A User-Specific Machine Learning Approach for Improving Touch Accuracy on Mobile Devices

Daryl Weir, Simon Rogers, Roderick Murray-Smith
School of Computing Science
University of Glasgow
18 Lilybank Gardens
Glasgow, G12 8QQ, UK
darylw@dcs.gla.ac.uk, {Simon.Rogers, Roderick.Murray-Smith}@glasgow.ac.uk

Markus Löchtefeld
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
Campus D3-2
D-66123 Saarbrucken, Germany
markus.loechtefeld@dfki.de

ABSTRACT
We present a flexible Machine Learning approach for learning user-specific touch input models to increase touch accuracy on mobile devices. The model is based on flexible, non-parametric Gaussian Process regression and is learned using recorded touch inputs. We demonstrate that significant touch accuracy improvements can be obtained when either raw sensor data is used as an input or when the device's reported touch location is used as an input, with the latter marginally outperforming the former. We show that learned offset functions are highly nonlinear and user-specific and that user-specific models outperform models trained on data pooled from several users. Crucially, significant performance improvements can be obtained with a small ($\approx$ 200) number of training examples, easily obtained for a particular user through a calibration game or from keyboard entry data.

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation: Input devices and strategies (e.g. mouse, touchscreen)

Author Keywords
Touch; Machine Learning; Regression; Gaussian Processes; Probabilistic Modelling

INTRODUCTION
In this paper, we investigate an alternative, data-driven approach to touch. We treat the problem as a Machine Learning (ML) task where we are interested in learning a function that maps an input (the device's reported touch location or the raw sensor values) to the *intended touch location*. We show later that the functions learned for individual users are highly user-specific and vary greatly across the touch surface, overcoming the limitations of a device specific offset as in [6]. We use a flexible non-parametric regression algorithm based on Gaussian Processes (GPs;[12]) to enable us to model complex nonlinear functions without having to make detailed parametric assumptions about the touch event (as in [14]). We show that using either the raw sensor values or the device's reported touch location as the input we can obtain significant improvements in accuracy using a number of recorded touch events lower than that collected per user in [6].

Given the increasing ubiquity of touch, there is a clear need for techniques which facilitate accurate input. One of the primary causes of touch error is the so called "fat finger problem" [3]: through the act of touching, the user occludes the very targets she is trying to touch and the softness of the finger/thumb produces a much larger, more ambiguous, contact region than intended [8]. This is particularly problematic when users type on a virtual keyboard with their thumbs, both when operating the device one-handed [11] (holding and typing with the same hand) or typing with two thumbs with the device in landscape mode. The result is a user-specific offset between the touch location reported by the device and that intended by the user.

There has been considerable research effort directed at creating models to account for this offset. Studies both in the wild [6, 7] and in the lab [8, 14] have succeeded in compensating for some of the error inherent in touch based interaction. Unfortunately these approaches are either based on custom hardware or on global compensation methods; none of them focused on the needs of specific users. The approach proposed here is both user-specific and uses only data available on commercial smartphones. Further, our model is fully probabilistic in its predictions and is naturally able to incorporate data from sensors other than the touch screen (e.g. accelerometer readings). This approach provides the basis for future research into propagating uncertainty from input to application and potentially opening rich new channels of interaction.

RELATED WORK
Even though HCI research focused on touch screens based on capacitive sensing has been conducted for over 25 years [10], current error rates suggest there is still room for improvement [6]. Precise capacitive touch input is clearly a desirable goal, given the increasing ubiquity of touchscreen devices. Such devices are employed in an extremely broad range of settings

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
UIST'12, October 7–10, 2012, Cambridge, Massachusetts, USA.
Copyright 2012 ACM 978-1-4503-1580-7/12/10...$15.00.

— even input through fabric enclosure (e.g. inside a pocket) has been successfully explored [16].

In lab studies, [8] investigated the relationship between finger orientation and touch error and introduced 'Ridgepad' – a device that could infer finger orientation through fingerprint scanning. Similarly, [14] showed that it is possible to infer finger orientation using an array of capacitive sensors and sophisticated Bayesian statistical techniques. Both [8] and [14] showed that there is considerable diversity between users, and between finger and thumb use. This suggests that significant improvements could be made if user- and context-specific models were built. Unfortunately, the improvements in touch performance described by [8] and [14] come at significant hardware and computational costs respectively and therefore do not offer solutions that are currently practical.

In two recent large-scale studies, [6, 7] used a crowd-sourcing approach to collect vast quantities of touch data from a variety of users and devices. They investigated simple tapping as well as text entry with applications deployed in the Android Marketplace. Their data led them to create a device-specific compensation function which they then showed improved accuracy [6]. For the text-entry scenario a simple shift function helped to decrease the error rate by 9.1% globally [7]. This approach increases the accuracy of touch input, but we argue that there is still room for improvement. In [9] it was shown that the personal perceived touch point is important, in addition to the physical characteristics of the touching finger or thumb. Users tend to use visual features of their fingers for their mental model that determines the corresponding point on the screen that their touch hits. Since these mental models can differ between users, personalized models would have a bigger influence than general models. This is a key motivation for our approach – no matter how good the sensing hardware is at converting sensor inputs into a touch location, there will always be user-specific biases. For example, in Figure 1 we show the smoothed sensor values for a touch event (white is high valued, black low). The grey circle shows the touch location reported by the device, which looks very reasonable given the sensor values. However, the white circle represents the point the user was trying to touch. This offset between the *intended* touch point and the *measured* touch point will be highly user-specific (see e.g. [9, 14]) and it is this error that we aim to reduce in this work.

In both [9] and [6], the techniques proposed give deterministic values for the intended touch location — there is no information about the uncertainty of predictions. There is a growing body of research showing the value of propagating uncertainty from input to the application level [20, 18]. [15] explicitly models the uncertainty in a touch input and uses this to negotiate the handover of control between the user and the system. The authors demonstrate a map browsing application which automatically scrolls towards nearby points of interest when user input is uncertain (i.e. when the finger is lifted from the screen slightly.) The work of Schwarz [17] is also relevant. The authors propose a framework for modelling uncertain interaction which allows a system to consider several possible actions and assign them probabilities based on user input. Definitive action is only taken when the probability for an event reaches a certain threshold. The predictions of our model are Gaussian distributions with full covariance structure, which could be used as input to such a probabilistic interaction framework.

Personalized models have been used commonly for key-target resizing on soft keyboards [2] but the disadvantages of these techniques is that the approach is only applicable for keyboard input since they are based on natural languages technologies. Similarly, in [4] the authors use machine learning to classify which key a given touch was targetting. Again, this requires specific information about the layout of the keyboard and is not generally applicable to all touches. Our approach combines the personalization and learning process with the power of a model which increases touch accuracy for all tasks.

TOUCH AS A MACHINE LEARNING PROBLEM

Our approach to overcome the problems discussed above is to introduce a machine learning technique that allows us to create user specific models based on either raw touch sensor data or the device's reported touch location. We are able to access the raw sensor data for the Nokia N9 MeeGo smartphone, which has an array of capacitive sensors laid out in a grid. We base most of our experimentation on this device as it is the only commercial product for which we have access to raw sensor data. Our task is therefore to find a mapping between either a vector of sensor values or a 2-dimensional reported touch location and the corresponding intended 2-dimensional touch location on the 854x480 pixel display. Figure 1 shows the sensor values for a typical touch towards the top of the device (the intended touch point is represented as a white circle). It is clear that a typical touch produces a significant sensor reading in several sensor positions simultaneously. As already mentioned, the exact pattern of sensor activation for a particular touch will depend on the user and on how they are performing the touch – with a finger or a thumb say – and it is not clear what form this function will take. We therefore turn to a flexible non-parametric regression algorithm: Gaussian Process regression.

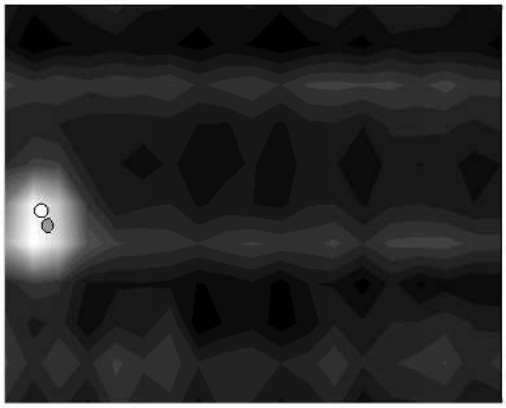


Figure 1. Sensor outputs (black = low; white = high) for a touch aimed at the white circle. The N9's reported touch location is indicated with a grey circle.

Gaussian Process Regression

Gaussian Processes (GPs) [12] are a popular statistical technique for regression and classification. In this work, we are interested in learning a function that can map between inputs $\mathbf{s}$ and intended 2D touch location (x, y): $(x, y) = f(\mathbf{s})$. In our experiments, the input $\mathbf{s}$ will be either the raw sensor values, or the touch location given by the device. GPs allow us to do this without making any parametric assumptions about the form of $f(\mathbf{s})$. Instead, we supply a mean function $\mu(\mathbf{s})$ (which is set to zero in all of our experiments) and a covariance function, $C(\mathbf{s}_n, \mathbf{s}_m)$, which defines how similar the nth and mth outputs should be based on inputs $\mathbf{s}_n$ and $\mathbf{s}_m$. Readers are pointed to [12] for extensive details on GP regression and classification.

Given N training examples, $\mathbf{x}_1, \ldots, \mathbf{x}_n, \ldots, \mathbf{x}_N$ and associated locations, $(x_1, y_1), \ldots, (x_n, y_n), \ldots, (x_N, y_N)$, our GP regression approach proceeds as follows. Firstly, we turn the problem from a two-dimensional regression into a one-dimensional regression by stacking all of the locations into a single vector:

$$\mathbf{z} = [x_1, \ldots, x_n, \ldots, x_N, y_1, \ldots, y_n, \ldots, y_N]^{\mathrm{T}}.$$

Note that this does not preclude us from modelling possible dependencies between x_n and y_n (see below). We now build an $N \times N$ covariance matrix, $\mathbf{C}$, where the n, mth element is calculated by evaluating the covariance function $C(\mathbf{s}_n, \mathbf{s}_m)$. This matrix is then stacked up to produce the full $2N \times 2N$ covariance matrix, $\widehat{\mathbf{C}}$:

$$\widehat{\mathbf{C}} = \begin{bmatrix} \mathbf{C} & \alpha\mathbf{C} \\ \alpha\mathbf{C} & \mathbf{C} \end{bmatrix},$$

where α controls the strength of the dependence between x_n and y_n. Formally, the covariance between x_n and x_m (or y_n and y_m) is given by $C(\mathbf{s}_n, \mathbf{s}_m)$ whilst the covariance between x_n and y_m is given by $\alpha C(\mathbf{s}_n, \mathbf{s}_m)$. If $\alpha = 0$, we are effectively using independent regression models for the x and y locations.

Finally, we assume a small amount of additive Gaussian noise (with variance σ^2). This overcomes possible problems resulting from trying to map very similar input sensor values to different intended touch locations.

Our aim is to be able to predict (x^*, y^*) for a new set of input values $\mathbf{s}^*$. To do this, we create a vector comprising the covariance function evaluated between $\mathbf{s}^*$ and the N input vectors in the training set:

$$\mathbf{c} = [C(\mathbf{s}^*, \mathbf{s}_1), \ldots, C(\mathbf{s}^*, \mathbf{s}_N)],$$

which is then stacked up similarly to the training covariance matrix:

$$\widehat{\mathbf{c}} = \begin{bmatrix} \mathbf{c} & \alpha\mathbf{c} \\ \alpha\mathbf{c} & \mathbf{c} \end{bmatrix}.$$

The GP prediction is a 2-dimensional Gaussian with mean and covariance given by:

$$\begin{aligned} (x^*, y^*) &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \boldsymbol{\mu} &= \widehat{\mathbf{c}} \left[\widehat{\mathbf{C}} + \sigma^2\mathbf{I}\right]^{-1} \mathbf{z} \\ \boldsymbol{\Sigma} &= C(\mathbf{s}^*, \mathbf{s}^*) - \widehat{\mathbf{c}} \left[\widehat{\mathbf{C}} + \sigma^2\mathbf{I}\right]^{-1} \widehat{\mathbf{c}}^T. \end{aligned}$$

Note that the inversion of the $2N \times 2N$ matrix is not prediction specific and can therefore be done just once after training data has been collected. Note that if $\alpha = 0$, this predictive Gaussian will have no covariance between the two dimensions (i.e. the top right and bottom left elements of $\boldsymbol{\Sigma}$ will be equal to zero). Both α and σ^2 are parameters that need to be optimised.

Choice of covariance function

Many different covariance functions have been used for GP regression. In this work, we have used the following, standard, combination of a linear and Gaussian covariance:

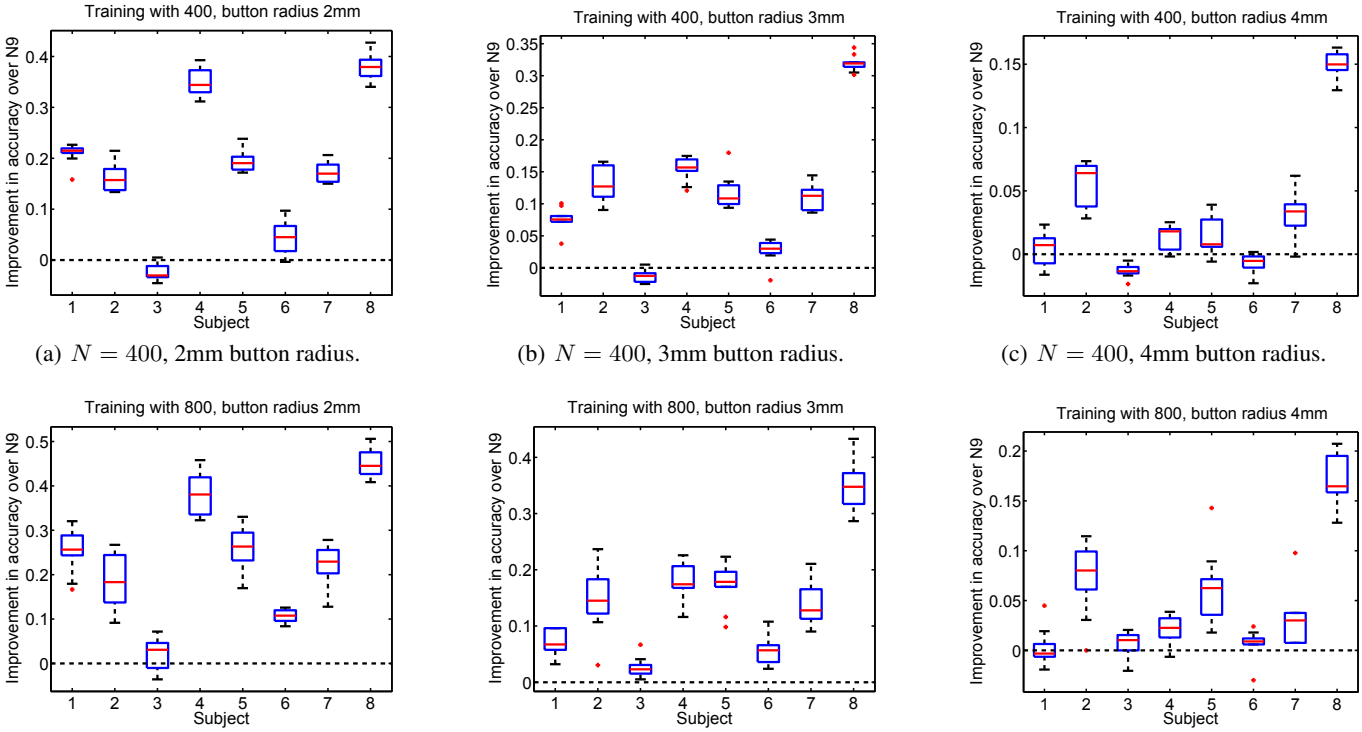
$$C(\mathbf{s}_n, \mathbf{s}_m) = b\left(a\mathbf{s}_n^T\mathbf{s}_m + (1-a)\exp\left\{-\gamma||\mathbf{s}_n - \mathbf{s}_m||_2^2\right\}\right),$$

where a controls the relative influence of the linear and Gaussian terms and γ controls the length scale of the Gaussian. Cross validation on a gives an indication of the non-linearity of the mapping from the input space to the intended touch location. We shall see later that using the sensor values as input results in a highly non-linear function, whereas using the device's reported touch location as our input gives an even balance of the linear and non-linear terms. This covariance function provided excellent performance, but an exploration of other covariance functions is an avenue for future work.

EXPERIMENTAL SETUP

We built a simple data collector in Python and PyGame that ran natively on the Nokia N9. The software displayed crosshairs on the screen that the users had to touch. The crosshairs were randomly located in the area that is normally occupied by the landscape keyboard of the N9 to simulate text entry in landscape mode. For each touch on a crosshair, the system recorded the intended location (i.e. the screen location of the crosshair), the values of the capacitive sensors and the location reported by the N9 for the touch event. After the user lifted his finger from the screen another crosshair would be displayed. We obtained data from a total of 8 participants (3 female), aged between 23 and 34, all of whom performed 1000 touches holding the device in landscape mode, using either thumb to touch (see Figure 2). All but one of our participants owned smartphones and therefore were used to operating a touch-screen device.

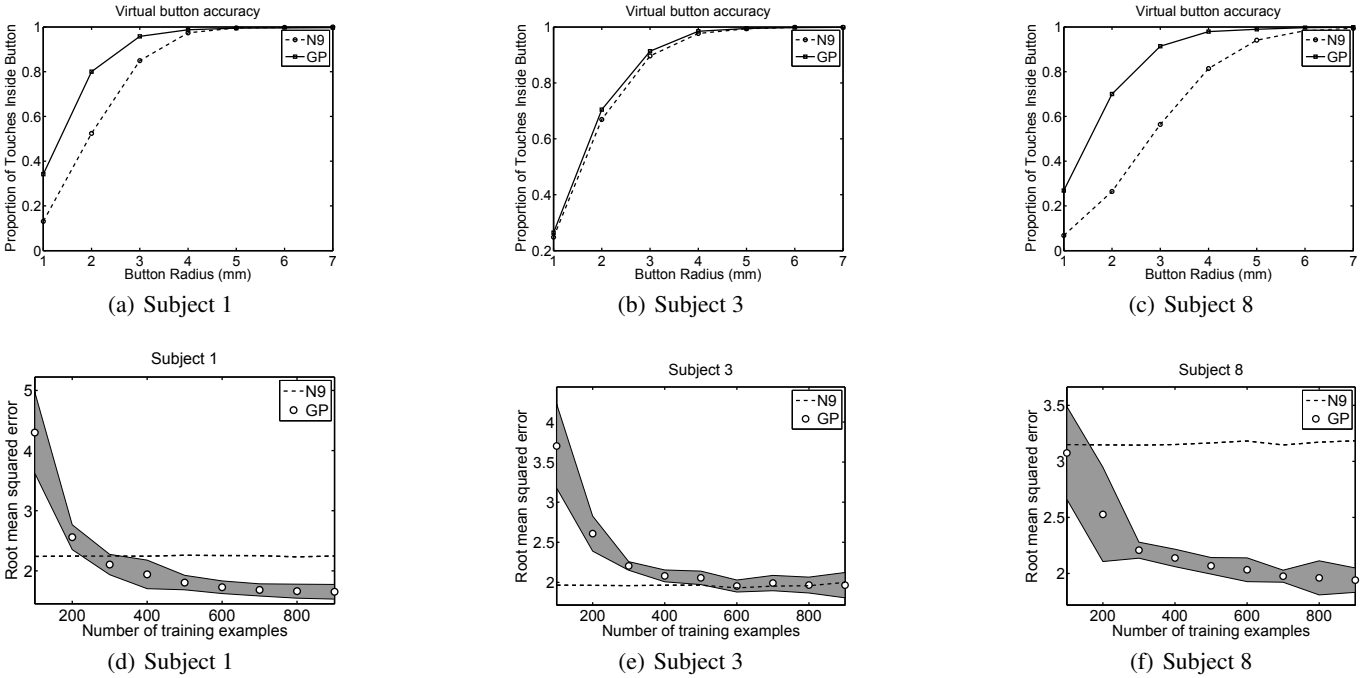
For all our experiments, we rescaled the touch locations such that they were in the unit square centered on the origin, with no loss of generality. This was done so that the same model parameters can be used across different devices with different native resolutions. We transform the data back to a real world scale when computing virtual button accuracy and RMS error on our predictions.



(a) $N = 400$, 2mm button radius. (b) $N = 400$, 3mm button radius. (c) $N = 400$, 4mm button radius.

(d) $N = 800$, 2mm button radius. (e) $N = 800$, 3mm button radius. (f) $N = 800$, 4mm button radius.

Figure 3. Improvements in accuracy rate for different virtual button radii and numbers of training examples when the raw sensor values are used as an input. The value on the y-axis is the proportion of touches inside the button for the GP model minus the proportion computed for the N9's native touch location. An increase of say 0.1 corresponds to a 10% decrease in error rate. Boxplots show the distribution of improvement over 10 repetitions.



(a) Subject 1 (b) Subject 3 (c) Subject 8

(d) Subject 1 (e) Subject 3 (f) Subject 8

Figure 4. Comparison of performance between the N9 touch events (N9) and our Gaussian Processes predictions using the raw sensor values as input (GP). Top row: Accuracy for different virtual button sizes (800 training points). Bottom row: Learning curves. Subject 1 represents an average user, subject 3 a user that already performs far above average already with the N9 screen and subject 8 represents a user that doesn't own a touch-screen based phone.

Figure 2. Experimental setup: participants held the phone in both hands and used either thumb to touch.

RESULTS

In our first set of experiments, we use the raw values from the capacitive sensor to predict the true touch location. We predict the location directly rather than an offset, since in principle this algorithm could take the place of whatever technique is currently used to predict device location. 5-fold cross-validation was performed for subject 1 over a small range of values for $a, \gamma, b, \alpha, \sigma^2$ and these values were then used across all subjects. It is important to note that these parameters control the GP covariance function, which defines how smooth the resulting predictive functions can be, rather than defining the functions themselves. In practice this means that the results for subject 1 are likely to be slightly optimistic, and those for all other users pessimistic. Ideally, a cross-validation would be performed on a per-user basis but the necessary computation is an unreasonable demand in a mobile setting — any deployment of the system would have to use predetermined covariance parameters such as these. The chosen parameter values were: $\gamma = 0.05, b = 5, a = 0.1, \alpha = 0.9, \sigma^2 = 10^{-3}$. The sensor data is pre-processed prior to use by first setting to zero any values that are less than 150 (we found this was the typical background noise in the absence of touch input) and then normalising each touch so that the sum of the sensor values is equal to 1. The mean of the predictive Gaussian provided by the GP is used as the inferred touch location.

Predicting from raw sensor values

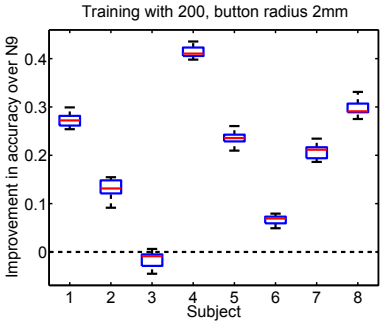
We begin by looking at virtual button accuracy, following [8] and [14]. Assuming a circular button is placed around the centre of the crosshair at which the user was aiming, we can compute the proportion of touches that would fall within this circle. In Figure 3 we show the improvement in proportion for the GP regression over the N9s native location sensing. Positive values mean the model is an improvement over the N9s native algorithm. The data were analysed for three different virtual button sizes (2mm radius, 3mm, 4mm) and two different sizes of training set (400 and 800 touches). In each case, we repeated the experiment 10 times. For each repetition, we randomly sampled $N = 400$ or $N = 800$ touches from the 1000 touches for each user and used the remaining (600 or 200) samples for testing. Note that as we are interested in personalised touch models, separate regression models are trained for each user. The average reductions in error rate across all users are 23.47% (2mm buttons), 14.20% (3mm) and 4.70% (4mm).

For all users except subject 3, we see notable performance increases for buttons of radius 2mm and 3mm. These improvements are all statistically significant (paired t-test, $p < 0.05$), with the exception of subject 3. An increase of 0.2 corresponds to a potential decrease in error rate of 20%. Improvements are lower for 4mm buttons reflecting the acceptable N9 performance at this radius. For 4mm buttons the improvements are statistically significant for subjects other than 1, 3 and 6. To place these results in context, in Figure 4 we show more detailed results from subjects 1,3 and 8 (representing a wide range of performances). The top plots ((a)-(c)) show the button radius results for buttons of radius 1mm to 7mm. The solid curve is the performance of the GP regression and the dashed curve that of the N9s native algorithm. For all subjects, results are indistinguishable at 6mm but large improvements are evident at smaller button sizes for subjects 1 and 8. Subject 3 is the only subject for which we see no improvement. Inspecting Figure 4(b), we can see that both models perform very well for this subject — both the N9s algorithm and the GP perform well, and so very little improvement is possible. It is also interesting to note that subject 8 is the only user in the study who does not own a touch-screen device and is a user for which we see very large improvements.

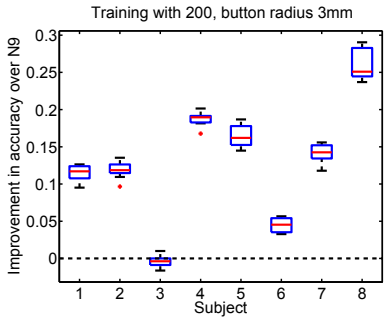
The bottom row in Figure 4 shows how the root mean squared error between the intended and inferred touch location (in mm) varies as the training set size increases. The solid line gives the GP performance and the shaded area shows plus and minus one standard deviation. This was obtained by randomly taking a subset of touches for training and then testing on the remainder. The N9 performance is averaged over the same set of test points (variance in N9 performance was too small to visualise). We notice that for subjects 1 and 8, there is a large drop up to 400 training examples and then performance appears to plateau. In [6], participants recorded on average ~ 1000 touches each and therefore our proposed approach could be trained using data collected in a similar manner. Subject 3 also plateaus at around 400 training examples, but does not show any improvement. Again however, this can be explained by the excellent N9 performance for this user (a root mean squared error value of less than 2mm).

Prediction from device location

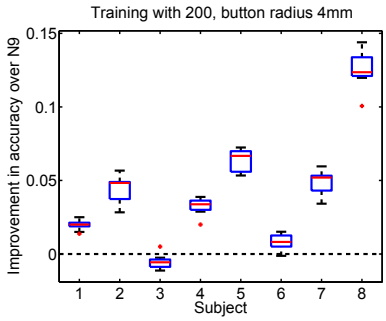
In our second set of experiments, we used the N9's reported touch location as input instead of the raw sensor values. This is motivated by the fact that it is not currently possible to obtain raw sensor data on most touchscreen devices. This approach is similar to the offsets computed in [6] in that it is adding a post-processing step to the device's reported touch location but differs in two respects. Firstly, we are learning a continuous, user-specific offset function which can, in theory, have a different value at any input location and secondly, the GP regression does not restrict us to any particular parametric function family, such as the 5th order polynomials used by Henze et al. We once again optimise the GP covariance parameters by performing a cross-validation (us-



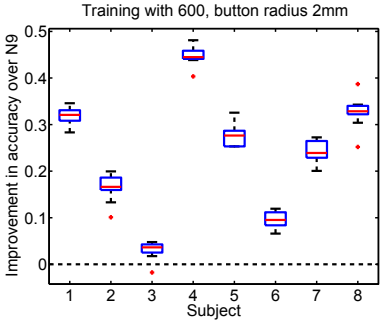
(a) $N = 200$, 2mm button radius.



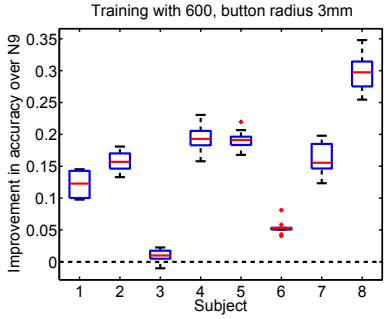
(b) $N = 200$, 3mm button radius



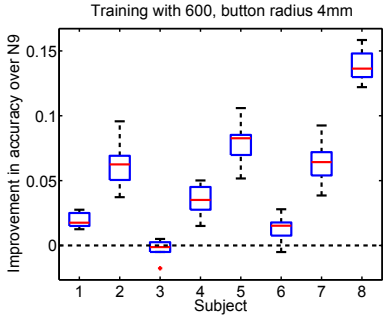
(c) $N = 200$, 4mm button radius.



(d) $N = 600$, 2mm button radius.

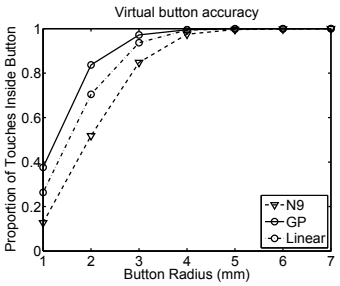


(e) $N = 600$, 3mm button radius.

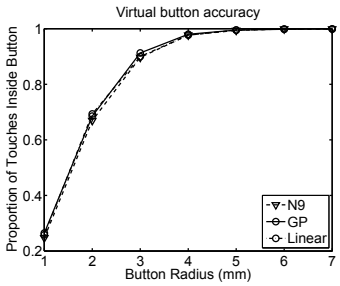


(f) $N = 600$, 4mm button radius.

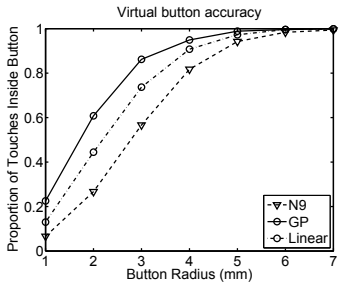
Figure 5. Improvements in accuracy rate for different virtual button radii and numbers of training examples when the N9's touch location is used as an input. The value on the y-axis is the proportion of touches inside the button for the GP model minus the proportion computed for the N9's reported touch location. An increase of say 0.1 corresponds to a 10% decrease in error rate. Boxplots show the distribution of improvement over 10 repetitions.



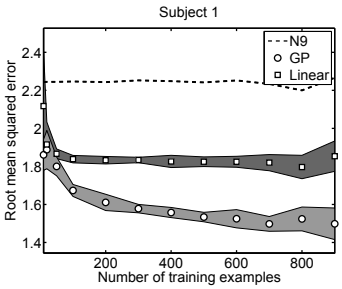
(a) Subject 1



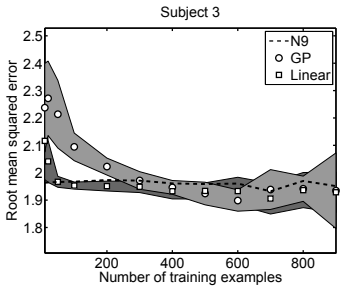
(b) Subject 3



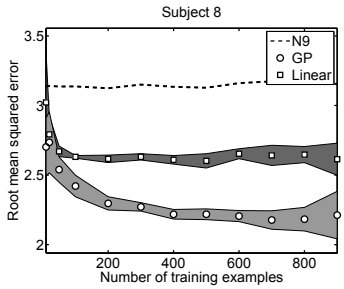
(c) Subject 8



(d) Subject 1



(e) Subject 3



(f) Subject 8

Figure 6. Comparison of performance between the N9 touch events (N9), a simple linear regression model (Linear) (e.g. as in [6] and our Gaussian Process' predictions using the N9's reported touch location as input (GP). Top row: Accuracy for different virtual button sizes (800 training points). Bottom row: Learning curves. Subject 1 represents an average user, subject 3 a user that already performs far above average already with the N9 screen and subject 8 represents a user that never used a touch screen based phone before.

ing user 5; randomly chosen). The optimal parameters were: $\gamma = 100, b = 1, a = 0.5, \alpha = 0.3, \sigma^2 = 10^{-3}$.

Figure 5 shows the improvements in performance offered by this method over the N9's reported touch location. We found that similar improvements can be made with fewer training examples than are required when using sensor data as input and so show the performance for $N = 200$ and $N = 600$ for buttons of radius 2mm, 3mm and 4mm. Once again, we notice significant (paired t-test, $p < 0.05$) performance improvements for all subjects except subject 3. These improvements are for just 200 training examples – collection of which would represent a very low overhead for the user. Assuming that an increase in performance of 0.1 corresponds to a reduction in error rate of 10%, the average reduction in error rate across all users is 23.79% (2mm buttons), 14.79% (3mm) and 5.11% (4mm).

The work in [9] suggests a theoretical lower limit of 2.15mm for the radius of a button which can be acquired with 95% accuracy. Their technique can acquire targets of radius 2.7mm, but doing so requires an overhead camera and an algorithm to extract salient features from the outline of the finger. Clearly these are not reasonable restrictions in a mobile setting. Our method achieves 95% accuracy with targets of radius between 2.8mm and 4mm, depending on the user. For all subjects bar one, this still represents an improvement over commercially available sensing technology of the N9.

In Figure 6 we show more detailed results for subjects 1, 3 and 8. We also give the performance of a simple linear regression model (see e.g. [13] p. 22) trained on the same data. For subject 3 we again see no significant improvement but for subjects 1 and 8 we see that both the linear regression and the GP outperform the N9's native algorithm and that the GP outperforms the linear regression, suggesting that the touch offset is not a linear function of the touch location. Notice also that for subjects 1 and 8, the performance is never worse than that of the N9, whereas in predicting the intended location from the raw sensor values, the performance was worse when there was little training data (see Figure 4(d) and (f)). This can be explained by the GP prior mean function, which is set to zero. When very little data is available, the GP will produce an output close to zero. For the offset prediction, this will give reasonable results as the model output will not vary much from the N9 output. However, when predicting the location rather than the offset, it will give very poor performance unless the touch happens to be close to the origin.

When using the N9 touch location as an input, it is possible to visualise the offset functions that are learned across the 2D input space. These are shown for subjects 1, 3 and 8 in Figure 7. The left hand panels show the predicted offsets across the input space, binned into a 10×5 grid. The size and direction of the offset for each touch which falls in a given box is shown relative to the center of that box. This binning is done only for ease of visualisation — all calculations were done with the continuous offset functions. The centre and right panels show the continuous horizontal and vertical offset functions. White corresponds to a large positive value (to the right or up for horizontal and vertical offsets respectively) and black a large negative value (to the left and down). We see very small offsets for subject 3, reflecting the fact that the N9 input is already very accurate. For users 1 and 8, it is immediately clear that the offset functions are highly non-linear. The horizontal offset functions have two reasonably flat areas separated by a steep ridge. Recall that the user is operating the system with two thumbs (see Figure 2) and this ridge represents the point at which the thumb they are using changes. The vertical offsets are more complex, likely reflecting the change in thumb orientation as it reaches up and down.

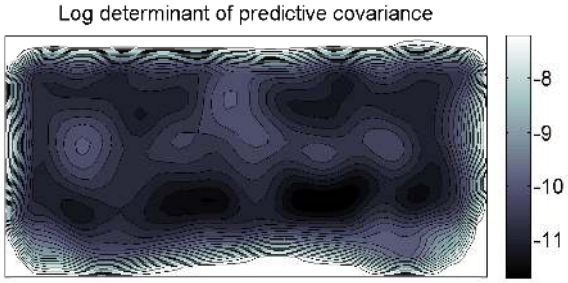


Figure 9. Log determinant of the predictive covariance matrix for subject 1 when the N9's touch location is used as an input.

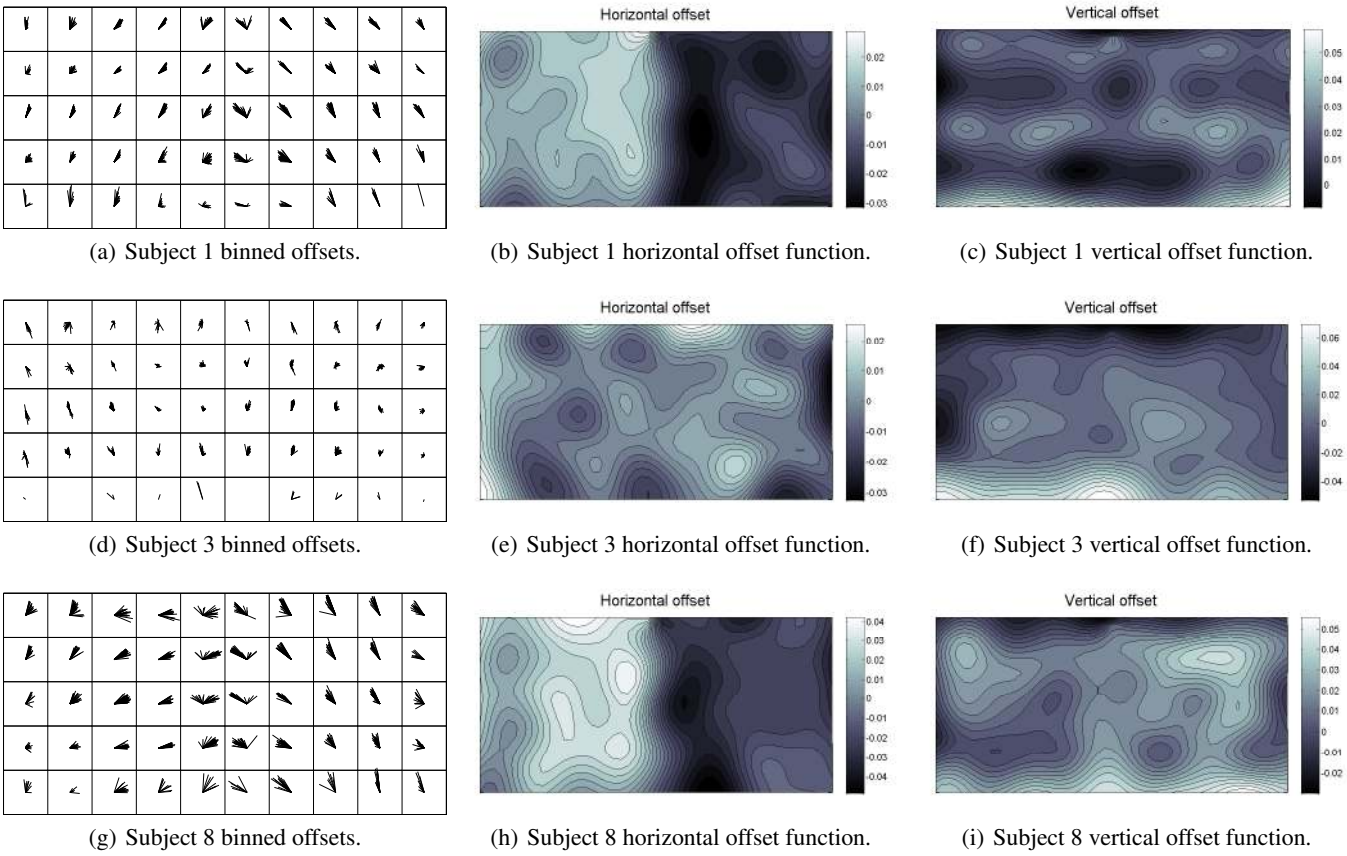
To show the relative limitations of the linear model, Figure 8 shows the linear offsets learned for subject 8. Comparing these with the bottom row in Figure 7, we can clearly see the subtleties lost by the linear model. In particular, the horizontal offset in the linear model changes smoothly from left to right, completely missing the sharp change required towards the centre of the device where the user changes thumbs.

Predictive covariance

The Gaussian Process prediction consists of a Gaussian density. Thus far we have just used the mean of this Gaussian as a point estimate. The covariance of the Gaussian is also useful as it describes the uncertainty in the prediction and could therefore be used at the application level. In Figure 9 we visualise the log determinant of the predictive covariance matrix (higher corresponds to greater uncertainty) for subject 1 when the N9 touch location is used as an input. The highest uncertainty is at the edges, representing the limit of the recorded data. There are also areas of higher uncertainty down the centre and in the centre left and right areas. The centre area can most likely be explained by the user changing thumbs in this area – they will not always change at exactly the same horizontal position and therefore uncertainty in the required offsets is natural. This information about the uncertainty is a further advantage over the polynomial function of [6], which only gives a point estimate of the offset. We discuss other uses of predictive uncertainty below.

Generalising Results

To ensure that the results described above were not specific to a particular device and orientation combination, we car-



(a) Subject 1 binned offsets. (b) Subject 1 horizontal offset function. (c) Subject 1 vertical offset function.

(d) Subject 3 binned offsets. (e) Subject 3 horizontal offset function. (f) Subject 3 vertical offset function.

(g) Subject 8 binned offsets. (h) Subject 8 horizontal offset function. (i) Subject 8 vertical offset function.

Figure 7. Offsets learned for subjects 1, 3 and 8. All panels represent the lower half of the N9 screen when in landscape mode. The left hand plot shows the offsets inferred by the GP for the actual touch data and the right two the continuous horizontal and vertical offset functions learned by the GP.

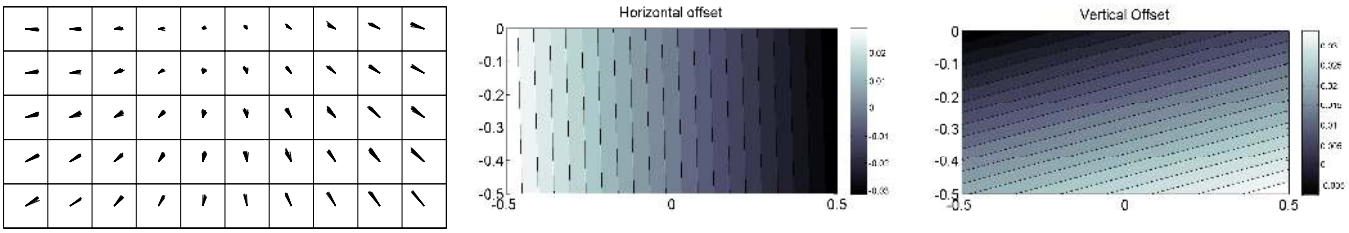
ried out a number of additional experiments. First, we had two subjects (1 and 6) each generate 1000 additional touches, with the N9 in a portrait orientation. Users held the device in one hand and touched using the thumb on that hand. The touch targets were generated in the region corresponding to the device's portrait virtual keyboard. While significant conclusions cannot be drawn for a sample of two, the results of this probe are included to contextualise our core results.

Figure 10 shows our results. Again, we computed the virtual button accuracy as our performance metric ((a) and (c)). We used the raw sensor values of the touches as the input to the GP. For both users, we see that the GP is more accurate than the N9's native algorithm. These GPs were based on the optimal parameters found from the cross validation on the landscape data, showing that the covariance parameters generalise well across orientation. The learning curves (10(b) and (d)) show that the GP outperforms the N9 after approximately 300 training points are generated. An interesting future research direction is the development of a model which can predict touch location regardless of device orientation. This is potentially possible since even when touching the same location, the sensor readings are likely to be different across the different orientations.

We also replicated our experiment on a Google Nexus One running Android. Since this device does not expose the raw sensor values in the screen, we were restricted to considering the prediction problem using the device's reported touch location as input. We had two subjects (1 and 3) generate 1000 touches in landscape mode. The protocol was identical to the original experiment. Virtual button accuracies and learning curves for the GP, Android phone and a linear model are shown in Figure 11. Once again we see that the GP outperforms both competing models for all button sizes. Note also that one of the users shown is subject 3, for whom the GP offered no improved performance on the N9. On the Android phone, we see a large improvement for this user when using the GP. As with the N9, the learning curves are below the device performance for even very small quantities of training data. This can again be attributed to the zero mean of our GP.

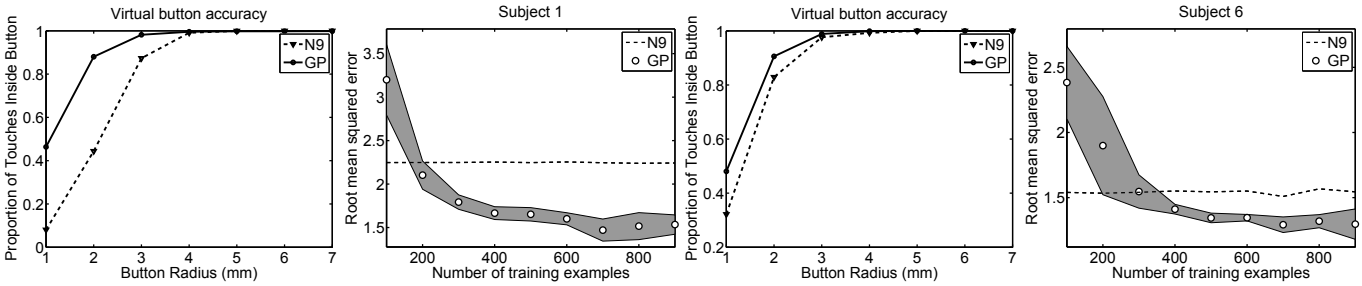
Importance of user-specificity

In Figure 7 we saw that the offset functions learned varied greatly across users. To quantify what this means in terms of performance, we carried out an additional experiment where we trained on a mixture of data from all subjects and evaluated the test performance of the resulting model on the data for each individual. Figure 12 shows our results. For each subject, two box plots are shown. The first shows the improvement in accuracy for a 2mm button over the N9 when training on 600 of that subject's touches. The second shows the performance improvement when the model is trained on



(a) Subject 8 binned offsets. (b) Subject 8 horizontal offset function. (c) Subject 8 vertical offset function.

Figure 8. Offsets learned by linear regression for subject 8.



(a) Subject 1 (b) Subject 1 (c) Subject 6 (d) Subject 6

Figure 10. Virtual button accuracy and learning curves for portrait touches

600 touches drawn at random from the data for all subjects. This experiment was done using the device location as the GP input. Improvements were also seen for 3mm and 4mm buttons (results not shown).

In all cases, the performance improvement for the user-specific models is higher than that for the model trained on all subjects. The differences are statistically significant for all subjects (paired t-test, $p < 0.05$). For subject 3, whose touch behaviour is significantly different from the norm (as evidenced by the earlier results), training on the other subjects' data results in a notable *decrease* in performance compared to the N9. Thus it is clear that there is significant variation between users, and that training user-specific models is advantageous.

DISCUSSION AND FUTURE WORK

Data Collection

Key to this approach is the collection of training data. There are several ways in which this could be done with minimal effort from the user, described below. Investigating the practical feasibility of these approaches is an area for future research.

One method for generating the data would be to make use of a calibration game [5]. In [6] a similar approach has been applied and was very successful. With such a tapping game it would be easily possible to generate the required number of touches in a recreational setting. Note that in [6], users provided, on average, many more touches than needed for our approach. An alternative would be to use keyboard entry as a source of training data. Since the approach described in this paper only requires approximately 200 touches (when predicting from the device's reported touch location) to perform

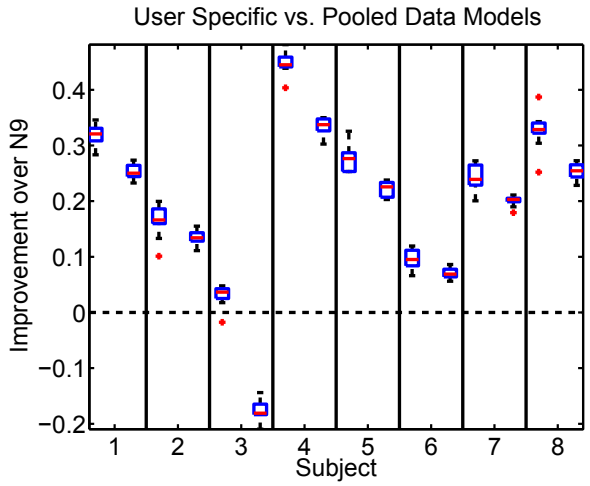
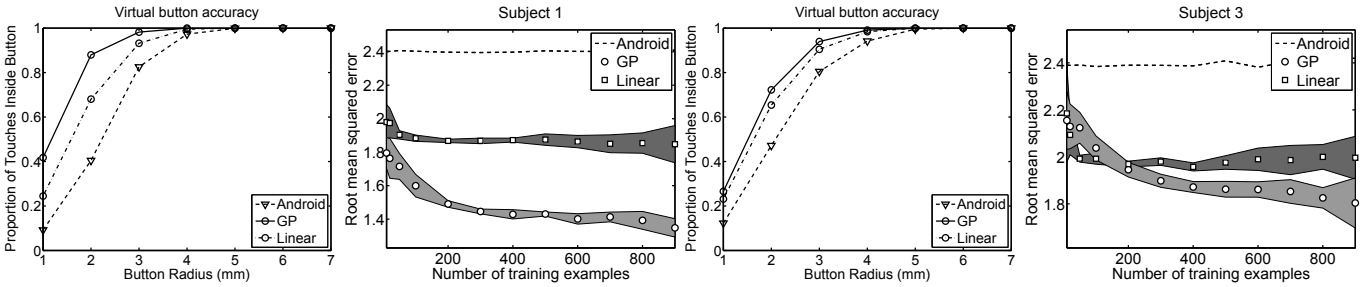


Figure 12. Improvements in accuracy rate for 2mm virtual buttons when using a user-specific model (left hand box plot in each case) and a model trained on all users (right hand box plot).

significantly better than the native touch location, writing one long email might already provide enough input data to train. Such an approach could be used in conjunction with the automatic keyboard optimisation proposed by [2]. In addition, the success of [7] showed that it is possible to combine typing and a calibration game.

Another possibility would be to introduce a new unlock method that collects the required data. Most mobile phones can be secured by a four digit PIN needed to unlock the device and this can serve as a data collection point. By arranging the numbers randomly on the screen the user



(a) Subject 1, Android (b) Subject 1, Android (c) Subject 3, Android (d) Subject 3, Android

Figure 11. Performance on Android phone

would need to touch on different places each time he wants to unlock his device. This could generate the needed touches in approximately 50 unlock procedures. Furthermore the approach would be safe against so-called smudge attacks [1] - reconstruction of frequently used patterns through oily residues on the screen - since the positions of the unlock digits would vary and no common pattern should emerge. The only drawback of this mechanism would be that users would not be able to generate a muscle memory to quickly unlock the device.

For longitudinal usage an iterative recalibration might be necessary since users might generate certain habits in touching or holding the device. A constant update e.g. using the keyboard approach as discussed above could be a feasible way of adapting the touch accuracy that might actually be capable of coping with temporary impairments such as a sprained thumb.

Predictive variance and uncertainty handover

The GP regression is naturally probabilistic, providing as its prediction a Gaussian density. Whilst we have only used the mean value in our accuracy calculations, the covariance could be useful from both a design perspective and an interaction perspective. Knowing which areas of the input space are the hardest to model (for example, the vertical strip in the centre when the user is using two thumbs in landscape mode, see Figure 9) can help in deciding where to place different sized buttons. Being able to model this in a user-specific manner could lead to devices that could optimise layout for accuracy as more data is acquired. From an interaction point of view, including uncertainty in input opens up the possibility of building schemes where autonomy can be transferred from the user to the device as shown in [15]. For example, uncertainty in touch locations could be used to produce a distribution over which key the user intended to press when operating a keyboard. This distribution could be fed forward into a probabilistic language model allowing for potential improvements in text entry accuracy. In [17] Schwarz et al presented a model which used uncertainty information to select between several small, closely packed buttons. The authors used a Gaussian centered on the detected touch area to represent the uncertain touch, but it is clear that the touch area is not necessarily a good indication of the intended target. The distributions produced by the GP automatically account for the offset and allow the full covariance structure to be learned for all parts of the space — using this information in application frameworks is a promising research direction which could open up new channels for rich touch interaction.

Computational complexity

Key to the success of a method like this is the ability to deploy it in real time on a mobile device. The GP regression approach has two distinct phases: training, involving inverting a square matrix of size $2 \times N$ (where N is the number of training touches) and predicting, which involves creating a covariance matrix and performing a matrix multiplication for each new touch event. The multiplications required for predictions will scale with the number of training examples. In our experiments, we found significant improvements for 400 training points (when the sensor data is used as input) and 200 (when the device's reported touch location is used). With matrices of this size, the operations are easily feasible on all modern mobile devices. In addition, our experiments used randomly placed training locations and it is highly likely that smarter position of these (perhaps using an active learning strategy), coupled with a sparse regression algorithm (e.g. [19]) could reduce the size of training sets further.

Incorporating additional data

GP regression provides a principled and flexible framework for incorporating additional data into the prediction task through combinations of covariance functions (see e.g. [12]). For example, one could include the current output from the accelerometers built into most mobile devices. This could allow the model to automatically adapt to different device orientations or different device usage contexts. For example, the model could adapt to give greater uncertainty in the predictions when the user is walking. In the text-entry scenario described above, the increased uncertainty would automatically give more influence to the probabilistic language model. The ability of the GP to probabilistically incorporate streams of data from multiple sensors is a key strength of our approach, and makes the GP a powerful tool for modelling uncertain interaction.

Multiple users on one device

One drawback of a user-specific model is when multiple users use the same device. One way to overcome this would be to

allow the user to switch the adaptive model on or off, depending on who was using the device. Alternatively, it may be possible to automatically infer the particular user from the touch characteristics (particularly if raw sensor data is used) or estimated keyboard error rates (e.g. if the user starts making many mistakes) and switch the predictive model on and off accordingly.

CONCLUSIONS

In this paper we demonstrated the feasibility of a user-specific machine learning approach based on GP regression to increase user specific touch accuracy. Using raw sensor input from a capacitive touch screen we were able to improve touch accuracy on average by 23.47% for a 2mm button, 14.20% for a 3mm and 4.7% for a 4mm button. We also demonstrated that the approach using the touch location reported by the device as input resulted in slightly better performance, with accuracy increasing on average by 23.79% for a 2mm, 14.79% for a 3mm and 5.11% for a 4mm button. This has the potential to both improve the user experience and allow for smaller sized buttons.

We found that our approach worked in both portrait and landscape modes and across two different devices. As shown in [6], user-specific offsets exist in all devices studied and our improvements are therefore unlikely to be restricted to the two devices studied here.

A key finding in this research is that optimal offset functions are highly nonlinear, both horizontally and vertically and vary dramatically over the touch area and over users. For most users, the proposed GP approach considerably outperforms a linear model trained on the same data. User-specific models also outperformed models trained on pooled data, highlighting the diversity in user behaviour.

This work provides the basis for a wide range of further research. We have a flexible GP regression framework with which we can incorporate data from multiple sensors to intelligently deal with the inherent uncertainties of touch interaction. With this rich predictive model, we can easily build improved systems for important tasks such as text entry. Further, we can use the predictive uncertainty to respond appropriately and automatically to a range of usage contexts, or to negotiate the handover of autonomy between user and device.

ACKNOWLEDGEMENTS

Daryl Weir is supported by a PhD studentship from the Scottish Informatics and Computer Science Alliance (SICSA). Nokia provided equipment and funding to support this research.

REFERENCES

1. Aviv, A. J., Gibson, K., Mossop, E., Blaze, M., and Smith, J. M. Smudge attacks on smartphone touch screens. In *WOOT '10*, 1–7.
2. Baldwin, T., and Chai, J. Towards online adaptation and personalization of key-target resizing for mobile devices. In *IUI '12*, 11–20.
3. Baudisch, P., and Chu, G. Back-of-device interaction allows creating very small touch devices. In *CHI '09*, 1923–1932.
4. Findlater, L., and Wobbrock, J. O. Personalized Input: Improving Ten-Finger Touchscreen Typing through Automatic Adaptation. 815–824.
5. Flatla, D. R., Gutwin, C., Nacke, L. E., Bateman, S., and Mandryk, R. L. Calibration games: making calibration tasks enjoyable by adding motivating game elements. In *UIST '11*, 403–412.
6. Henze, N., Rukzio, E., and Boll, S. 100,000,000 taps: Analysis and improvement of touch performance in the large. In *MobileHCI '11*,
7. Henze, N., Rukzio, E., and Boll, S. Observational and experimental investigation of typing behaviour using virtual keyboards on mobile devices. In *CHI'12*,
8. Holz, C., and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *CHI '10*, 581–590.
9. Holz, C., and Baudisch, P. Understanding touch. In *CHI '11*, 2501–2510.
10. Lee, S., Buxton, W., and Smith, K. C. A multi-touch three dimensional touch-sensitive tablet. In *CHI '85*, 21–25.
11. Parhi, P., Karlson, A. K., and Bederson, B. B. Target size study for one-handed thumb use on small touchscreen devices. 203–210.
12. Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
13. Rogers, S., and Girolami, M. *A First Course in Machine Learning*. Chapman and Hall / CRC, 2011.
14. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. Anglepose: robust, precise capacitive touch tracking via 3d orientation estimation. In *CHI '11*, 2575–2584.
15. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. Fingercloud: uncertainty and autonomy handover in capacitive sensing. In *CHI '10*, 577–580.
16. Saponas, T. S., Harrison, C., and Benko, H. Pockettouch: through-fabric capacitive touch input. In *UIST '11*, 303–308.
17. Schwarz, J., Hudson, S., Mankoff, J., and Wilson, A. D. A framework for robust and flexible handling of inputs with uncertainty. In *UIST '10*, 47.
18. Schwarz, J., Mankoff, J., and Hudson, S. Monte carlo methods for managing interactive state, action and feedback under uncertainty. In *UIST '11*, 235.
19. Tipping, M. Sparse Bayesian learning and the Relevance Vector Machine. *Journal of Machine Learning Research 1*, 211–244.
20. Williamson, J. *Continuous Uncertain Interaction*. PhD thesis, University of Glasgow, 2006.