Observational and Experimental Investigation of Typing Behaviour using Virtual Keyboards on Mobile Devices

Niels Henze
University of Oldenburg
Oldenburg, Germany
niels.henze@uni-oldenburg.de

Enrico Rukzio
University of Duisburg-Essen / Lancaster University
enrico.rukzio@uni-due.de

Susanne Boll
University of Oldenburg
Oldenburg, Germany
susanne.boll@uni-oldenburg.de

ABSTRACT
With the rise of current smartphones, virtual keyboards for touchscreens became the dominant mobile text entry technique. We developed a typing game that records how users touch on the standard Android keyboard to investigate users' typing behaviour. 47,770,625 keystrokes from 72,945 installations have been collected by publishing the game. By visualizing the touch distribution we identified a systematic skew and derived a function that compensates this skew by shifting touch events. By updating the game we conduct an experiment that investigates the effect of shifting touch events, changing the keys' labels, and visualizing the touched position. Results based on 6,603,659 keystrokes and 13,013 installations show that visualizing the touched positions using a simple dot decreases the error rate of the Android keyboard by 18.3% but also decreases the speed by 5.2% with no positive effect on learnability. The Android keyboard outperforms the control condition but the constructed shift function further improves the performance by 2.2% and decreases the error rate by 9.1%. We argue that the shift function can improve existing keyboards at no costs.

Author Keywords
touchscreen; virtual keyboard; mobile phone; public study.

ACM Classification Keywords
H.5.2 Interfaces and Presentation: User Interfaces - Input devices and strategies.

General Terms
Design, Human Factors, Experimentation.

INTRODUCTION
Since the introduction of the iPhone, mobile phones with touchscreens began to dominate the smartphone market. Today, all major phone makers have touchscreen devices in their portfolio. In contrast to earlier devices, today's smartphones are operated by touching the screen with the fingers and only a few devices have a physical keyboard. Instead, users rely on virtual keyboards that are operated by touching the screen.

CHI'12, May 5–10, 2012, Austin, Texas, USA.

While touchscreens and virtual keyboards have been studied for years, understanding users' touch behaviour remains challenging. Previous work usually studies the effect of single aspects, such as key size or keyboard layout, on the users' performance. Due to limited resources corresponding user studies are often conducted with a homogenous sample and a single device. Such studies usually try to seek a balance between internal validity (the extent to which variance is due to the test conditions) and external validity (the extent to which results are generalizable). Experimenters control most variables and conduct studies with a small number of participants in a lab (high internal and low external validity). Many results from related work are therefore based on the performance of male right-handed students from a technical discipline that live in the same region i.e. no equal gender split and mainly participants from the authors' institution.

In contrast to previous work, our aim is to observe and manipulate the touch behaviour of a diverse sample, a large number of devices, and various contexts. To collect the required large amount of keystrokes on a virtual keyboard we developed a mobile typing game. To attract a large number of participants the game has been published to the Android Market. Our approach thus allows studying a large number of users with varying backgrounds in a large number of realistic contexts with their own devices (low internal validity due to a high variance but high external validity). This allowed analysing the typing performance of users whose behaviour would have been significantly altered in a very controlled setting. Because external factors cannot be ruled out and we have little control over the participants the study has a low internal validity as there was no possibility to control any contextual factors. The flip side is that the diversity of the environment provides a higher external validity than common lab studies.

After discussing related work, we describe the game that we developed to collect the data. We provide an overview about the data we collected after publishing the game to the Android Market. Following this, an analysis of the touch distribution is provided that shows how touch contacts are skewed relative to the keyboard's keys' centre. Afterwards, three approaches to influence the users' typing behaviour are proposed. We report how we evaluated these approaches in an experiment by publishing an update of the game to the Android Market. We show that our adapted shift function improves the performance, elevating the position of the keys' labels is not beneficial, and informing the user about the touched position decreases the error rate but also decreases the speed. We close the paper with a conclusion and an outlook on future work.

RELATED WORK

In the last decade and in particular since the emergence of the iPhone we observed a shift from using keypads and stylus-based interaction to finger-based interactions with touch-screens on mobile devices. One important aspect to be considered in the design of virtual keyboards is that the output resolution of such a touchscreen is much higher than the input resolution of a human thumb or finger. This leads to the "fat-finger-problem" due to the difficulty to select small targets with a much larger finger and the aspect that the finger occludes the target as well. Current smartphones address this aspect e.g. through a visual confirmation of what has been touched or trough callouts that show the region currently touched in order to perform fine granular selections.

One strand of research focuses on interaction techniques that allow the selection of small targets with a finger without changing the size of the target while achieving an acceptable error rate. In Shift [20] this has been achieved through callouts showing a copy of the area occluded by the finger in a non-occluded area and the possibility to move a pointer in the callout via finger movement to select the desired target. In TapTap [15] the occluded area is also shown in a callout but here a zoomed in copy of the occluded area is shown and the user has to touch the desired target in the callout with a second touch. In Escape [21] the small targets are visually changed and indicate a direction in which the user has to drag its finger after touching it in order to select it. Those interaction techniques are not well suited for text input as additional interactions are required, which requires more time and a higher mental effort, when compared with a simple touch.

Further research focused on the optimal size of targets while considering the trade-off between finger size and user interfaces design. For almost perfect accuracy targets need to be larger than 20 x 20 mm [10]. This means that current touch-screen phones would be able to display only around 8 targets while showing no other information. According to the iOS Human Interface Guidelines [1] the optimal size of a tapable UI element on the iPhone is 6.74 x 6.74 mm which is a compromise between an acceptable error rate and the available screen size. A significant body of research investigates the influence of target size and context on time needed for selecting a target and the error rate [12, 19]. Considered contextual aspects were e.g. the actual task (e.g. inspired by Fitts' law or text input), device- and display-size and -type, thumb size [2], activity [17], touch feedback [9] or one-handed or two-handed interaction. The outcome is often a suggestion regarding an optimal target size and location under consideration of the given context and an assumption regarding acceptable error rate, task load or user satisfaction.

Relatively little research analysed how the actual location of a target on the screen or a device's orientation affects effectiveness and efficiency. Early research, focusing on fixed touch-screens mounted on a table, showed that users touch slightly below the actual target if the screen is tilted away from the user and that they touch above the target if its tilted towards the user [18]. Other research showed that the location of targets on the screen has an effect on effectiveness, efficiency and user satisfaction. Himberg et al. developed an adaptive numerical on-screen keyboard that observes where the user is touching the display in relationship to the displayed key [5]. This information is used to adapt the shape of the virtual keys to improve the error rate. Similar work by [8] uses geometric pattern matching to reduce the error rate for stylus-based text entry. [3] developed an anchored keyboard adaptation and a simulation suggests that it reduces the errors rate. Holz and Baudisch investigated how crosshairs are targeted and present a model that can reduce the error offset [6].

Karlson showed that regions which are easily to reach with the thumb when considering one-handed interaction achieve the best task performance and lowest perceived difficulty [7]. Karlson concludes that frequently used buttons should be placed in those regions. Perry and Hourcade showed again that targets within easy reach of the thumb can be reached quicker but the accuracy is best when the targets are located on the left, right and top edges of the screen [14]. Park et al. analysed the success rate, error rate and convenience of 25 regions of a touchscreen when using one-handed thumb input [13]. The authors also analysed the offset between indicated target and actual touch events. They observed location-specific offsets and discuss the idea of adjusting the location of the touch recognition area to improve the overall performance. Those findings have been extended by Henze et al. who analysed those offsets using a very large data set and showed that a corresponding compensation function can reduce the error rate significantly [4]. Recent work of Ruchenko et al. tried to improve the performance of virtual keyboards through data collected in a game [16]. They showed the positive effects of providing feedback about where users touched so user could adapt their behaviour. Unfortunately, only 6 persons participated in the laboratory study and the potential advantages of key-target resizing were only shown in a simulation and were not tested.

Our paper is the first that analyse the offset between the displayed keyboard keys and the actual hit locations based on a very large data set collected in a realistic context. This allows us, in contrast to previous research that was performed in laboratory settings or which is based on a small number of touch events, to calculate those offset vectors very precisely. Furthermore are we the first to show that the application of corresponding compensation functions improves performance and reduce error rate significantly for a very large number of users who typed in various contexts using various devices.

DESIGN OF THE GAME

To collect a large number of keystrokes on a virtual keyboard from a number of different devices and diverse participants we decided to collect data using a mobile typing game. During the design of 'Type It!' we had to find a balance between providing players with a game that is worth playing and a test application that collects meaningful data.

Game play

The game play focuses on collecting basic keystrokes that form independent words. Words are presented to the player and the task is to type these words. The game is structured in

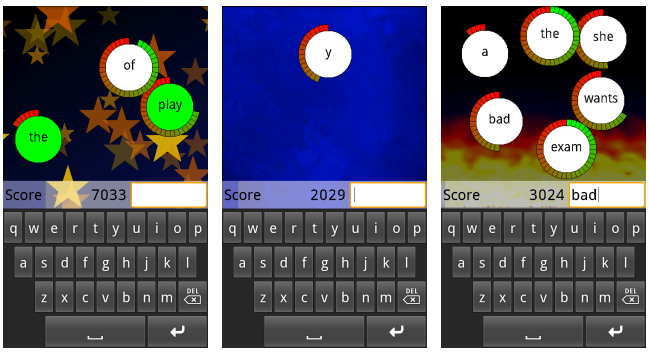

Figure 1. Screenshots of the game's three stages: stars, water, and fire.

three stages called stars, water, and fire. Each stage contains four levels and each level consists of multiple words that must be typed. As shown in Figure 1 the keyboard is displayed in the lower half of the screen and the words are shown in the upper part of the screen. While playing, words are presented in white circles with a fixed size. A circular progress bar around the circles shows the remaining time until the word must have been typed. The bar is coloured from red to green to also highlight the remaining time. While the time to type a word expires, the progress bar gets shorter. The available time to type a word depends on the level and the number of characters. Depending on the level, multiple words are presented simultaneously and can be typed in any order.

A word's characters must be typed to complete it. While typing, the characters appear in a textbox just above the keyboard. The player must confirm the words by either tapping the space bar or the enter key. If a word has been typed correctly the word's background becomes green, the progress bar accelerates, and a rattle sound is played. If the progress bar gets empty the word disappears. To make a game out of the basic task the player must complete a word in a certain timeframe. The timeframe is reduced from word to word while the player proceeds through a level and also depends on the word's number of characters. Players receive a penalty point if a word has not been completed in the given timeframe. The game is lost when the player collected three penalty points in one level. Players receive scores when they complete a word. The faster a word is typed the higher the score.

To increase the study's internal validity, the same keyboard is used for all devices. We used the source code of the standard Android 2.2 ('Froyo') keyboard as basis. The Android keyboard is designed to scale across different devices, screen sizes, and resolutions. We adapted the keyboard by removing keys that are not required to play the game and added code to measure the players typing behaviour. An interesting aspect of the Android keyboard is that the position of touch events is internally shifted upwards by 10 'density-independent pixels' (dp). Dp is an abstract unit based on the physical density of the screen. These units are relative to a 160 dots per inch screen and designed so that 160dp is one inch. According to the Android Developer Guide the ratio of dp-to-pixel changes with the screen density, but not necessarily in direct proportion[1]. In addition, if there is free space to the left (as for the 'a') or to right of a key (as for the 'l') this area is part of the keys interactive region. Touching, for example, on the free space left of the 'a' is still considered as typing on the 'a'.

level	max. characters	source
1	4	MacKenzie et al.
2	5	MacKenzie et al.
3	1	one character
4	7	MacKenzie et al.
5	1	one character
6	2	two characters
7	3	three characters
8	2	two random characters

Table 1. The text sources used for the first eight levels of the game.

We made the game visually appealing to motivate intensive usage. Each stage has a different animated background shown in Figure 1. The total score is shown above the keyboard next to the text box. Furthermore a player receives "badges" when successfully completing a level or achieving other goals. To increase the long term motivation we implemented a global and a local high score lists shown in Figure 2. Players can share their score via twitter if they achieve a high score.

Levels and text sources

To increase the external validity and the players' fun we use words with different length and from different sources throughout the levels. Table 1 provides an overview about the first eight levels and the used words. For most levels we randomly select words from the phrase set provided by [11] with a fixed maximum number of characters. In addition, for some levels we use all words with two or three characters from the Official Tournament and Club Word List for Scrabble[2] and words consisting of one or two random characters. We also vary the available time to complete the words and the number of words. In general, the game gets more challenging from level to level. While the first levels are very easy we assume that the very last level is impossible to finish successfully.

Measures and consent

We collect various data about the used devices and the performance of the players. An identifier for each installation is derived from a device's "Android ID" to anonymize the data. Furthermore, we collect the user's locale (e.g. "en_GB" or "es_ES"), the device's name (e.g. "GT-I9000" for the Samsung Galaxy S), the time zone as well as the width and the height of the virtual keyboard in pixels. During the game we record the presented words and the position where the player taps the virtual keyboard. We record the position where the player's finger initially hit the screen and the position where the finger lifts-off the screen. We do not record intermediate movement as this would have led to very large data sets to be logged and transmitted to our server later on. For the taps we record the position before the Android keyboard shifts the

[1]Android Developer Guide - More Resource Types: `http://developer.android.com/guide/topics/resources/more-resources.html`

[2]Scrabble tournament and club word list: `http://www.isc.ro/en/commands/lists.html`

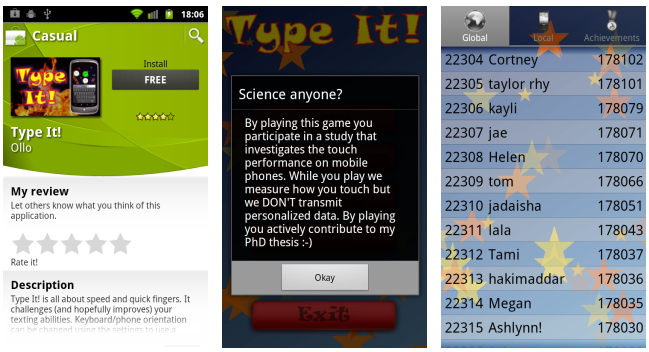

Figure 2. Type It! in the Android market (left), a modal dialog that informs the player about the study (centre), and the high score list (right).

touch events by the 10dp mentioned above. For all events we log the time elapsed since the start of the level.

The properties of the used device are transmitted to our server when a game is started and the data collected while playing is transmitted after a level is finished. The data is stored internally on the phone and retransmitted after the next level if the transmission failed. We do not store data that allows identifying individual players or installations. We inform players about the fact that data is collected to act ethically and to conform to corresponding legislation in many countries. The modal dialog shown in Figure 2 tells players that they are about to participate in a study when the game is started for the first time. In addition, the description in the Android Market briefly outlines our intention, what we record, and what we are trying to achieve with the collected data.

PUBLISHING IN THE MARKET

We published Type It![3] in the Android Market on April 29, 2011. Figure 2 shows the appearance of the game in the Android Market. Till July 31, 2011 the game got installed 89,262 times according to Google's Developer Console. In total the game received 880 ratings with an average of 3.98 on the five point scale (the higher the better). On our server we collected data from 80,424 installations but only 72,945 installations provided meaningful data (see below). We provide an overview about the data in the following.

Demographics

We collected data from devices with 581 different names. Most of these names appeared, however, only a few times and are rather exotic such as the "Pulse Mini MG Mod". For 104 names we collected data from more than a hundred installations. As the mobile network operators give different names to the same device type there are in fact much less different devices than the 581 names suggest. The Galaxy Tab, for example, appears with at least six different names. After harmonizing the names for common devices, the 15 most common devices represent 44.13% of all installations.

The collected locales and time zones show that there is a bias towards western countries among the players. The most common locales are English speaking US ("en_US", 65.94%) and English speaking Great Britain ("en_GB", 10.44%). This is followed by Germany ("de_DE", 1.84%), Spanish speaking US ("es_US", 1.68%), and France ("fr_FR", 1.37%). The other 191 locales together result in 18.72% including 64 further English locales representing 4.77% of all installations. The recorded time zones show a similar picture. The only non US American or European time zone among the ten most common ones is Asia/Calcutta.

[3]Type It! in the Android Market: `http://tiny.cc/type_it`

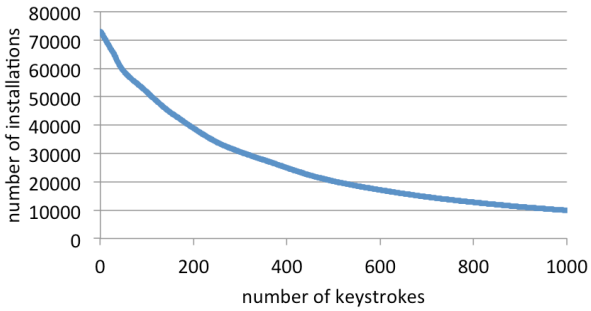

Figure 3. Number of collected keystrokes. The graph must be interpreted as on y installations more than x keystrokes have been recorded. E.g. 40.000 installations contributed more than 200 keystrokes.

Collected data

While we received data from 80,424 installations, not all of them provided meaningful data. We only use data from 72,945 installations because installations provided inconsistent data or we did not record a single played level. In total 952,487 levels have been played and on average 13.06 levels (SD=58.88) have been played on each installation. On 45.94% of the installations less than 5 levels have been played. There are, however, a few very intensive players and 21 installations, for example, contributed more than 1,000 levels each. The number of keystrokes per installation is analogue. In total 47,770,625 keystrokes have been recorded and on average 654.89 keystrokes (SD=4,149.46) have been produced on each installation. Figure 3 provides an overview about the number of keystrokes per installation.

OBSERVED TOUCH BEHAVIOUR

We computed the distribution of the positions where the players' fingers lift-off the screen for each key using our entire dataset. To compute the distribution, we could either assign a touched position to the key that fits the presented word or to the key that is recognized by the keyboard. As the touches are normally distributed and the error rates are low we assign the position to the key recognized by the keyboard. Figure 4 shows the touch distribution for the two most common devices: the Optimus One, a device with a 3.2 inch screen by LG, and the Ascend, a device with a 3.5 inch screen by Huawei. In the following we analyse the horizontal and the vertical skew in the distribution of taps.

Vertical offset

For each key, we analysed the distribution of touch events by computing the distance between the centre of the touch distribution and the centre of the visual area of the respective key.

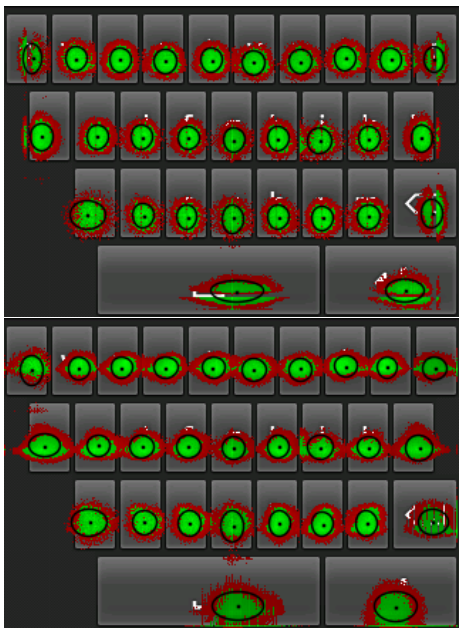

Figure 4. Touch distribution of the Optimus One based on 2,407,164 keystrokes (top) and the Ascend based on 4,589,967 keystrokes (bottom). Green regions cover 50% of all taps. Red and green regions combined cover 80% of the taps. Black dots show the distributions' centre and black ellipses one standard deviation.

On average the centre of the touch distribution is 10.60 pixels (SD=8.79 pixels) below the keys' centre for the Optimus One and 8.02 pixels (SD=8.34 pixels) below the keys' centre for the Ascend. The 95% confidence intervals for the centre of the touch distributions are $\pm$0.01 pixels wide. Taking the screen's physical size into account this means that on the Optimus One players hit on average 2.24 mm below the keys' centre and on the Ascend 1.85 mm below the keys' centre.

The deviation from the centre varies for the different character keys but the difference between the two devices is consistent for all keys. Using the distance between the keys' centres and the centre of the touch distribution for the 26 character keys as the sample, an unpaired t-test shows that the distance is significantly different ($p<.10^{-12}$, d=.91). Even though, we found this difference using post-hoc analysis the high effect size and the low significance level let us assume that we did not observe the difference by chance. Table 2 shows how far, on average for all keys, the players input is shifted below the centre of the respective key. The 95% confidence intervals for the centre of the touch distributions are $\pm$0.02 pixels wide or smaller. There is no correlation between the physical size of the offset and the size of the screen (r=.13) but a correlation between the physical size of the keys' distributions standard deviation and the size of the screen (r=.89). Thus, the data suggest that the vertical offset does not increase with an increasing screen size but the variability does.

Horizontal offset

While the vertical offset is very consistent across all character keys the horizontal offset varies across the keyboard and is much smaller. For the above mentioned Optimus One, the

device	screen	pixel offset	mm offset	SD
Ally	3.2in	13.23px	1.15mm	1.15mm
Droid Inc.	3.7in	14.72px	1.48mm	1.39mm
Glacier	3.8in	6.93px	0.72mm	1.32mm
Galaxy S	4.0in	12.72px	1.39mm	1.33mm
Captivate	4.0in	13.78px	1.50mm	1.34mm
Vibrant	4.0in	12.80px	1.39mm	1.35mm
Fascinate	4.0in	12.98px	1.41mm	1.37mm
Epic 4G	4.0in	13.55px	1.48mm	1.43mm
Desire HD	4.3in	9.05px	1.06mm	1.49mm
Evo 4G	4.3in	10.88px	1.27mm	1.54mm

Table 2. Average vertical offset for ten devices with a resolution of 480x800 pixels. 95% confidence intervals are $\pm$0.02 pixels or smaller.

average horizontal offset is negligible 0.03 mm and the Ascend's offset is 0.07 mm. For the 50 most common devices the average horizontal offset is below 0.1 mm. This is mainly caused by the different offsets for individual keys. E.g. for the Optimus One the horizontal offset for 'a' is 5.58 pixels/1.18 mm ($\text{conf}_{95\%} \pm 0.05$ pixels) while the offset for the 'l' is -5.13 pixels/1.08 mm ($\text{conf}_{95\%} \pm 0.06$ pixels). Thus, the individual keys must be treated separately. Figure 5 shows the average horizontal offset for all devices with a resolution of 480x800 pixels among the 50 most common devices. A negative value means that players typed right of a key's centre and a positive value means that players typed left of the key's centre. One must, however, notice that the top row covers the whole width of the screen while the two lower rows have free space to the left that also activates the key. In addition, the row in the centre has also free space to the right. With this in mind it can be seen that the average touch position is slightly skewed towards the centre of the screen for all but those keys with adjacent free space next to them (i.e. 'a', 'z', and 'l').

Discussion

Analysing the collected data, we found that players' taps are systematically skewed towards the bottom of the screen along the vertical axis. We also found that taps on keys that have no adjacent free space next to them are skewed towards the centre of the screen along the horizontal axis. Furthermore, we could show that the touch-distribution differs for different devices even if they have the same resolution and the same screen size.

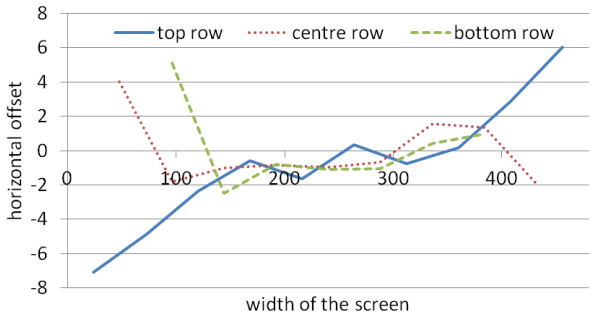

Figure 5. Average horizontal offset (in pixels) for the character keys of the 21 most common devices with a resolution of 480x800 pixels. Negative values mean that players taped right of a key's centre and positives values means that players taped left of the key's centre.

Besides the general limitations of our approach the results are mainly limited by the fact that comparing different devices means conducting a quasi-experiment. Thus, we cannot know if the observed differences are because of the devices itself or because of other factors. E.g. some devices might be preferred by a person group that tends to have smaller hands than the average (e.g. kids), users with different background might hold the device differently, and the devices' form factor might also have an impact. As our knowledge about the players is very limited we cannot factor out those aspects. Furthermore, the analysis is based on data collected using the standard Android keyboard that shifts the users input. Even though players still tap below the keys' centre and the majority is certainly not aware of this shift, we cannot know the influence of this approach.

The analysis of touch behaviour revealed a systematic skew in the distribution of taps. By shifting the users' input by 10dp towards the top of the screen the Android keyboard already tries to compensate this systematic skew. To our knowledge, however, no published work actually showed if this rather simple compensation function improves the users' performance. Another potential approach to influence the users' touch behaviour is to change the design of the keys. Assuming that the users try to hit the keys' labels, shifting the position of the labels towards the upper part of the keys might influence the users to also move their taps upwards. Finally, it seems reasonable to assume that users are not aware that their touch distribution is distorted. Visualizing the position where the device recognized a touch event, users might be able to adjust their behaviour according to the provided feedback.

INFLUENCING TOUCH BEHAVIOUR

To analyse the three approaches to influence the users touch behaviour proposed in the previous section we design according implementations which are discussed in the following.

Shifting touch positions

To analyse how shifting the users' input influences the touch behaviour we use three different ways to shift the users' taps. The first technique, that we call 'no shift', does not shift the touch events and simply uses the touch events' raw position. As the second technique, that we call 'native shift' we use the standard Android keyboard that shifts the touch events by 10 density independent pixels towards the upper part of the screen.

For the third technique, called 'adapted shift', we derived a compensation function from the data described in the previous section. The technique follows the assumption that it is best to shift the users' input in a way that moves the centres of the touch distributions to the centres of the keys. Figure 6, exemplarily visualizes the compensation function computed for the Galaxy S. For each key, we use the centres of the distribution of touch events as support points (highlighted by a white dot in Figure 6). For each key a vector to the key's centre is derived (shown in blue). Using linear interpolation the shift vectors for the corners of all keys are computed based on the support points. To avoid shifting taps off the keyboard the respective component of the vectors located at the keyboard's

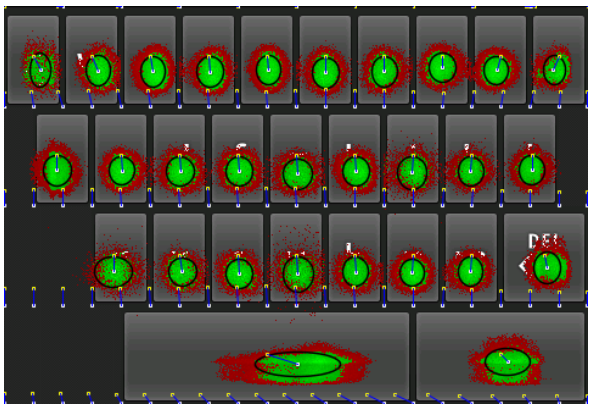

Figure 6. Touch distribution and computed compensation function for the Galaxy S based on 3,242 installations that provided 1,831,489 keystrokes. The white dots show the origin of the shift vectors, the yellow dots their destination, and the blue lines show the actual vectors.

border are set to zero. E.g. the y-component of vectors located at the upper boundary of the keyboard is set to zero to avoid shifting taps above the keyboard. Finally, shift vectors for further positions are derived using linear interpolation to produce an array of 21x5 shift vectors. All touch events on the keyboard are shifted according to the surrounding vectors.

The touch distribution differs for different devices. Therefore, a compensation function is computed for each of the 50 most common devices in the dataset. For the remaining devices a function is computed using data from all devices with this particular resolution. However, all resolutions with less than 150,000 keystrokes in the dataset are rejected which leads to 58 different functions. The compensation functions are integrated in the Android keyboard and applied before touch events are processed by the keyboard.

Shifting key labels

This approach is based on the assumption that users are influenced by the locations of the keys' labels and, at least to some degree, try to hit the label. The labels are either shifted to the upper part of the keys ('elevated labels') or not ('default labels'). To maintain the labels' font size and leave room for larger upper case characters the labels are only slightly shifted. The centre of the labels is shifted from the centre of the key to the upper third of the key depending on the device's resolution. E.g. for the Optimus One the labels are shifted by 11 pixels towards the top of the screen, the same amount of pixels that players touches are skewed towards the bottom of the screen. As the labels' shift only depends on the device's resolution, it only roughly approximates the skew we found in the previous section. The horizontal position is not altered. Figure 7 shows the difference between the default and the elevated labels.

Showing touched positions using dots

The third approach tries to inform the user about the touched position in an unobtrusive way. A small red dot appears as soon as the user's finger touches the screen. The dot follows the touch events until the finger lift-off again and remains at

Figure 7. The two different positions of the keys' labels. The green labels show the default Android keyboard and the white labels show the elevated labels.

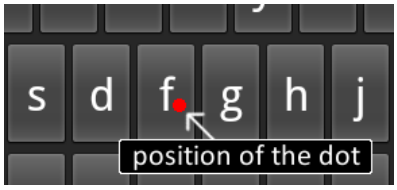

Figure 8. Keyboard that shows a red dot at the position where the user touches the screen after typing an 'f'.

this position. Thereby, the dot always shows the last touched position. We refrain from showing former touch positions or the movement of the finger to keep the visualization as simple as possible to increase the understandability and minimize the obtrusiveness. Figure 8 shows the keyboard after the user just tapped on the 'f'.

EXPERIMENT

We conducted an experiment to analyse the effect of the proposed approaches. The game was published as an update to the Android Market after integrating the three approaches. In the following we describe the design of the experiment, followed by the results, and a discussion of our findings.

Design

The experiment follows an independent measures design to allow players adapting to the respective keyboard. The three approaches described in the previous section are used as the independent variables. Table 3 provides an overview about the independent variables and their degrees of freedom. In total, the design results in 12 conditions. Installations are randomly assigned to one of the conditions when the game is started for the first time or when the updated version is started for the first time. By randomly assigning installations to one condition we assume that players are almost evenly distributed across the 12 conditions.

Three measures are used to assess the players' behaviour. We assess the players' *speed* by counting the number of keystrokes and measuring the time to complete a level to derive the number of keystrokes per second. In addition, we determine the players' *performance* by counting the number of keystrokes that contribute to the given words. E.g. keystrokes that produced characters that are deleted afterwards are excluded. This is used to compute the number of correct keystrokes per second. Finally, we assess the *error rate* by dividing the number of keystrokes that lead to incorrect characters or compensate errors by the total number of keystrokes.

independet variable	degrees of freedom
touch	no shift native shift adapted shift
label	default labels elevated labels
dot	no dot dot

Table 3. The three independent variables and their degrees of freedom.

Results

We deployed an update of the game to the Android Market on July 31, 2011 and collected data until September 19, 2011. We use the data provided by installations that updated or newly installed the game. In total we received data from 26,586 installations. For the analysis we only consider data provided by devices that can use one of the computed compensation functions. Thereby, we consider only data from the 50 most common devices and from the 8 most common resolutions. Furthermore, we removed the first played level from the data provided by each installation and data from all installations that contributed only a single level or provided inconsistent data, to reduce the noise in the data.

We use data from 13,013 installations that contributed 6,603,659 keystrokes by playing 120,662 levels for the following analysis. On average an installation contributed 9.27 levels (SD=25.54) and 507.47 keystrokes (SD=1,556.69). The average number of installations per condition is 1084.42 (SD=36.24), the average number of played levels per condition is 10,055 (SD=938.46), and the average number of keystrokes per condition is 550,304 (SD=53,537). Because the device types and players' locales are consistent with our first observation we refrain from a detailed description.

In the following we analyse the effect of the independent variables using a three-way independent analysis of variance (ANOVA). The differences between the individual conditions are analysed using Fisher's Least Significant Difference (LSD) post-hoc test. The levels of significance when comparing the conditions are shown in Figure 12. The term 'control condition' is used for the condition with no shift, default labels, and no dot - i.e. the Android keyboard without shifting the touch events by 10dp.

Speed: To assess the effect of the independent variables on the players' speed we compare the number of keystrokes per seconds. The ANOVA show that the three main effects as well as all interaction effects are significant ($p<.001$). Figure 9 shows the average number of keystrokes per second for all conditions. In particular, the 'native shift with default labels and no dot' together with the two 'adapted shift with default labels' conditions result in a significantly higher number of

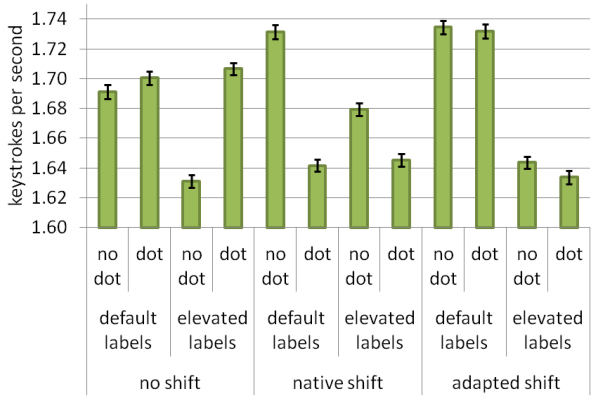

Figure 9. Average speed assessed through the number keystrokes per second (error bars show standard error).

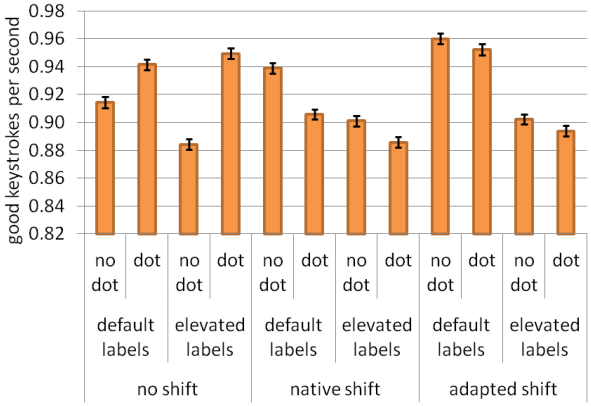

Figure 10. Average performance assessed through the number of correct keystrokes per second (error bars show standard error).

keystrokes per second than all other conditions. The difference between these three conditions is, however, not significant. Players are 2.5% faster with one of the three conditions compared to the control condition and players are 6.4% faster with the fastest condition (adapted shift with default labels and no dot) compared to the slowest condition (no shift with elevated labels and no dot).

Performance: To assess the players' performance we investigate the independent variables' effect on the correct keystrokes per second (i.e. those keystrokes that are not errors and do not compensate an error). The ANOVA show that the two main effects touch and label ($p<.001$) as well as the effect of dot ($p<.05$) are significant. All interaction effects are also significant ($p<.001$). Figure 10 shows the performance for all conditions. For all conditions with native shift or adapted shift, elevating the labels results in a lower performance (between 2.2% and 6.1%). For the same conditions with native shift or adapted shift, the dot also results in a lower performance (between 0.8% and 3.5%). The results for the conditions without shift are mixed. For these conditions, however, the dot increases the performance. Comparing the native shift and the adapted shift with default labels and no dot, shows that that performance increases by 2.3% with the adapted shift. Compared to the control condition, the adapted shift with default labels and no dot (the condition with the highest performance) has a 5.0% higher performance.

Error rate: To determine the error rate we divide the number of keystrokes that are errors or compensate an error by the number of the remaining keystrokes. The ANOVA shows that all main effects as well as all interactions are significant ($p<.001$). Figure 11 shows the error rate for all conditions. Pair wise comparison of conditions with the same shift function and the same labels show that the dot always reduces the error rate (between 3.8% and 18.3%). Compared to the control condition, the native shift with default labels and no dot (the standard Android keyboard) has a 2.2% higher error rate. Adapted shift with default labels and no dot (the Android keyboard with adapted shift) decreases the error rate by 7.1% compared to the control condition. The native shift with default labels and dot (the condition with the lowest error rate) has a 16.4% lower error rate than the control condition.

Learnability with dots: The assumption behind showing the touched position to the user was that users can adapt their behaviour using the provided feedback. Therefore, we analyse how the error rate, with and without dots, changes over time. We computed the error rate after playing 2 to 11 levels. To determine the difference between installations in the 'no dot' conditions and those in the 'dot' conditions we normalized the error rate to a percentile scale. Thus, 100% is the average error rate over all conditions after playing a certain number of levels. A larger percentage means an above-average error rate while a lower percentage means a below-average error rate. Figure 13 contrasts the results for installations in the 'dot' conditions with installations in the 'no dot' condition. The correlation between the number of played level and the difference of the error rate is r=-0.41. This small inverse correlation suggests that the difference of the error rate slightly decreases when players get more experienced. Thus, players in the 'dot' conditions do not further improve their error rate compared to players in the 'no dot' conditions.

Discussion

Using an experiment we compared three approaches to influence the users' behaviour when typing on a virtual keyboard

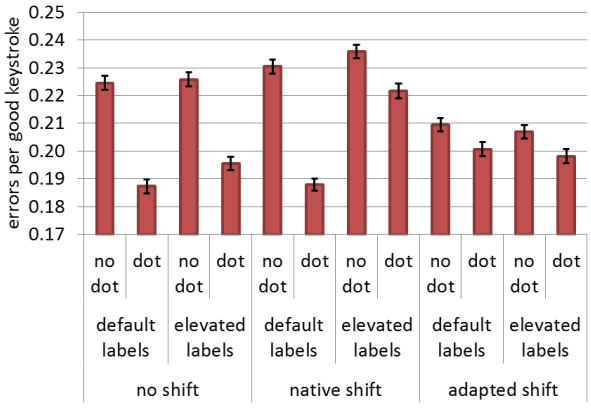

Figure 11. Average error rate assessed by dividing the number of keystrokes that are errors or compensate an error by the total number of keystrokes (error bars show standard error).

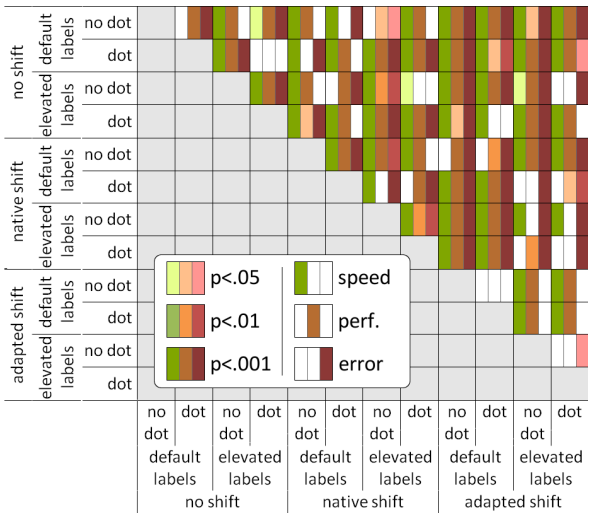

Figure 12. Significance levels for comparison of the individual conditions and the three depended variables using Fisher's LSD.

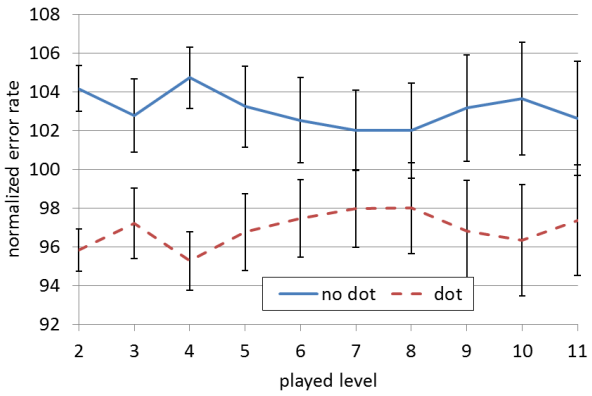

Figure 13. Normalized error rate for installations in the 'dots' and the 'no dots' conditions for the first 11 played levels. Error bars show standard error.

using data provided by 13,013 installations, 120,662 played levels and 6,603,659 touch events.

Shifting touch positions: Using the default labels and no dot the shift function provided by the Android keyboard results in 2.4% higher speed, 2.7% higher performance, and a 2.2% higher error rate compared to the control condition. The advantage for the adapted shift function is even higher. Using the adapted shift results in 2.6% higher speed, 5.0% higher performance, and a 7.7% lower error rate compared to the control condition. The results show that the native shift function improves the users' typing. The adapted shift function can, however, further improve the users' performance by 2.2% and decrease the error rate by 9.1% when compared to the standard Android keyboard.

Shifting key labels: For all conditions with a shift function elevating the labels' position decreases the speed, decreases the performance, and increases the error rate for the Android keyboard. The only condition that improved by elevating the labels is without shift and with dot. Still, the overall results strongly suggest that elevating the labels' position to the upper part of the key does not improve the users' typing.

Showing touched positions using dots: It is found that visualizing where the user's finger lift-off the keyboard using a dot decreases the error rate for all conditions. Adding the dot to the control condition, for example, decreases the error rate by 16.9% and adding the dot to the Android keyboard decreases the error rate by 18.3%. For all conditions with the native or the adapted shift function, however, the dot decreases the speed up to 5.2% and also decreases the performance. To reduce the error rate users must pay attention to the dot. We assume that this attention spend on the dot reduces the speed and as a consequence also the performance.

Limitations: In line with previous work that investigates typing on virtual keyboards one limitation of the study lies in the task that is used to collect the data. While other work asks participants to copy text, often in a highly controlled environment using a single device, our task is part of a typing game. Using a public game, however, has the advantage that the study has a high external validity. It can be assumed that users have more diverse backgrounds and use the game in more diverse contexts than what can be achieved in common lab studies. Furthermore, the study is not limited to one specific device but considers a broad range of devices. This is especially important because our observation of the touch behaviour showed that the device influences how users type.

As we collected data from players that installed the game at their own will we have very limited control over the participants. We cannot control who plays the game and when they stop playing. E.g. the probability that players with a low performance stop playing early might be higher than for players with a high performance. The used approach, however, enables to attract a truly large sample from all over the world. Instead of conducting a study with a small homogeneous sample, as it is often common in other studies, the large sample and the participants diversity increases the study's validity.

CONCLUSION

We investigated the touch behaviour on mobile devices' virtual keyboards. To observe the behaviour of a large number of users we developed a typing game. Using this game we collected 47,770,625 keystrokes that have been produced by 72,945 installations playing 952,487 levels. Analysing the data shows that players touch systematically below the centre of the keys (e.g. 12.72px or 1.39mm for the Galaxy S). Furthermore, we found no strong correlation between the touch skew and the devices' size. It can be concluded that other factors, such as the devices' form factor or the devices' user groups, have a stronger effect on the typing behaviour. We used the collected data to identify three approaches that might influence the typing behaviour positively. We developed a function that compensates the systematic skews found in the touch distribution. Furthermore, we shift the keys' labels to the upper part of the keys and show the position where the user's finger lift-off the screen using a simple dot.

To compare the three approaches we conducted an experiment by publishing an update of the game. For the experi-

ment with three independent variables we collected data from 13,013 installations that contributed 6,603,659 keystrokes by playing 120,662 levels. The results disprove our assumptions that elevating the position of the keys' labels is beneficial. Showing the users where they touch using a dot clearly improves the error rate. The error rate for the standard Android keyboard, for example, is decreased by 18.3%. The dot, however, also decreases the typing speed and has no positive effect on learnability. The usefulness of this feedback therefore depends if the error rate or speed is more important. We showed that the simple shift function provided by the standard Android keyboard improves the users' speed but provides no relevant improvement for the error rate. The adapted shift function that we derived from our observation, however, further improves the performance by 2.2% and decreases the error rate by 9.1% compared to the Android keyboard. Because this shift function can be used as a drop-in replacement for the Android keyboard's shift function we assume that it can improve the typing on current smartphones at no costs.

The conducted studies have a low internal validity but, compared to common lab studies, a very high external validity. Our approach has the advantage that the results are based on a very large number of users, the participants are likely representative for Android users, and the data has been collected in real life contexts from users using their own devices. We assume that the very large number of observed users is by nature representative for smartphone users. There are, however, also disadvantages of the used approach. Studies with a high external validity, such as ours, enable to observe the users real behaviour. In contrast, studies with a high internal validity that investigate isolated aspects might be better suited to find the causes for the observed behaviour.

The collected data provides a rich source of information. We would like to share the data with other to enable further analysis. In particular, analysing the data provided by different devices could provide interesting insights. Furthermore, the data could be used for analysing touch sequences to dynamically adapt the keyboard while typing words. We are interested in using the game to investigate further aspects that influences the users' typing behaviour. E.g. testing further visual keyboard designs and shift functions. We also aim at investigating more radical keyboard designs and if other widgets can be investigated using mobile games.

Acknowledgments: Parts of this work were conducted within the context of the Emmy Noether research group Mobile Interaction with Pervasive User Interfaces funded by the German Research Foundation (DFG).

REFERENCES

1. Apple Inc. iOS Human Interface Guidelines, 2010.
2. Balakrishnan, V., and Yeow, P. A study of the effect of thumb sizes on mobile phone texting satisfaction. *Journal of Usability Studies 3* (2008).
3. Gunawardana, A., Paek, T., and Meek, C. Usability guided key-target resizing for soft keyboards. In *Proc. IUI* (2010).
4. Henze, N., Rukzio, E., and Boll, S. 100,000,000 taps: Analysis and improvement of touch performance in the large. In *Proc. MobileHCI* (2011).
5. Himberg, J., Häkkilä, J., Kangas, P., and Mäntyjärvi, J. On-line personalization of a touch screen based keyboard. In *Proc. IUI* (2003).
6. Holz, C., and Baudisch, P. Understanding touch. In *Proc. CHI* (2011).
7. Karlson, A. Interface Design for Single-Handed Use of Small Devices. In *Proc. UIST* (2008).
8. Kristensson, P.-O., and Zhai, S. Relaxing stylus typing precision by geometric pattern matching. In *Proc. IUI* (2005).
9. Lee, S., and Zhai, S. The performance of touch screen soft buttons. In *Proc. CHI* (2009).
10. Lewis, J. R. Literature review of touch screen research from 1980 to 1992. In *IBM Technical Report 54.694* (1993).
11. MacKenzie, I., and Soukoreff, R. Phrase sets for evaluating text entry techniques. In *Adjunct Proc. CHI* (2003).
12. Parhi, P., Karlson, A., and Bederson, B. Target size study for one-handed thumb use on small touchscreen devices. In *Proc. MobileHCI* (2006).
13. Park, Y., Han, S., Park, J., and Cho, Y. Touch key design for target selection on a mobile phone. In *Proc. MobileHCI* (2008).
14. Perry, K., and Hourcade, J. Evaluating one handed thumb tapping on mobile touchscreen devices. In *Proc. GI* (2008).
15. Roudaut, A., Huot, S., and Lecolinet, E. TapTap and MagStick: improving one-handed target acquisition on small touch-screens. In *Proc. AVI* (2008).
16. Rudchenko, D., Paek, T., and Badger, E. Text text revolution: A game that improves text entry on mobile touchscreen keyboards. In *Proc. Pervasive* (2011).
17. Schildbach, B., and Rukzio, E. Investigating selection and reading performance on a mobile phone while walking. In *Proc. MobileHCI* (2010).
18. Sears, A. Improving touchscreen keyboards: Design issues and a comparison with other devices. *Interacting with computers 3* (1991).
19. Sears, A., and Zha, Y. Data entry for mobile devices using soft keyboards: Understanding the effects of keyboard size and user tasks. *International Journal of Human-Computer Interaction 16* (2003).
20. Vogel, D., and Baudisch, P. Shift: a technique for operating pen-based interfaces using touch. In *Proc. CHI* (2007).
21. Yatani, K., Partridge, K., Bern, M., and Newman, M. Escape: a target selection technique using visually-cued gestures. In *Proc. CHI* (2008).