International Journal of Human - Computer Studies 99 (2017) 37–47

Contents lists available at ScienceDirect

Int. J. Human–Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

A Monte Carlo simulation approach for quantitatively evaluating keyboard layouts for gesture input

Rylan T. Conway[a,*], Evan W. Sangaline[b]

[a] Physics Department, University of California, Davis, CA, USA
[b] National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, MI, USA

ARTICLE INFO

Keywords:
Touchscreen keyboards
Gesture input
Model-based design
Monte Carlo simulation

ABSTRACT

Gesture typing is a method of text entry that is ergonomically well-suited to the form factor of touchscreen devices and allows for much faster input than tapping each letter individually. The QWERTY keyboard was, however, not designed with gesture input in mind and its particular layout results in a high frequency of gesture recognition errors. In this paper, we describe a new approach to quantifying the frequency of gesture input recognition errors through the use of modeling and simulating realistically imperfect user input. We introduce new methodologies for modeling randomized gesture inputs, efficiently reconstructing words from gestures on arbitrary keyboard layouts, and using these in conjunction with a frequency weighted lexicon to perform Monte Carlo evaluations of keyboard error rates or any other arbitrary metric. An open source framework, Dodona, is also provided that allows for these techniques to be easily employed and customized in the evaluation of a wide spectrum of possible keyboards and input methods. Finally, we perform an optimization procedure over permutations of the QWERTY keyboard to demonstrate the effectiveness of this approach and describe ways that future analyses can build upon these results.

1. Introduction

The advent of smartphones and tablets has made the use of touchscreen keyboards pervasive in modern society. However, the ubiquitous QWERTY keyboard was not designed with the needs of a touchscreen keyboard in mind, namely accuracy and speed. The introduction of gesture or stroke-based input methods significantly increased the speed that text could be entered on touchscreens (Montgomery, 1982; Zhai and Kristensson, 2003; Zhai et al., 2009; Kushler and Marsden, 2006). However, this method introduces some new problems that can occur when the gesture input patterns for two words are too similar, or sometimes completely ambiguous, leading to input errors. An example gesture input error is illustrated in Fig. 1. A recent study showed that gesture input has an error rate that is about 5–10% higher compared to touch typing (Bi et al., 2013). With the fast and inherently imprecise nature of gesture input the prevalence of errors is unavoidable and the need to correct these errors significantly slows down the rate of text entry. The QWERTY keyboard in particular is poorly suited as a medium for swipe input. Characteristics such as the “u”, “i”, and “o” keys being adjacent lead to numerous gesture ambiguities and potential input errors. It is clearly not the optimal layout for gesture input.

The rise of digital keyboard use, first on stylus based keyboards in the 1990s and then on modern touchscreens a decade later, has led to a lot of research and development in breaking away from QWERTY to a layout that is statistically more efficient. This work resulted in various improved layouts for digital stylus keyboards such as the OPTI keyboard (MacKenzie and Zhang, 1999), the Metropolis and Hooke keyboards (Zhai et al., 2000), and the ATOMIK keyboard (Zhai et al., 2002). In addition to statistical efficiency, attempts were also made to improve statistical efficiency while simultaneously making the new layout as easy to use for novices as possible (Zhai and Smith, 2001).

More recently, a few keyboards have been introduced that improve text input for certain situations on modern smartphones and tablets: optimizing for the speed of two-thumb text entry on tablets (Oulasvirta, 2013); optimizing tap-typing ambiguity (the SWRM keyboard) and simultaneously optimizing single-finger text entry for speed, reduced tap-typing ambiguity, and familiarity with the QWERTY keyboard (the SATH keyboard) (Dunlop and Levine, 2012); and optimizing the autocorrect feature itself to simultaneously increase the accuracy of word correction and completion (Bi, 2014).

Most of the aforementioned work was done specifically for touch typing since that is the most common form of text input on touchscreen devices. However, the relatively recent rise in popularity of gesture typing has led to some interesting new keyboard layouts that were specifically optimized for improved gesture typing performance. The

* Corresponding author.

http://dx.doi.org/10.1016/j.ijhcs.2016.10.001
Received 14 December 2015; Received in revised form 4 October 2016; Accepted 6 October 2016
Available online 11 October 2016

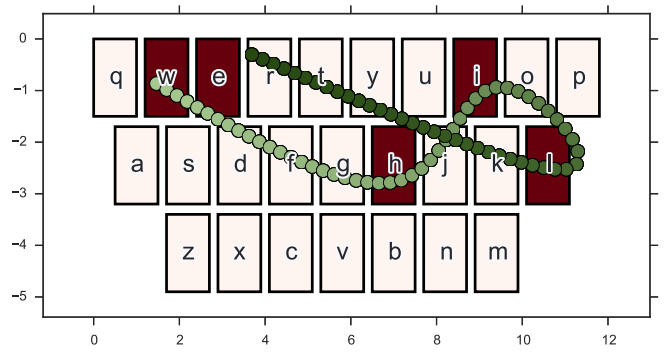

Fig. 1. A gesture input collision between the words “while” and “whole”. The gesture input pattern, represented by the series of green markers, was intended to represent the word “whole” but instead was incorrectly matched with the word “while”. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Square OSK and Hexagon OSK keyboards were optimized to maximize gesture input speed using Fitt's law (Rick, 2010). Various optimizations were also done by Smith, Bi, and Zhai while maintaining some familiarity with QWERTY by using the same layout geometry and only changing the letter placements. This resulted in four new keyboards: the GK-C keyboard, which is optimized to maximize gesture input clarity; the GK-S keyboard, which is optimized for gesture input speed; the GK-D keyboard, which was simultaneously optimized for gesture clarity and speed using Pareto optimization; and the GK-T keyboard, which was simultaneously optimized for gesture clarity, gesture speed, and QWERTY similarity (Smith et al., 2015).

Evaluating and comparing various keyboard layouts is a difficult problem given the complexity and variability associated with text entry. Measuring text entry and error rates from user based trials is typically done to evaluate or directly compare the effectiveness of various keyboards and input methods. These studies usually require the test subjects to transcribe a set of predefined phrases using a specified input device. Text entry evaluations of mini-QWERTY keyboards (Clarkson et al., 2005), chording keyboards (Lyons et al., 2006), handwriting recognition systems (Kristensson and Denby, 2009), and various gesture input systems (Castellucci and MacKenzie, 2008; Wobbrock, Chau, and Myers, Wobbrock et al.2007) have all been done using this approach. The main downside of this approach is the fact that in day-to-day use most users spend very little time transcribing phrases. The majority of text entry is done by composing original phrases. Therefore, text entry evaluations from transcription based user studies are not realistic and can introduce unintended biases into the results. Vertanen and Kristensson showed how these biases can be mitigated by including composition tasks in user trials to complement the standard transcription tasks (Vertanen and Kristensson, 2014).

Despite the recent work done to improve text entry evaluations with user based studies the metrics used for optimization are typically based on surrogate models of the actual performance characteristic of interest. For example, the gesture clarity metric used by Smith et al. (2015) is correlated with how frequently words are correctly reconstructed but does not measure this directly. The reason that these approximate measures have been used is that accurately evaluating real keyboard reconstruction error rates would require an immense amount of user input data. Modern optimization techniques typically evaluate and compare hundreds of thousands of different keyboard layouts, making it completely infeasible to obtain the necessary user data. The methodology that we propose allows for the direct evaluation of gesture reconstruction error rates, or any other desired metric, by simulating realistic user interactions with a keyboard. This is similar to the approach used by Fowler et al. when they simulated noisy tap typing input to estimate the effect of language model personalization on word error rate (Fowler et al., 2015).

To demonstrate the effectiveness of our methodology we will show how it can be used to find a keyboard layout that minimizes gesture input errors. This requires accurately modeling gesture input for a given layout, interpreting the input, and quantifying how frequently typical inputs would be misinterpreted. We employ several different models for gesture input and a dictionary of the most common words in the English language to simulate realistic usage and take into account variations between users. We also attempt to develop a highly accurate algorithm for recognizing gesture inputs that is not limited to a specific keyboard layout. It should be noted that although this paper focuses on the error rate performance, the overall methodology can be used to evaluate and compare keyboard layouts based on any performance measure.

Finally, In order to address the problem we designed and built an open source software framework, called Dodona, for exploring different input methods. This framework is well suited for examining a wide range of possible keyboard designs and input models. It was built with optimization in mind and has a focus on efficient implementations and extensibility. The library is freely available on GitHub (Conway and Sangaline, 2015) and was used to perform the analysis and generate all keyboard related graphics presented here.

2. Modeling swipe input

An extremely large dataset of gesture inputs is needed in order to accurately evaluate the error rate of a given keyboard layout. The only way to obtain such a dataset on a reasonable time-scale is to generate gesture input data based on models of user input. To accomplish this we developed several models which can take a word and produce what we refer to as a gesture input vector, a sequential series of $(x,\ y,\ t)$ points that represent discrete samples along a gesture input pattern. We then used words that were randomly generated based on their frequency of use in the English language to feed into these models and generate realistic sets of input.

2.1. Input vectors and interpolations

In general, our input model can produce either a “random vector” or a “perfect vector”. The former is used for realistic, inexact gesture input while the latter represents the ideal input pattern that is free from variation. To construct random vectors we begin by drawing control points for each letter in a given word from a two dimensional Gaussian distribution that's centered around each corresponding key on the keyboard. The x and y widths of the Gaussian, in addition to the correlation in the offsets between subsequent control points, can be changed as parameters of the input model. We then interpolate between these control points for each letter to produce a continuous gesture input as a function of time. This is then sampled at evenly spaced intervals along the entire interpolation in order to produce an input vector with a set number of points. Perfect vectors are constructed in the same way but use the centers of the keys as control points. The idea that their exists a unique perfect vector for each word in the lexicon was first introduced by Kristensson and Zhai in their seminal paper about the SHARK^2 text input system for stylus keyboards (Kristensson and Zhai, 2004). In their work they refer to perfect vectors as *sokgraphs*.

We chose to implement a variety of different interpolations to account for the variations in individual gesture input style. We settled on five different interpolation techniques: a straight-line spatial interpolation, a natural cubic spline, a cubic Hermite spline (Bartles et al., 1998), a monotonic cubic Hermite spline (Dougherty et al., 1989), and a modified natural cubic spline where the first and last segments are required to be straight lines.

Using randomly generated control points with various interpolation techniques allows us to capture a large range of input possibilities. This is demonstrated in Fig. 2, which shows five different possible swipe