DUCK: a deDUCtive soft Keyboard for visually impaired users

Mathieu RAYNAL[a,1] and Philippe ROUSSILLE[a]
[a] *University of Toulouse – IRIT*
Toulouse, France

Abstract. Touch screens rapidly and significantly replace physical keyboards on mobile devices. Hence, text entry is now dependent on soft keyboards that are widely used by sighted people, but raise accessibility issues for visually impaired users. These users rely on tactile exploration with vocal feedback of the whole screen for entering text, which is time consuming. We designed a software keyboard that aims reducing tactile exploration and speeding up text entry for VI users. It relies on the selection of the first letter of a word and rapid and inaccurate typing of the remaining letters. It then proposes a list of words having the same first letter and a similar total distance between letters. The evaluation with twelve VI users showed that this keyboard is very efficient for words larger than five characters. It also helps preventing certain typing errors.

Keywords. Text-input, visually impaired users, touch screen, soft keyboard

1. Introduction

Although initial use was limited to calling and sending small text messages (SMS), mobile devices have evolved, and now support the use of different services, including web applications and social networks. The emergence of tablets also promoted, at the expense of desktop computers, the use of mobile applications for most common office tasks (email, note taking, presentation support, etc.). At the same time, physical keyboards have almost disappeared from mobile devices, allowing additional space for larger screens. Mobile devices now integrate large tactile screens with multitouch interaction as the main input interaction technique, and software keyboards for text input. The layout of these virtual keyboards is very flexible, and provides advanced functions for improving text input (prediction system, deduction, completion, etc.)

However, touch-based mobile devices without physical keyboards cause accessibility issues. Visually impaired (VI) users largely rely on physical keyboards to type text but also to select commands within the applications. On mobile devices with touch screens, the lack of tactile cues (such as the presence of physical keys) makes the task very difficult, if possible. On such devices, all the interactions are spatially related to items displayed on the screen. VI people are hence forced to explore the whole layout in order to find the elements with which they want to interact.

In the case of text entry, the issue is amplified because exploration of the whole display is required for each character to be entered. Indeed, the most common text input method for the VI is based on the finger exploration of the screen with simultaneous vocalization through voice synthesis of the key being under the finger (e.g. VoiceOver from Apple, Inc., or TalkBack from Google Inc.). Therefore VI users must constantly

[1] Mathieu RAYNAL – mathieu.raynal@irit.fr

explore the screen to locate a specific key. Although the voice feedback can be improved with vibration cues, this process, called "painful exploration" [1], is slow and requires much effort.

We designed and evaluated a software “deductive” keyboard called DUCK (deDUCtive Keyboard) aimed at decreasing painful exploration. DUCK is based on the observation that a great majority of VI users know the spatial layout of their physical keyboard. With DUCK the user must explore the software keyboard layout once, at the beginning of a word input. Then, he just has to type as many times as the number of letters in the expected word without focusing on spatial accuracy. At the end of each word, a deduction system proposes a list of probable words according to the user's taps. We made the hypotheses that this keyboard would save exploration time and increase text input speed.

In the following sections, we make an overview of the main existing input methods specific to VI users, on mobile device. Then we describe the principles of the DUCK keyboard as well as the algorithm that allows the keyboard to make words deduction. We also report on the experimental comparison of DUCK with classical text entry methods. This evaluation was done with twelve VI users. We finally discuss the results and the improvements that we are currently designing in order to improve DUCK efficiency and user satisfaction.

2. Related works

The biggest challenge a VI user is facing when entering text on a mobile device is localizing the different characters of the software keyboard. The most common solution consists in vocalizing the key being under the user's finger through voice synthesis (see e.g. Apple VoiceOver or Google TalkBack). Characters are entered when the user lifts his finger from the screen. The major drawback is the need for the user to browse the whole keyboard with his finger in order to find the expected character. Even with a good knowledge of the spatial layout, the user must acknowledge the characters' position for each input.

Although it is functional, this solution is time consuming. Then different virtual keyboards have been designed specifically for VI users. They are often based on the Braille alphabet where letters are coded in a matrix with three rows and two columns. Each character corresponds to a code involving the six dots differently. BrailleType [5], for instance, relies on dividing the screen into six equal-sized cells (three lines and two columns), each one corresponding to a Braille dot. The user must tap within the areas to activate all the dots needed to select a character. The system validates the entry after a certain time. The TypeInBraille [4] system is also based on Braille. In that case, the selection of characters in not based on the screen division but on multitap interaction. The user must carry out three successive multitaps on the screen to select a character. The three multitaps correspond to the three lines of the Braille matrix. For each line, the user must select one of the four possible tapping: activate the left dot, the right dot, both dots, or no dot for the current row.

Although non-visual text input based on Braille is functional, it is important to note that "*fewer than 10 percent of the 1.3 million people who are legally blind in the United States are Braille readers*". Therefore, these systems are limited to a small part of the VI users. The time needed to learn a new keyboard layout, and to be efficient with it, is extremely long [3]. Learning Braille can take up to four months for uncontracted Braille or two years for contracted braille. Most of the users will thus not make that effort in order to use a keyboard on a mobile device.

We aimed to design a keyboard that improves text input speed for VI users, but that does not require a specific learning before use. Therefore, we relied on the regular language specific layout, which is known by all the VI users who use a physical keyboard. The challenge was speeding up text input speed without producing too many errors.

3. DUCK Keyboard

3.1. Design principles

We based the DUCK design on different observations and predictions that we made during brainstorming and preliminary design tests with VI users. Firstly, most of the VI users have a previous knowledge about the layout of their physical keyboard. Secondly, the systematic vocal feedback provided after each character input is time consuming and sometimes useless. Then, it could be quicker to type a whole word before providing this feedback. However, a short feedback that acknowledges each touch event is essential. Obviously, a correction function and corresponding non-visual interaction technique are mandatory.

DUCK is a software keyboard displayed on the maximum surface of the screen. In order not to lose prior knowledge of the spatial layout, the location of the characters and most important keys is preserved in a specific language.

When entering a word, the user first drags one finger onto the keyboard. During this exploration, the user ears a vocal feedback corresponding to the touched character. For validating the first letter, he/she must release the finger. In order to finish entering a full word, he must hit the touchscreen as many times as the number of remaining letters in that word. The only instruction is to type those letters according to the prior knowledge of the spatial layout of the keyboard, without focusing on spatial accuracy. There is no more vocal feedback for these remaining letters. A short and simple audio feedback (beep) is provided for each tap. Once the user terminates entering the right amount of letters, he must validate the input with a two fingers press on the touchscreen. DUCK then calculates the Euclidian distance between the successive hits, and looks for the words having the same first letter and a similar total distance. The four most probable words are provided within a list. The first word is read and spelled. The user can either select the current word with another two fingers press, or cycle through the list with a one finger press. In case of error, the user can cancel the whole word input with a two fingers swipe to the left.

As previously mentioned the user must first explore the keyboard and listen to vocal feedback in order to select the first letter of the targeted word. We drastically reduce the number of solutions by doing so.

3.2. Deductive algorithm

In order to describe the algorithm, we assume a case where the user wants to type a word of N characters beginning with the letter L. Let E be the set of four words available in the dictionary, starting with L, and containing N letters. As the user typed a sequence of N hits, $h=[h_i]$, where $h_i = (x_i,y_i)$ are the coordinates of the hit. Similarly, every word w in E can be described as a sequence of characters c_i. Since the corresponding key and its position on the layout are known, let $position(c_i)=(x'_i,y'_i)$ be the coordinates of the center of the key corresponding to the character marked c_i. To compute the best possible word for the sequence h, the algorithm computes the distance between the sequence of hits h and each candidate w of E:

$$D(h, w) = \sum_{i=1}^{N} \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2}$$

Finally, DUCK returns the best four matches, which are the words having the same first letter and minimizing the distance between the sequence of hits h and this word.

Of course, the dictionary file can be changed. It is a file describing the possible words, and the position of the keys on the actual layout. We provide a tool to create dictionaries from a list of words and a specific layout.

3.3. Gesture-based interactions

In addition to typing letters and words, the user can move one finger up and down to select either the symbols or punctuations display. There is no prediction or deduction for the symbols or punctuations. In such a case, DUCK turns back to a standard "search and validate" keyboard. The user must move the finger onto the screen until the character is located. Releasing the finger validates the input of the character.

4. Material and methods

Device: We did a comparative study between DUCK and a “VoiceOver like” keyboard named VODKA. With VODKA, each key is spoken when the user passes his finger over. The last chosen key is validated when the finger is lifted. Users can erase the last typed letter by swiping two fingers leftwards. For both keyboards, it is possible to enter a space character by sliding two fingers to the right. Spaces are however automatically added with DUCK as soon as the word is validated. The corpus of words used in this experiment contained 336,531 words extracted from the Gutenberg French dictionary. These words were used by the deduction algorithm to generate the list of potential words. Both keyboards were based on the same layout. It was a subset of the AZERTY layout including alphabetic keys only, arranged over three rows of ten keys with the last four keys of the third row (where the punctuation symbols usually are) left blank. It is possible to easily change the layout with an XML file describing the position of each key.

The experiment was run on a Samsung Galaxy SIII, using the 4.1 Android. It has a Quad-core 1.4 GHz Cortex-A9 for CPU, and 1 GB of RAM. The touchscreen had a size of 136.6 × 70.6 mm with a resolution of 720 × 1280 pixels. Both keyboards were coded in Java for a direct use on the smartphone. The voice synthesis was rendered by the IVONA text-to-speech engine, which we selected for clarity and natural sounding.

Participants: We recruited twelve participants, six males and six females (average age 32.4 years, standard deviation 16.2 years). Ten of them were legally blind with no functional perception. Two users had light perception only. None of them had sufficient visual capacities to perceive the general shape of the virtual keyboard. All the participants declared having a good knowledge of the AZERTY layout. Eight persons use it every day for their own usage. The four remaining are experts in the AZERTY keyboard because they teach how to use it to VI students. Among our users, two owned and used text-entry input on a tactile device on a daily basis.

The experiment was in accordance with the principles of the Declaration of Helsinki, and the French Data protection act. The study was also approved by a local ethics committee. Each user signed a written consent form before starting the experiment.

Procedure: The task consisted in typing a short phrase after dictation by the text-to-speech synthesis (TTS), using either DUCK or VODKA. The subjects were asked to type as fast as possible, while minimizing the number of errors. Half of the users started with the DUCK session, while the others started with the VODKA session. Within each session, the user had to type eight short sentences after dictation. Each participant had the same sixteen sentences to type, but they were allocated with a different pseudo-random order. The sixteen sentences were simple in order to ease the memorization and typing, as well as minimizing the number of typos. They included four to seven words chosen among the most frequent words of the French language only. We also made sure that the letter frequency observed in these sentences was the same as the one observed for the French language. We made sure that these sentences were short and simple enough so that the users could easily remember them and didn't have to listen to the sentences multiple times.

Each session started with a brief familiarization phase for the current keyboard. The subjects were taught how to input letters, short words, long words, and finally sentences. After the familiarization, the subject started the experiment *per se*. At any time, he was allowed to replay the current phrase by sliding two fingers up. He could listening to his actual input by sliding two fingers down, as well as erasing the last input (word for DUCK, letter for VODKA) by sliding two fingers left. The users were not constrained holding the phone in a particular way or using specific fingers. In addition, they were free to correct the mistakes or not before validating the input.

All the interactions, including user events (finger move, finger press, and finger release), text entry (typed characters and words), and deduction lists were recorded in a XML file. All the questionnaires were passed orally.

5. Results

We recorded the text input of twelve different users typing sixteen sentences of four to seven words (total of 192 sentences and 1032 words) with the two keywords. For each subject, the whole experiment lasted from 50 to 165 minutes, including a 15 minutes break between the two sessions.

The following sections present the quantitative and qualitative results, including inferential statistics. The significance level (α) of the statistical tests was always set to 0.05. Bonferroni corrections were systematically applied for multiple comparisons.

5.1. Sentence and word input speeds

We computed the average CPS on all the typed sentences by all the users. Firstly, we computed the text entry speed per sentence. Here, the length of the sentence (including the space characters between words and at the end of a sentence) was divided by the time (in seconds) needed to enter that sentence. This speed is computed in characters per second (cps). The average sentence input speeds were 0.37 cps (SD=0.12) for VODKA and 0.38 cps (SD=0.15) for DUCK, the median being 0.33 CPS for both. As the speed distributions were not normal (Shapiro-Wilk, $W = 0.95$, $p<0.01$), we used a Wilcoxon test to compare them, which showed that the speeds were not statistically different ($W = 2115$, $p = 0.819$).

However, this time took into account the corrections made by the user during his input. In order to assess the pertinence of our deduction system, we choose to take only the words that were correctly typed (namely words that were correctly typed on the first try by the user) into account.

Word entry speed corresponds to the duration between the entry of the first character of a word and the final selection of the whole word (by pressing the space bar for VODKA or the two digits tap for DUCK), divided by the length of that word. It is important to note that the whole typing time for DUCK includes the time needed for the selection of the first character, the remaining N-1 taps, and the validation phase. The validation phase consists in listening one to many words in the list, and selecting the expected one. The average word entry speeds were 0.383 cps (median=0.28, SD=0.20) for VODKA and 0.398 cps (median=0.32, SD=0.18) for DUCK. The Wilcoxon test showed that these speeds were significantly different (W=774, p = 0.048).

Figure 1 shows the word entry speed according to word length. It appears that it is very efficient to type two letters words with VODKA but not with DUCK. The plot also shows that the time needed to type a character with VODKA increases with word length. The speed for twelve letters word is only 0.4 cps. On the contrary, DUCK is getting more and more efficient to type long words. We compared the speeds measured for each word length using a series of Wilcoxon tests with a Bonferroni correction. It appears that DUCK is systematically more efficient than VODKA for words longer than six characters. Specifically, the plot in Fig 1 shows that the speed to type twelve letters words with DUCK is closed to 0.8 cps, i.e. twice the speed measured with VODKA.

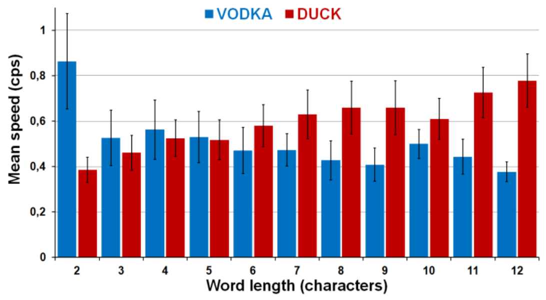

Figure 1: Average typing speed (in CPS) according to word length. The speed to type a character within a word depends on the length of that word.

5.2. Word typing without validation

As mentioned before, the time needed to type a word with DUCK includes two phases. The first phase, which we called "tapping", includes the time needed to find and select the first letter of a word of N characters, and the time needed to do the following N-1 taps. The second phase, which we called "validation", includes the time needed to choose the correct word in the list. The position of the expected word in the list depends on the difference between the measured typing distance and the expected word distance.

Figure 2 shows the time needed to input a word with VODKA (T_V) and DUCK (T_D). It also shows the time needed for tapping (T_{TAP}) and validating (T_{VAL}) with DUCK. These different durations were computed according to word length (2 to 12 characters). The least square linear regression shows that the typing time with VODKA (blue line) increases with a coefficient of 2.7 (R^2=0.95). The whole typing time with DUCK (red line) is a bit less affected by word length (y=1.2, R^2=0.95). These lines intersect at 3.7 letters, which means that words longer than four letters were quicker to type in DUCK.

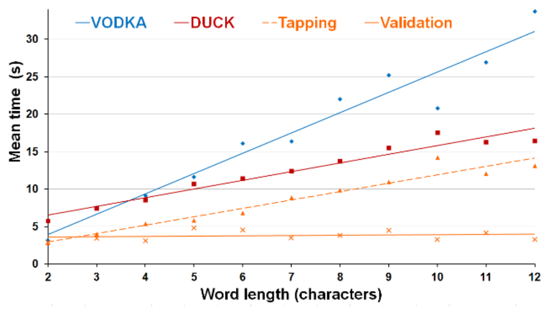

Figure 2: Word typing time (s) for DUCK (T_D) and VODKA (T_V). The tapping (T_{TAP}) and validation (T_{VAL}) times needed with DUCK are also indicated ($T_D=T_{TAP} + T_{VAL}$). The lines indicate least square regressions for each series of data.

The DUCK tapping time (dotted orange line) is inferior but highly correlated to the whole input time with a very similar coefficient (y=1.1, R^2=0.94). As expected, the validation time (orange) does not depend on word length (y=0.03, R^2=0.03). It is nearly equivalent to the y-intercept (a=3.5) and constant regardless of word length. It appears that the tapping time for DUCK is always shorter than the typing time for VODKA.

Interestingly the difference of time needed to type long words with the two keyboards is increasing. Obviously, the whole input time (including the validation phase) is mandatory with DUCK and underplays this observation. However, splitting the data at length of six yields a significant result ($p<0.001$), showing that the validation step needed with DUCK is fully compensated for words longer than five characters.

5.3. *Text input accuracy*

The Minimum String Distance (MSD provides the minimal number of operations needed to transform one string to the other. DUCK and VODKA yielded similar MSD scores (94% for DUCK and 96% for VODKA), which illustrate a similar tolerance to letter-level mistakes. Because DUCK was primarily designed to enter full words and not letters, we designed a measure similar to the MSD but addressing the word level instead of the character level. For each sentence, we counted the total number of words that were typed, and the words that were typed correctly. We then defined the Minimum Word Distance (MWD):

$$MWD = \frac{W_{correct}}{W_{total}} \times 100$$

Wilcoxon tests showed that MSD rates are not different. On contrary, there was a significant difference for the MWD rates (Wilcoxon; W=486, $p<0.001$). MVD for DUCK is clearly superior to VODKA's (92% vs. 41%).

Considering the overall input, including the twelve users (i.e. 192 sentences), there were 156 additional letters for VODKA, and 62 missing ones. There were 218 additional letters with DUCK, and 118 missing ones. We also computed the number of word insertions and word omissions in sentences typed using DUCK: 63 omissions and 32 additions.

6. Discussion

The results show that both keyboards yield similar results in terms of global input speed. Obviously, DUCK is very efficient to tap a succession of characters but the validation time, which is mandatory, partially impairs that efficiency. It appears that the validation time is constant (3.5 s in average), and probably hard to reduce. Hence, input with DUCK gets more efficient when the validation phase is compensated, which is the case for long words. It appeared that the gain is proportional to the number of characters in the word, and that the validation time is fully compensated for words of five letters long. It means that DUCK is more efficient than a classical software keyboard for words of six letters long.

One improvement of DUCK may target the validation time which is a bit long. It is not an issue related to the deduction algorithm because the expected word is generally located on the top of the list, even for shorter words. In fact, it is the interaction that we designed that is not suitable for words smaller than three characters. The user loses too much time selecting the expected word in the list, especially when the word is not in first position. Indeed, the further the word is in the list, the longer the user will need to validate.

The algorithm is henceforth *deductive*: we do not use the shape of the word as an additional signature. We simply aim at correcting imprecise keyboard layout knowledge. By using a more complex algorithm like the ones used in SWYPE or SHARK[2], and giving words an optimal place in the list by using user heuristics we could get better deduction results.

However, although the deduction algorithm works fine in most cases (91% of the words are in the first or second positions), some deduction problems remain. The first one is related to the similarity of two words. Two words that have the same length and characters that are close to each other on the layout will be always suggested together. For example, “these” and “there” only differ by the fourth letter. Since these two letters are close to each other on the keyboard (one key difference), the algorithm will not make the distinction between the two words. In fact, the ranking in the list depends on the distance between the tap, and the ‘r’ and ‘s’ keys.

7. Conclusion

We presented the DUCK keyboard which, from a first letter correctly typed and a set of approximative hits can deduce the desired word by the user. Due to the validation phase, this system doesn't offer an advantageous solution for short words (less than four characters long). However, it is really efficient for words longer than four characters. While it might not be suited for smartphones (because of shortened words and abbreviations), it ends up being a pertinent solution for note-taking tasks on tactile displays like tablets (office use & word processing).

8. References

[1] Bonner, M. N., Brudvik, J. T., Abowd, G. D. and Edwards, W. E. No-look notes: accessible eyes-free multi-touch text entry. *In Pervasive Computing*, 409-426. Springer, 2010.

[2] Kristensson, P.-O. and Zhai, S.. 2004. SHARK2: a large vocabulary shorthand writing system for pen-based computers. In *Proc. of* UIST '04. ACM, New York, NY, USA, 43-52.

[3] MacKenzie, I.S. and Zhang, S. X. 1999. The design and evaluation of a high-performance soft keyboard. In *Proc. of* CHI '99. ACM, New York, NY, USA, 25-31.

[4] Mascetti, S., Bernareggi, C. and Belotti, M. Typeinbraille: Quick eyes-free typing on smartphones. In *Proc of ICCHP'12*, 615-622, Berlin, Heidelberg, 2012. Springer-Verlag.

[5] Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J. and Gonçalves, D. Brailletype: Unleashing braille over touch screen mobile phones. In *Proc. of INTERACT'11*, 100-107, 2011. Springer-Verlag.