TiltText: Using Tilt for Text Input to Mobile Phones

Daniel Wigdor, Ravin Balakrishnan
Department of Computer Science
University of Toronto
dwigdor | ravin @dgp.toronto.edu
www.dgp.toronto.edu

ABSTRACT
TiltText, a new technique for entering text into a mobile phone is described. The standard 12-button text entry keypad of a mobile phone forces ambiguity when the 26-letter Roman alphabet is mapped in the traditional manner onto keys 2-9. The *TiltText* technique uses the orientation of the phone to resolve this ambiguity, by tilting the phone in one of four directions to choose which character on a particular key to enter. We first discuss implementation strategies, and then present the results of a controlled experiment comparing *TiltText* to *MultiTap*, the most common text entry technique. The experiment included 10 participants who each entered a total of 640 phrases of text chosen from a standard corpus, over a period of about five hours. The results show that text entry speed including correction for errors using *TiltText* was 23% faster than *MultiTap* by the end of the experiment, despite a higher error rate for *TiltText*. *TiltText* is thus amongst the fastest known language-independent techniques for entering text into mobile phones.

Keywords: Text entry, mobile phones, tilt input

INTRODUCTION
Most mobile phones are equipped with a simple 12-button keypad, which is an inherently poor tool for generating phrases for a 26-letter alphabet. Using traditional text-entry techniques, such as *MultiTap*, an average text message of 7 words requires roughly 70 key presses. Given estimates (www.gsmworld.com) that in 2003 nearly 500 billion text messages will be sent worldwide from mobile phones, entry using current techniques will require approximately 35 trillion key presses worldwide this year. While much research effort has gone into devising a variety of more efficient text input techniques [9, 15] which have all shown various improvements to the status-quo, none has yet emerged as a new standard. As such, there remains considerable opportunity for researchers to influence this area by developing new techniques.

UIST '03 Vancouver, BC, Canada

TiltText
We have developed a new text input technique, called *TiltText*, which uses the standard 12-button mobile phone keypad augmented with a low-cost tilt sensor. Similar to *TiltType* described by Partridge et al. [10], *TiltText* uses a combination of a button press and tilting of the device to determine the desired letter. Our technique differs from *TiltType* in the keypad used and in the sensing algorithms, which we discuss in detail later in this paper. The standard phone keypad mapping assigns three or four alphabetic characters, and one number, to each key. For example, the `2` key also has the characters `a`, `b`, and `c` assigned to it. *TiltText* assigns an additional mapping by specifying a tilt direction for each of the characters on a key, removing any ambiguity from the button press. The user presses a key while simultaneously tilting the phone in one of four directions (left, forward, right, back) to input the desired character (Figure 1). For example, pressing the `2` key and tilting to the left inputs the character `a`, while tilting to the right inputs the character `c`. By requiring only a single keypress and slight tilt to input alphanumeric characters, the overall speed of text entry can be increased. Further, unlike some techniques [15] that improve on the status quo, *TiltText* is not language dependent, and thus can be used by experts without visually attending to the display screen.

In this paper we first review related work, then discuss implementation issues, then present an experiment comparing the performance of *TiltText* to the most common existing technique, *MultiTap*. We conclude by discussing the characteristics of *TiltText* compared to other techniques, including implications for evaluation metrics.

Figure 1. *TiltText*. The center picture shows the untilted phone where pressing a key enters its numeric value. Left picture: left tilt enters first character on key. Top picture: forward tilt enters second character. Right picture: right tilt enters third character. Bottom picture: tilting towards the user enters fourth character if one exists for that key.

RELATED WORK

There are two areas of research that are relevant to our work: text input techniques for mobile phones, and the use of tilt transducers in mobile devices.

Text Input Techniques for Mobile Phones

A small number of mobile phones today utilize QWERTY style keypads that enable text entry with techniques similar to typing on a regular keyboard, albeit at a much smaller physical scale (e.g., Nokia 5510 www.nokia.com). More recently, hybrid devices that combine phones with PDAs, such as the Handspring Treo (www.handspring.com) and PocketPC Phone (www.microsoft.com), utilize pen-based text input techniques common to PDA's such as Graffiti. While these devices are making small inroads into the mobile phone market, the vast majority of mobile phones are equipped with the standard keypad (Figure 2) which has 12 keys: `0-9`, `*`, and `#`.

Figure 2. Standard 12-key mobile phone keypad

Entering text from a 26 character alphabet using this keypad forces a mapping of more than one character per button of the keypad. A typical mapping has keys `2-9` representing either three or four characters, with space and punctuation mapped to the other buttons. All text input techniques that use this standard keypad have to somehow resolve the ambiguity that arises from this multiplexed mapping. There are three main techniques for overcoming this ambiguity: *MultiTap*, two-key, and linguistic disambiguation. We now review these techniques briefly, and refer the reader to Soukoreff and MacKenzie [15] for a more comprehensive review that is beyond the scope of the present paper.

MultiTap

MultiTap works by requiring the user to make multiple presses of each key to indicate which letter on that key is desired. For example, the letters `pqrs` traditionally appear on the `7` key. Pressing that key once yields `p`, twice `q`, etc. A problem arises when the user attempts to enter two consecutive letters on the same button. For example, tapping the `2` key three times could result in either `c` or `ab`. To overcome this, *MultiTap* employs a time-out on the button presses, typically 1-2 seconds, so that not pressing a button for the length of the timeout indicates that you are done entering that letter. Entering `ab` under this scheme has the user press the `2` key once for `a`, wait for the timeout, then press `2` twice more to enter `b`. To overcome the time overhead this incurs, many implementations add a "timeout kill" button that allows the user to skip the timeout. If we assume that `0` is the timeout kill button, this makes the sequence of button presses to enter `ab`: `2,0,2,2`. *MultiTap* eliminates any ambiguity, but can be quite slow, with a keystrokes per character (KSPC) rate of approximately 2.03 [8].

Two-key Disambiguation

The two-key technique requires the user to press two keys in quick succession to enter a character. The first keypress selects the appropriate group of characters, while the second identifies the position of the desired character within that group. For example, to enter the character `e`, the user presses the `3` key to select the group `def`, followed by the `2` key since `e` is in the second position within the group. This technique, while quite simple, has failed to gain popularity for Roman alphabets. It has an obvious KSPC rate of 2.

Linguistic Disambiguation

There are a number of linguistic disambiguation schemes that utilize knowledge of the language to aid the text entry process. One example is *T9* (www.tegic.com) that renders all possible permutations of a sequence of button presses and looks them up in a dictionary. For example, the key sequence `5,3,8` could indicate any of 27 possible renderings (3x3x3 letters on each of those keys). Most of these renderings have no meaning, and so are rejected. Looking each of them up in a dictionary tells the system that only `jet` is an English word, and so it is the one rendered. Ambiguity can, however, arise if there is more than one valid rendering in the language, in which case the most common is presented. For example, the sequence `6,6` could indicate either `on` or `no`. If the system renders the wrong word, a "next" key allows the user to cycle through the other valid permutations. An analysis of this technique for entering text from an English corpus found a KSPC close to 1 [8]. Newer linguistic disambiguation techniques such as *LetterWise* [9] and *WordWise* (www.eatoni.com) also perform similarly well, with subtle advantages over earlier techniques. While these all have excellent KSPC rates, the success of linguistic-based systems depends on the assumption that users tend to enter "English like" words when sending text messages. As Mackenzie et al. [9] note, users often use abbreviations, and not complete English when text messaging. Further, users of text messaging often communicate in acronyms or combinations of letters and numbers (e.g., `b4` for `before`). Another problem with these linguistic techniques is that users have to visually monitor the screen in order to resolve potential ambiguities, whereas the *MultiTap* and two-key techniques can be operated "eyes-free" by skilled users.

As a result of these limitations of current keypad text input techniques, the quest for a widely applicable, low KSPC, text input technique continues.

Using Tilt Sensors in Mobile Devices

Several researchers have recently proposed interesting interaction techniques that are enabled by incorporating a low-cost tilt sensor within mobile devices [3-7, 10, 12, 13]. While some of this prior art (e.g., [3-7, 12]) do not concern

text entry techniques per se, they do add to the set of possible interactions that could take advantage of tilt sensors embedded in mobile devices, thus providing further justification for the incremental cost of the sensor.

Of particular relevance to our work are two techniques for text entry that use tilt information. Both of these techniques focus on very small devices lacking a large number of buttons, and were not optimized or evaluated for speed of entry. *Unigesture* [13] used tilt as an alternative to button pressing, eliminating the need for buttons for text entry. Rather than having the user make one of 8 ambiguous button presses (as is the present case with mobile phones), *Unigesture* has the user tilt the device in one of 7 directions to specify the group, or "zone", of the character that is desired. The ambiguity of the tilt is then resolved by using dictionary-based disambiguation.

TiltType [10] refines *Unigesture* by adding the combination of button pressing and tilt for entering unambiguous text. *TiltType* was designed to enter text into a small, watch-like device with 4 buttons. Pressing a button triggered an on-screen display of the characters that could be entered by tilting the device in one of eight directions, the appropriate tilt was then made, and the button released. *TiltType* has the same root concept as our *TiltText* technique, in that tilt is used to disambiguate button presses.

Our present work builds upon *TiltType* in several significant ways. First, neither *TiltType* nor *Unigesture* were designed for use with mobile phone keypads, as we are proposing with our *TiltText* technique. We believe that using the standard mobile phone keypad will significantly increase the viability of tilting text input as a real, usable, technique. Second, while *TiltType* uses eight tilt directions, we only use a maximum of four tilt directions, reducing the accuracy demands on the user when tilting. Third, the algorithm used for detecting tilt in the *TiltType* technique is one which we dub *key tilt*, which, as is discussed later in our paper, is not the most optimal tilt detection mechanism for speedy text entry. We develop two alternative tilt detection mechanisms that improve upon *key tilt.* Finally, we present the results of a controlled experiment that provides the first set of usability data with regards to using tilt for text input.

DESIGN ISSUES

TiltText uses the orientation of the phone along two axes to disambiguate the meaning of button presses. Tilting the phone to the left selects the first letter of the key, away from the body the second, to the right the third, and, if present, towards the body the fourth (see figure 1). Pressing a key without tilting results in entering the numeric value of the key. Space and backspace operations are carried out by pressing unambiguous single-function buttons (as in *MultiTap*).

Supporting both lowercase and uppercase characters would require a further disambiguation step since a total of seven characters per key would need to be mapped for keys `2`-`6` and `8`, and nine characters each for the `7` and `9` keys (Figure 2). Adding case sensitivity could be done by either requiring the pressing of a "sticky" shift-key, or considering the magnitude of the tilt as a disambiguator where greater magnitude tilts result in upper case letters, as Figure 3 illustrates. The latter technique, however, would likely make eyes-free entry more difficult.

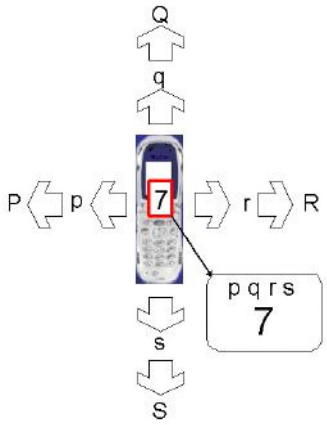

Figure 3. Uppercase text entry with *TiltText*. Tilting beyond a threshold makes the character uppercase.

Techniques for Calculating Tilt

The tilt of the phone is taken as whichever direction has the greatest tilt relative to an initial "origin" value. After exploring various options during our development process, we have found that there are three main ways to determine the tilt value: *key tilt*, *absolute tilt*, and *relative tilt*.

Key Tilt

With this technique, first seen in the *TiltType* work, the amount of tilt is calculated as the difference in the value of the tilt sensors at key down and key up. This requires the user to carry out three distinct movements once the button has been located: push the button, tilt the phone, release the button. We conducted a pilot experiment comparing a *TiltText* implementation that used *key tilt*, and found that user performance with this implementation was much slower than the traditional *MultiTap* technique. For this reason, *key tilt* was not used in our final experiment.

Absolute Tilt

This technique compares the tilt sensor's value at any given time to a "fixed" absolute origin. Only two distinct movements are required to enter a character: tilt the phone, then press the key. In contrast, *key tilt* requires that the key be pressed first, phone tilted, then key released. However, this approach is also not ideal, since in practice users do not maintain a constant arm posture. In order for the tilt value to be meaningful, the fixed origin will have to be reset every time the user's gross arm posture changes. Further, when using *TiltText* to enter two characters requiring tilt in opposite directions, more movement is required using this absolute approach, since the first tilt must be undone, then the new tilt applied. For example, entering the letters `ac` using the `2` key requires an initial tilt of some angle Δ to

the left to enter the a. Then, the user has to tilt the same angle Δ in the reverse direction to return to the origin, before tilting another angle θ to the right to enter the letter c. The total amount of movement is 2Δ + θ, instead of the smaller Δ + θ that one may expect. However, one advantage of this method over *key tilt* is that if successive characters with the same tilt direction are to be entered, then the user can keep the phone tilted at that direction for the successive keypresses.

Relative Tilt

This approach calculates the tilt relative to a floating origin that is set when a tilt gesture begins. The beginning of a gesture is determined by continuously watching for a change in orientation or a change in the direction of a tilting gesture. This approach solves both problems of the absolute tilt method. Since all tilts are relative to the beginning of the gesture, there is no absolute origin that need be reset when changing arm position. Further, opposite direction tilts do not require double tilting, since the second tilt's origin is the end of the first tilt's gesture. So, entering the letters ac requires a tilt of some angle Δ to the left to enter a, then another tilt of angle θ to the right to enter the c, for a total movement of Δ + θ. Note that, just like with *absolute tilt*, when entering only letters, we can enter successive characters with the same tilt direction without re-tilting the phone, by looking at the last significant tilt.

EVALUATION

Goals

We sought to compare the performance of *TiltText* to other techniques for text entry into mobile phones. For this first experiment, we chose as a comparison the most commonly implemented technique, *MultiTap*, because it is the common baseline in almost every other evaluation of text entry techniques reported to date [9, 14, 15]. As such, while the present study only directly compares *TiltText* to *MultiTap*, we can indirectly make comparisons of *TiltText*'s performance relative to other techniques via the results reported in the literature.

Apparatus

Hardware

We used a Motorola i95cl phone. The phone was equipped with an Analog Devices ADXL202EB-232 2-axis accelerometer board to enable tilt sensing, connected to the phone via a serial cable (with the addition of an external power line).

An implementation of a relative tilt system would require regular sampling from the tilt sensor. Unfortunately, our hardware allowed only a reliable rate of ~10 Hz. Pilot studies, using a relative tilt implementation of *TiltText,* showed that participants' text entry speed increased to a point such that our mechanism for determining origins (repeated sampling of tilt at 10Hz) proved insufficient for determining tilt accurately. The resulting device-caused errors in recognition made entering text at higher speeds frustrating. From these pilot studies, we believe that rates of 20-50Hz would be required for a robust relative tilt implementation. As a result, despite the limitations discussed earlier, we had to implement an absolute tilt approach, allowing the user to reset the origin at any time by holding the phone at the desired orientation and pressing "0". The additional movement required by this approach, however, is acceptable for our evaluation purposes because if we can demonstrate that *TiltText* performs well despite this additional movement, then any more robust implementation using a relative tilt approach can only do better. In other words, our evaluation is biased against our new technique.

Because the ADXL board can detect a tilt of only fractions of a degree, only a very small tilt of the phone is necessary to disambiguate a button press. The maximum of the tilt in either axis was taken to be the intended tilt, with a 10% bias towards forward/back. This bias is included based on our pilot studies which revealed a tendency to pitch to the dominant side when tilting forward with the wrist.

Software

The software to read tilts and render text, as well as conduct the experiment, was written in Java 2 Micro-Edition (source at www.dgp.toronto.edu/research/tilttext) using classes from both the Mobile Devices Information Profile (MIDP 1.0) and proprietary i95cl specific classes.

The experiment was conducted entirely on the mobile phone rather than simulating a mobile phone keypad on some other device. All software, including those implementing the text entry techniques, and data presentation and collection software used in the experiment ran on the phone. No connection to an external computing device beyond the tilt sensor was required. Of the major techniques for text entry into mobile phones evaluated in the literature over the last several years, this seems to be the first to do so on an actual mobile phone keypad / display. We believe that this helps to more closely represent real use, and so enhances the external validity of our experiment.

Our *MultiTap* implementation used the i95cl's built-in *MultiTap* engine, with a 2 second timeout and timeout kill. We only considered lowercase text entry in this evaluation. As such, the *MultiTap* engine was modified slightly to remove characters from the key mapping that were not on the face of the button, so that the options available were only the lower case letters and numeral on the key. This matches the traditional *MultiTap* implementation in past experiments, such as *LetterWise* [9] .

Participants

Ten participants volunteered for the experiment. They were recruited from within the university community. There were 5 men and 5 women of whom 3 were left-handed and 7 right-handed. Participants were pre-screened so that no one with any experience composing text using either technique was included. Participants did not receive any tangible compensation for their participation.

Procedure

Participants entered short phrases of text selected from among those in MacKenzie's English phrase dictionary (www.yorku.ca/mack/phrases2.txt). These phrases were chosen because they have been used in previous text entry studies involving *MultiTap* [9], allowing us to leverage this previous work. This corpus' high correlation of frequencies of letters to English is a benefit, although it does not take into account abbreviations commonly used in texting.

The desired text phrases were shown to participants on the screen on the phone. For consistency with past *MultiTap* experiments, participants were instructed to enter text only with the thumb of the hand with which they held the phone, and not to change hands during the experiment.

Timing began when participants entered the first character of the phrase, and ended when the phrase was entered completely and correctly. If an erroneous character was entered, the phone alerted the user by vibrating, and the user was required to correct their error. With this procedure, the end result is error-free in the sense that the correct phrase is captured. Also, the phrase completion time incorporates the time taken to correct for errors.

Before beginning each treatment, participants were told to read and understand the displayed phrase before entering it, and were given instructions for that treatment as follows:

MultiTap instructions: to enter a character using the *MultiTap* technique, first find the key that is labeled with that character. Press that key repeatedly until the desired character is reached. Press once for the first character, twice for the second, three times for the third, and, if present, four times for the fourth. Once you have found the correct letter, and are ready for the next one, you simply repeat the process. If the letter you wish to enter next is on the same key, you must first either press the "right" arrow on the phone or wait two seconds for the cursor to advance.

TiltText instructions: The technique works by tilting the phone in the direction of the letter you wish to enter, then pressing the key on which it is inscribed. For the first letter, tilt left. For the second letter, tilt forward. For the third letter, tilt to the right. For the fourth letter, tilt towards you. The direction of tilt is measured relative to the "centre" or "origin" position of the phone. You can reset the origin at any time by pressing the `0` key.

The experimenter then demonstrated the relevant technique. To ensure that participants understood how the technique worked, they were asked to enter a single phrase that would require tilting in all four directions for *TiltText*, or two successive letters on the same key for *MultiTap.*

Additional instructions were given for both techniques to describe space and delete keys, as well as to enter an extra space at the end of the phrase to indicate completion. The process for error correction was also explained to them. Participants were also directed to rest as they liked between phrases, but to continue as quickly as possible once they had started entering a phrase.

Design

A within-subjects design was used. Participants were randomly assigned to two groups of 5 participants each. The first group performed the experiment with the *MultiTap* technique first, followed by *TiltText*, while the second group did it in the reverse order.

For each technique, participants were asked to complete two sessions of 8 blocks of trials each. Each block required the entry of 2 identical practice phrases, followed by 20 different phrases randomly selected from the corpus. The selection of phrases for each of the 16 blocks were done before the experiment, and presented in the same order to each participant. Phrases were selected such that all blocks had similar average phrase length. The same set of phrases and blocks were used for both techniques. In other words, all participants were required to enter identical phrases in the same order, the only difference being which technique they used first. Participants were asked to rest for at least 5 minutes between each block, and each session of 8 blocks each was conducted on separate days. At least 24 hours passed between sessions for the different techniques to limit interference. In summary, the design was as follows:

10 participants x
2 techniques (*MultiTap* and *TiltText*) x
2 sessions per technique x
8 blocks per session x
20 phrases per block (excluding practice phrases)
= 6400 phrases entered in total.

Results

Data Summary

The data collected from 10 participants took an average of 10.3 minutes per block. A total of 145360 correct characters of input were entered for the 6400 phrases.

Physical Comfort

Some participants reported that their thumb became sore while using both techniques. When this was reported, the participants were encouraged to rest until they felt comfortable to proceed. No participant reported pain or discomfort in their wrist or arms.

Text Entry Speed

We use the standard wpm (words-per-minute) measure to describe text entry speed. This is traditionally calculated as characters per second * 60 / 5. Because timing in our experiment started only after entering the first character, that character should not be included in calculations of entry speed. Thus, for the purposes of these computations, the length of a phrase is n-1 characters. Also, to signify completion, users had to enter an extra space at the end of each phrase. However, our timing considers the entry of the last real character of the phrase to be the end time.

The average text entry speed for all blocks were 11.76 wpm and 10.11 wpm for *TiltText* and *MultiTap* respectively. Overall, *TiltText* was 16.3% faster than *MultiTap.*

The means for the first block of trials were 7.42 wpm and 7.53 wpm, for *TiltText* and *MultiTap* respectively.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.