch detection. For experiments in this section and Sec.4.2, Kinect is kept at a distance of 0.8m from the surface. 26 square regions with the same size of 1.5cm × 1.5cm are selected to be the virtual keys corresponding to 26 alphabet characters from 'a' to 'z'. The data consists of 1800 depth frames in 640×480 resolution. The first 300 frames are used to train the (3D) background. Touching action is performed at arbitrary regions in the remaining frames. The accuracy is determined as the proportion between the number of error frames and the total number of used frames. From the experiment, we choose Δ_{min} = 10mm and Δ_{max} = 13mm.

4.2 Estimating the Accuracy of Virtual Keyboard

The purpose of this experiment is to measure the accuracy of our virtual keyboard with the optimum parameters Δ_{min} and Δ_{max} determined in Sec.4.1. The data consists of 10 strings with 154 characters in total. For each string, we type 5 times to the virtual keyboard. The accuracy of each typing time is the percentage of correct touch events. After typing all of 10 strings, we calculate the average accuracy of five times over all strings. Table 1 shows the results.

Table 1. Accuracy of virtual keyboard in 5 typing times

Test	Result
1	94.156%
2	94.805%
3	92.208%
4	94.805%
5	96.104%
Average	94.416%

From Table 1, we conclude that our system archives the average accuracy of 94.416% in touch detection of our virtual keyboard.

4.3 Estimating the Efficiency of Word Prediction Based on Word Frequency

In this experiment, the efficiency of word prediction is measured by the percentage of keystrokes that users can save when they type texts in the same topic as that of the current dictionary. We measure two methods: offline –word frequencies are fixed; and online – the word frequencies are updated gradually after each time a user types a word. A dictionary containing words and their frequencies is built from 100 articles in a single topic (e.g. business, technology, health, etc). 100 other articles in the same topic are selected to test the system. Each article in the training set and test set has 2000-6000 words. The result is illustrated in Figure 3.

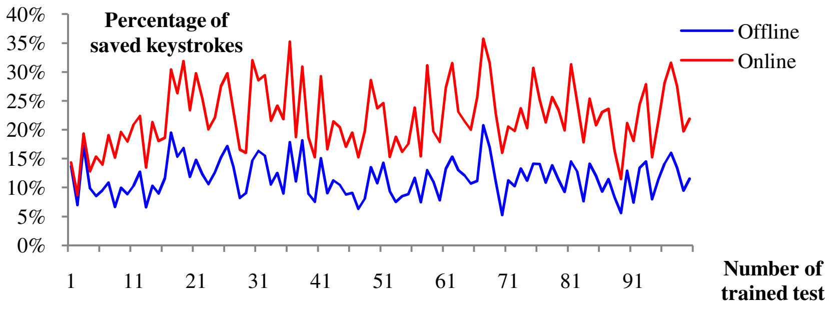


Fig. 3. Percentage of saved keystrokes when applying two methods (online and offline) of word prediction based on word frequencies

From the experimental results, we conclude that with the same text and topic, when the number of typed words is small, the accuracy of two methods do not have much difference. When the number of words increases, the efficiency of online method becomes higher. On average, users can save 11.61% keystrokes with offline method and 22.19% with online method.

4.4 Estimating the Accuracy of Word Correction Based on Word Frequency and Neighborhood Characters

This experiment aims to measure the accuracy of system to correct typing mistakes of users based on the frequencies of words in the same specific topic and the neighborhoods of the typed characters. Let P_{error} be the probability that a character is mistyped to a neighborhood character, we measure the percentage P_{fix} of corrected mistakes after using our method. A dictionary are built from 40 articles in a single topic and 40 other articles are used as testcases. Each article has 2000-6000 words and the total number of characters is 85627. Recall the Sec.3.3, we choose $P_t(a, b) = 5\%$ if $a \neq b$. The result of this experiment is in Figure 4.

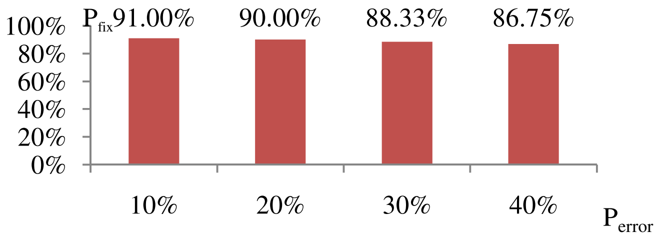


Fig. 4. Result of the efficiency of word correction using word frequencies and neighborhood characters

From the experimental result, we conclude that using word frequencies and neighborhood characters to correct typing mistakes can solve 89.02% on average of these mistakes when typing in the same topic. This percentage decreases when the initial error rate increases.

5 Conclusions

We propose to develop a table top with smart virtual keyboards for multiple users to work in a collaborative environment. To eliminate noise in Kinect, we use a median filter in both background training and touch event detection. The virtual keyboard system can be apply over any relatively flat area with the accuracy of 94.416%. The position, layout, and language of each virtual keyboard can be customized easily.

Besides, the accuracy of our virtual keyboard is enhanced by word predicting function that learns from both train data and user profile to suggest and correct the typed words quickly. This feature helps users to save up to 11-22% of keystrokes and correct 89.02% mistakes in typing documents in the same topic with the dictionary.

We are currently develop further features for our tabletop environment and improve the system to apply on any types of surfaces (curve, sphere, etc). Besides various dictionaries are being built for different topics and categories, even for programming languages (e.g. C++, Pascal, C#).

References

1. Tobias Schwirten, L.: Radar Touch, `http://www.radar-touch.com/`
2. Marquardt, N., Kiemer, J., Greenberg, S.: What caused that touch? expressive interaction with a surface through duciary-tagged gloves. In: ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, pp. 139–142. ACM (2010)
3. Wilson, A.D.: Using a depth camera as a touch sensor. In: ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, pp. 69–72. ACM (2010)
4. Findlater, L., Wobbrock, J.: Personalized input: improving ten-finger touchscreen typing through automatic adaptation. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2012, pp. 815–824. ACM (2012)
5. Findlater, L., Wobbrock, J.O., Wigdor, D.: Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces. In: Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, pp. 2453–2462 (2011)
6. Hartmann, B., Morris, M.R., Benko, H., Wilson, A.D.: Augmenting interactive tables with mice & keyboards. In: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, UIST 2009, pp. 149–152. ACM (2009)
7. Morris, M.R., Lombardo, J., Wigdor, D.: Wesearch: supporting collaborative search and sensemaking on a tabletop display. In: Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, CSCW 2010, pp. 401–410. ACM (2010)
8. Tuddenham, P., Davies, I., Robinson, P.: Websurface: An interface for co-located col-laborative information gathering. In: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces, ITS 2009, pp. 181–188. ACM (2009)
9. Klinkhammer, D., Nitsche, M., Specht, M., Reiterer, H.: Adaptive personal territories for co-located tabletop interaction in a museum setting. In: Proc. of the ACM International Conference on Interactive Tabletops and Surfaces, ITS 2011, pp. 107–110. ACM (2011)
10. Dippon, A., Echtler, F., Klinker, G.: Multi-touch Table as Conventional Input Device. In: Stephanidis, C. (ed.) Posters, Part II, HCII 2011. CCIS, vol. 174, pp. 237–241. Springer, Heidelberg (2011)
11. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE, 257–286 (1989)
12. Fredkin, E.: Trie Memory. CACM 3(9), 490–499 (1960)