User-Specific Touch Models in a Cross-Device Context

Daniel Buschek
University of Munich
Amalienstr. 17, 80333 Munich, Germany
buschek@cip.ifi.lmu.de

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

ABSTRACT

We present a machine learning approach to train user-specific offset models, which map actual to intended touch locations to improve accuracy. We propose a flexible framework to adapt and apply models trained on touch data from one device and user to others. This paper presents a study of the first published experimental data from multiple devices per user, and indicates that models not only improve accuracy between repeated sessions for the same user, but across devices and users, too. Device-specific models outperform unadapted user-specific models from different devices. However, with both user- and device-specific data, we demonstrate that our approach allows to combine this information to adapt models to the targeted device resulting in significant improvement. On average, adapted models improved accuracy by over 8%. We show that models can be obtained from a small number of touches (≈ 60). We also apply models to predict input-styles and identify users.

Author Keywords

Touch; Machine Learning; Regression; Probabilistic Modelling; User-Specific; Device-Specific

ACM Classification Keywords

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

INTRODUCTION

It is desirable that devices adapt to the user, not the other way round. Machine learning techniques can help touch devices to adapt, using models of touch behaviour. In this context, modelling so far mainly focussed on data from a single device per user, more for practical than conceptual reasons. To examine user- and device-specific touch influences, we present a modelling approach with respect to data from multiple devices per user. User-specific offset models are trained to map actual to intended touch locations. We discuss concepts to adapt and transfer these models across devices and users. We further utilise our transfer approach to examine user and device variability. In a practical application, we deploy these models to improve touch accuracy and find that users benefit from their models not only on the same device in one session, but across devices and repeated sessions as well. Finally, we further apply offset models to predict input-styles and identify users.

MobileHCI '13, August 27 - 30 2013, Munich, Germany

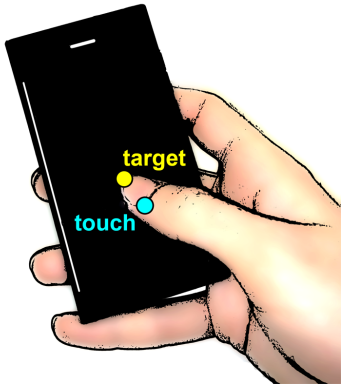

Figure 1. Touch offset. The user intended to hit a target to the upper left of the actual touch location. The difference between both locations is a two-dimensional offset vector. Given previously recorded touch data, offset models can predict the target location for future touches to improve the user's touch accuracy.

Precise input on a touchscreen is hindered by several well-known problems: The finger covers the relevant area of the screen, so the target is no longer visible ([14], [7]). The screen is flat and offers no tangible cues about the content. Users might also visually target with a different part of their finger than the one with which they actually touch the screen, see for example [8]. These problems lead to an offset between the intended and the actual touch location sensed by the device. Figure 1 shows an example of this situation.

Related research shows that size and direction of touch offsets depend on the screen location, the user and the device: Users show different offsets at different positions on the screen [16], which could in some way be attributed to the movement and size of the finger, the input-style (e.g. one hand and thumb [9]), and so on. Characteristics vary between users, which suggests a user-specific approach [16]. Device-specific influences exist as well, like size and shape, the aspect ratio of the screen or the resolution [6].

The remainder of this paper is structured as follows: First, we describe a general method to model user-specific touch

offsets. Second, we present several approaches to transfer models across devices and users, based on different assumptions. We then report results from our user study, in which we applied these concepts to touch data. Here, we discuss user-specificity, a comparison of user- and device-specific influences and performances when applied to improve touch accuracy. We then apply offset models to two other tasks, input-style prediction and user identification. Finally, we draw a conclusion and show possibilities for future work.

RELATED WORK

A user-specific machine learning approach to model offsets between intended and actual touch locations is presented in [16]. The authors used Gaussian Process (GP; [10]) models in a landscape orientation, two-thumbs input setting and reported highly non-linear surfaces. They also demonstrated the relevance of user-specific modelling. They mainly focussed on a single device per user.

In a larger study, offset functions were derived from touch data collected via the android app market [6]. The authors found an improvement in accuracy achieved with these functions in a portrait orientation, one thumb setting, similar to the setup we observe in this paper. For each user there is only one phone in the dataset. In contrast, we collected data on multiple phones per user.

Other approaches improve touch accuracy with rich information about the finger: [7] inferred the pose with a fingerprint reader. [12] used additional sensors and advanced Bayesian techniques. Although these approaches are successful, they require custom hardware or costly computations. Both [7] and [12] reported noticeable differences between users, which further suggests a user-specific approach for offset models.

Touch accuracy has also been improved with special input methods. [13] proposed techniques with area of interest magnification and magnetic pointers. [14] use callouts in ambigious situations to avoid target occlusion. However, these concepts require more effort per touch or change the touching procedure. [2] and [3] model user-specific touch behaviour to improve typing on software keyboards. They adapt key-target sizes and key-layouts, but such improvements only apply to this one task. We focus on a more abstract offset model, from which users can benefit throughout all tasks and without additional interface elements.

OFFSET MODELS

An offset model is defined as a function f that maps actual to intended touch locations. Figure 2 shows examples. They are derived from training examples using a machine learning approach. A training example is a tuple (a, t), where a denotes the actual location (touch) and t the intended location (target). A model is user-specific, if it is trained on data from a single user. It is device-specific, if training data comes from a single device. In this paper, we mainly use a quadratic model, i.e. a linear regression model with constant (bias), linear and quadratic terms. This was motivated by the observation that additional terms (e.g. cubic) did not have considerable influence. However, it is not the purpose of this paper to define the best possible model. We rather present model transfer concepts which can be applied to any suitable model.

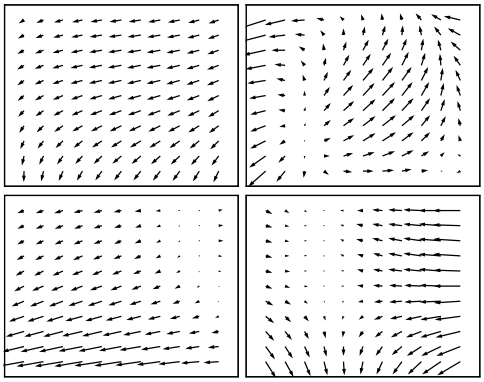

Figure 2. Four offset models. Each box represents the lower half of the screen in portrait orientation. Arrows indicate size and direction of offsets. They are scaled to better reveal the pattern.

Training

We start with N training tuples $x_1, ..., x_n, ..., x_N$ with touch and target locations $(a_1, t_1), ..., (a_n, t_n), ..., (a_N, t_N)$. Note that both a_n and t_n are 2D screen coordinates.

We define M basis functions $\Phi_1, \Phi_2, ..., \Phi_M$ to model non-linear surfaces with a linear (in parameters) model. For a quadratic model, we set $M = 5$ with $\Phi_1(x, y) = 1$; $\Phi_2(x, y) = x$ and $\Phi_3(x, y) = y$; $\Phi_4(x, y) = x^2$ and $\Phi_5(x, y) = y^2$.

We build the $N \times M$ design matrix X with $X_{nm} = \Phi_m(x_n)$, the m-th basis function evaluated for the n-th training example. Thus each row of X contains one training example and each column contains one input dimension.

Let t_x, t_y denote vectors containing the x- and y-coordinates of all N training targets t_n, respectively. Using the maximum likelihood solution, we arrive at two weight-vectors $w_x = (X^T X)^{-1} X^T t_x$ and $w_y = (X^T X)^{-1} X^T t_y$, which define our model: $f_x(a) = {w_x}^T a$ and $f_y(a) = {w_y}^T a$. Note, that we model horizontal and vertical offsets as functions of both x and y, in contrast to related research, see [6]. The models in figure 2 clearly show that this additional flexibility should not be omitted.

Prediction

For a new touch $a = (x, y)^T$ we build the vector $a' = \Phi(a)$. In our quadratic model we get $a' = (1, x, y, x^2, y^2)^T$. We can predict $t' = (x', y')^T$, with $x' = f_x(a')$ and $y' = f_y(a')$. Variances associated with these predictions can be computed as well: ${\sigma'}^2 = \hat{\sigma}^2 {a'}^T (X^T X)^{-1} a'$, where $\hat{\sigma}^2$ is the maximum likelihood estimate for the dataset noise variance:

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^{N} (t_n - w^T a_n)^2$$

Predictive variance could be used as an indicator for uncertain user intention for autonomy handover ("H-Metaphor"; [5]), as shown in [11].

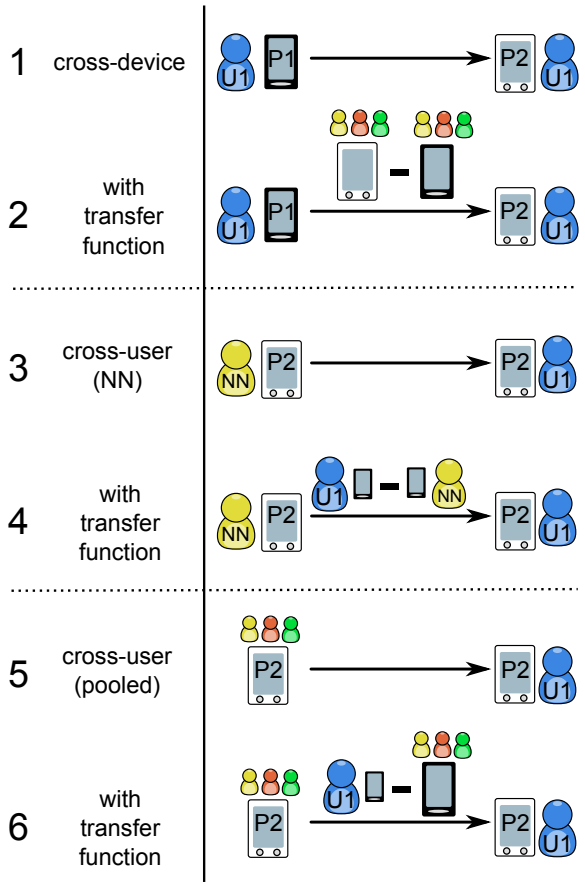

Figure 3. Visualised overview of model transfer approaches to derive a model for user $U1$ on phone $P2$. Cross-device transfer (1 and 2) assumes that a model of the same user from another device $P1$ will also work for that user on $P2$. Adaptation is possible based on the differences of both devices (2). Device-models are trained on data from all users to average out user-specific effects and capture device-specific ones. On the other hand, cross-user transfer (3-6) assumes that a model directly from $P2$ is a better choice, despite being trained on data from another user. In approach 3 and 4, we rely on the nearest neighbour (NN) of $U1$ on $P1$, assuming that both users will also be similar on $P2$. With data from that user on $P1$, adaptation is possible by comparing users on that phone and assuming similar differences on $P2$ as well (4). Instead of a single similar user, the last two approaches use data from all users of $P2$, i.e. device-models, assuming that users are similar on one device.

Collecting training data

Training data is a limitation in any practical application of offset models. However, in this paper, we are primarily interested in the possibility of model transfer itself. This also helps to improve our understanding of user- and device-specific effects on touch offsets, as well as possible relationships between them. In a deployed application, we could collect training data with a calibration phase or game [4]. Although in our study, we collected 200 random training touches per user, we show (in section Training set size) that modelling works well with a small subset (≈ 60). This number could be further reduced with an appropriate choice of target locations, instead of random ones. The user could also benefit from an existing model. It could be related to another user on the same device or to the same user on another device. Here, a model transfer is needed, but no new data or effort from the user.

MODEL TRANSFER

We want to derive a model for user u on phone P. No training data is available for this user and phone combination, but we have data from u on another phone. We also have data from other users on P. We have two options to derive the unknown model for u on P:

- **Cross-device transfer**: We use a model from the same user, but from a different device.
- **Cross-user transfer**: We use a model from a different user, but from the same device.

Figure 3 presents possible approaches. Their performance depends on the variability of users and devices: If offsets are highly user-specific, then a cross-device approach is the better choice because we stick to the same user. If the device has greater influence on the offsets than the user, we should prefer a cross-user approach, staying on the same device. We present both options in more detail in the following sections.

Transferring models across devices

The task of model transfer for user u from phone P_1 to another phone P_2 is defined as follows (see 1 in figure 3): A user-specific model is trained on data from u on P_1. This model is then tested on data from u on P_2. Therefore, we refer to P_1 as the training or original phone, and to P_2 as the testing or targeted phone.

We need to handle possible device differences, like different algorithms for sensor data processing or physical differences. We address these issues with a transfer function $f(x) = PM_2(x) - PM_1(x)$. This function is defined as the difference of predictions from two phone models PM_1 and PM_2. These models are trained on data from all (other) users from phone P_1 and P_2, respectively (see 2 in figure 3). Predictions from the user's own model are then altered using f. Figure 4 shows some examples.

For example, let us assume the case of a certain user u. We have data for that user on one phone P_1, but now want to derive a model for this user on another phone P_2. First, a model M_u is trained on that user's data from P_1. Then, both phone models PM_1 and PM_2 are trained for the relevant devices P_1, P_2 with data from all (other) users. Now we can define the transfer function $f(x) = PM_2(x) - PM_1(x)$. Finally, the prediction for a new touch (a, t) from that user is given by $t' = M_u(a) + f(a)$. Comparing t' with t allows to evaluate the quality of the prediction.

Transferring models across users

In a similar fashion, we can transfer models between users instead of phones. This task is defined as follows: For a given pair of phones P_1, P_2 and two users u, n a model is trained on data from user n with P_2 and tested on data from user u on that phone. This approach requires additional consideration, since, in general, there are a lot more users than phones. This raises the question which user n of phone P_2 to choose to train a model for u on that phone.

A possible approach (see 3 and 4 in figure 3) is to find the nearest neighbour (NN) of u, using the Euclidean distance

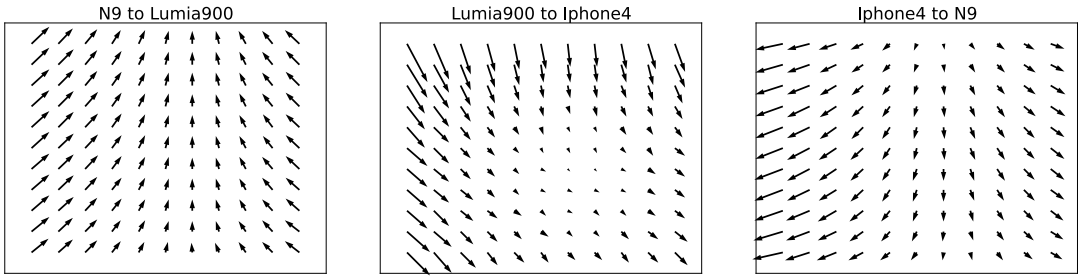

Figure 4. Cross-device transfer functions. When transferring offset models from one device to another, these functions are evaluated and added to the models' predictions to counter device-specific differences between both phones. They are defined as the difference in predictions of two device models. Device-specific models are trained on data from all users of that device to average out user-specific effects and capture only device-specific ones.

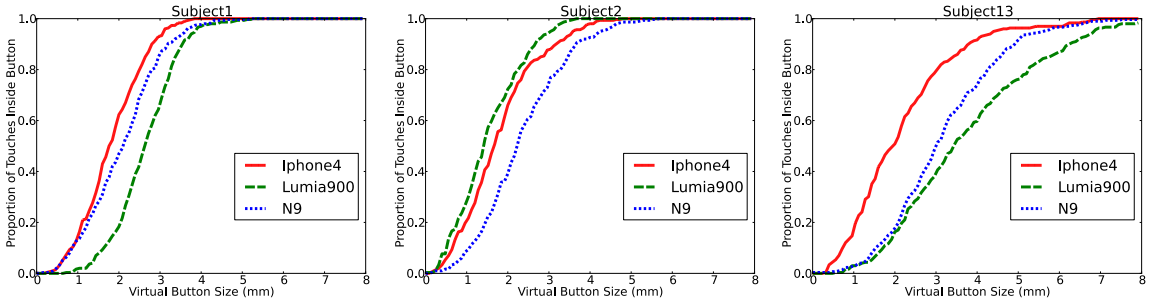

Figure 5. Virtual button accuracy (VBA) plots are cumulative histograms of offset vector lengths. The y-axis gives a user's performance (% hits) if targets in the experiment are treated like circular buttons of radius x. A steep slope indicates many small offsets and thus high accuracy. Subject 1 and 2, in the left and middle figures, show more experienced smartphone users, whereas subject 13 to the right owned no such device.

of the model parameter vectors for models from P_1 as the distance function. Here we assume that users who show similar behaviour on one phone, will do so as well on another phone. More subtly, this entails that users respond to new devices in a similar fashion. A cross-user transfer function $f(x) = M_u(x) - M_n(x)$ can then be defined as the difference of predictions from two user models M_u and M_n. These models are trained on data from P_1 from user u and n, respectively. To adapt the model of user n to be deployed for user u on the targeted phone P_2, its predictions are altered with f.

For example, let us assume again that we only have data for u on one phone P_1, but are interested in a model for another phone P_2. First, a model M_u is trained based on data of u from P_1. Then, we find the NN n of u on that phone. We train the model $M1_n$ for P_1 and $M2_n$ for P_2. We also define the transfer function $f(x) = M_u(x) - M1_n(x)$ as the difference between both users on the original phone P_1. Finally, the prediction for a new touch (a, t) from u on the targeted phone P_2 is given by $t' = M2_n(a) + f(a)$.

Instead of nearest neighbours we can train models on data from all (other) users of the original phone (as $M1_n$) and of the targeted phone (as $M2_n$), respectively (see 5 and 6 in figure 3). Here, we assume small variance between users.

Comparison

We presented two different model transfer approaches: The first one takes a model from the same user, but from another device and transfers it across devices. To overcome device differences, we proposed a transfer function based on the differences of device-specific models, created from data of all users. The second approach takes a model from the targeted device, but from another user. We suggested to find the nearest neighbour of the user in question, assuming both users to be similar on the other device as well. Here, a transfer function uses the differences between both users on the original phone. Alternatively, we train models with data from all users, assuming small variance between users on one phone.

EXPERIMENTAL SETUP

We implemented a web-app to collect touch data: Crosshairs (two lines of 20 pixels length) are shown on the lower half of the screen, one at a time. The lower half was chosen because it represents the usual keyboard area. Target locations are determined using JavaScript's Math.random() function to pick random x and y coordinates for each target. The centre of the target was at least 5 pixels away from the edges of the input area. The task is to tap on the target. The next target is then shown regardless of whether the user did actually hit the target or not. No feedback is given about where or how the user touched. Each participant touched 300 targets using the phone with the right hand and thumb in portrait orientation. They were asked to sit down and neither put emphasis on accuracy nor speed, but rather tap "naturally", like they would do when typing a message. Thirty different users completed the game on 13 different smartphones in total. Not every participant used every phone. However, the most data was collected for the Iphone 4 (25 users), Lumia 900 (22 users) and

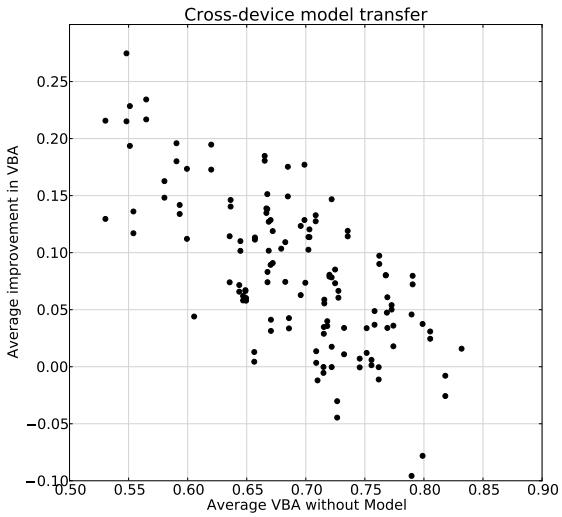

Figure 6. Cross-device model transfer results. Each dot represents a tuple (u, P_1, P_2), where u denotes the user, P_1 the original phone, and P_2 the target phone. The x-axis gives the average value of the VBA-plot for user u with phone P_2 without any model. This indicates the baseline accuracy. The y-axis then shows the improvement achieved with a model transferred from P_1. Improvement is the average difference between a VBA-plot derived from the cross-device approach and the baseline plot.

Nokia N9 (24 users). Our analysis focuses on this data, but also considers observations from other phones. Nine users came back for a second session. All in all, we collected over 45000 touches.

RESULTS

We evaluated the performance of the model transfer approaches with the dataset described above. We also compared user and device variability, as well as the variability of one user between repeated sessions on the same phone. Unless stated otherwise, we use relative coordinates, which means that, say, a target in the upper right corner has the same coordinates on every phone, regardless of the actual physical size of the device. Therefore, models are stretched or shrunk when transferred across devices. Relative coordinates are preferable, because not every application might have access to physical device sizes. However, careful consideration is needed regarding the influence of scaling effects.

Virtual button accuracy

Figure 5 presents a basic observation: The plots show cumulative histograms of offset vector lengths for three users. The y-axis shows a user's performance if targets in the experiment are treated like circular buttons of radius x. We therefore refer to these histograms as virtual button accuracy (VBA) plots. This choice of visualisation is useful to compare users: A steep slope indicates many small offsets and thus high accuracy. When we refer to an improvement in VBA in the following sections, we mean the (average) difference between two such VBA plots from different conditions: For example one with model application, one without. This gives a convenient measure of improvement with values between 0 and 1. It also allows for better intuition regarding a possible application with user interfaces, because we can refer to (virtual) buttons of specific sizes as well, by computing the difference of two plots for a fixed value on the x-axis. We associate a difference of 0.1 with 10% improvement in accuracy.

id	approach	units	avg	2 mm	3 mm	4 mm
1	cross-device	relative	5.25	15.17	11.56	5.77
		mm	4.59	13.65	10.08	4.83
2	with transfer function	relative	8.23	24.10	16.40	7.80
		mm	8.16	24.09	16.19	7.46
3	cross-user (NN)	relative	3.91	11.11	8.47	4.14
		mm	3.25	9.41	6.99	3.49
4	with transfer function	relative	6.20	17.59	13.32	6.64
		mm	6.48	18.96	13.45	6.59
5	cross-user (pooled)	relative	6.16	18.06	12.60	6.14
		mm	6.16	18.04	12.61	6.14
6	with transfer function	relative	8.12	23.76	16.20	7.70
		mm	8.06	23.78	16.09	7.36

Table 1. Model transfer results (improvements in %)

Model transfer

Figure 6 visualises the results of model transfer with the cross-device approach, using the three devices Iphone 4, Lumia 900 and N9. The shape of the distribution reveals that users with a high baseline accuracy tend to benefit less from their models, as there is less potential for improvement. In a few cases, the models could not help. However, in the vast majority of constellations the cross-device approach greatly improved accuracy. Table 1 summarises the results. The average improvement is 8.23% with relative coordinates, averaged over virtual buttons of a radius up to 8 mm. It is worth noting the average improvements for specific smaller sizes as well (2/3/4 mm). For comparison: Android's recommended width/height for touch targets is 7-10 mm ($\approx$ 3.5-5 mm radius), including gaps [1]. We furthermore find that the cross-device transfer functions add significant improvement (paired t-test, $p < 0.05$). Therefore, we state that using such functions is an appropriate way of adapting models across devices in this context.

For cross-user transfer, the approach with offset models trained on pooled data from all users outperformed the nearest neighbour (NN) approach significantly (paired t-test, $p < 0.05$). Again, we also observed a significant positive effect of transfer functions. Comparing all results, we find that the cross-device approach with transfer functions outperforms both cross-user approaches significantly (paired t-test, $p < 0.05$), although the difference to the cross-user approach with pooled models and transfer functions is relatively small. Note, that these two approaches use the same data: The user's own model, and models from all users per device. In conclusion, we find that if there is only device-specific data (approach 5), this gives better models than only user-specific data (approach 1). However, if we have both (2 and 6), then user-specific models give better predictions (2) than device-specific models (6). Table 1 further shows the results obtained with physical coordinates (mm from bottom right corner of the screen). The differences in the average cases with transfer functions are not significant (paired t-test, $p < 0.05$).

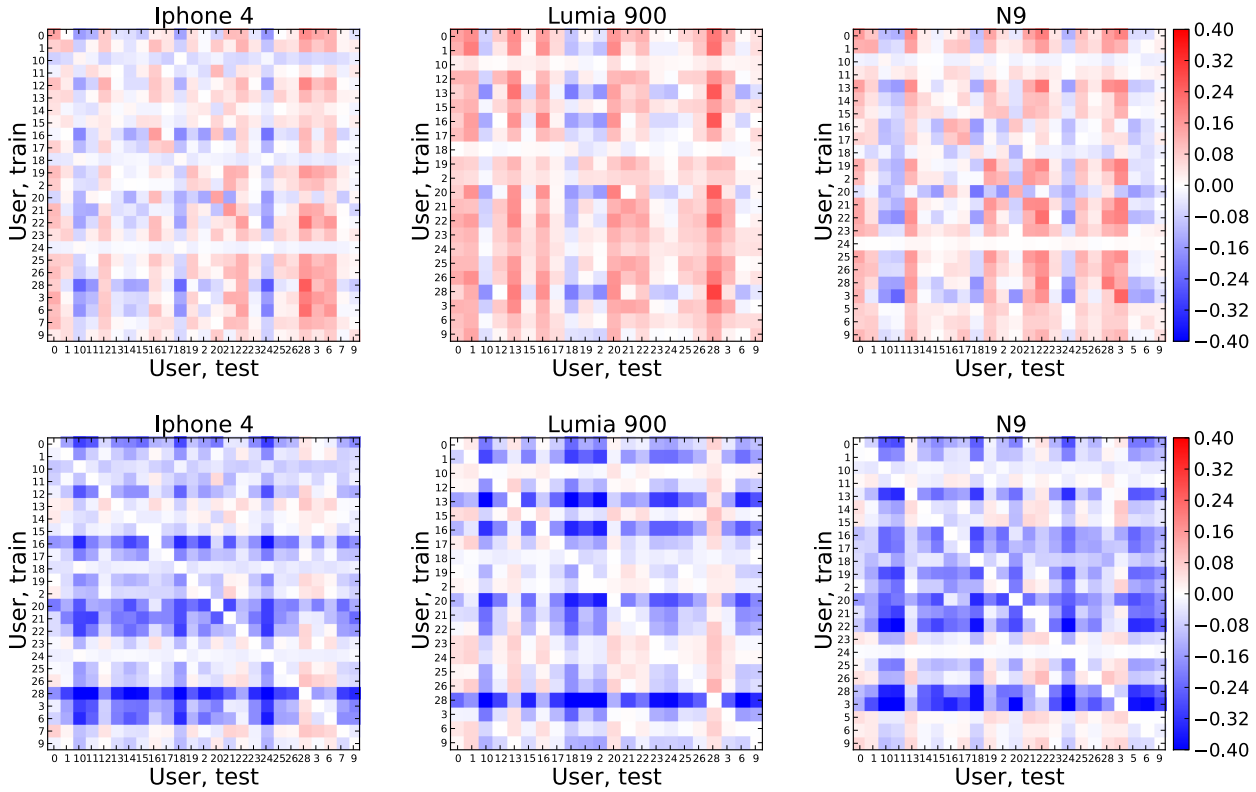

Figure 7. Cross-user model application improvements. In each matrix, an entry e_{ij} shows the improvement achieved when the model of subject i is applied to the data of subject j on the given phone. Along the diagonal ($i = j$), where models are applied to the same user, a 10-fold cross-validation was used to avoid training on test data. In the top row we see the absolute improvements, while in the bottom row we see relative improvement, i.e. the differences to the improvement achieved with the user's own model. Thus, the diagonals contain zeros in the bottom row of matrices.

User-specificity

We further trained models from one user on one phone and computed the improvement in VBA when applied to data from another user on the same phone. Figure 7 presents the results. Notice the red columns in the top row matrices: Some users benefit not only from their own model, but also from models of most other users. Some of these cases correspond with less touch experience - users, who owned no touchscreen device. This confirms our observations from model transfer (see figure 6). Blue cells in the three top plots appear when models had a negative influence on accuracy. The average improvements with models from the same user on the same phone (diagonals) are: 10.2% (average), 30.4% (2 mm), 19.1% (3 mm) and 8.5% (4 mm). We can compare these values with table 1 to note performance costs of model transfer.

Looking at relative improvements (bottom row), we find a lot of negative blue values, indicating less improvement than the user's own model. This shows the importance of user-specific information: In most cases, the user's own model is among the best models or the top one. The light red cells show, that in some cases the model of another user is slightly better. However, in the majority of constellations, models from other users turn out to be much worse. Notice the deep blue rows in these plots, which indicate users with the most "specific" behaviour: Their models don't work well for any other user.

User and device variability

In the previous section, we compared models across users on one phone and found that best results are achieved with the user's own model. It is of further interest whether this is still the case after model transfer. This leads to the following analysis: For a given user u and two phones P_1, P_2, we trained 1000 models by subsampling 200 of the 300 touches from u on the first phone P_1. We then transferred these models to the targeted phone P_2, using our cross-device approach. We computed the RMSE of these models, using the data from u on P_2. For each subsample, we also picked another user of P_2 at random and trained a model by subsampling in the same way from that user's data on P_2. We computed their RMSE when applied to the data of u on P_2 as well. If user-specificity transfers across devices, then the transferred models should achieve better results on the targeted phone than the models from other users, despite the fact that those are trained on data from that phone. If it is the other way round, then the device-specific information is more important.

Table 2 summarises the results of a paired t-test. The first and second column give the phones P_1, P_2 used for training and testing the model, respectively. The third column presents the proportion of users for which the own model achieved a lower RMSE E_u than the RMSE E_d of models from other users, as indicated by the t-values. Finally, the rightmost column

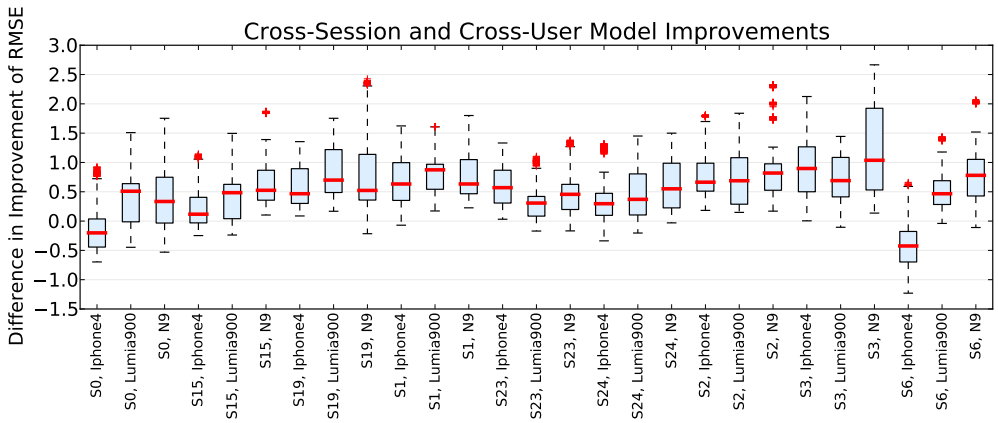

Figure 8. Cross-session and cross-user model improvements. Each Boxplot represents the differences in improvements between 1000 pairs of subsampled models: A (cross-session) model from the same user from a previous session, and a (cross-user) model from another randomly chosen user. In 25 of 27 cases, the user's own model from a previous session outperformed models from other users. This indicates that models stay similar between repeated sessions for one user.

P_1	P_2	units	$E_u < E_d$	$p < 0.05$
Iphone 4	Lumia 900	relative mm	86% 82%	86% 82%
Iphone 4	N9	relative mm	96% 91%	96% 91%
Lumia 900	Iphone 4	relative mm	91% 91%	91% 91%
Lumia 900	N9	relative mm	100% 100%	100% 100%
N9	Iphone 4	relative mm	91% 91%	91% 91%
N9	Lumia 900	relative mm	100% 100%	100% 100%

Table 2. User and device variability results

P_1	P_2	units	$E_u < E_d$	$p < 0.05$
Iphone 4	Lumia 900	relative mm	45% 45%	45% 36%
Iphone 4	N9	relative mm	78% 70%	78% 70%
Lumia 900	Iphone 4	relative mm	77% 77%	77% 77%
Lumia 900	N9	relative mm	95% 95%	95% 95%
N9	Iphone 4	relative mm	91% 82%	91% 78%
N9	Lumia 900	relative mm	66% 71%	66% 71%

Table 3. User and device variability results, pooled models

shows the proportion of users for which this effect was significant ($p < 0.05$). We observed results in clear favour of the user-specific information: In all phone combinations, transferred models from the same user performed significantly better than models from another user from the targeted device for more than 82% of the users. In conclusion, adapted models from the same user should be preferred over models from (single) other users of the targeted phone.

In a second evaluation, we trained device-specific models on data pooled from all users of the targeted phone P_2. We then compared these models to the transferred models. Table 3 summarises the results (paired t-test) like before. They are in favour of user-specific information: In four of six phone combinations, the transferred and adapted models from the same user performed significantly better than the pooled models from the targeted device for more than 77% of the users.

Both cases targeting the Lumia 900 stand out with lower results from the perspective of the user-specific model. We found that models are significantly more similar between users on the Lumia 900 than on the two other phones (paired t-test, $p < 0.05$). If users are similar anyway, user-specific information is not so important, while the other models benefit from being trained on the targeted phone. We repeated the analysis with a physical coordinate system to exclude negative model scaling effects, since the Lumia is larger than the other phones. Table 3 shows these results as well. They shifted in favour of the user's own models for "N9 to Lumia 900", but not for "Iphone 4 to Lumia 900". In the other cases, cross-device performance was the same or decreased. The Lumia's size might still be the reason for increased user similarity, but this will need further evaluation.

Session variability

So far, we addressed user-specific and device-specific issues. However, even the same user on the same device will not show the exact same touch behaviour all the time. This session variability is a potential source of problems for model applications: Models should still be valid if the user picks the phone up another time. Otherwise, they must be updated. To find out how touch offsets change between sessions, we asked participants to come back and repeat the experiment. Nine users returned with a gap of at least one week. We collected their touch data in the same way and on the same phones again.

To measure session variability, we compared models between sessions and users. Therefore, a user's touch behaviour is stable between two sessions on one phone if the models from both sessions are more similar to each other than models from this user compared to other users' models. In the following, the distance between models is defined as the euclidean distance of their weight-vectors.

For each user and phone, we trained 1000 models by subsampling 200 of the 300 touches from the first session. We computed the distances of these models to the model trained on all 300 touches from the second session of that user on the same phone. This gives distances related to session variability. For each subsample, we also picked another user at random and trained a model by subsampling in the same way from that user's data on this phone. We then computed the distance between both subsampled models. This gives a distance related to user variability. Applying a paired t-test on these pairs of distances lead to the following results: In 22 of 27 cases, models were significantly more similar to each other between sessions than between users ($p < 0.05$). In the five other cases, models were more similar between users than between sessions. Three of these five exceptions belong to a single user.

We applied the same concept again to evaluate improvements across sessions: Instead of model distances we measured the reductions in root mean squared error (RMSE) achieved with subsampled models (from the first session or from another user) on the data of the new session. Figure 8 shows the differences in improvements between cross-session and cross-user models. In all but two cases, the cross-session models from the same user outperformed models from other users significantly (paired t-test, $p < 0.05$). These results indicate that user-specific models are stable across sessions. Updates might still become necessary because of reasons not covered in this experiment, e.g. long-term changes or hand injury.

Training set size

Since we use a supervised learning task, we are interested in training models with as little data as possible. We thus evaluated model performances with varying training set sizes: For each user and phone, we compared their RMSE with models from fewer data to the one achieved with the maximum of 200 touches. With 60 randomly chosen training taps, models on average performed only 3.3% worse (i.e. their RMSE was 3.3% higher) than the maximum on the Iphone 4, 3.5% on the Lumia 900 and 3.4% on the N9. In conclusion, useful offset models can be trained from a small number of touches, probably less than an average message or e-mail requires.

APPLICATIONS

We have applied offset models to improve touch accuracy. However, there are other potential application scenarios as well. We present some of them in the following sections.

Input-style prediction

In our experiment, we also gathered a few sets of touches with the left hand and thumb. Using this data, we demonstrate an application of offset models to predict the input-style continuously as the user interacts with the device. In general, we need a model for each input-style. We can train these models on given data, from a calibration game or derive them from existing models from another phone or user, as described in this paper. We can then compute the likelihood of new inputs given each model. An application could report these likelihoods, process them further or simply decide for the input-style with the highest likelihood. In the following application, we focus on the task of hand prediction in portrait orientation, that means we want to predict whether the user is tapping with the left or right thumb. However, the same concept is also applicable to any other input-style.

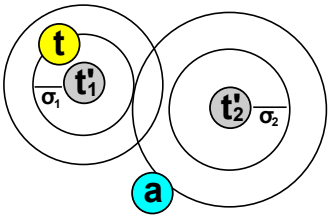

Figure 9. Input-style prediction. A touch location a is observed for the target location t. Circles indicate two distributions with means t'_1, t'_2 and standard deviations σ_1, σ_2, given by the predictions of the left and right hand model for this touch a, respectively. In this particular case, the likelihood of t is obviously higher given the distribution parametrised by the prediction t'_1, σ_1 of the right hand model. In conclusion, we predict that a is a touch with the right hand.

We define two classes C_1 (right hand) and C_2 (left hand). and train a model M_1 on data from the right hand and another model M_2 on data from the left hand. Let $x_n = (a_n, t_n)$ denote new inputs, with a_n representing the touch location and t_n the target location, as before. First, we compute the predictions t'_1, t'_2 and predictive variances σ_1^2, σ_2^2 with both models M_1, M_2, respectively. We then compute the likelihoods $P(x_n|C_1), P(x_n|C_2)$ of our observation, using Gaussian distributions with these predictions as their parameters:

$$P(x_n|C_1) = p(t_n|t'_1, \sigma_1^2) = \mathcal{N}(t'_1, \sigma_1^2)$$

$$P(x_n|C_2) = p(t_n|t'_2, \sigma_2^2) = \mathcal{N}(t'_2, \sigma_2^2)$$

This concept is presented in figure 9. We further define two priors $P(C_1), P(C_2)$. In the simplest case we set both to 0.5. Predictions are then given by Bayes' Theorem as follows:

$$P(C_1|x_n) = \frac{P(x_n|C_1)P(C_1)}{P(x_n|C_1)P(C_1) + P(x_n|C_2)P(C_2)}$$

$$P(C_2|x_n) = \frac{P(x_n|C_2)P(C_2)}{P(x_n|C_1)P(C_1) + P(x_n|C_2)P(C_2)}$$

For a simple decision we report "right" if $P(C_1|x_n) > P(C_2|x_n)$, and "left" in the other case. In the example of figure 9, the observation $x_n = (a, t)$ is much more likely given the right hand model, thus we predict a right hand touch.

In an evaluation with seven subjects on the Iphone 4, this approach achieved an average accuracy of 84% in predicting the correct hand for the current touch, using only 30 randomly chosen taps per class as training examples. Possible extensions to this approach could consider users to be less likely to switch than to stick to the current input-style. This

t	# training	$s = 25$	$s = 50$	$s = 100$	$s = 200$
1	30	69	69	69	69
	100	71	71	71	71
5	30	73	75	77	77
	100	75	78	79	80
10	30	69	74	77	79
	100	71	76	79	81

Table 4. User identification results (accuracy in %)

can be included as prior knowledge by adjusting the priors $P(C_1), P(C_2)$. Furthermore, we can treat successive touches as a joint distribution instead of single observations and/or average over the last t observations for each class. This increases stability so that the prediction will not switch instantly, but only if enough evidence has been accumulated over the last touches. Finally, the approach presented here is perfectly capable of handling additional features besides touch offsets, like accelerometer sensor values. These could further increase the accuracy of the prediction or allow for richer observations, like a walking condition.

Readers are invited to try our demo application with their smartphones. It is available as a web-app using HTML 5 and JavaScript at: `http://www.dcs.gla.ac.uk/taps/demos/`

The demo shows that training and prediction are feasible to perform in a web browser. This might be interesting for responsive mobile web design. Layouts could be adapted depending on the input-style to help users reach important elements or to avoid occlusion of content by the fingers.

User identification

For user identification, we utilise user-specificity and the observation that users are more similar to their own data from previous sessions than to other users. In the same way we predicted left or right handed input, we can predict which of two users is currently tapping. Table 4 shows the results for 300 pairs of users on an Iphone 4. The first column denotes the number of last taps t, which are used for prediction as a joint distribution. The second column contains the number of randomly chosen training examples. The remaining columns present the average accuracy in percent, when we feed test examples alternating between both users in packs of size s, i.e. assuming users would hand over the phone every s touches. These results show the benefit of considering more than just the last touch and great potential for user identification with offset models, using a small set of training examples.

Interface elements as targets

Both applications described above require target locations to compute offsets. In practice, it is unlikely to have only one point-like target on the screen. However, interface elements can be used to derive target locations as well. Let us consider a set of buttons: For a given touch, we do not know which button the user wanted to hit, but we can assume that they wanted to hit something, i.e. the touch has intention. Therefore, we can use a Gaussian mixture model, with a Gaussian distribution centred on each button (see figure 10). We compute the predictions of each class and their likelihoods given

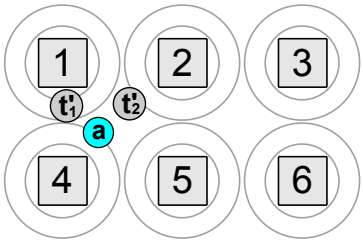

Figure 10. A Gaussian mixture model for an interface: For the touch location a and the predictions t'_1 (right), t'_2 (left), we compute the mixture model likelihood of the predicted points for each model (right and left) and normalise these to give a probability for each model. In this case, a is more likely to be a right hand touch aiming for button 1, than a left hand touch aiming for button 2. Therefore, we can predict the input-style without a given target location.

t	# training	$s = 25$	$s = 50$	$s = 100$	$s = 200$
1	30	69	69	69	69
	100	70	70	70	70
5	30	74	77	75	75
	100	77	79	80	80
10	30	69	76	76	75
	100	71	78	80	79

Table 5. Input-style prediction results, using buttons (accuracy in %)

the mixture model. We evaluated this idea for the input-style prediction task on an interface with 3×2 squared buttons with seven users on an Iphone 4. Table 5 presents the results. We observed average accuracies of up to 80%. In conclusion, we find potential for inference with interface elements as well.

CONCLUSIONS AND FUTURE WORK

We presented a machine learning approach to model user-specific touch offsets. We proposed transfer functions to adapt these models to other devices or users. This modelling approach can handle data from multiple devices per user without changing individual user- or device-specific models themselves. Thus we can easily adapt predictions to any user and device combination. Our approach models devices in terms of touch behaviour as well, so no detailed device-specific information like shape, sensors or internal algorithms is needed. In conclusion, this provides a flexible foundation to work with user-specific touch models in a cross-device context. Our framework can easily be extended to handle richer models as well. They could include additional features, like sensor data, which offers a broad range of possibilities for future work.

Our specific offset model is a linear regression model with quadratic terms. This choice is based on general observations and clearly limits the possible complexity of behaviour which can be modelled. However, the transfer framework can be applied to any offset model.

We demonstrated that useful models can be obtained from a small number of training examples ($\approx$ 60 in a portrait, one thumb setting). This could possibly be reduced even further with sparse modelling approaches and an analysis of where to best sample touches on the screen, as shown in [15].

We evaluated these concepts in a user study. Here, we applied our modelling approach to improve touch accuracy. Without transfer functions, we found that average device models outperform a user's own model from a different device significantly with average improvements in touch accuracy of 6.16% (device-specific) against 5.25% (user-specific). With both user- and device-specific data, we can combine this information using transfer functions. Adapting user-specific models with device-specific ones resulted in an average improvement of 8.23%. For specific virtual button sizes we observed 24.10% for 2 mm, 16.40% for 3 mm, and 7.80% for 4 mm. Adapting device-specific models with user-specific ones achieved 6.20% with nearest-neighbours and 8.12% with pooled models. In conclusion, device-specific information can leverage user-specific data in our approach: Each on their own, device-specific models outperform user-specific models (from a different device). If however both are available, then user-specific models should be preferred for prediction and device-specific models for adaptation.

Our study has some limitations: First, due to the lack of feedback, concentration could have degraded over the course of the trial. Thus, observed targeting performances might have to be considered pessimistic. Second, the task only involved interaction with abstract targets (crosshairs). Therefore, an interesting direction for future research is to compare offsets between different actual interface elements and UI styles.

We further examined user-specificity on one phone. Here we found that the user's own model outperforms models from other users in most cases. Transferred and adapted models also outperformed models from other users trained directly on the targeted device. We observed the same tendency when transferred models are compared to device-specific models trained on multiple users, with the exception of one targeted phone. These results support a user-specific approach for modelling touch offsets and give rise to application ideas, which make use of the variability between users, e.g. user identification.

On the Lumia 900, the largest of the phones in our study, we found users to be more similar to each other. Large screens could diminish the benefit of user-specific information. This should be further explored, also with other larger devices, e.g. tablets.

We evaluated how stable touch offset patterns stay between repeated sessions for a given user and device. We found that users benefit more from their own model from a previous session than from other users' models. Therefore, we suggest that applications do not need to recalibrate offset models in each session.

Finally, we presented an application which uses offset models to predict the current input-style. We implemented this concept to predict left- or right-handed interaction on a mobile website, also demonstrating the feasibility of our approach in a web-context. We observed an average accuracy of 84%. We also presented a task of user identification, where we predicted which of two users is currently tapping with an average accuracy of up to 81%. Utilising input- or user-specific aspects and session-stability of offset models is a promising approach for inference based on continuous touch input.

ACKNOWLEDGEMENTS

Nokia provided equipment and funding to support this research.

REFERENCES

1. Metrics and Grids - Android Developers. Retrieved May 09, 2013, from: http://developer.android.com/design/style/metrics-grids.html.
2. Baldwin, T., and Chai, J. Towards online adaptation and personalization of key-target resizing for mobile devices. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, ACM (2012), 11–20.
3. Findlater, L., and Wobbrock, J. Personalized input: Improving ten-finger touchscreen typing through automatic adaptation. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM (2012), 815–824.
4. Flatla, D., Gutwin, C., Nacke, L., Bateman, S., and Mandryk, R. Calibration games: making calibration tasks enjoyable by adding motivating game elements. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 403–412.
5. Flemisch, F., Adams, C., Conway, S., Goodrich, K., Palmer, M., and Schutte, P. The h-metaphor as a guideline for vehicle automation and interaction. Tech. rep., NASA/TM-2003-212672, 2003.
6. Henze, N., Rukzio, E., and Boll, S. 100,000,000 taps: analysis and improvement of touch performance in the large. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, ACM (2011), 133–142.
7. Holz, C., and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2010), 581–590.
8. Holz, C., and Baudisch, P. Understanding touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 2501–2510.
9. Parhi, P., Karlson, A., and Bederson, B. Target size study for one-handed thumb use on small touchscreen devices. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, ACM (2006), 203–210.
10. Rasmussen, C., and Williams, C. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.
11. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. Fingercloud: uncertainty and autonomy handover incapacitive sensing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2010), 577–580.
12. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. Anglepose: robust, precise capacitive touch tracking via 3d orientation estimation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 2575–2584.
13. Roudaut, A., Huot, S., and Lecolinet, E. Taptap and magstick: improving one-handed target acquisition on small touch-screens. In *Proceedings of the working conference on Advanced visual interfaces*, ACM (2008), 146–153.
14. Vogel, D., and Baudisch, P. Shift: a technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, vol. 28, Citeseer (2007), 657–666.
15. Weir, D., Buschek, D., and Rogers, S. Sparse selection of training data for touch correction systems. To appear in MobileHCI 2013.
16. Weir, D., Rogers, S., Murray-Smith, R., and Löchtefeld, M. A user-specific machine learning approach for improving touch accuracy on mobile devices. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, ACM (2012), 465–476.