Uncertain Text Entry on Mobile Devices

Daryl Weir[1], **Henning Pohl**[2], **Simon Rogers**[1], **Keith Vertanen**[3], **Per Ola Kristensson**[4]
[1]University of Glasgow, UK, [2]University of Hannover, Germany,
[3]Montana Tech, Montana, USA, [4]University of St Andrews, UK
darylw@dcs.gla.ac.uk, henning.pohl@hci.uni-hannover.de, simon.rogers@glasgow.ac.uk,
kvertanen@mtech.edu, pok@st-andrews.ac.uk

ABSTRACT
Users often struggle to enter text accurately on touchscreen keyboards. To address this, we present a flexible decoder for touchscreen text entry that combines probabilistic touch models with a language model. We investigate two different touch models. The first touch model is based on a Gaussian Process regression approach and implicitly models the inherent uncertainty of the touching process. The second touch model allows users to explicitly control the uncertainty via touch pressure. Using the first model we show that the character error rate can be reduced by up to 7% over a baseline method, and by up to 1.3% over a leading commercial keyboard. Using the second model we demonstrate that providing users with control over input certainty reduces the amount of text users have to correct manually and increases the text entry rate.

Author Keywords
Mobile text entry; keyboard error correction

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces—*Input devices and strategies, Interaction styles*

INTRODUCTION
Touchscreen keyboard input is inherently *uncertain*. This uncertainty stems from a number of sources. When trying to press a key, users often touch a location offset from their intended target—this is the well known *Fat Finger Problem* [32]. Further, users might be unsure about the spelling of some words, leading to incorrect input. Finally, touch sensor readings also introduce a degree of uncertainty.

There is a growing body of research on using probabilistic techniques to model this uncertainty in order to improve the performance of touch interactions. For text entry, there are two primary approaches. The first is offset modelling, which aims to predict the intended touch target given the recorded touch location. Weir et al. [37] used a Gaussian Process (GP) regression approach to predict the distributions over possible touch locations. Goodman et al. [9], Kristensson and Vertanen [17] and Bi and Zhai [2] used information about the location of on-screen targets to obtain the probability that a given touch was meant for a given target. The second approach is language modelling, using statistical properties of text to obtain the most likely interpretation of a stream of key presses. For example, Goodman et al. [9] showed that it was possible to reduce error rates by a factor of 1.67 to 1.87.

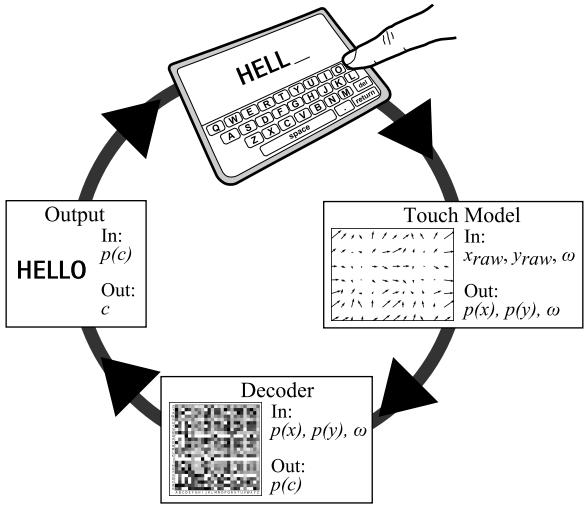


Figure 1. We present an autocorrect method for typing on a pressure-sensitive touchscreen. Given a touch location $(x_{\text{raw}}, y_{\text{raw}})$ and a touch pressure (ω), a touch model assigns probabilities to keys while a language model assigns probabilities to possible words. A decoder algorithm combines these to make a prediction on the entered text. Pressure information can be used to control the text correction behavior.

We present a flexible decoder that combines state-of-the-art probabilistic touch models with a long-span language model (Figure 1). Our decoder searches for the most likely text given the uncertainty of the tap locations, possible missing key presses, and possible extra key presses. Using our decoder we have tested a variety of correction strategies, such as whether the decoder is free to change previous characters.

We consider two touch models representing different approaches to modelling and controlling touch uncertainty. The first is GPType, which uses a GP to model user specific touch offsets and their variance in different areas of the screen. The GP is based on training data and hidden from the user. Thus GPType represents an *implicit* uncertainty model, mirroring existing autocorrection techniques in commercial devices.

In our second system, ForceType, we allow users to explicitly control the level of uncertainity in their input via pressure information. This gives users control over the relative influ-

ence of the language model (LM) and allows them to handle situations where typical correction systems might fail. For example, when typing words not found in the training corpus of the LMs, mixing languages, or using slang words, text may be autocorrected to something unintended. Using existing systems, users have to take action to revert to the entered text. While users can be casual and allow their phones to correct their mistakes for most text, they can elevate their level of control for phrases where they feel the autocorrection algorithm may go astray.

RELATED WORK

Text Entry Correction Approaches

Several on-screen keyboard correction methods have been explored in the literature [16]. Goodman et al. [9] were the first to combine a language model and a touch model to increase accuracy on soft keyboards. They determined the best key sequence for a touch sequence by maximizing the likelihood, significantly reducing error rates. Kristensson and Zhai [20] presented an alternative approach that used pattern matching to identify the words that most likely corresponded to users' tap sequences. Gunawardana et al. [10] presented an approach with dynamic key-target resizing which defined a minimum target size, preventing key targets from shrinking too much. Rudchenko et al. [29], with their *Text Text Revolution* game, showed that per-user touch models and personalized key-target resizing can further improve typing accuracy. Kristensson and Zhai [19] presented the gesture keyboard *Shark*^{2}, which combined a shorthand writing system with a language model. Goel et al. [8] showed how accelerometer data can be used to correct typing errors resulting from inaccuracies due to walking.

Offset Modelling

On any touchscreen device, the location a user touches and the location the user *intended* to touch are offset. This has been studied as a perception problem based on how users target things hidden by their fingers [14]. This is also caused in part by the softness of the finger causing a large touch area—the so-called *Fat Finger Problem* [32].

A body of research exists based on modelling and correcting touch offsets. For example, Henze et al. [11] collected millions of touches in an Android typing game and modelled offsets using polynomials to improve text entry accuracy. Weir et al. [37] demonstrated offset modelling using a GP regression model, and in particular showed the importance of user specificity in offset models. Bi and Zhai [1] extend Fitts' law to finger touch using a dual Gaussian distribution approach. Bi and Zhai [2] further extend this to handle target selection on a touchscreen using their *Bayesian touch criterion*, demonstrating a significant increase in accuracy.

Pressure Input

Pressure has been used in interactive systems for a wide range of applications. Ramos et al. [26]'s *pressure widgets*, showed how stylus pressure can be used in selection tasks and how many pressure levels can be discriminated. *Force Gestures* by Heo et al. [12] augmented tapping and dragging operations with pressure to extend the available gesture set. A more general investigation of pressure-based interaction was done by Stewart et al. [34] who looked at how holding a device influences target acquisition times.

There is some work on combining typing with pressure data. For keypads, McCallum et al. [23] showed how pressure-based disambiguation allows faster text entry than multi-tap. Shi et al. [31] showed that using a fisheye function for pressure-based selection reduces error rates. Brewster and Hughes [3] used pressure to allow the selection of letter case when typing. Clarkson et al. [5] presented a way of marking messages as urgent when typed more forcefully which is similar to our approach of giving more weight to more forceful typing. In *One-press control* [6], pressure is used to create a new set of modifier keys. *TypeRight* [13] is a physical keyboard that prevents errors by increasing the resistance of keys for out of vocabulary (OOV) words.

Autonomy Handover

The ForceType system presented in this paper allows users to influence how much control they want to give to autocorrection. Flemish et al. [7] referred to this concept as the *H-metaphor* and liken it to riding a horse where people can 'loosen or tighten the reins' to vary their level of control. Williamson [38] looked at interactions as a control process, where the level of uncertainty influences how much control is exerted and what kind of feedback has to be provided. Uncertain input for GUIs has been investigated by Schwarz et al. [30], who tried to enable vague input handling for standard widgets. Pohl and Murray Smith [25] frame the change of control as a *casual interaction continuum*, where users themselves choose how much they want to engage, allowing them to refrain from tight control when close interaction with their device is frowned upon socially.

THE GPTYPE SYSTEM

GPType is a correction system consisting of three parts: (1) a LM which assigns probabilities to sequences of text, (2) a touch model, which assigns probabilities to each key on a keyboard given a touch location, and (3) a decoder which combines the LM and touch model probabilities and decodes what the user was trying to type.

Language Model

We adapted a language model that has previously been used for a thumb-typing touchscreen keyboard [24]. The Twitter-based language model was trained based on 778M tweets sent between 12/2010 and 6/2012. Duplicate tweets, retweets, and non-English-language tweets were eliminated via a language-identification module [21] (with a confidence of 95%). Based on a tweet's source string we removed all tweets that did not originate from a mobile device. Tweets were split into sentences and we only kept sentences where all words existed in a list of 330K words drawn from Wiktionary, Webster's dictionary, the CMU pronouncing dictionary, and GNU aspell. The final dataset consisted of 94.6 M sentences, 626 M words, and 2.56 G characters.

The language model was built using the SRILM toolkit using a vocabulary of A-Z, space, apostrophe, comma, period, exclamation point and question mark. The character-based 7-gram language model was smoothed using Witten-Bell and

no count cutoffs. The model was then entropy-pruned to reduce the memory footprint in anticipation of using the model on a mobile device. The final model had 225 K n-grams and a compressed disk size of 2.0 MB.

Touch Model

To generate input probabilities for the decoder, we use GP regression to model each user's touch offset function. GPs are flexible, non-parametric statistical tools commonly used for regression and classification. Given 2D touch locations $\mathbf{s} = (x, y)$, the GP learns a function which maps to the offsets (Δ_x, Δ_y). The GP is defined in terms of its mean function $\mu(\mathbf{s})$, which we choose to be zero since in the absence of data we wish to predict no offset, and its covariance function $C(\mathbf{s}_n, \mathbf{s}_m)$, which defines how similar the n-th and m-th outputs should be given the corresponding inputs. A full treatment of GP regression and classification is given in [27], and an example of using a GP for touch offset modelling can be found in [37].

Following Weir et al.[37], we choose a linear combination of a linear and a Gaussian covariance function:

$$C(\mathbf{s}_n, \mathbf{s}_m) = a\mathbf{s}_n^T \mathbf{s}_m + (1-a)\exp\left\{-\gamma||\mathbf{s}_n - \mathbf{s}_m||_2^2\right\},$$

where a controls the relative influence of the terms, and γ is the scale parameter of the Gaussian function.

The advantage of the GP over a parametric method such as a polynomial (used in e.g. [11]) is that the prediction is a distribution over possible offset values. For text entry, we use this to obtain probabilities over the different keys on the keyboard for a given touch. The probability of a given key is found by integrating the density function of the predictive Gaussian over the rectangular area of the key. In practice this is not possible analytically, so we approximate the probability by sampling from the Gaussian, counting which keys the samples fall into, and then normalising these counts to obtain probabilities.

The sizes of the predictive Gaussians, and consequently the probabilities produced by the model, are learned from training data. Thus, in areas of the screen where the offsets are more variable, the Gaussians are larger and the probabilities more diffuse. This gives the language model more latitude to correct inputs in these areas. The GP models the uncertainty in the touch interaction and uses information about that uncertainty to make predictions. This is an advantage over other probabilistic approaches, such as that proposed by Bi and Zhai [2], which model the touch uncertainty with fixed parameters for all users. The tradeoff is that users must provide calibration touches before using the system.

Decoder

We created a decoder which searches for the most probable character sequence given a sequence of taps. Each "tap" is in reality a probability distribution over every keyboard character. These tap distributions were set according to the GP tap model as well as the ForceType model.

As we will describe shortly, we explored four different correction strategies in our interface. Each strategy involved changing how much of the tap sequence the decoder was allowed to change. To facilitate this, the decoder allows some taps to be marked as fixed. Fixed taps served as language model context, but were not eligible for change during the decoder's search.

The decoder searches in the space of all possible character sequences for the non-fixed taps in an observed sequence. During this search, a tap would most likely generate the actual key hit. But the decoder also allows substitution with all other possible keys according to the provided tap distribution.

Our decoder allows an observed tap to be deleted without generating a character. Similarly, it explores inserting all possible characters without consuming an observation. Both insertion and deletion hypotheses incur a configurable penalty.

During its search, the decoder makes use of a character language model. As each output character is generated, we combine the tap probability with the probability of the character given the language model by taking a weighted product. The relative contribution of the tap distribution and the language model is controlled by a configurable scale factor.

The search over all possible character sequences is exponential in the length of the sequence. Pruning is thus critical to ensure real-time performance. During its search, any hypothesis that becomes less probable then the current best answer is pruned. Additionally, we employed beam width pruning. In beam width pruning, the decoder tracks the best hypothesis found thus far for each position in the tap sequence. Hypotheses that are too improbable (i.e. outside the beam width) compared to the best one at a particular position are pruned. By varying the beam width, we can control the speed accuracy tradeoff during recognition. The free parameters of the decoder were optimized with respect to the data we used to learn the GP probabilities.

STUDY 1: GP AND CORRECTION STRATEGIES

In order to build our text correction system, we required training data. This data came in two parts: (1) training points for the GP offset model, and (2) typing data to which we could apply various correction strategies to identify the optimal technique. This section details the procedure conducted to collect this data and the details of the resulting model.

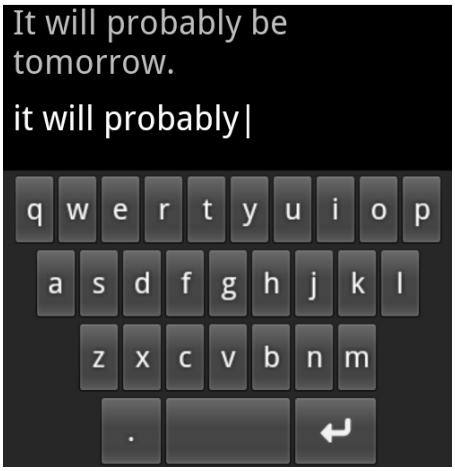
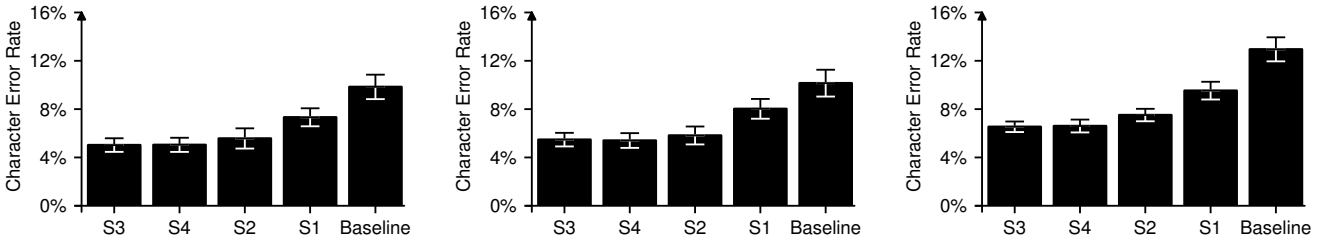


Figure 2. A screenshot of our logging application. The target phrase, the currently entered text and our simplified keyboard are shown.



(a) Sitting **(b) Standing** **(c) Walking**

Figure 3. Character error rates after applying our four different correction strategies to the typing data gathered in our model building study, separated by mobility condition (Study 1). Plots show mean and standard error across all participants. The baseline method represents the literal keys touched.

We built our models based on data collected from 10 participants (4 female) who volunteered for three 45 minute sessions. Participants ranged in age from 19–31 (mean = 25.4, sd = 4.25). All participants were smartphone owners, and rated themselves as intermediate to expert users of touchscreen devices. At the end of the data collection period, participants were paid £10 for their time.

To collect training data for the GP, we built a custom keyboard running on a *Samsung Galaxy SII* smartphone, with Android version 4.0. The keyboard had a simplified layout consisting only of the alphabetic keys, space bar, period and an enter key. Participants were asked to press a sequence of keys until 10 presses were logged for each key. When a key was pressed, the keyboard logged the time and touch location, as well as the location of the center of the requested key. Touches which were more than 3 key widths from the intended key were filtered out, as these were deemed likely to be mistakes. Participants repeated this process in three mobility conditions: sitting, standing, and walking. We chose not to use a pace setter for the walking condition, instead asking participants to move at whatever speed they were comfortable walking and typing. Participants walked in a set path, and their laps were timed so that walking speed could be determined. We trained our GPs offline using MATLAB. The covariance function parameters were optimised on a user specific basis using 10-fold cross validation to find the model which minimised the RMS error between the offset touch locations and the centers of the target keys. The implicit assumption that users target the key center is without loss of generality—if in truth they target some other feature, the offsets would simply be shifted.

Each participant also provided typing data. We used phrases from the Enron Mobile Email dataset [36] as stimuli. This phrase set consists of phrases drawn from genuine mobile emails and has been shown to result in similar text entry performance as the MacKenzie and Soukoreff [22] phrase set [18]. We filtered out any phrases containing characters not on our keyboard, and then removed sentences with fewer than 4 or more than 10 words. This left a set of 427 phrases. Participants were shown a random subset of these and asked to type them as quickly and accurately as possible using our custom keyboard. As there was no backspace key, participants had to leave any mistakes uncorrected. This was done so that we could evaluate the quality of our model's corrections on the text as entered. During each of the three sessions, participants typed for 10 minutes in each mobility condition. The order of mobility conditions was counterbalanced across both participant and session. Participants were instructed to hold the phone in two hands and type with their thumbs. The stream of touches for each typing session was passed through the best GP for that participant to obtain offset touch locations and key press probabilities. These probabilities were then passed through the decoder to obtain the corrected sentences. We evaluated four correction strategies:

S1 Single-Key Correction—for each key press, we input the letter with the highest combined LM/GP probability, holding all previously entered characters fixed.

S2 Modifiable Context Correction—as S1, except that the decoder is also free to change previously entered letters.

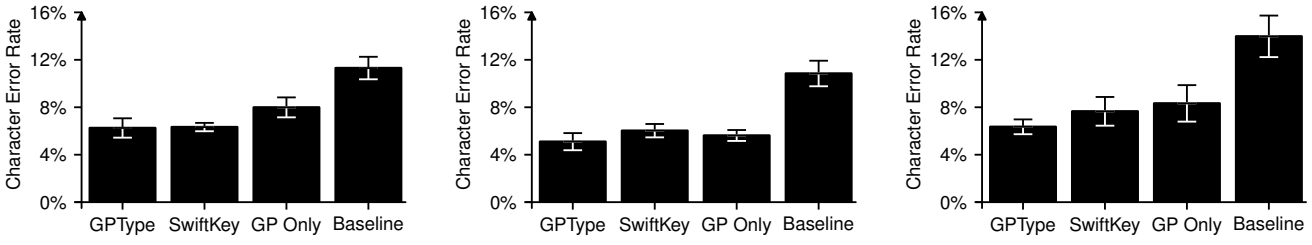
S3 Word Correction—when the user types a word delimiter (space or period), the system uses the LM and GP to identify the most likely word, holding previously entered words as fixed context.

S4 Single-Key + Word Correction—combination of S1 & S3.

We measured performance in terms of the character error rate (CER) after correction, averaged across all phrases and all users for a given mobility condition. CER is the number of characters that need to be inserted, substituted, or deleted in order to transform the corrected text into the reference text, divided by the number of characters in the reference text. For each correction strategy, we also tested a range of values for the cost and maximum number of insertions and deletions, the beam width of the decoder's search, and the relative scaling of the probabilities from the LM and GP.

Our results are summarised in Figure 3. These plots show the mean CER achieved for each correction strategy, along with the baseline error between the characters as typed and the stimuli.

In general, we found that S3 and S4 were the optimal correction strategies — there was no significant difference between them for any mobility. S2 was slightly worse than either of these. In theory S2 represents the most thorough search, since at each new tap the system is allowed to change all previous letters. In practice, multiple such corrections can introduce insertion or deletion errors which accumulate over time, leading to the increased error rate seen here. S1 is the worst of the correction schemes, since it is only able to fix cases where the wrong key was hit, and cannot correct transposition errors



(a) Sitting **(b) Standing** **(c) Walking**

Figure 4. Character error rates for the two keyboards we evaluated, separated by mobility condition (Study 2). Plots show mean and standard error across all participants. The baseline method represents the literal keys touched, while GP Only shows the keys hit after the mean GP offset is applied.

such as typing 'teh' in place of 'the'.

The observed error rates were not significantly different between the sitting and standing mobility conditions, but the error rates for the walking condition were significantly higher (paired t-test, $p < 0.05$). The lowest error rates obtained using the optimal correction strategies were 5.02% for sitting, 5.47% for standing, and 6.5% for walking.

STUDY 2: COMPARISON WITH SWIFTKEY

We evaluated our best correction system in a study. The goal of this study was to determine whether the benefits shown in our offline simulations were reflected in a live typing task. We also wanted to assess how our system compared to an existing commercial soft keyboard. For this we chose SwiftKey[1], a popular Android keyboard which also uses language modelling to perform intelligent correction.

We ported the GP, LM and decoder to Android and adapted our custom keyboard from the model building study. We used S3 as our correction style, since it was as good at correcting as S4 and required fewer LM searches. We used 5 examples of a user's touch per key to train the GP, rather than the 10 used in model building, in order to prevent input from slowing down due to large matrix operations.

Participants

We recruited a further 10 participants (3 female) to take part in this evaluation. Ages ranged from 18–28 (mean = 22.4, sd = 4.22). 8 participants were smartphone owners and considered themselves expert users. The other 2 did not own smartphones and had little experience using touch screen devices. No participant in this study took part in the previous study. Participants were paid £10 for their time.

Apparatus

Participants typed using our custom logging application, again running on a Samsung Galaxy SII. This smartphone has a 4.3 inch screen with a 480×800 pixel resolution. A screenshot is shown in Figure 2. The stimulus phrase appears at the top of the screen, and the text entered by the participant is shown below. The time remaining in the current typing task is shown in the upper right. Visually, this application was identical to the one used in the model building study. The backspace key in the logging app was disabled, as we wanted to evaluate the quality of the corrections made by our system without participants manually backspacing to correct errors.

As mentioned above, we also collected typing data using the SwiftKey keyboard. The keys on our custom keyboard have the same size and layout as those in SwiftKey, so the two logging interfaces were very similar. As SwiftKey is a third party product, it was not possible to disable the backspace key and so participants were instructed not to use it and accept any corrections from the keyboard.

Procedure

The procedure for the study consisted of three sessions. In the first session, participants provided calibration data for the GP in each of the three mobility conditions. This was collected in the same way as in the model building study. Afterwards, participants typed for five minutes, while seated, on each of the two study keyboards to familiarise themselves with the layouts. As in Study 1, participants typed phrases from the Enron Mobile Email set [36]. In each of the other two sessions, participants performed 10 minutes of typing in each mobility condition. They used GPType in one session, and SwiftKey in the other. All tasks were carried out in a meeting room, with a clear path marked for walking.

Design

Our study was a within-subjects 2x3 factorial design with factors: Keyboard (levels: GPType, SwiftKey) and Mobility (levels: Sitting, Standing, Walking). The presentation order of the keyboards was counterbalanced across participants, and the order of mobility conditions was partially counterbalanced across participants and between sessions. A full counterbalancing was impossible with 3 levels and 20 sessions.

Results

We measure performance of the corrections produced by each keyboard in terms of CER between the corrected text and the stimulus phrase. Our results are summarised in Figure 4, which shows the mean and standard error across all participants, separated by mobility condition. The baseline is the CER between the keys hit by the participant's touches and the stimulus phrase. Also shown is a *GP Only* condition, in which we apply the mean offset from the GP to the user's touch and see which key was hit by the resulting touch. The GP Only condition does not use the decoder or LM.

Both keyboards offer a significant improvement over the baseline in all mobility conditions (paired t-test, $p < 0.05$). The

[1] `http://www.swiftkey.net`

CER reduction over the baseline for GPType was 4.9% for sitting, 5% for standing, and 7.6% for walking. Further, GPType offers a small but significant improvement over SwiftKey in the standing and walking mobility conditions (approximately 1% reduced CER for the standing condition and 1.3% for walking). No significant difference between the keyboards was observed in the sitting condition. This is likely an effect of the participants' touch offsets becoming more pronounced when standing or moving. Interestingly, the standing condition had the lowest CER for both baseline and the evaluated keyboards. It is unclear why this should be the case.

Also of interest is the fact that the GP Only condition is significantly better than the baseline. This indicates that both the offset modelling and the decoding play a role in producing the observed reduction in error rate. By computing the mean touch offset, many substitution errors can be corrected. The decoder can then decrease the error rate further by fixing transposition errors and performing insertions or deletions.

We also measured participants' text entry rates for both keyboards. However, we saw no significant difference between GPType and SwiftKey. This is perhaps to be expected, given that the physical layout of the keyboards was the same and that backspace was disabled, so the observed entry rates do not subsume manual error correction.

THE AUTOCORRECT TRAP

So far we have explored a novel approach of error correction for touchscreen keyboards. Current touchscreen keyboards use a wide range of other autocorrection techniques (e.g., adaptive but clamped target resizing [10]). These techniques, as well as our GPType method presented here, implicitly model uncertainty: the user has little influence on the way autocorrection works. In any such system, users might be able to reject a proposed correction, delete a character and retype with autocorrection switched off, or select the corrected word and pick from the originally typed version, or other suggestions. While this theoretically allows users to control text correction according to their needs, there are still widespread frustrations and situations where text prediction falls short. When typing unknown words, using regional dialects, or mixing languages, autocorrection is often not flexible enough.

We thus propose giving users control over how their phones correct text. Empowering users to vary the level of error correction per word allows users to fall back to automatic error correction for phrases they deem correctable while being able to tighten the reins during phrases they feel their phones cannot handle. This assumes that users have a sense of whether a phrase is hard for the text prediction on their phone or not. To get an idea of how the capabilities of auto-correction are viewed by users, we conducted a study.

We designed a questionnaire listing 20 phrases of varying levels of difficulty—some consisting only of common English words and others with proper nouns, slang, and/or words or phrases borrowed from other languages (e.g., *summa cum laude*). Participants were recruited using mailing lists and social media. For each phrase, participants were asked to rate on a 5-point Likert scale whether they thought autocorrect would change it or not when entered on a smartphone.

We received 28 responses (8 female, ages 17–44, mean = 29.0, sd = 7.5) to the questionnaire. Asked to rate their English language skills, 86% of participants rated themselves as functionally native speakers. The rest of the participants still rated themselves at a near native level.

To establish a ground truth on whether our phrases would be autocorrected or not, we entered them into three smartphones—an iPhone 5 running iOS 6.1, an LG E700 running Windows Phone 7.5, and a Samsung Galaxy S3 Mini using the SwiftKey keyboard on Android 4.1. We noted for each phone which phrases were autocorrected. In cases of disagreement, where some phones corrected and others did not, we used majority voting to produce a singe binary value for each phrase. Next, we split the user ratings for corrected and uncorrected phrases and produced histograms showing the counts for each rating. By normalising these histograms we obtain discrete probability distributions over the ratings for the two groups of phrases. Our results are shown in Figure 5. For phrases which would actually be autocorrected, the most probable rating is 1 (certain correction) and the least probable is 5 (no correction). The monotonic decrease across the other ratings shows a clear trend, indicating our respondents had a good sense of phrases that would be autocorrected.

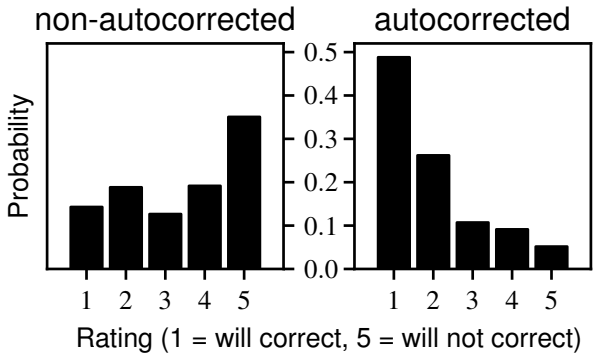


Figure 5. For a set of 20 phrases, we asked 28 people whether they thought their phone's autocorrect would change it when entered, or leave it unchanged. Participants gave a rating between 1 (will definitely be changed) to 5 (will definitely not be changed). We classified each phrase by testing it for autocorrection on several different phones. This figure shows the probability of each point on the rating scale for both classes (changed and unchanged).

For phrases that were not corrected, the trend is less pronounced. The most likely rating is for no correction and there is a remarkably high probability that users rate these uncorrectable sentences as certain or near certain corrections. In essence, users overestimate how likely autocorrect systems are to take action. This is not necessarily an issue—if a user takes action to prevent autocorrection when none would have occurred, there is no difference to their final input. These results motivate the need for a system with finer control over automatic error correction behaviour. The distributions over ratings are significantly different (Wilcoxon test, $p \ll 0.01$) for the two groups of phrases. This suggests that users have a functional mental model of the way autocorrect operates and would be able to identify situations where preventing autocorrection is desirable.

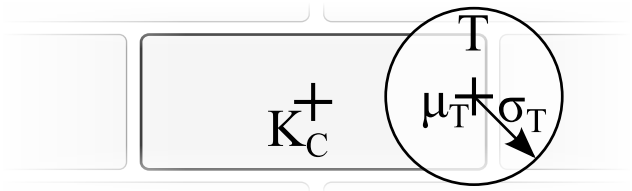


Figure 6. The touch model used by our correction system ForceType. For each key K, we evaluate the likelihood of its centre K_C under a Gaussian on the touch point μ_T. The standard deviation σ_T is controlled by pressure. Higher pressure causes a narrower distribution.

FORCETYPE: PRESSURE AS CERTAINTY

Given that users already have a mental model of which words are likely to be autocorrected, we desire an input modality which enables them to express certainty about words they do not want corrected. In essence, we wish to allow users to negotiate control between themselves and the language model. For normal typing, the autocorrect functions as it does on current phones, but when the user suspects the word they are entering is unknown to the phone's dictionary they can smoothly limit the autocorrect behaviour. We choose pressure as the factor to control this negotiation. When the user wishes to indicate certainty, they simply press harder on the keys. Srinivasan and Chen [33] have shown pressure to be controllable by users and the idea of pressing hard for a different input behaviour is simple to grasp.

Other research has considered negotiated uncertainty through user input. Rogers et al. [28] use the height of the finger above the screen as a proxy for uncertainty when browsing a map. However, although 'hover' input of this form is now being introduced on commercial phones, such as the *Samsung Galaxy S4*, it is not an appropriate choice for a typing application. Subramanian et al. [35] showed that users display unintentional drift when varying hover height, so controlling uncertainty via this modality may be suboptimal—hence our choice of the pressure modality.

Correction Model

We use the same language model and decoding algorithm as for GPType, but rather than generating key press probabilities from a GP we use a pressure dependent touch model. The likelihood of a key given a touch is computed as a Gaussian with a standard deviation that changes dependent on the pressure. In particular, we take the standard deviation as: $\sigma_T = C/\omega_T$, where C is a constant and ω_T is the pressure for touch T. Thus for high pressure touches the variance is small and the probability mass is concentrated in the pressed key, whereas for lower pressure touches the probability mass is spread over the keyboard, allowing correction to take place. The touch model is illustrated in Figure 6.

The constant C needs to be calibrated appropriately. We chose a value such that the distribution for a 'typical' touch had a standard deviation equal to half a key width. To determine what defined 'typical', we asked 10 participants to repeatedly write *the quick brown fox jumps over the lazy dog*, an English pangram, on our input device, a pressure sensitive touch pad called the ForcePad (more information on this hardware is given in the next section). This was done without presenting any feedback about what was typed, and in the absence of any correction scheme. This provided us with pressure information for typing with every key on the keyboard. From a histogram of the logged pressure information, it was the clear the data were not normally distributed and had high positive skew. We chose to fit a gamma distribution to the data. Next, from the 2333 collected datapoints we removed 144 outliers (>2 sd away from the mean — these were primarily points where the hardware reached its upper sensing limit). We then refitted a gamma distribution to the remaining data. The mean of this distribution was 168.28 g/cm^2 (sd = 106.20 g/cm^2). We then chose C such that the Gaussian distribution for a touch with this mean pressure had a standard deviation equal to half a key width.

STUDY 3: FORCETYPE EVALUATION

We conducted a user study to determine the effects of pressure-based scaling of text correction. We hypothesize that:

H1 Pressure-adaptive error correction requires fewer word corrections by the user than constant-scale text correction.

H2 Users are faster when typing words unknown to the error correction algorithm when utilising pressure adaptation.

To test these hypotheses, we had participants type a series of phrases in two conditions: with and without pressure adaptation. For the latter, we use the same correction model but rather than adapting the variance of the touch Gaussian based on pressure, we used a fixed value, chosen such that the standard deviation was equal to one key width. This corresponds to the standard deviation in the pressure sensitive model when the touch has mean pressure and approximates the behaviour of autocorrect on modern smartphones.

We used a between-subjects design for the study, so that each participant used only one correction model. This decision was made due to the difficulty of having a participant learn the pressure sensitive model and then asking them to "type normally" for the other condition. The conditions could thus not be properly counterbalanced in a within-subjects design.

In order to assess the impact of using the model on the text entry speed for the pressure group, the phrase set used needs to contain both *correctable* phrases, containing only words known to the correction model, and *uncorrectable* phrases, containing at least one word unknown to the model. The former should not require pressure typing invocations if the user has a good understanding of the language model behavior, while the latter set will require one or more invocations.

Participants

We recruited 16 participants (5 female, age 21–39, mean = 25.69, sd = 4.48), where all but three were smartphone owners. On average, participants had over 3 years of smartphone experience. On a 1–5 scale (1 = beginner, 5 = functionally native), participants rated their English language skills at 3.06. Each participant was randomly assigned to one of the two conditions—text entry with pressure-adaptive autocorrect and text entry without. After the experiment, participants were given a small non-monetary gratuity.

Apparatus

For pressure sensing and finger tracking we use a *Synaptics ForcePad* sensor. The device weights 526 g and mea-

sures $14.2 \times 12.2 \times 1.1$ cm. The touch-sensitive area covers 10.9×6.9 cm on the device's surface. The sensor's diagonal thus is 12.9 cm—within roughly 6 % of a *Samsung Galaxy S III's* 12.19 cm. Overall, the Forcepad is slightly larger but considerably heavier (an S III only weighs 133 g) than current mobile devices. However, it does provide a comparable experience when holding the device in landscape mode and typing. Thus, we feel confident the Forcepad is a valid placeholder device for evaluating how future mobile devices incorporating pressure sensing touch could be used.

The ForcePad provides capacitive tracking with pressure information for up to 5 fingers. Force readings are provided with 6 bit resolution at 67 Hz. We take the maximum value in the last 10 frames as the pressure value for a touch. The force sensitivity of the ForcePad made us choose this device instead of for example approaches using accelerometers to infer typing pressure [15]. Such approaches are typically limited or inaccurate, and in the first instance we wanted accurate values to assess how users controlled the system. However, we consider an exploration of using these techniques to bring ForceType to modern smartphones as an important avenue for future work.

Figure 7. Modified Synaptics ForcePad used in the *ForceType* system.

The ForcePad is an input-only device and cannot display visual feedback or instructions directly on its surface. Using an external monitor for feedback, however, would not allow us to evaluate normal typing behaviour. We thus modified the ForcePad by attaching a 132×32 px LCD to the device on which we can display three lines of text in a medium-sized font. The LCD was glued to the top of the ForcePad increasing the overall weight to 651 g without impeding hands holding the device. The modified device is shown in Figure 7. An Arduino is used for control and allows the connected PC to set display content via the serial port. This combination of ForcePad and LCD enables us to simulate future mobile devices with pressure-sensitive touch. One limitation of this hardware is there is no visual feedback on corrections before they happen—this is another reason that exploring pressure proxies on current smartphones is a focus for future work.

To simulate an onscreen keyboard, we glued a keyboard overlay on the ForcePad. We used a modified version of the iOS landscape keyboard with all buttons triggering mode switches removed (so as not to confuse participants). The glued-on keyboard does not introduce new haptic cues and thus provides an experience comparable to current smartphone keyboards.

Procedure

We used a subset of the English *NUS SMS Corpus* (version 2012.04.30) as phrase set [4]. This dataset contains 41537 text messages, sent primarily by users in Singapore, India, and the USA. Text messages often contain slang and shorthand, making them a good example of the a text where autocorrect fails. However, not all messages are equally appropriate for our evaluation and we removed all messages that:

- are shorter than 15 or longer than 50 characters
- contain any character not in the set given by the ISO basic latin alphabet plus the space and period characters
- are shorter than three words
- contain unknown one-letter words

This filtering leaves us with set of 5733 phrases. Our main concern is our requirement for unknown words in the messages—words that can not be found in a standard dictionary known to a language model. To determine such unknown words, we make use of the built-in Android `en-us` dictionary. We now split our phrase set into two parts:

correctable phrases Phrases that only contain words found in the dictionary ($n = 1272$).

uncorrectable phrases Phrases that contain at least one word not found in the dictionary ($n = 4461$).

For every participant, 20 correctable and 20 uncorrectable phrases were picked at random. While phrases were chosen randomly, a post-hoc analysis showed that there was no significant difference across users in the number of OOV words per sentence ($p > 0.4$). Participants were not provided with an indication whether a phrase was correctable or not.

At the beginning of the study, we gave participants the chance to familiarise themselves with our prototype. Participants then had to complete 40 trials. In each trial, they were shown the complete phrase on a screen in front of them and then had to copy that phrase. During text input, the phrase (clipped to the display size) was shown alongside the response text on the device. We asked participants to copy the phrases accurately—if they noticed a mistake, whether from a language model correction or their own typing, they were instructed to correct it. After using backspace to delete part of a word, autocorrect was disabled until the next word. This is equivalent to the behaviour on phones and prevents infinite correction loops. Participants had to submit each phrase with the return key.

Results

One participant was excluded from the analysis, as the touch offsets were so large the touches rarely hit the correct key.

Errors

As our performance measure, we use Active Correction Rate (ACR), defined as the proportion of words which the user has to actively correct by backspacing and retyping. This is a similar approach to Hoffman et al. [13], where they looked at the number of times backspace needed to be pressed. If **H1** is accepted, we should see a decrease in ACR since users can use pressure to prevent autocorrection of non-standard words. We chose this metric over CER, since in this study users were able to manually change unwanted corrections. In GPType, corrections made by the system could not be changed, since

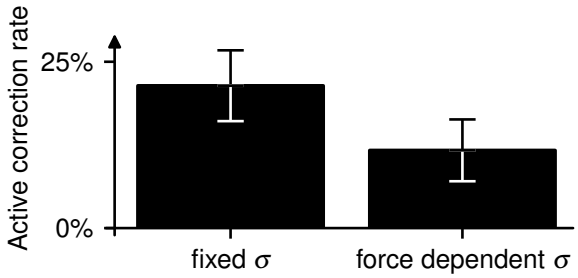


Figure 8. ForceType requires significantly fewer active corrections from users when entering text. Required corrections dropped by ≈ 10 percentage points. Errors bars are 1 sd.

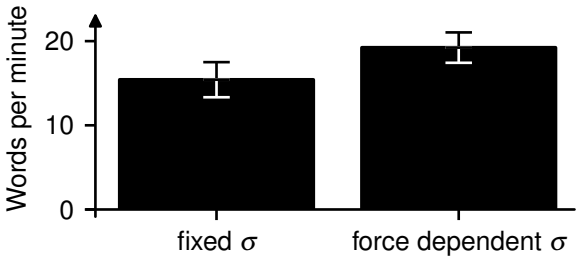


Figure 9. ForceType enabled users to enter phrases $> 20\%$ faster. A significant increase over the baseline. Errors bars are 1 sd.

we evaluated the quality of the corrections, not the ability of users to control the system. Our choice is supported by the results—the mean CERs for the two study groups were not significantly different ($\sim 3\%$ in each case).

Figure 8 shows the ACR. Using pressure adaptation, the average user had an ACR of 10.86%, compared to a value of 19.48% for users without pressure adaptation. This is a significant decrease compared to the baseline condition (independent two-sample t-test: $t_{12} = -3.48, p < 0.005$). We can thus reject the null hypothesis and accept **H1**.

The ACR values for both groups are quite high. Since the phrases are from an SMS corpus, many of the words are not found in common English. Unfamiliarity with the form factor of our apparatus may also have increased ACR.

Entry Rate

Entry rate was measured using words per minute (WPM), with a word being defined as five consecutive characters, including spaces. With pressure adaptation active, users typed 19.23 WPM, while without they only typed 15.42 WPM (Figure 9). This is a significant increase in typing speed (independent two-sample t-test: $t_{12} = 3.5002, p < 0.005$). We can reject the null hypothesis and also accept **H2**.

We also looked at the impact of uncorrectable phrases on typing speed. A small drop in speed could be expected, as users have to invest more mental effort in processing those phrases. We saw WPM go down by 3.97 (ForceType) and 2.61 (baseline). Both changes are significant ($p < 0.05$), while the difference between the two changes is not ($p > 0.13$). Thus, ForceType and the control are both equally affected by uncorrectable phrases, resulting in a small performance drop. Overall, ForceType resulted in a faster text entry rate.

DISCUSSION

The goal of our work was to show the advantages of combining a probabilistic touch model with a language model and decoding algorithm to increase text entry accuracy. With GPType, we used a touch offset model to correct for systematic errors in a user's touches and then compute probabilities over keys. This successfully reduced character error rates in a real typing task to a level comparable with a leading commercial product. Indeed, when users were standing or walking our system was slightly better than the commercial keyboard.

Our second system, ForceType, tackled cases where traditional autocorrect systems failed. By allowing users to convey certainty in their input using pressure, autocorrect can be dynamically turned off to allow entry of words not in standard word lists, such as acronyms or words from local dialects. ForceType was successful in increasing typing speed and reducing the number of cases where users had to manually correct their entered text.

These two systems represent different approaches to the uncertainty inherent in text entry. In GPType, we model this uncertainty in a manner that is hidden from users, allowing the system to account for and correct many commonplace errors. In ForceType, the uncertainty is *explicitly* controlled by the user, allowing finer control over the system when the user can anticipate undesired autocorrection behaviour.

It is likely that neither of these approaches is strictly better than the other. In future work, it would be interesting to combine offset modelling with explicit pressure control and see if input accuracy can be further increased.

CONCLUSIONS

Text entry is a ubiquitous activity on modern mobile devices, but continues to present challenges. The process is naturally very uncertain, both because of physical effects (e.g. the Fat Finger Problem) and due to uncertainty of intent—existing correction systems often assume users are trying to type words known to a dictionary, but this is certainly not always the case. In this paper we have presented two systems that improve the text entry experience by combining a state-of-the-art decoder and LM with a model capturing individual users' physical uncertainty (GPType) and an input system that allows users to explicitly control uncertainty (ForceType).

Evaluating GPType, we showed competitive reductions in character error rate for real typing tasks, without reducing the text entry rate. GPType reduced the character error rate by 4.9% over the baseline for sitting, 5% for standing, and 7.6% for walking users. Further, GPType obtained an average 1% decrease in CER over SwiftKey, a leading commercial keyboard, for standing and a 1.3% decrease for walking users.

Evaluating ForceType, we showed that users could successfully prevent autocorrection of words they did not want to be changed by varying their input pressure as they typed, resulting in faster overall text entry rates.

ACKNOWLEDGMENTS

This research was supported by Scottish Informatics and Computer Science Alliance (SICSA).

REFERENCES

1. Bi, X., Li, Y., and Zhai, S. FFitts Law: Modeling finger Touch with Fitts' law. *Proc. CHI '13*. 1363–1372.
2. Bi, X. and Zhai, S. Bayesian touch - a statistical criterion of target selection with finger touch. *Proc. UIST '13*. 51–60.
3. Brewster, S.A. and Hughes, M. Pressure-Based Text Entry for Mobile Devices. *Proc. MobileHCI '09*. 9:1–9:4.
4. Chen, T. and Kan, M.Y. Creating a Live, Public Short Message Service Corpus: the NUS SMS Corpus. *Language Resources and Evaluation*, 2012:1–37.
5. Clarkson, E.C., Patel, S.N., Pierce, J.S., and Abowd, G.D. Exploring Continuous Pressure Input for Mobile Phones. Tech. Rep. GIT-GVU-06-20, Georgia Tech, 2006.
6. de Jong, S., Kirkali, D., Schraffenberger, H., Jillissen, J., de Rooij, A., and Terpstra, A. One-Press Control: A Tactile Input Method for Pressure-Sensitive Computer Keyboards. *Proc. CHI EA '10*. 4261–4266.
7. Flemish, F.O., Adams, C.A., Conway, S.R., Goodrich, K.H., Palmer, M.T., and Schutte, P.C. T he H-Metaphor as a Guideline for Vehicle Automation and Interaction. Tech. Rep. TM-2003-212672, NASA, 2003.
8. Goel, M., Findlater, L., and Wobbrock, J. WalkType: Using Accelerometer Data to Accomodate Situational Impairments in Mobile Touch Screen Text Entry. *Proc. CHI '12*. 2687–2696.
9. Goodman, J., Venolia, G., Steury, K., and Parker, C. Language Modeling for Soft Keyboards. *Proc. AAAI 2002*. 419–424.
10. Gunawardana, A., Paek, T., and Meek, C. Usability Guided Key-Target Resizing for Soft Keyboards. *Proc. IUI '10*. 111–118.
11. Henze, N., Rukzio, E., and Boll, S. Observational and Experimental Investigation of Typing Behaviour Using Virtual Keyboards for Mobile Devices. *Proc. CHI '12*. 2659–2668.
12. Heo, S. and Lee, G. Force Gestures: Augmenting Touch Screen Gestures with Normal and Tangential Forces. *Proc. UIST '11*. 621–626.
13. Hoffmann, A., Spelmezan, D., and Borchers, J. TypeRight: A Keyboard with Tactile Error Prevention. *Proc. CHI '09*. 2265–2268.
14. Holz, C. and Baudisch, P. The Generalized Perceived Input Point Model and How to Double Touch Accuracy by Extracting Fingerprints. *Proc. CHI '10*. 581–590.
15. Iwasaki, K., Miyaki, T., and Rekimoto, J. Expressive Typing: A New Way to Sense Typing Pressure and Its Applications. *Proc. CHI EA '09*. 4369–4374.
16. Kristensson, P.O. Five challenges for intelligent text entry methods. *AI Magazine*, 30(4), 2009:85–94.
17. Kristensson, P.O. and Vertanen, K. Asynchronous multimodal text entry using speech and gesture keyboards. *Proc. Interspeech 2011*. 581–584.
18. Kristensson, P.O. and Vertanen, K. Performance Comparisons of Phrase Sets and Presentation Styles for Text Entry Evaluations. *Proc. IUI '12*. 29–32.
19. Kristensson, P.O. and Zhai, S. SHARK^2: A Large Vocabulary Shorthand Writing System for Pen-Based Computers. *Proc. UIST '04*. 43–52.
20. Kristensson, P.O. and Zhai, S. Relaxing Stylus Typing Precision by Geometric Pattern Matching. *Proc. IUI '05*. 151–158.
21. Lui, M. and Baldwin, T. langid.py: An off-the-shelf language identification tool. *Proceedings of the ACL 2012 System Demonstrations*. 25–30.
22. MacKenzie, I.S. and Soukoreff, R.W. Phrase Sets for Evaluating Text Entry Techniques. *Proc. CHI EA '03*. 754–755.
23. McCallum, D.C., Mak, E., Irani, P., and Subramanian, S. PressureText: Pressure Input for Mobile Phone Text Entry. *Proc. CHI EA '09*. 4519–4524.
24. Oulasvirta, A., Reichel, A., Li, W., Zhang, Y., Bachynskzi, M., Vertanen, K., and Kristensson, P.O. Improving Two-Thumb Text Entry on Touchscreen Devices. *Proc. CHI '13*. 2765–2774.
25. Pohl, H. and Murray-Smith, R. Focused and Casual Interactions: Allowing Users to Vary Their Level of Engagement. *Proc. CHI'13*. 2223–2232.
26. Ramos, G., Boulos, M., and Balakrishnan, R. Pressure Widgets. *Proc. CHI '04*. 487–494.
27. Rasmussen, C.E. and Williams, C.K.I. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.
28. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. FingerCloud: Uncertainty and Autonomy Handover in Capacitive Sensing. *Proc. CHI '10*. 577–580.
29. Rudchenko, D., Paek, T., and Badger, E. Text Text Revolution: A Game That Improves Text Entry on Mobile Touchscreen Keyboards. *Proc. Pervasive '11*. 206–213.
30. Schwarz, J., Hudson, S., Mankoff, J., and Wilson, A.D. A Framework for Robust and Flexible Handling of Inputs with Uncertainty. *Proc. UIST '10*. 47–56.
31. Shi, K., Irani, P., Gustafson, S., and Subramanian, S. PressureFish : A Method to Improve Control of Discrete Pressure-based Input. *Proc. CHI '08*. 1295–1298.
32. Siek, K.A., Rogers, Y., and Connelly, K.H. Fat Finger Worries: How Older and Younger Users Physically Interact with PDAs. *Proc. INTERACT'05*. 267–280.
33. Srinivasan, M.A. and Chen, J.s. Human Performance in Controlling Normal Forces of Contact with Rigid Objects. *Proc. ASME'93*, vol. 49. 119–125.
34. Stewart, C., Rohs, M., Kratz, S., and Essl, G. Characteristics of Pressure-Based Input for Mobile Devices. *Proc. CHI '10*. 801–810.
35. Subramanian, S., Aliakseyeu, D., and Lucero, A. Multi-layer interaction for digital tables. *Proc. UIST '06*. 269–272.
36. Vertanen, K. and Kristensson, P.O. A versatile dataset for text entry evaluations based on genuine mobile emails. *Proc. MobileHCI '11*. 295–298.
37. Weir, D., Rogers, S., Murray-Smith, R., and Löchtefeld, M. A User-Specific Machine Learning Approach for Improving Touch Accuracy on Mobile Devices. *Proc. UIST '12*. 465–476.
38. Williamson, J. *Continuous Uncertain Interaction*. PhD thesis, University of Glasgow, 2006.