MODELLING UNCERTAINTY IN TOUCH INTERACTION

DARYL WEIR

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

SCHOOL OF COMPUTING SCIENCE

COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

SEPTEMBER 2014

Abstract

Touch interaction is an increasingly ubiquitous input modality on modern devices. It appears on devices including phones, tablets, smartwatches and even some recent laptops. Despite its popularity, touch as an input technology suffers from a high level of measurement uncertainty. This stems from issues such as the 'fat finger problem', where the soft pad of the finger creates an ambiguous contact region with the screen that must be approximated by a single touch point. In addition to these physical uncertainties, there are issues of uncertainty of *intent* when the user is unsure of the goal of a touch. Perhaps the most common example is when typing a word, the user may be unsure of the spelling leading to touches on the wrong keys.

The uncertainty of touch leads to an *offset* between the user's intended target and the touch position recorded by the device. While numerous models have been proposed to model and correct for these offsets, existing techniques in general have assumed that the offset is a deterministic function of the input. We observe that this is not the case — touch also exhibits a random component. We propose in this dissertation that this property makes touch an excellent target for analysis using probabilistic techniques from machine learning. These techniques allow us to quantify the uncertainty expressed by a given touch, and the core assertion of our work is that this allows useful improvements to touch interaction to be obtained.

We show this through a number of studies. In Chapter 4, we apply Gaussian Process regression to the touch offset problem, producing models which allow very accurate selection of small targets. In the process, we observe that offsets are both highly non-linear and highly user-specific. In Chapter 5, we make use of the predictive uncertainty of the GP model when applied to a soft keyboard — this allows us to obtain key press probabilities which we combine with a language model to perform autocorrection. In Chapter 6, we introduce an extension to this framework in which users are given direct control over the level of uncertainty they express. We show that not only can users control such a system succesfully, they can use it to improve their performance when typing words not known to the language model. Finally, in Chapter 7 we show that users' touch behaviour is signficantly different across different tasks, particularly for typing compared to pointing tasks. We use this to motivate an investigation of the use of a sparse regression algorithm, the Relevance Vector Machine, to train offset models using small amounts of data.

Acknowledgements

As with any PhD thesis, this document would not exist without the contributions of a great many people. First and foremost, I wish to thank my advisor, Simon Rogers, for countless useful insights and for motivating me throughout my PhD. Thanks also to the rest of my supervisory team, Muffy Calder and Mark Dunlop, for their input. Roderick Murray-Smith and John Williamson also provided numerous valuable insights throughout the process.

My gratitude goes out to everyone I've shared office space with over the past four years for moral support, stimulating conversations and advice. In no particular order, thanks to Lauren, Mel, Stuart, Daniel, Daniel, Bea, Henning, Mozghan, and Xiaoyu. Honorary office mate status goes to Roseanne for all the cake.

I was lucky enough to spend three months working for Microsoft in Tampere, Finland during my studies, and I thank Seppo Turunen and Mika Kuulusa for helping to arrange that.

Thanks to all my collaborators and everyone who participated in the experiments.

I owe a significant debt to my friends for keeping me sane through what was often a stressful time. Hollie, Thomas, Emma, Dave, Mark, John, Keith — you've been incredible. My mum Jackie, dad Paul and brother Bradley have been a constant source of support and friendship, and I can't express how much that's meant. Thank you all from the bottom of my heart. To the many people who I've undoubtedly missed from this list, sorry for that and thanks for the help.

To Martha and James Kenyon, without whom I'd be a very different person indeed.

Table of Contents

List of Figures

Chapter 1

Introduction

Touch interaction is a ubiquitous input modality on devices from phones to laptops, and even on newer technology like smartwatches. However, touch is a process with a high level of inherent uncertainty. This uncertainty arises from a number of sources. These include: the 'fat finger problem', which describes the fact that the soft pad of the finger creates an ambiguity in touch location; occlusion, where users obscure the targets they aim at when touching; sensing issues; and uncertainty of intent — for example, when typing a word the user might not know the spelling, leading to touches on the wrong keys.

The combined result of these factors is an *offset* between the targets a user tries to touch and the position at which their touch is recorded. This has been modelled both in commercial devices (the original Android keyboard added a 10 pixel vertical shift to all touches) and in research. However, existing models typically assume that this is a systematic effect and treat the corrected input as certain. This is not necessarily the case.

Touch also has a random component. If a user tries to accomplish the same task multiple times, their touches will fall with some random spread. This uncertainty is typically ignored in existing models of touch, but we believe that this is a mistake.

The random component of touch makes it a candidate for analysis using probabilistic techniques from machine learning. In this thesis, we shall show the results of a number of applications of such techniques to the touch problem. We show that by taking the uncertainty of the process into account, performance gains can be made. Moreover, we show that users can actually make use of a system in which they can explicitly control the level of uncertainty they express to the system with a touch input, and that this can be be beneficial when entering text.

1.1 Thesis Statement

Touch interaction is ubiquitous on modern mobile devices, but is still prone to inaccuracy and error. This is due in large part to the inherent uncertainty related to touch, which stems from a number of sources: ambiguity from the soft finger touching small targets; uncertainty introduced by the hardware; and uncertainty of intent (e.g. the user being unsure of a word's spelling while typing). The core assertion of this work is that by modelling this uncertainty, the quality of touch interactions can be improved.

We support this assertion through a number of experiments. In particular, we propose that probabilistic modelling of the offsets between a users intended and recorded touch locations can be used to allow more accurate selection of small targets. Further, this technique can also improve the accuracy of typing on a soft keyboard. This improvement is shown through the results of a number of user studies, highlighting the benefit of implicit uncertainty modelling. Additionally, this approach is contrasted with an explicit uncertainty model, in which the user can control the ability of the system to correct their input. We show that both approaches have merit and can lead to increased performance in the text entry task. Finally, we present the results of several studies into the training data requirements of the models described in the rest of the thesis, and a sparse solution which potentially allows these models to be deployed in the real world with small training sets.

1.2 List of Contributing Papers

The work described in this thesis has led to three conference papers. These are as follows:

1. D. Weir, S. Rogers, R. Murray-Smith, and M. Löchtefeld, "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 Press, 2012, pp. 465–476.

2. D. Weir, D. Buschek, and S. Rogers, "Sparse selection of training data for touch correction systems," in *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services — MobileHCI 13*. ACM Press, 2013, pp. 404–407.

3. D. Weir, H. Pohl, S. Rogers, K. Vertanen, and P. O. Kristensson, "Uncertain text entry on mobile devices," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems — CHI 14*. ACM Press, 2014, pp. 2307–2316

Chapter 4 of this thesis is based on the first of these papers, Chapters 5 and 6 are based on the second paper, and Chapter 7 contains elements of the third paper. The analyses given in this thesis are expanded treatments of the work in the papers, together with some additional unpublished research.

1.3 Overview of Thesis and Research Contributions

The contributions made by this thesis are:

- A flexible offset model based on Gaussian Processes which describes the non-linear, user-specific behaviour seen in touch in a fully probabilistic way.
- A text entry system which combines this touch model with a statistical language model, representing a novel approach to touch variability in the text entry task.
- A study of the use of input pressure as a mechanism for negotiating uncertainty with a text entry system, in order to prevent autocorrections on words outside the system's vocabulary.
- A detailed examination of the differences in touch behaviour across different tasks, finding that touch behaviour for typing is significantly different than when pointing at arbitary targets
- An exploration of the use of sparse machine learning techniques to learn touch models with very small amounts of training data.

The remainder of the thesis is laid out as follows:

Chapter 2 discusses the research literature on which the remainder of the thesis builds. This includes a thorough explanation of the nature and cause of the problems with touch, and a review of the techniques that have been proposed to solve these problems. Additionally, some discussion of the idea of autonomy handover between interactive systems and their users is given, with examples from previous work.

Chapter 3 introduces the probabilistic modelling techniques used in the rest of the thesis — Gaussian Process regression and Relevance Vector Machine regression.

Chapter 4 presents the use of Gaussian Process regression to model touch offsets. The results of a user study demonstrate that offsets are highly non-linear and user-specific, and that the GP model allows very accurate selection of small targets.

Chapter 5 applies the GP offset model to the text entry task. This makes use of the probabilistic nature of the GP predictions, in conjunction with a statistical language model, to

build an autocorrect system which is shown in a user study to perform as well as a leading commercial keyboard.

Chapter 6 presents an alternate interaction approach to our uncertainty model, in which users are given explicit control over the uncertainty they communicate to the system. This allows them to prevent unwanted autocorrections when entering text that is not in the vocabulary of a language model. This Chapter makes two contributions: first showing that users can successfully control such a system; and secondly demonstrating that this control allows them to enter text with fewer manual corrections.

Chapter 7 discusses the variation of offset behaviour across tasks. It shows that offset models trained on target acquisition tasks (where users press buttons or crosshairs) do not generalise well to predicting touch offsets when a user types. This implies that a successful touch system requires multiple offset models. Given this, it is important to be able to build these models using as little training data as possible, and to identify which corrective model to apply at any given time. Thus, the Chapter also investigates the use of Relevance Vector Machines to build sparse offset models using 10 or fewer training points. It is shown that even with such a small amount of data, performance gains can be made.

Chapter 8 summarises the work, presents potential avenues for future research, and concludes the thesis.

Chapter 2

Background

Summary. This chapter provides an overview of the relevant research literature related to this work. This includes a review of touch interaction techniques, the problems inherent to touch, and existing approaches to improving touch accuracy. In addition, we discuss the concept of autonomy handover.

2.1 Introduction

This thesis presents a series of models for touch interaction based on some machine learning techniques, which are discussed in Chapter 3. In particular, these models are designed to improve the quality of touch interactions on mobile devices. There is an extensive history of research on touch, and Section 2.2 summarises the important previous work. This includes the development of touch hardware and the rise of multi-touch, as well as work looking at the problems facing users of touch technology. A look at existing models for correcting touch errors is also given.

In addition to modelling and correcting touch errors, the other primary motivation for the work presented here is the idea of uncertainty handover. This is the concept of allowing users to control the level of freedom a system has to correct their input, automate a task or take some 'smart' action. This is often useful behaviour, but there are many situations where users might wish to retain fine control. Section 2.5 gives an overview of work on this subject.

2.2 Touch Sensing Technology

2.2.1 Early Touch Systems

Touchscreen technology has existed for some time. The first touchscreens were created by Johnson [2] at the Royal Radar Establishment in 1965. His system was based on the change in capacitance observed when touching a grid of wires overlaid on a monitor. He envisaged the use of the technology in improving the efficiency of Air Traffic Controllers, but noted that the potential applicability might extend to all computer displays. The system was quite simple, supporting the detection of only a single finger.

Other capacitive touch sensors were developed at CERN [3] in the 1970s to aid in the control of their particle accelerators. These transparent sensors more closely resemble the technology found in modern devices, but were still limited to single touch detection.

Around the same time, the University of Illinois obtained a patent [4] for a touch screen based on infrared sensing. Fingers were detected when they broke a series of infrared beams just above the screen. This had the advantage of being able to detect any opaque object near the screen, unlike capacitive sensors which could detect only capacitive objects like fingers or metal styli. This technology was commercialised in 1983 in the Hewlett-Packard HP-150 personal computer.

Resistive touch screen technology was first patented by Samuel Hurst in 1975 [5]. Hurst's company Elographics also mass produced the first resistive screen in 1982. This technology works by applying voltages to a series of thin layers and measuring the response. Touches press the layers together and alter the voltage response, allowing coordinates to be calculated.

Resistive technology has the advantage of being able to work with any stylus-like object, unlike capacitive screens which only work with capacitive objects like the human finger. This allows users to operate these screens while wearing gloves or using an arbitary pen, for example. The technology is also cheap to implement — the most basic implementation requires only four wires. For this reason resistive technology was the primary choice in portable devices like PDAs for many years, and has only recently been supplanted by capacitive screens in the majority of smartphones. One of the primary reasons that capacitive screens have displaced resistive ones is that resistive sensing can only reliably detect a single contact area, rather than many as is possible with capacitive screens. Today, resistive screens are typically found only on low-end phones.

2.2.2 Multitouch Devices

Optical Sensing

Multitouch sensing — that is, detecting more than one point of contact at a time — was first developed at the University of Toronto in 1982 [6]. Their system, the Flexible Machine Interace, consisted of a frosted glass panel with a camera under it. When fingers were in contact with the panel, they created dark spots which could be observed by the camera and mapped to touch coordinates in software. This system did not offer any visual feedback, and acted only as an input device.

Other early camera based multitouch systems include VideoPlace [7] and DigitalDesk [8], which allowed users to interact with images projected on the environment around them. These papers broke ground in their creation of multitouch gestures such as 'pinch-to-zoom', which are ubiquitous on modern mobile devices. However, the sensing technology was based on somewhat clunky camera setups which were expensive and not portable.

Over the course of the next 20 years, a number of tabletop multitouch systems were created, such as ActiveDesk [9] and DiamondTouch [10]. These systems used rear projected displays and optical sensing. Additionally, DiamondTouch could distinguish individual users (or rather, the seats in which they were sitting) to facilitate colocated collaborative work. Wu and Balakrishnan [11] studied the types of gesture possible in such an environent — including multiple fingers, multiple hands, or even multiple users. These tabletop systems were the forerunners for commercial products such as the Microsoft PixelSense (formerly known as the Surface) [1] which allow natural collaborative interactions.

ThinSight [12] is a thin optical touch sensor which can be embedded behind existing LCD panels. It uses infrared sensing to detect fingers and parts of the hand close to the sensor through the display. Touch features are extracted using computer vision techniques. The authors posit the technology could be used to add multitouch to conventional screens. However, in recent years the reduction in cost of capacitive sensors has allowed capacitive screens to be added to commercial laptops, removing the need for this technology.

Resistive Sensing

Resistive touch screens, while very popular in commercial devices, are relatively understudied in the research literature. Of course, these screens have been present on the devices used for studies on PDAs, but the screen technology itself has seen little innovation compared to optical or capacitive sensing.

[1] `htp://www.pixelsense.com`

One of the primary drawbacks of resistive screens is that they can normally only report a single touch point. This is a limitation of the physics of the sensing — multitouch requires the input space to be scanned in a grid, while resistive screens simply apply a voltage to the whole screen to get a single coordinate value for each axis. Multitouch implementations have been demonstrated at technical shows on resistive screens by companies such as Stantum[2]. This technology does not seem to have made it to commercial devices, however, and the implementations are not explained thoroughly. The US Patent Office has also issued a patent for multitouch input on a resistive panel using a combination of finger and stylus input [13].

This patent was issued in 2010, three years after the release of the iPhone popularised capacitive touch technology for multitouch. Resistive screens are now typically found on low-end devices, which may explain the lack of research focus, and the failure of multitouch to reach commercial resistive screens.

Capacitive Sensing

Multitouch on capacitive sensors was also pioneered at the University of Toronto. The Fast Multiple-Touch-Sensitive Input Device (FMTSID) [14, 15] was a tablet capable of sensing an arbitrary number of touch inputs. It reported both touch locations and an approximation of the touch pressure (based on area of contact and compression of the capacitve material) for each finger. Like the Flexible Machine Interface, the FMTSID was an input only device and could not display information, similar to some modern drawing tablets.

Westerman [16] studied the problem of finger tracking and hand pose estimation as part of his PhD thesis. He created a number of algorithms for this which were subsequently commercialised by his company FingerWorks. The company sold touch enabled keyboards and gesture pads which supported a range of multitouch gestures to enhance user productivity. The company was purchased by Apple and their multitouch algorithms incorporated into the original iPhone.

Rekimoto introduced SmartSkin [17], a flexible framework for building interactive surfaces of arbitrary size using a mesh of capactive sensors. This system introduced a number of bimanual gestures for manipulating objects projected onto the mesh, and was capable of estimating the distance of a user's hands from the mesh to a reasonable accuracy.

The development of transparent capacitive multitouch controllers was much slower than the progression of systems using visual touch tracking. It appears that much of the development of such systems took place in industry, rather than in the research community, which focused on optical tracking to a much greater extent. The first product allowing capacitive multitouch

[2] `http://www.stantum.com`

and display on the same screen was the *Lemur*, a music controller released in 2004[3]. After that, capacitive touch displays were rare until the introduction of the iPhone in 2007.

Multitouch Mobile Devices

The world's first smartphone was arguably the IBM Simon, released in 1992. It was certainly the first phone with a touch screen, and had many interface elements that were a precursor to the touch interfaces found on phones today. However, it supported only single touches and was hindered by the processing and network technology of its time, and was discontinued after only a few months.

The growth of smartphones began seriously in 2007 when the Apple iPhone was announced. This introduced multitouch movements such as pinch-to-zoom and natural 'flicking' gestures to a wide audience for the first time. Since then, smartphones and related multitouch devices like tablets and newer laptops have become ubiquitous. The market for these devices is huge — in 2013, it is estimated that around one billion smartphones were shipped, amounting to over half of all mobile phones made that year. There is clearly a need for accurate input on touchscreen devices, but as the next Section will discuss, there are a number of problems which hinder touch accuracy.

2.3 Problems with Touch

Even early in the development of touchscreens, it was known that there were potential sources of error. For example, in an early review of touchscreen technology in 1990, Sears et al. [18] observed that the angle of the screen with respect to the user could introduce bias in the user's touch locations. Since then, a wealth of research has been done exploring the accuracy of touch and the causes of such errors.

2.3.1 Touch Offsets

Studies have shown that users consistently touch below the center of an on-screen target if the screen is tilted away from them [19, 20]. In the literature this is commonly known as a *touch offset* — a mismatch between the user's intended touch location and the recorded touch location from the device. Figure 2.1 illustrates an example of this. For a touch aimed at the crosshair target, centred at position (x, y), the user's touch has been recorded at position (x', y'), shown with a circle. These positions are offset from one another by (Δ_x, Δ_y).

[3] `www.stantum.com`

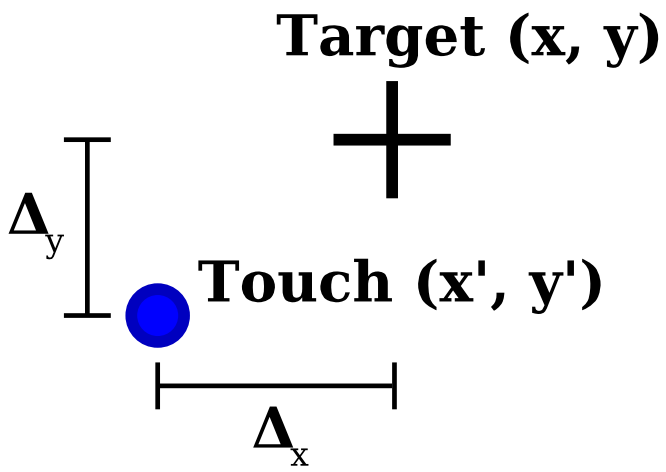


Figure 2.1: An example of a touch offset. The user aimed for the crosshair, but the system recorded their touch at the circle.

Beringer [19] found that the size of the offset increased as the screen was tilted further towards the horizontal. Sears [20] observed an additional effect on the offset based on whether the screen was mounted above, below or level with the user's eyeline. It should be noted that these studies used fixed workstation monitors. On a mobile device, the problem is exacerbated since the orientation of the screen can change rapidly. In particular, if a user operates a smartphone while walking the screen will move around with the impact of their steps and the observed offsets may become large.

Both Beringer and Sears experimented with algorithms to correct for offsets in software. They found that such corrective models increased accuracy and decreased selection time in target acquisition tasks. Moreover, Beringer observed that the observed offsets had a user-specific component — in general, it was better to adapt the reported touch location based on a user's own historical data than to use a model trained on data from multiple users. This makes the offset problem more pronounced, since it is not possible to learn a single corrective model.

Figure 2.2 illustrates another complication. It shows the recorded touch locations for two users when they touched the 'F' key on a smartphone 10 times (this data was taken from Chapter 7). One user (blue squares) has a low mean offset, and all of their touches are close to the key center. The other (green circles) has a more pronounced offset, touching below the key center with some touches straying into the key below. Note however that in both cases, there is a spread in the observed touch locations for a given user. That is, there is a random component to touch offsets in addition to the systematic component described above. This also seems to be user specific in some way — the first user's touches are quite spread out horizontally, but relatively close vertically, while the second user has a more vertically spread distribution. This creates an additional modelling challenge, and may help to explain

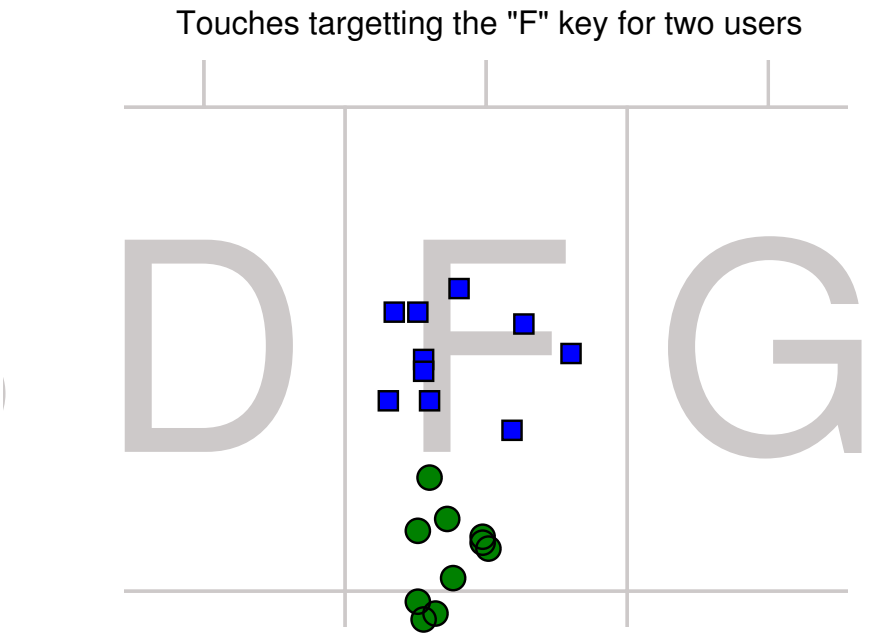


Figure 2.2: Touch locations for two users targetting the ‘F’ key 10 times on a soft keyboard.

why the probabilistic algorithms applied to touch in this thesis are successful; touch offsets follow a distribution rather than being an analytic function of the target location.

2.3.2 Baseline Accuracy for Touch

A number of studies have been done exploring the accuracy of touch. Such research typically aims to find a minimum target size which users are able to acquire with a given accuracy (95% is a commonly used goal). However, the results and recommendations found in the literature are quite varied, based on the conditions tested and the hardware used.

Desktop Touchscreens

On a large touch enabled monitor, Sears and Shneiderman found that 99% accuracy could be achieved for square targets of 8 mm on a side. This relied on the lift-off selection strategy, where the recorded touch point is taken where the finger leaves the screen, not where it first makes contact. On the same hardware, using the land-on selection strategy meant that the same accuracy could only be obtained with buttons of size 23 mm, nearly 3 times larger [21]. This suggests two phases to target acquisition on such screens: an inaccurate initial approach to the screen followed by a more controlled refinement of the finger location to make a selection. This prompted the authors to propose that touch screen targetting behaviour should be modelled by two applications of Fitts’ Law [22], one for each phase.

For touch screen keyboards using similar large desktop monitors, Sears et al. [23] showed that low error rates could be achieved for 8 mm keys but that typing speed rose steadily as the size of the keys increased to 22.7 mm.

Mobile Devices - Stylus Input

On PDAs, the results were somewhat different. Using a stylus for input, studies have suggested a minimum target size for soft buttons as low as 2.5 mm [24] and as high as 6 mm [25]. These numbers are quite low in comparison to the results above, suggesting stylus input is more accurate than finger touch. This is not surprising, since the contact area of a stylus was always smaller than the smallest targets used in these studies. Both of these studies were based on single selections, and found that targeting speed increased and errors decreased as button size increased.

However, when investigating stylus typing on PDAs, Mackenzie and Zhang found no effect on text entry speed between 6-10 mm buttons [26]. This result was confirmed by Sears and Zha [23] for buttons between 2 and 4 mm. This is unlike the results mentioned above for typing on a desktop monitor, where speed rose with larger buttons. However, this can be explained in terms of Fitts' Law [22], which estimates the motion time (MT) to acquire a target in terms of the distance D to the target and the width W of the target as:

$$MT = a + b \log_2 \left(1 + \frac{D}{W}\right),$$

where a and b are constants. The term $\log_2 \left(1 + \frac{D}{W}\right)$ is called the index of difficulty (ID). The ID captures the intuition that targets are harder to hit when further away, but easier to hit when bigger. The text entry studies scaled their keyboards uniformly, so that the interkey distance and key width changed in proportion to each other and thus the ID was the same for each condition. It is hence not surprising that the observed entry rate was the same across conditions. In the target selection studies, the target sizes changed but the distances between targets did not, so that the ID for each condition was different: thus the observed result that smaller targets were more difficult to hit.

Mobile Devices - Finger Input

Parhi et al. [27] investigated one-thumb usage on a PDA for both single target (checkboxes or confirmation buttons, for example) and multiple target (a numeric keypad) selection. They recommended a minimum size of 9.2 mm for single target tasks and 9.6 mm for multiple target tasks. This difference is rather small, and while their results showed statistical significance, it seems somewhat unlikely that such a small variation in target size would lead

to measurable performance differences in general usage. The authors also noted an effect of button size on entry rate for the multiple target task, even though the Fitts' Law index of difficulty was the same across all conditions. They explain this anecdotally by noting that for conditions where the buttons are smaller than the pad of the thumb, users took extra care in their selection process. This was not an issue in the stylus text entry experiments, since the stylus was always smaller than the keys.

Other studies have suggested a range of minimum values for reliably acquiring targets. Roudaut et al. [28] showed that users could reliably acquire 3 mm targets with their thumbs while holding a phone in one hand, but only by using a zooming technique which made the effective target size 9 mm. Vogel and Baudisch [29] used a related zooming method and their participants were able to acquire 2.6 mm targets with 90% accuracy. Using only the fingers and no targeting aid, the same accuracy was achieved only for 10 mm targets. Arguably only the last of these results is relevant, since the introduction of zooming means that those tasks are no longer direct pointing tasks. Rather, target selection becomes a two stage process where the zoom must be activated first, followed by pointing in the magnified space.

Holz and Baudisch [30] suggest that using regular finger touch methods, 95% accuracy is achievable for 8.6 mm targets. Using optical tracking, they suggest that the smallest target size which users can theoretically acquire is around 2 mm, although their offset techniques do not reach this limit.

In general, it seems that without corrective methods, users can only reliably acquire targets of sizes between 9 and 10 mm using finger touch on current devices. Any technique which allows acquisition of targets smaller than this minimum offers a useful improvement over commercially available options.

2.3.3 Other Factors Affecting Touch Offsets

In addition to target size and the orientation of the screen with respect to the user's eyes, a number of other factors have been shown to affect touch offsets.

For fixed desktop touchscreens, Beringer and Peterson [19] showed that the horizontal offset changed depending on which hand was used to touch targets, which they thought might be a result of either visual parallax or the angle of the hand with respect to the screen.

They also found differences in the observed horizontal offsets between a study where users had to select one from a grid of nine targets, and one where only a single target was shown at a time. This is an early indication the touch task can have an effect on the offsets. Subsequent studies have typically not considered this, focusing on the offsets in a specific task.

The way the user holds the device and performs touches has a noticeable effect. Wang and Ren [31] showed that users have different accuracies depending on which finger they use,

and that the angle of the finger with respect to the screen also caused accuracy changes. They found that the thumb and little finger were less accurate for target selection than the other three fingers.

Azenkot and Zhai [32] looked at text entry accuracy as a function of hand posture. They found that touch behaviour was significantly different between one-thumb, two-thumb and index finger usage. They used this to give a set of design recommendations for offsets in different regions of the screen for the different hand postures.

Goel et al. [33] built *WalkType*, a system designed to improve text entry for walking users. This was motivated by the observation that baseline error rates were higher when users walked and typed. This suggests mobility is also a factor affecting touch, which is reasonable to expect given the division of user attention and noisy device movements associated with walking.

All of these factors can contribute to the size and direction of touch offsets. A single model for increasing touch accuracy is thus extremely difficult to create. A more realistic goal is a set of models for each of the different scenarios together with a framework for detecting the usage context. Even this is a difficult proposition, since the range of possible contexts is potentially infinite. The space of necessary offset models rapidly becomes very large, and this issue has not yet been addressed by researchers.

Another alternative could be to develop a system which continuously updates an offset model based on the most recent touches observed. However, this would by nature be quite slow to adapt when a user changes input technique, say from two-thumb input to one-thumb. From a Bayesian standpoint, a preferable approach might be to run several offset models at once, and keep track of their running average accuracy over the past few touches. When a given model has a consistently lower error over a certain number of touches, its corrections can actually be applied to the user's input. Of course, in practice this is likely an unreasonable computational demand. It also requires the selection of some subset of the possible offset models as 'archetypal' in some sense of the touches we expect to see, a problem which is in itself difficult.

2.3.4 Causes of Touch Offsets

As described above, touch offsets can vary based on a range of factors. However, there are a number root causes which can explain much of this variation. First, and perhaps most well known, is the so-called 'fat finger problem' [34]. This is the issue that the area of a user's finger in contact with a touchscreen is often greater than the size of the targets they are trying to touch. Thus, when trying to select from a dense grid of targets, it is easy to inadvertently touch the wrong one. This problem is notable in text entry, where soft keyboards feature

many small, closely arranged targets. For example, the keys on the portrait keyboard of an iPhone 5 are around 6 mm by 4 mm, but the human thumb is typically much larger than this. The 50th percentile of thumb width is 23 mm for males and 21 mm for females [35]. Even if only a third of the thumb is in contact with a touch screen, the touch area is already bigger than the keys. Users have some ability to control this, since by touching at a sharper angle or with reduced pressure they can decrease the size of the contact area, but in general the fat finger problem is a significant source of ambiguity.

The fat finger problem may also be thought of as a mapping issue. The system ultimately interprets each touch as a single (x, y) coordinate pair, but the finger makes a contact area which is much bigger than a single pixel. The choice of the algorithm that maps this contact area to a single point will clearly have an effect on touch accuracy, particularly if the output of the algorithm does not align with the user's idea of where they are touching.

A related issue of touch is the occlusion problem [36]. This refers to the fact that as a touch actually happens, the finger is often occluding content on the screen, making it more difficult for users to know where they are touching. This lack of feedback to the user in where they have touched creates an additional source of error. This is closely tied to the fat finger problem, but also encompasses some slightly more general effects. For example, if stretching across a soft keyboard with the thumb, the parts of the hand not in contact with the screen can also occlude the other keys from view. If the next key to be typed is occluded in this way, the movement of the thumb to that key might be less accurate than if it could be seen clearly.

Sensing issues can also contribute to the inaccuracy of touch interaction. Typical capacitive sensors take the values obtained from an array of sensor pads and interpolate the maxima of these values to obtain a set of (x, y) coordinates. However, a sensor response can be induced by parts of the finger close to but not touching the array [37]. Thus, the interpolated position corresponds to the centroid of a region which may not match up to the physical contact area, introducing another small offset.

Further, capacitive sensors are subject to drift and bias based on their environment — changes in temperature and ambient electrical activity can affect the observed capacitances [38]. Since the implementations are typically closed, it is difficult to know how sensor manufacturers take this into account in their coordinate algorithms. Even if some compensation is made, the range of potential usage localities of mobile touchscreen devices precludes the creation of a universally applicable algorithm. Additionally, the compensations themselves can introduce new problems. For example, the iPhone applies a small corrective vertical offset to all touches. If two users on opposite sides of a table try to interact with the phone, the user looking at the phone upside down will find it difficult to touch accurately because this offset is now towards them, rather than away as the interface designers intended. These

effects create additional offset in the reported touch coordinates from devices.

Holz and Baudisch [39] approached the problem from a different direction. Much of the early work on touch offsets assumed that the fatness of the finger placed a hard limit on the achievable accuracy for touchscreens. This work reframed the problem as a *generalised perceived input point problem*, suggesting that even for a fixed screen, touch offsets are not just a function of the target location but can be affected by factors such as the 3D angle of the finger with respect to the screen. They hypothesise that by modelling some of these external factors, touch accuracy can be improved below the 'fat finger limit'.

By having participants repeatedly acquire a target on a capacitive touch pad, they showed that all three of finger yaw, pitch and roll had a significant effect on the observed touch offset. Using this, the authors built a system which could obtain these angles by scanning a user's fingerprints. They were able to compute intended touch locations to a level which allowed accurate selection of targets smaller than the supposed limit for finger touch — around 7.9 mm for 95% accuracy, an size reduction of about 20% over the state of the art at that time.

The authors' later work further refined their model of touch [30]. Here, they considered users' mental models of the touch process as an explanation for the offset problem. They found evidence that users target using visual features on the *dorsal* side of their fingers — for example, aligning the center of their fingernail with the intended target. This makes sense, since the dorsal side is all that is visible to them while touching. However, in practice the contact area on the palmar side of the finger does not align with these visual features. Since capacitive sensing is only able to detect the palmar side, offsets are introduced. Essentially, the authors pose the problems of touch as perceptual mismatch between sensor and human. This is in line with the early work of Sears [20], who explained the vertical offset he found on tilted touchscreens as a parallax effect of the viewing angle.

Using a visual tracking system, they tested accuracy when assuming a range of features were used to acquire targets. The most successful model was the so-called projected center model. This assumes that users target by aligning the horizontal center of their finger and the vertical center of their fingernail with the target in order to touch accurately. Under this model, 95% touch accuracy was obtained for targets of width 4.9 mm, which is around half the size of minima reported in other work. Of course, the drawback of this approach is that it relies on camera tracking of the finger, which is not possible on current mobile devices in everyday usage environments. Despite this, the conceptual model presented in this work is very useful in understanding touch.

Recently, work has been done to better understand touch in terms of human motor control, in the vein of Fitts' Law [22]. Bi et al. [40] proposed a dual-Gaussian-distribution hypothesis to explain touch targeting behaviour. This says that touch targeting is best modeled by two in-

dependent Gaussians. One models the basic speed-accuracy tradeoff inherent in the human motor system, and the other the absolute precision of touch independent of this tradeoff. From this, they derive Finger-Fitts (or FFitts) Law, an extension of Fitts' Law formulation specifically for touch. This was shown in three experiments to better account for the observed variance in touch points than the original Fitts' formulation, as well as providing better predictive power for user performance in a given touch system. Of course, using a mixture of distributions is likely to fit better than a single one from a purely mathematical perspective, but the approach here is in line with earlier findings by Sears and Schneiderman[21] that target acquisition in touch is a two stage process: the finger is first moved to the approximate location, then fine selection with the fingertip is performed.

Grossman and Balakrishnan [41] give an alternative probabilistic formulation of pointing behaviour. They note asymmetric effects in the spread of touches in the direction the finger is moving as compared to the spread in the perpendicular direction. This affects the probability of a touch aiming at an interface element depending on the direction from which the finger approaches. This is an indicator that previous touch location may have a bearing on touch accuracy, and should potentially be considered in corrective models.

2.4 Approaches to Improve Touch Accuracy

The offset problem is well understood in the HCI community, and as touch technology has grown more popular there has been considerable research effort directed at improving the accuracy of touch input. Broadly speaking, this research can be divided into three overall approaches:

- models that explicitly learn a function mapping between recorded and intended touch locations;
- models that provide some targeting aid or parametric model to improve accuracy, but do not explicitly model offsets;
- interface-specific models, which rely on knowledge of the task and/or interface layout to increase accuracy.

There is some overlap between these broad categories. For example, the work in Chapter 5 of this thesis combines both an offset model and task specific knowledge to increase text entry accuracy. However, these categories provide a good framework for discussing the relevant work.

2.4.1 Offset Models

As described above, Beringer [19] was the first to build an explicit offset model for a desktop touchscreen. His simple model computed the mean x- and y-offsets from a training block of 108 touches, and applied this to subsequently collected data. This reduced the mean x-offset from 2 mm to 1 mm and the mean y-offset from 1.6 mm to less than 0.3 mm. It was noted that computing the offset from data from the same user reduced the error more than using data pooled from a population of users. This is an early indication of the user specificity of offsets.

Potter et al. [42] employed a small corrective offset such that the reported touch location was slightly above the finger. The same offset was applied to all users in their study, and details of how the size of the offset was chosen are not given. Their primary interest was in comparing the land-on and lift-off selection strategies, but they still reported that this offset gave an increase in selection accuracy when compared to having the touch point be right under the finger.

Park et al. [43] studied the optimal placement of soft buttons for one-handed use on a mobile phone. They divided the screen into 25 rectangular regions of size 13.9×10.4 mm and rated these according to number of errors and ease of reach by the thumb. As a preliminary study, they computed the mean offset for touches targeting two of the regions with the highest error rates, and evaluated a simple offset model that applied these mean offsets to any touches falling in those regions. This was shown to increase selection accuracy in those regions by up to 20%, suggesting that even basic offset models can provide a substantial benefit to the user.

In their attempt to explain the touch offset problem, Holz and Baudisch produced two corrective models. The first was Ridgepad [39], a high precision touch device that scanned a user's fingerprints to determine the angle of their finger with respect to the screen. They applied a corrective offset based on these angles which considerably reduced touch errors. They also employed optical tracking using an overhead camera [30] to extract visual features of the finger to aid targeting. The best model for this system assumed the intended touch point was the projection of the center of the fingernail onto the touch pad. While both of these approaches significantly reduced error when compared to using the centroid of the touch contact as the intended point, they require non-standard hardware that cannot be readily used in a mobile setting. This is a notable drawback given that the majority of touchscreen devices are phones or tablets.

Roudaut et al. [44] generalised the study of offsets from planar to curved surfaces. They found a significant effect of curvature on both the size and variability of offsets. Placing a target on convex curvature (protrusions from a surface) tended to decrease both the mean offset length and the spread in observed touch locations. On concave surfaces (depressions in

a surface), the mean offset length was increased but the spread was reduced. There was also a relationship between the level of curvature and the relative size of the observed increases and decreases. Spherical displays have been studied in the literature (e.g. [45]) and are being commercialised by companies such as PufferFish[4]. Thus a good understanding to how offsets generalise to arbitrary surfaces is important.

Henze et al. [1] conducted a large scale in-the-wild study of touch behaviour on smartphones. They deployed a game to the Android market in which players had to hit a series of circular targets as quickly as they could. This allowed the authors to collect a large amount of data about users' offsets when hitting targets in different areas of the screen and on many different devices. In total, they logged over 120,000,000 touch event from more than 90,000 installations of the game. Using this data, they learned two offset models. In the first, they divided the screen into a 30×30 grid and learned the mean offset to apply to each grid cell. In the second, they learned continuous fifth order polynomial functions which mapped the touch coordinates to offset values. Both models were shown to significantly reduce the number of missed targets in a follow on evaluation. These functions were learned for all devices of a given screen resolution, rather than on a user-specific basis, which potentially means the models learned were not optimal.

The authors later carried out a second study on a similar scale, this time studying text entry on soft keyboards [46]. They released a second Android game, where players had to type words as fast as possible in order to gain points. They gathered 47,000,000 keystrokes from over 70,000 installations. From this they obtained per-key distributions of touch locations for each device in the dataset, and learned corrective functions for the systematic offsets. They tested these functions in a follow up study, and showed that they led to a significant decrease in text entry errors over the default Android keyboard. Again, these results were not user-specific, creating potential for further improvement of the models.

Buschek et al. [47] studied the way in which offsets vary across devices. They had users complete a crosshair targeting task on several different smartphones and learned linear regression models for the offsets on each user-device combination. They then demonstrated that it was possible to learn transfer functions which adapt a user's offset model from one device to another, or adapt a user's model to suit a different user on the same device. This is useful since it allows a user to change to a new phone and still enjoy the increased touch accuracy afforded by an offset model without providing a new set of training data. These adapted models allowed for more accurate target selection than the device baseline in all cases. An interesting finding of this work was that adapting based on other users of a given device was more effective than adapting an individual's model from another device. This suggests that there is a bigger difference in offsets across devices than between users on the same device.

[4] `http://www.pufferfishdisplays.co.uk`

However, models tailored to a specific user on a specific device always outperformed pooled data models for the device, in line with the findings of previous research.

Discussion

A range of techniques to model offsets have been studied: subtraction of the global offset mean; applying the same offset to any touch in a given region of the screen; continuous polynomial predictive functions; and corrections based on the angle of the finger with respect to the screen. Existing research has varied in whether offsets are applied at a device- or user-specific level, despite evidence that the latter is the superior choice in terms of predictive accuracy.

The methods described above all make the decision to model offsets as a purely systematic effects. Even Roudaut et al. [44], who acknowledge that the observed touches for a given target have a measurable spread, do not account for this random effect in their corrective model. We believe this is a failing of the existing techniques — what if a touch falls right on the boundary of two targets? In such a situation a deterministic offset model will assign the touch with certainty to one of those targets. If instead the uncertainty in the touch location was modelled, we would be able to say something about the probability that a touch was intended for each of the targets, potentially allowing the system to make a more 'intelligent' response based on other available contextual information. This motivates the probablistic approach taken to touch in this thesis.

2.4.2 Other Approaches

There are many other approaches that have been shown to increase touch accuracy without explicitly modelling offsets.

Targeting Aids

A number of techniques exist based around the idea of removing or reducing the effect of finger occlusion. The most basic of these is the lift-off selection strategy discussed previously and studied in e.g. [42, 20]. In this, a cursor is shown when the user touches down but selection is deferred until the finger leaves the screen, allowing fine positioning of the touch target.

An extension of this is the work of Albinsson and Zhai [48]. This paper introduces a family of interface widgets that allow pixel level selection of targets on touchscreens. Examples include CrossKeys, where an initial touch creates a set of crosshairs which can be adjusted with a set of directional arrows to finely position the intended target; and *2DLever*, which places

an onscreen lever near the original touch, and coarse movements of one end are translated into fine movements at the other, allowing accurate selection. These tools were shown to be extremely accurate, but introduce a considerable time burden and take up a large amount of screen space.

Zooming is another popular targeting aid. For example, *TapTap* [28] works by introducing a two stage selection process for touches on a mobile phone. The first touch defines an area of interest, which is then shown zoomed in a cutout window. A second touch within this window allows fine selection of targets within the zoomed area. While occlusion is still possible under this model, in general the zooming was sufficient to select small targets. However, the drawback is that it doubles the number of touches required, and is thus not suited for applications such as text entry.

Another zooming technique is described by Olwal and Feiner [49] which involves rubbing a region of the screen to create a zoomed fisheye view. Targets can then be selected in the zoomed region. However, the authors did not carry out an evaluation of this technique. It is likely that zooming again makes target selection simpler, but if the targets are densely packed (such as on a map) then occlusion can still occur.

Another common idea is to place the selection cursor at an offset to the finger location, so that the selection process is not occluded. *OffsetCursor* [42] was the first system to do this. It was shown that using OffsetCursor reduced errors when compared to standard touch, but introduced a time penalty to selection. These findings were confirmed by Ren and Moriya [50] for stylus-based touch devices.

One of the failings of OffsetCursor is that original implementations used a fixed offset for all touches, which made some areas near the edge of the device untargetable. Later work, such as that of Esenther and Ryall [51] or Benko et al. [52], used multiple finger pointing to control the size and direction of the cursor offset, preventing these issues. Their techniques were also shown to allow more accurate pointing than direct touch, but again they made selection slower. The additional cognitive load of moving a cursor offset to the finger slows down the interaction.

An issue of using a cursor offset is that it is unnecessary for large targets. However, all early implementations of the technique had the offset active for all touches. *Shift* [29] corrects for this by combining elements of zooming and offset cursors. For touches where the user needs no assistance, the system functions like a regular touch screen. For fine target selection, users press and hold on the screen, causing a zoomed view of the area occluded by the finger to be shown in an unoccluded cutout window. Moving the finger controls a cursor in this offset window to allow selection of the occluded targets. This technique performed similarly to OffsetCursor in terms of accuracy for all targets, and allowed faster selection of large targets. Further, it was subjectively preferred by users in the study. Both techniques allowed

accurate selection of 6 pixel targets using fingers on devices designed for use with a stylus.

Similar techniques are also employed on commercial devices. Most smartphone keyboards, for example, display a shifted image of the currently pressed key when the finger is down to help prevent occlusion issues.

One important note with regard to these targetting aids as opposed to direct touch input is that they change the fundamental task by introducing a two stage interaction. Users must first make the cognitive step to activate the targetting aid, then perform selection in the aided space. This makes these techniques difficult to compare directly in terms of selection accuracy, since the activation step incurs different time penalties for each technique. Perhaps information throughput rate might be a better metric to compare these different techniques in future research.

Back of Device Interaction

A somewhat different approach to the occlusion problem is to move touch interaction to the back of the device, so that the screen is not occluded at all. Baudisch and Chu [53] have argued that this is particularly useful for devices with extremely small screens, where a finger can occlude much of the available display space. In a study using small devices with screens from 2.4 inches to less than 1 inch, they showed that back of device pointing allowed accurate target selection regardless of screen size, but front touch (coupled with the *Shift* technique described above) failed for screens smaller than 1 inch. This is important as commercial availability of small touch enabled devices like smartwatches increases.

Back of device input has been shown to be both possible and usable on existing hardware. *BlindSight* [54] placed a touch sensitive keypad on the back of a mobile phone so that users could interact eyes-free while making calls. The authors showed that input accuracy reached useful levels, and that study participants preferred BlindSight to a visual baseline condition. *HybridTouch* [55] allowed users to scroll content on a PDA using back of device drag gestures, and these were also shown to be popular with participants.

While gestures and pointing on small devices are good, for larger devices back of device input is quite inaccurate. Wigdor et al. [56] implemented 'under the table' touch on a tabletop display, and found that 4.5 cm targets were necessary for 95% accuracy. This is considerably larger than the minimum target size reported for other touch systems.

Of course, one of the limiting factors for back of device interaction is the lack of visual feedback — users are often unsure where there fingers are touching. To overcome this limitation, a number of transparent or pseudo-transparent systems have been developed. *LucidTouch* [36] uses a rear mounted video camera to keep track of the user's fingers behind a tablet PC. An transparent overlay on the screen shows where the user is touching, while still showing

the regular on-screen interface. Users reacted positively to this technology, but interestingly only found the overlaid visualisation useful for certain tasks. The authors do not comment on how fast or accurate users were with this system, reporting only subjective preferences for the different conditions in the study. This makes it difficult to assess how useful this technology was for solving the occlusion problem which back of device interaction aims to solve.

Ohtani et al. [57] looked at the accuracy and selection time on a visually transparent display allowing both front and back of device touch. They found that users were faster at selecting targets when using only the front of the display, but more accurate when able to use both front and back. There was so significant difference in accuracy between front-only and back-only touch.

There has also been work on using back of device information to improve the user experience for conventional touch. Mohd Noor et al. [58] used the position of the hand at the back of the device to predict where and when a touch on the front would occur. These predictions were accurate to a distance of around $2\,\text{cm}$ up to $200\,\text{ms}$ before the user touched down. While not inherently more accurate than other touch models, this approach allows interesting interface improvements. For example, if we can predict which icon a user will touch, content for the corresponding application or function can be preloaded to create the illusion of reduced loading times and a smoother overall user experience.

Schoenleben and Oulasvirta [59] looked at typing using a touch keyboard on the back of a specially modified tablet. Their system was designed for ten-finger touch typists, keeping the standard mapping of fingers to keys on the back-of-device keyboard to allow users to transfer their expertise. After 8 hours of training, participants could type at up to 46.2 words per minute. Buschek et al. [60] extended this model by adding a machine learning algorithm for moving the key locations based on the observed typing dynamics of individual users. Their model included considerations of hand anatomy when weighting potential key movements, and was shown to provide performance improvements of up to 40% over the basic keyboard.

Particle Filtering

There has also been work on estimating the location (and other features) of a finger near a touch sensor using techniques from machine learning. In particular, a family of algorithms known as Sequential Monte Carlo techniques [61] or particle filters have been used. These algorithms treat the features of the finger — position, angle, width and so on — as random variables following some distribution. The algorithms approximate this distribution over time using a set of samples, or particles. The particles at a given time step are generated by resampling particles from the previous time with probability based on how well they match

the data observed at the sensors. An advantage of this is that estimates of the uncertainty in each parameter can be obtained, indicating how certain the system should be about an input.

FingerCloud [62] used a particle filter to track the (x, y, z) position of a finger. Traditional touch screens only report input when the finger is in contact with the screen, but adding the third dimension allows interesting new applications. In the paper the authors describe a single-finger map browsing application where the height of the finger controls the level of zoom. On existing phones this is typically controlled by pinch gestures, which are difficult to perform one handed.

In later work, the authors introduced *AnglePose* [37], which added a number of other features to the model of the finger. In particular, the 3D *pose* of the finger, consisting of position, width and yaw, pitch and roll angles, is tracked. A user study showed that explicitly modelling these features allowed 95% selection accuracy of circular buttons only $6\,\text{mm}$ in diameter. An interpolation of the sensor values, as conventionally used in capacitive touch, obtained the same accuracy only for $10\,\text{mm}$ buttons. The particle filter model was further shown to be more accurate than an iPhone for selecting densely packed targets. One explanation for the observed increase in accuracy is that modelling the parts of the finger not in contact with the screen helps to reduce the 'shadowing' effect on the sensors described in Section 2.3.4.

One drawback of the particle filtering approach is the relative computational intensity. A number of samples from the distribution (often upwards of 100) are maintained, and the parameter values from these are averaged to produce inputs. The resampling and weighting process for these particles is quite demanding. Additionally, adding multitouch to the model involves adding a complete set of parameter values for each finger which needs to be tracked, further increasing the load. Thus, while the technique is accurate and requires no new interaction metaphor over traditional touch, it requires relatively powerful hardware to run well.

The main difference between particle filter models and the techniques presented later in this thesis is that the former requires the specification of a fixed set of parameters of the finger, while our techniques assume only that a smooth mapping between our inputs and the touch offsets exists. A fixed finger model has both advantages — it becomes easier to generate simulated test data — and disadvantages — the assumptions made may not hold in practice. Both techniques are capable of providing uncertainty estimates, which we consider to be an advantage for reasons described previously.

2.4.3 Interface Specific Models

The third category of approaches to improving the accuracy of touch input is those models which are specific in some way to a particular interface or task. The models discussed so far have been evaluated for selecting arbitrary targets, without knowledge of the semantic meaning of those tasks. However, in HCI there is often a great deal of prior knowledge about a given interaction. For example, in text entry we have information about the relative likelihood of hitting each letter in the alphabet — 't' is much more common than 'z'. This could be further conditioned on what the user has already typed. This Section looks at models that leverage task-specific knowledge in order to allow more accurate input.

Using Target Location

The most basic form of interface-specific knowledge which can be exploited is the location of interactive elements on the screen. If we assume that each touch on a device is intended to accomplish some goal (this is not always the case, but distinguishing accidental from intentional input is a hard problem), then it can be assumed that the touch was targeting some button or other interface element on the screen, rather than empty space. Since the interface for a given task is specified by the designer in advance, we can use use knowledge of this layout to help the user touch.

An early implementation of this idea is the *BubbleCursor* [41], a variation on the lift-off selection strategy where the cursor is a circle with variable area. The cursor expands to encompass nearby targets, and gives clear visual feedback about what is selected. While originally presented for use with a mouse, it has since been adapted to function on touch screens (e.g. by Go and Endo [63]). This cursor prevents the user from failing to select if their touch should fall in a small gap between buttons.

A similar idea is the Bayesian Touch framework of Bi and Zhai [64]. This builds on the same authors' Finger-Fitts Law [40], which was discussed earlier. Essentially this formulates touch as an uncertain, ambiguous process. Rather than attempting to map finger touches to one location, their model aims to calculate the probability that a given touch was meant for each of the on-screen buttons or widgets. The most probable interface element is then selected, even if the original touch did not fall within its visual boundary (VB). Performing selection in this manner reduced error rates when compared to VB- or nearest-neighbour-based selection.

Text Entry

Text entry is perhaps the most ubiquitous task on mobile devices, whether it be for email, SMS, web browsing, or document creation. In addition to knowledge of the layout of the buttons on the keyboard, we have a great deal of knowledge of the sorts of things that users type. Extensive statistical analyses of text corpora have given us information about the relative frequencies of letters and words. This information can be exploited when interpreting the intented target of a touch.

Many models for text entry rely on using statistical language models to assign probabilities to letter sequences. Goodman et al. [65] were the first to work with a language model and a soft keyboard in conjunction. They learned Gaussian distributions for likely key press locations from historical typing data, and used this to together with the language model probabilities to search over interpretations of a sequence of touches. This was shown to reduce errors by a factor of 1.67 over the uncorrected input.

Kristensson and Zhai [66, 67] combined a language model with a shorthand writing system in SHARK^2. Users enter text by drawing 'sokgraphs', a gesture consisting of trajectories that pass through all the letters in a word in order. The system matches words by searching the lexicon of the language model for words whose letters contain the letters in the gesture. This was shown to be usable when searching over vocabularies of up to 57,000 words.

The authors later work improved the pattern matching of the keyboard [68]. Using geometric pattern matching, they were able to identify the intended words for gestures even in cases where the gesture missed all the letters. This technology was designed with stylus writing in mind, but has since been commercialised in all major smartphone operating systems, initially as the ShapeWriter keyboard.

Research has also looked at personalising soft keyboards based on a user's touch history. Himberg et al. [69] looked at moving the centroids of the keys on a soft numeric keypad based on how users targeted those keys. Go and Endo [63] extended this technique to a full keyboard in *CATKey*. Both systems displayed visual feedback about how the shapes and locations of the keys changed over time, but neither study found a significant effect of the adapatation on accuracy or entry rate. However, users displayed subjective preferences for the adapted keyboards even though they provided no quantitative benefit.

Findlater and Wobbrock [70] took a similar approach, using a classification approach to assign touches to keys based on a user's history of previous touches. They compared adaptation with and without visual feedback, and found that moving the key centroids but not changing the visual appearance of the keyboard led to a significant improvement in text entry rate. Visual feedback, on the other hand, had either no effect or in some cases actually decreased performance. While users may prefer the visual appearance of the updating keyboard layout,

it appears that this actually hinders them as they continually try to change their behaviour based on new layouts.

Some models have also had success by adapting the size of key targets based on the next-letter probabilities from a language model. In *BigKey* [71], this adaptation was done visually. For example, if the characters in the current word were 'th', the system would make the vowels and 'r' keys bigger as they are more likely to be the next characters typed. This improved both speed and accuracy over a regular soft keyboard.

Gunawardana et al. [72] performed the same type of key target resizing but hid the visual effects from the user. However, they noted that this process can sometimes prevent users from hitting a key entirely if a neighbouring key is much more likely under the language model. Thus, they added a feature which defines a small 'anchor' region at the geometric center of each key, such that touches on the anchor will always trigger that key regardless of overlap from resize neighbours. Through simulation, they showed that this approach was superior to key target resizing alone.

While these resizing algorithms were based on touch data from many users, Rudchenko et al. [73] showed that additional performance benefits were possible when training on user-specific data. The personalised models gave a relative improvement of 21.4% over the pooled data resizing algorithm. This matches the results found for other touch models described earlier. The paper also introduces a game called Text Text Revolution which can generate the data needed to train the resizing algorithm. The authors point to games as a good source of training data for touch models, as playing them is not a burden on the user as an explicit calibration process might be. This is a similar argument to that given by Henze et al. [1, 46] in their large scale data gathering through Android games.

Other sensor information can also be leveraged to improve text entry. *WalkType* [33] combines touch information with data from the accelerometers in a smartphone. It uses a classification based approach trained on typing data from 16 users to identify the intended keys for users typing while walking. This was shown to significantly reduce uncorrected errors and increase typing speed for walking participants. This paper provides evidence that the variance in a user's touch distributions is sensitive to the context of use — walking introduces additional noise.

2.4.4 Limitations of Existing Models

The models described above have all been successful in some respect at improving the touch experience. There are a number of common themes in terms of the drawbacks and limitations of these technologies. For example, many rely on custom hardware such as cameras or back

of device sensing, which makes it unlikely that they can be deployed to existing commercial devices.

Additionally, many systems are built using historical touch data gathered from many users. However, numerous studies have shown that touch behaviour is highly user specific, and that these pooled data models tend to perform worse than equivalent models trained on a user's own touches. There is a need for greater attention to this in touch models.

Another common issue is the use of parametric models, such as the polynomials used to describe offsets by Henze [1] or the 'hinged rectangle' finger model used by Rogers et al. [37]. These encode implicit assumptions about touch behaviour which may not hold in practice. Machine learning offers a number of flexible, non-parametric approaches which make no such assumptions, but these have thus far gone largely unused by the HCI community. One of the primary contributions of this thesis is an application of such techniques to touch.

Many approaches to touch also ignore the uncertainty and variability of touch. Offset models typically assume that there is a fixed offset vector given a single target location, but this is not the case. The random component of touch offsets is typically neglected, which can lead to bad decisions about what to do with a given touch. For example, consider a touch falling between two targets. An uncertain model might assign a probability of 55% that the touch was intended for the first target and 45% that it was meant for the second. If a certain model always assigns to the first target (say), information is lost. In typing, having information about these probabilities could be used to update the assignment of a touch to a key based on previous or subsequent touches.

Finally, task- or interface-specificity can be looked at as a limitation. While encoding knowledge about the types of touch the system can expect allows for higher input accuracy, it also introduces the notion that each new interface needs a new touch model. This leads to a potential explosion in the amount of training data which must be collected. It would be better to have a single model which described touch in a general sense, or — perhaps more reasonably — a small number of models which cover the most common tasks and require only small amounts of training data.

2.5 Autonomy Handover

Besides the need for accurate methods for touch input, the other primary motivator of the work in this thesis is the existing work on *autonomy handover*. This is the idea that in many interaction tasks, the system will attempt to automate its behaviour in some way to make the user experience better. However, there may exist situations where this is not what the user wants — if this is the case, then it is advantageous to have a model where a user can take fine control in certain situations and prevent automated behaviour from the system.

As a concrete example, consider text entry. Modern smartphones have sophisticated autocorrect algorithms which, in the majority of situations, correct for mistakes the user might make. However, sometimes the user might want to enter a word unknown to autocorrect — such as proper nouns, words from other languages, or slang — without having their input changed. For example, the Scottish word 'scunner' is autocorrected to 'scanner' on all major smartphone operating systems. In a system that allows autonomy handover, the user should be able to take control and prevent such unwanted autocorrections from happening. Chapter 6 of this thesis details the creation and evaluation of a text entry system with this property.

This smooth negotiation of control between user and system has been described as the 'h-metaphor' [74], referring to the analogy of the interplay between a rider and horse. If a rider makes certain, frequent and deliberate movements, the horse follows instructions exactly. If the rider's control is vaguer, the horse will default to learned behaviour patterns and will take over more control from the rider. The notion of tightening and loosening the reins maps well to the interaction model described above.

In his doctoral work, Williamson [75] formulated a theoretical framework for interface design which frames interaction as a continuous control process. Under this model the system is constantly trying to infer a distribution over potential user goals, and take action accordingly. The idea of autonomy handover is important to this work, as is the representation and propagation of input uncertainty through the system. Touch is a natural mechanism for such a representation — we have seen that touch has a natural uncertainty, but that it is typically ignored when performing input. By modelling this uncertainty, we can potentially obtain new channels for interactions.

One existing application of this metaphor in a capacitive touch system is FingerCloud [62], described earlier in the context of a particle filtering system. This system uses the height of the finger above the sensor (which can be measured up to around 1 cm) as a proxy for certainty in a map-browsing application. The browser has a built-in set of interesting locations, and when the finger is held high above the screen it tends to gravitate the view towards these locations. This corresponds to a situation where the user is looking at the map near them but does not have a destination in mind — they want the system to show them interesting places. However, when they have a goal in mind they can touch close to the sensor, indicating certainty and preventing unwanted movements of the map.

Schwarz et al. [76] developed a framework for handling interactions with uncertain inputs. Uncertainty is propagated through the system in a way that defers making a decision until interaction cannot be made otherwise. This allows the system to maintain multiple hypotheses about what the user is trying to do and update the relative probabilities of these based on observed sensor information over time. They discuss a number of potential applications

using this framework, such as allowing selection when a touch covers multiple buttons by integrating a probability mass function for the touch over the buttons and selecting the more likely one. A similar approach is proposed to assign mouse clicks to one of a number of on-screen targets, allowing improved GUI pointing for users with motor impairments.

The same authors later formalised the statistical representation of their framework using Monte Carlo sampling of the intent distributions [77]. They implemented a mobile app that could model multiple possible interpretations of a gesture, and update visual feedback as the probabilities of the different interpretations changed.

Pohl and Murray-Smith framed the process of autonomy handover as a spectrum of interactions ranging from focused to casual [78]. Casual interactions can be utilised in situations where it is not socially acceptable or physically safe to interact, such as waving a hand over a phone to pause the music it is playing. Focused interactions are the more usual tasks where engagement with the device is necessary, such as picking up the phone to search for a new song to play. The authors propose utilising the volume around a device as the medium for casual interactions, with the device itself reserved for focused interactions.

In addition to these formulations and systems where uncertainty is explicitly modeled to control the autonomy tradeoff, this process can also be implicit. For example, although the authors do not frame it in this way, WalkType [33] can be thought of in these terms. When the user is walking, the system automatically infers this from the accelerometer readings and is empowered to make bigger corrections to account for the greater uncertainty in touch while walking. At slower paces or at rest, the user has more control over their input. The uncertainty model here is hidden from the user and the control tradeoff is not explicit, but it does still exist.

2.6 Conclusion

This Chapter has reviewed the literature related to the work that will be presented in the rest of the thesis. An examination of the problems of touch has been given, with particular focus on the idea of touch offsets between a user's intended target and the location reported by the touchscreen. A discussion of the causes of these offsets followed, along with a look at the existing models which have been used to model and improve touch. Some of the limitations of these existing models were discussed as motivation for this thesis.

Additionally, the idea of autonomy handover — negotiation of control between user and system using uncertain inputs — was introduced. This important but relatively underused model of interaction was discussed, and a number of examples from the literature were presented.

Chapter 3

Bayesian Regression Models

Summary. This chapter introduces a number of statistical techniques which will be important throughout the rest of this thesis. In particular, a brief discussion of Bayesian techniques is given, followed by the introduction of Gaussian Processes (GPs) and Relevance Vector Machines (RVMs).

3.1 Introduction

This thesis describes the design and implementation of a number of probabilistic models of touch. To do this, we make use of several existing machine learning algorithms. This chapter describes the Bayesian approach to regression, which is the task of learning continuous functions which map inputs $\mathbf{x}$ to outputs $f(\mathbf{x})$. In an offset modelling task, the inputs are typically positions recorded by a device, and the outputs are the offsets.

The use of Bayesian regression is motivated by the observation in the previous chapter that existing offset models have neglected the random component of touch. For example, Henze et al. [1] used fifth order polynomials to model offsets. This assumes that if a phone records a user's touches in a given place, their intended target can be inferred exactly. We have seen that this is not the case — given a target to touch multiple times, users will touch in slightly different places at each repetition.

In Bayesian regression, the model predictions are not deterministic values but rather probability distributions over the output value. From these distributions we can quantify how uncertain we should be about a given prediction, and use this predictive uncertainty as an additional input. In particular, in Chapter 5 we use the predictive uncertainty to obtain the probabilities that a touch on a soft keyboard was intended for each of the keys.

We begin by deriving the general predictive distribution for a Bayesian linear regression model, and then focus on two closely related models that we shall make use of in later

chapters — the Relevance Vector Machine (RVM) and Gaussian Process regression.

The GP can model complex non-linear functions without making detailed parametric assumptions about the form of the mapping, and provides robust estimates of the predictive variance of its predictions. The RVM builds predictive models which are sparse in the training data, so that predictions are conditioned on only a small subset of the available data. This allows efficient training and prediction, at the cost of some power in modelling the predictive variance.

3.2 Bayesian Linear Regression

In this Section, we will define Bayesian linear regression, a probabilistic treatment of a common data analysis task. We will derive the predictive distributions necessary to perform inference in this framework.

One of the simplest regression models that is regularly encountered is the best fitting straight line for a set of data. The equation of such a line is

$$y = w_1 + w_2 x,$$

where w_1 is the intercept of the line with the y axis and w_2 is the gradient of the line. The regression task consists of finding appropriate values for w_1 and w_2.

This is a specific example of a more general class of models known as linear regression models. In these, the variable y is expressed as a linear combination of regressands $\mathbf{x} = [1, x_1, \ldots, x_D]$ plus some additive noise ϵ, such that

$$y = \mathbf{x}^T \mathbf{w} + \epsilon.$$

A common assumption is that ϵ follows an independent, identically distributed Gaussian distribution with zero mean and some variance σ^2. The vector $\mathbf{w}$ is a set of weights w_i, one for each regressand. Determining these weights is the regression task. We often write $y = f(\mathbf{x}) + \epsilon$, with $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$.

There are two approaches to performing inference over $\mathbf{w}$. The first is to find a point estimate. The second, with which we will primarily be concerned in this Chapter, is the Bayesian perspective, which aims to infer a probability distribution over $\mathbf{w}$.

Bayes' rule is a fundamental result in probability theory, which (speaking informally) expresses how belief about the distribution of a quantity should change given the available data. It states that the *posterior distribution* $p(A|B)$ of a quantity A conditioned on data B

is given by:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}.$$

$p(A)$ is the *prior distribution* on A, in which we can encode our initial beliefs about A. $p(B|A)$, typically called the *likelihood*, encodes the degree of belief in B given A. $p(B)$ is the *marginal likelihood*, encoding the probability of B integrated over all values of A. The marginal likelihood is sometimes called the model evidence.

In Bayesian regression, our task is to find the posterior distribution of $\mathbf{w}$ given a set of observations $\{\mathbf{x}_i, y_i\}_{i=1}^N$. Letting $\mathbf{X}$ denote a matrix whose i-th column is $\mathbf{x}_i$ and $\mathbf{y}$ a column vector of the targets y_i, we have from Bayes' rule

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})}.$$

It is straightforward to show that the likelihood, assuming the Gaussian noise distribution mentioned above, may be expressed as

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{X}^T\mathbf{w}, \sigma^2\mathbf{I}),$$

another Gaussian distribution. $\mathbf{I}$ is the $N \times N$ identity matrix.

We also assume a Gaussian prior on the weights, with covariance matrix Σ

$$p(\mathbf{w}) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}).$$

It is a standard result in probability theory that if the prior and likelihood are Gaussian, the posterior will also be Gaussian. In particular, it can be shown that here we obtain

$$p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \sim \mathcal{N}(\bar{\mathbf{w}}, \mathbf{A}^{-1}),$$

where $\mathbf{A} = \sigma^{-2}\mathbf{X}\mathbf{X}^T + \mathbf{\Sigma}^{-1}$ and $\bar{\mathbf{w}} = \sigma^{-2}\mathbf{A}^{-1}\mathbf{X}\mathbf{y}$. This distribution is the end result we are looking for, allowing us to make estimates of the parameters $\mathbf{w}$ or perform Bayesian inference over all possible values.

An established necessity in regression is the process of regularisation, in which the estimated weights are forced to be 'small' in some sense by adding a penalty to the objective function. This process has two primary benefits. First, it prevents very large weights when the problem is ill-conditioned, making the regression solutions more stable. Secondly, forcing the weights to be smaller than the training data might otherwise suggest, the learned models are smoother, such that they fit the training data slightly worse but can generalise better to unseen test points. This is the principle of Occam's Razor — the simplest available model which fits the

data is often the best.

An advantage of the Bayesian approach to regression is that the assumption that the weights should be small is included directly into the inference by the specification of the prior. The Σ^{-1} term in A is equivalent to the regularisation parameter that would have to be added in a non-Bayesian treatment.

3.2.1 Predictions

To make predictions on test data with the model derived above, we average over all possible values for the weights, weighted by their probability. This is a sharp contrast to the non-Bayesian case, where a single value of the weights (often calculated as the posterior mean of the weight distribution above) is used to make predictions. The Bayesian treatment gives a distribution of values of the function output $f(\mathbf{x}_*)$ at the point $\mathbf{x}_*$. Writing f_* to denote $f(\mathbf{x}_*)$, we have

$$\begin{aligned} p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) &= \int p(f_*|\mathbf{x}_*, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{y}) d\mathbf{w}) \\ &= \mathcal{N}(\sigma^{-2}\mathbf{x}_*^T \mathbf{A}^{-1}\mathbf{X}\mathbf{y}, \mathbf{x}_*^T \mathbf{A}^{-1}\mathbf{x}_*). \end{aligned}$$

The distribution is again Gaussian, with a mean given by the product of the posterior mean weight and the test input, which makes sense given the Gaussian distribution of the weights. The variance is quadratic in the test input, so that there is more uncertainty for larger test inputs.

3.2.2 Input Projection

Linear models have that name because they are linear in the sense of the weights, not necessarily in the sense of the data. That is, the regressands can each be a potentially non-linear transformation of the data. For example, consider fitting a cubic model to a set of one dimensional data:

$$y = w_1 + w_2 x + w_3 x^2 + w_4 x^3 + \epsilon.$$

This is a linear combination of regressands which can be written in the form $y = \mathbf{x}^T\mathbf{w} + \epsilon$ for $\mathbf{x} = [1, x, x^2, x^3]^T$, but the higher powers of x are not linear functions.

More generally, we can consider functions $\boldsymbol{\phi}(\mathbf{x})$ which map D-dimensional inputs $\mathbf{x}$ to an M dimensional *feature space*. As long as this mapping is not dependent on $\mathbf{w}$, we can perform a linear analysis in feature space where we learn M rather than D weights. This allows for more expressive models which are not simply linear in the data.

Defining $\mathbf{\Phi}$ as the matrix whose columns are the vectors $\boldsymbol{\phi}(\mathbf{x})$ for all points $\mathbf{x}$ in the training set, the analysis is very similar to the model derived above, with $\mathbf{X}$ replaced by $\mathbf{\Phi}$ at all points. This leads to a predictive distribution

$$f_* \sim \mathcal{N}(\sigma^{-2}\boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{A}^{-1}\mathbf{\Phi}\mathbf{y}, \boldsymbol{\phi}(\mathbf{x}_*)^T A^{-1}\boldsymbol{\phi}(\mathbf{x}_*)),$$

where now $A = \sigma^{-2}\mathbf{\Phi}\mathbf{\Phi}^T + \Sigma^{-1}$. Computing this distribution requires inversion of an $M \times M$ matrix. It can be shown (see Chapter 2 of [79] for a derivation) that this may be rewritten as

$$\begin{aligned} f_* \sim \mathcal{N}(&\boldsymbol{\phi}(\mathbf{x}_*)\mathbf{\Sigma}\mathbf{\Phi}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \\ &\boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{\Sigma}\boldsymbol{\phi}(\mathbf{x}_*) - \boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{\Sigma}\mathbf{\Phi}(\mathbf{K} + \sigma^{-2}\mathbf{I})^{-1}\mathbf{\Phi}^T\mathbf{\Sigma}\boldsymbol{\phi}(\mathbf{x}_*)), \end{aligned} \tag{3.1}$$

where $\mathbf{K} = \mathbf{\Phi}^T\mathbf{\Sigma}\mathbf{\Phi}$.

This predictive distribution requires inversion of a matrix of size N, the number of datapoints, which may be more convenient if the dimension M of the feature space is very high. This is the typical equation given for generalised Bayesian linear models.

3.2.3 Hyperparameter Selection

The analysis above makes the implicit assumption that we have knowledge about the prior distribution of the weights and the noise — that is, we have some good estimate for σ^2 and Σ. In general this is often not the case. The Bayesian response to this would be to place a prior on these parameters, obtain a posterior distribution and integrate them out when making predictions, as we did for the weights above. Unfortunately this typically results in predictive distributions which are not Gaussian, and often not tractable at all.

It is therefore common to approximate the value of these parameters by their maximum a posteriori (MAP) values. This means maximising the posterior distributions. A common choice for the covariance of the prior distribution of the weights is to assume they are all drawn from the same Gaussian, with variance o_w^2. These parameters of the priors are typically called *hyperparameters*.

Applying Bayes' rule again, the posterior of the hyperparameters can be written

$$p(\sigma^2, \sigma_w^2|\mathbf{y}, \mathbf{X}) \propto p(\mathbf{y}|\mathbf{X}, \sigma^2, \sigma_w^2)p(\sigma^2, \sigma_w^2).$$

If we place a uniform prior on the hyperparameters, maximising the posterior becomes equiv-

alent to maximising $p(\mathbf{y}|X, \sigma^2, \sigma_w^2)$, the marginal likelihood of the data $\mathbf{y}$. This is given by

$$p(\mathbf{y}|\mathbf{X}, \sigma^2, \sigma_w^2) = \int p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2) p(\mathbf{w}|\sigma_w^2) d\mathbf{w} \sim \mathcal{N}(0, \mathbf{C}),$$

which has zero mean since $\mathbf{y}$ is linear in $\mathbf{w}$ and the prior on $\mathbf{w}$.has zero mean. The covariance is $\mathbf{C} = \sigma^2\mathbf{I} + \sigma_w^2 \mathbf{\Phi}\mathbf{\Phi}^T$.

There is no closed form solution for maximising this distribution, so in practice the MAP estimates of the hyperparameters are found by gradient minimisation of the negative logarithm of the marginal likelihood. Tipping [80] gives a description of this procedure, and the goodness of the approximation to the predictive distribution obtained by using MAP estimates for σ^2 and σ_w^2.

3.3 The Relevance Vector Machine

The Relevance Vector Machine (RVM), introduced by Tipping [80], is a variant of the Bayesian regression techniques described above. We make use of this algorithm in Chapter 7. It has the property of *sparseness*, such that all but a small number of the weights in the model are zero or very close to zero, so that corresponding basis functions are not required for predictions. Thus, the RVM can make predictions efficiently using only a small subset of the training data. This is a significant benefit given that offset behaviour varies based on a range of factors, discussed in Chapter 2. This means that many offset models may be needed for different conditions, so training models with small amounts of data is an attractive property.

The RVM is defined similarly to the standard Bayesian regression model, but with a different prior on the weights. In particular, each weight has an independently defined prior, such that

$$p(w_j|\alpha_j) \sim \mathcal{N}(0, \alpha_j^{-1}),$$
$$p(\mathbf{w}|\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \mathbf{A}).$$

This is a joint prior over $\mathbf{w}$ with precision hyperparameters $\mathbf{A} = \text{diag}\,\boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_N]$ (a precision is an inverse variance).

The basis functions for the RVM are typically, but not necessarily, taken to be Radial Basis Function (RBF; also called Gaussian) kernels, defined as

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{|\mathbf{x} - \mathbf{x}_j|^2}{2l^2}\right).$$

There is thus one basis function for each training point $\mathbf{x}_j$.

The derivation of the mean and variance of the distribution over weights for the RVM is very similar to that for the general Bayesian regression case, and we obtain

$$\mathbf{w} \sim \mathcal{N}((\mathbf{\Phi}^{\boldsymbol{T}}\mathbf{\Phi} + \sigma^2\mathbf{A})^{-1}\mathbf{\Phi}^T\mathbf{y}, (\sigma^{-2}\mathbf{\Phi}^{\boldsymbol{T}}\mathbf{\Phi} + \mathbf{A})^{-1}).$$

As in the general case, the prior on the weights acts as a regularisation term, forcing the weights to be 'small'. However, rather than being equally penalised by a single variance σ_w^2, each weight in the RVM is individually penalised by its associated hyperparameter. The larger the value of α_j, the smaller the corresponding w_j. Sparseness is achieved when some of the precisions are effectively infinite, making the corresponding weights zero. This prunes the basis functions with zero weights, so that they have no effect on model predictions.

The learning procedure for the RVM actually causes these sparseness conditions to occur, such that the resulting model is based on only a small subset of the available data. The training points which survive are called Relevance Vectors, hence the name of the algorithm.

Figure 3.1 shows an example of an RVM used to fit some one-dimensional data. The training set consists of 10 points, shown as circles. The four green circles are the relevance vectors — these are the only points the RVM uses to make predictions. The mean prediction is shown by a blue line, and the 95% confidence interval of the predictions is shown by the shaded grey area. Note that these intervals do not encompass all of the data, suggesting the RVM may be too confident in its predictions. This is worth remembering when employing this algorithm.

3.3.1 Learning in the RVM

As with the general Bayesian model, the task of training the RVM is equivalent to maximising the marginal likelihood of the training data with respect to the hyperparameters, or equivalently minimising its negative logarithm. The predictive distribution for the weights is again based on the MAP estimates of the hyperparameters, since a full Bayesian treatment is not analytically tractable.

In his paper introducing the algorithm, Tipping [80] suggests in general the use of gamma priors on each of the α_j and an inverse gamma prior on σ^2, but in practice recommends the limiting case of improper uninformative uniform priors on these parameters. Additionally, since all hyperparameters are variances or precisions, they must be positive to have any meaning, so Tipping proposes to maximize the logarithm of each parameter. In order for the MAP estimate to be equivalent to marginal likelihood maximisation, the uniform priors have to be defined in logarithm space, which implies non-uniformity at the linear scale. It is this choice which is ultimately responsible for the sparseness of the RVM. The reason for this is

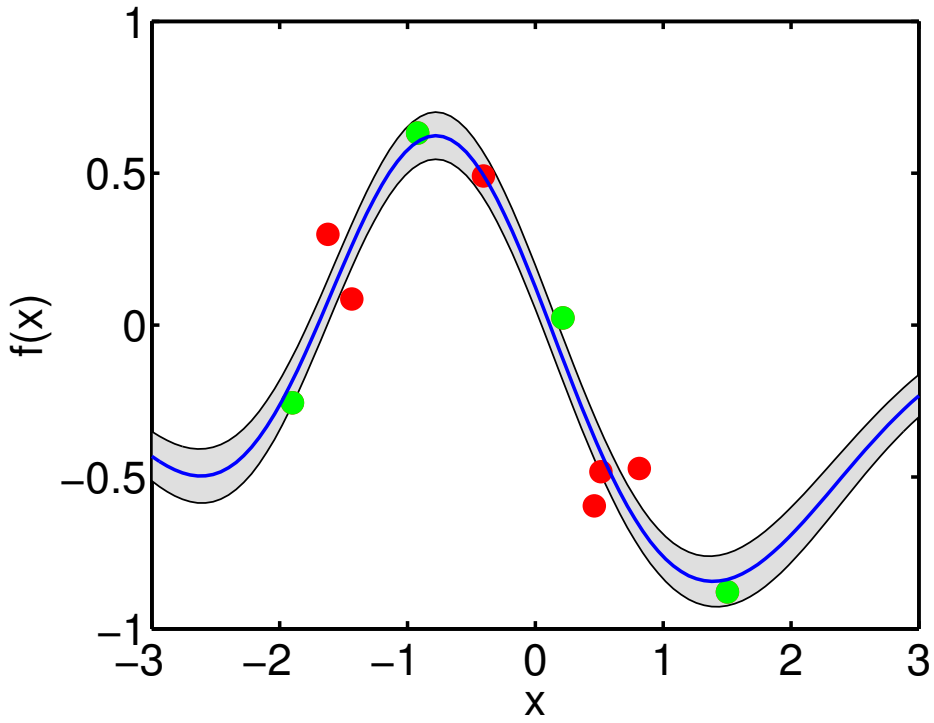


Figure 3.1: The predictive mean and 95% confidence intervals for an RVM trained on the 10 points shown by circles. The four green circles are the relevance vectors — only these are used for prediction.

quite complicated, and outwith the scope of this work. For a detailed discussion, see Chapter 2 in [81].

Expectation Maximisation

Tipping suggests that rather than using direct minimisation of the negative log marginal likelihod, the RVM should be trained using an Expectation Maximisation algorithm.

This approximate procedure works iteratively, alternating between an Expectation or E-step and a Maximisation or M-step. A full derivation of the procedure is not given here, but the E-step consists of computing the mean

$$\boldsymbol{\mu} = (\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi} + \sigma^2\mathbf{A})^{-1}\boldsymbol{\Phi}^T\mathbf{y}$$

and variance

$$\boldsymbol{\Sigma} = (\sigma^{-2}\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi} + \mathbf{A})^{-1}$$

of the posterior distribution of $\mathbf{w}$ using the current estimates of $\boldsymbol{\alpha}$ and σ^2.

The M-step then updates the hyperparameter estimates according to the following equations:

$$\alpha_j^{\text{new}} = \frac{1}{\boldsymbol{\mu}^{\mathbf{2}} + \boldsymbol{\Sigma}_{jj}},$$
$$(\sigma^2)^{\text{new}} = \frac{||\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2 + \text{Tr}[\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi}\boldsymbol{\Sigma}]}{N}.$$

The parameter estimates of the α_j and σ^2 are initialised using a prior, typically gamma or uniform.

Sequential Training

Both of the algorithms described above for training the RVM share a problem — they both begin by initialising all the weights to a non-zero value, and thus at least at first the training process uses all of the data. These algorithms are cubic in N, and this can be a signficant computational burden.

Indeed, the goal of a sparse algorithm is lower computational load by using only a subset of the data. In many problem domains datasets are too large to perform the training operations without the use of a cluster computer. Thus, an alternative training algorithm has been proposed which starts with all weights set to zero and adds basis functions sequentially [82].

This is based on the observation by Faul and Tipping that as a function of a single α_j, the optimisation of the marginal likelihood can be performed exactly [83]. The full algorithm is not given here, but essentially in each iteration it computes the increase in marginal likelihood for all bases not in the model, and adds the one that leads to the biggest increase. This process is repeated until the total change in $\log \boldsymbol{\alpha}$ is below a specified threshold, at which point the marginal likelihood has reached a local maximum and the model is considered trained.

We have not made use of this algorithm in this thesis, since the datasets we consider when applying the RVM are not large enough to cause computational issues. However, this algorithm may be useful in the future if implementing sparse regression on mobile devices.

3.4 Gaussian Processes

The approach to regression used in the Bayesian linear regression and RVM models is described by Williams [84] as the weight-space view. An equivalent approach which leads to the same result is the function-space view, where inference is performed over functions. To perform such inference, we need to define distributions over functions rather than scalar valued variables. To do this, we introduce the *Gaussian process*, defined as a *collection of random variables, any finite number of which have a joint Gaussian distribution*. This may

be thought of as a generalisation of the Gaussian distribution which governs the properties of functions. We will show shortly that this definition implies a distribution over functions.

The motivation for this approach is as follows. In the weight-space view, the form of the parametric model (the choice of basis functions) and the forms of the priors on the hyper-parameters can result in a prior distribution on the joint distribution of the function values. When predicting on test data, the quality of the predictive uncertainty depends on this implicit prior on functions. For many models, these function priors are defined implicitly by the other choices of the model, which can result in priors with undesirable qualities. In this case, we may conclude that the prior on functions was itself undesirable, but it was introduced almost accidentally by other assumptions in the model. The function-space view is motivated by the desire to begin the analysis by explicitly defining a prior over the functions we expect.

This also avoids the issue of having to select a particular class of function as basis functions in a parametric model. Rather, the GP (roughly speaking) assigns a prior probability over all possible functions, and lets us do inference in this space. This framework has a very attractive property — namely that if we consider functions evaluated at a finite set of points, we can perform inference and obtain posterior distributions over functions that are identical to those we would derive if we were able to analyse the infinitely many other points in the function domain. This framework offers both a sophisticated statistical view of regression with computational feasability, which is both rare and desirable.

The GP is a collection of random variables which represent the values of the function $f(\mathbf{x})$ at locations $\mathbf{x}$. Notationally, we index the random variables by the training points $\mathbf{x}_i$: $f_i := f(\mathbf{x}_i)$ is the random variable corresponding to the training example $(\mathbf{x}_i, y_i)$.

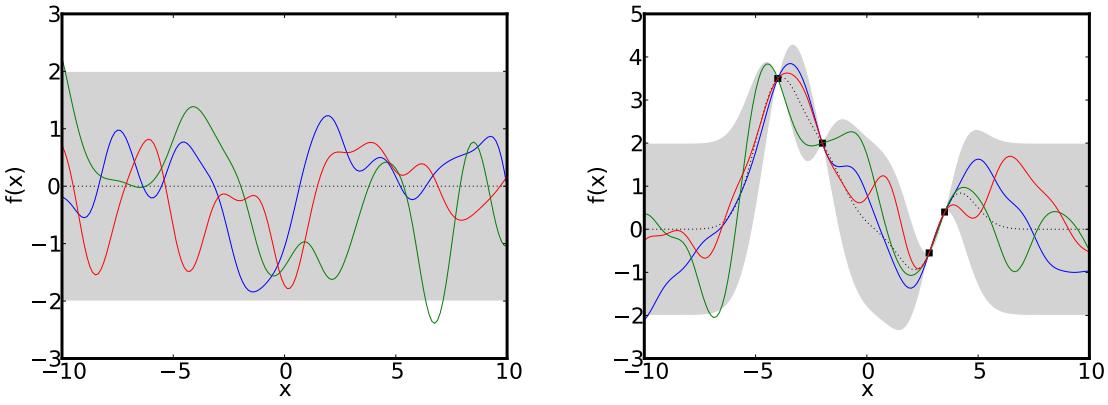
A Gaussian process is specified in terms of a *mean function* $m(\mathbf{x})$ and a *covariance function* $k(\mathbf{x}, \mathbf{x}')$. For a process $f(\mathbf{x})$, these are defined as

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]. \end{aligned}$$

The mean function is often taken to be zero, though it need not be. In all our usage of GPs, we will assume a zero mean function.

Given a zero mean function, the important features of the GP are defined entirely by the covariance function. The choice of covariance function implies a distribution over functions. We can sample from the distribution of functions evaluated at a certain set of points $\mathbf{X}_*$. To do this, we compute the covariance matrix $K(\mathbf{X}_*, \mathbf{X}_*)$ where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and sample a vector

$$\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}_*, \mathbf{X}_*)),$$



(a) Prior Samples. (b) Posterior Samples.

Figure 3.2: (a)Three functions sampled from a random GP prior with a Gaussian covariance function. They have different means but similar characteristic length scales. (b) Three sample functions from the posterior distribution obtained by training on the points shown as black squares.

and plot pairs of $(\mathbf{x}_*, f_*)$ values. Each vector sampled in this way represents a different function, evaluated at the points $\mathbf{X}_*$.

Figure 3.2(a) shows an example of this using the popular Gaussian covariance function

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-l||\mathbf{x}_i - \mathbf{x}_j||^2\right).$$

We can see that the three functions sampled look smooth and have different means, but appear to have similar characteristic length scales. Informally, this may be thought of as the distance you have to move in x space before the corresponding function values $f(x)$ become uncorrelated. The length scale gives a measure of how rapidly the functions can change. The parameter l in the covariance function controls this. Multiplying the covariance function by a constant factor b allows us to scale the overall variance of the random functions. We make use of this when we use GPs in later chapters.

3.4.1 Predictions

Drawing random samples from a GP prior is not generally interesting. We rather wish to condition the sampled functions on the training data. According to the GP definition, any subset of random variables has a joint Gaussian distribution. This therefore means the joint distribution of the training targets $\mathbf{f}$ and the outputs $\mathbf{f}_*$ at test points X_* is Gaussian under the prior. That is,

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right).$$

The posterior distribution over the test points may be inferred by conditioning this joint prior on the observations, which yields

$$\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(K(\mathbf{X}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f}, \\ K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{X}_*)).$$

Function values at the test points $\mathbf{X}_*$ can be sampled by computing the mean and covariance matrix of this Gaussian, and sampling from the multivariate Gaussian they define.

Figure 3.2(b) shows three functions sampled from the posterior distribution obtained by conditioning the prior described above on the four points shown as black squares. The mean predictive function is shown as a dashed line, and the shaded grey region shows the 95% confidence region for the GP predictions. Note that the functions interpolate the observations, and the predictive variances at the data are zero.

Note that his analysis assumes the case of no observation noise, which is rarely the case in reality. Typically we only observe a noisy variable $y = f(\mathbf{x}) + \epsilon$. As in the Bayesian linear regression model, we assume additive i.i.d. Gaussian noise with variance σ^2. This gives us a joint distribution between the observations and test targets of

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I} & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right).$$

Again, we condition on the observations we arrive at the predictive distribution

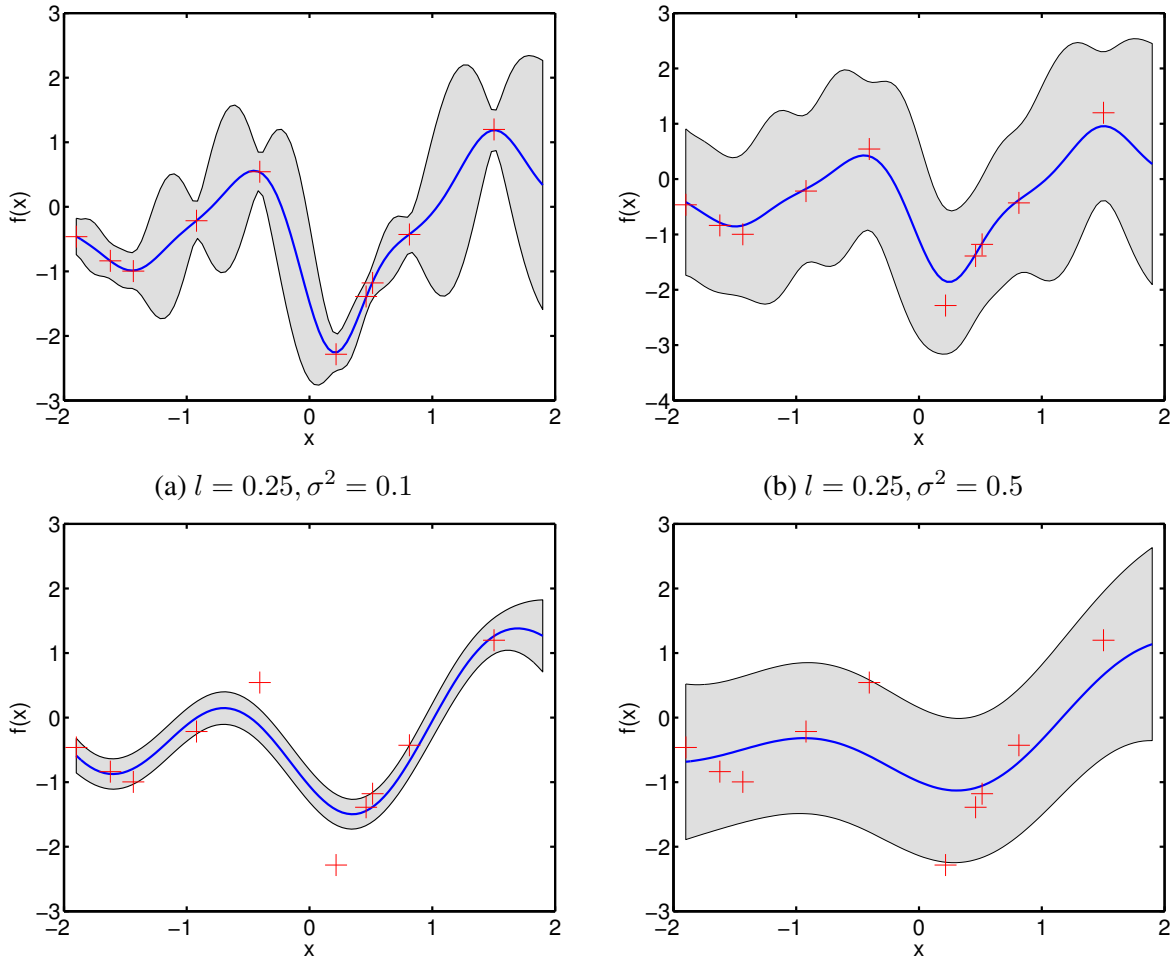
$$\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}]^{-1}\mathbf{y}, \\ K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}]^{-1}K(\mathbf{X}, \mathbf{X}_*)).$$

Note that this has the same form as the predictive equation 3.1 for the Bayesian linear regression model with projected inputs, where we have now defined the $K(.,.)$ notation for compactness. Thus the function-space view leads us to exactly the same results as the weight-space view.

3.4.2 Learning for the GP

As with the previously described models, the training task consists of optimising the marginal likelihood of the training data with respect to a set of hyperparameters. The noise variance σ^2 is one hyperparameter which is always present, while the others depend on the specific choice of covariance function. For example, the Gaussian covariance function described above has one additional hyperparameter, the length scale l.

The effect of varying the hyperparameters for this covariance function can be seen in Figure 3.3. We have sampled 10 points from the GP prior defined by a Gaussian covariance of a given length scale and noise level. These points are indicated by red crosses. Then we have trained four GPs on these points with different values of the hyperparameters.



(a) $l = 0.25, \sigma^2 = 0.1$ (b) $l = 0.25, \sigma^2 = 0.5$

(c) $l = 1, \sigma^2 = 0.1$ (d) $l = 1, \sigma^2 = 0.5$

Figure 3.3: Mean predictive functions and 95% confidence intervals for four GPs trained on the red crosses, each with different hyperparameter values.

Each plot shows the mean predictive function (blue line) and 95% confidence regions (shaded grey) for one of the GPs. The model in panel (a) was trained using the same length scale and noise variance as the generating model. The mean function interpolates the data exactly, and the predictive variance can be seen to increase as we move away from the data points. Panel (b) shows a model with the same length scale, but higher noise variance. The mean function no longer interpolates the data, as it is somewhat smoother. The predictive variance is also larger at all points. Effectively, more of the variation in the data has been explained as noise rather than fitted by the model.

Panel (c) shows a model with the same noise variance as the original data but a longer length

scale. The function is less complex than the true model, and the predictions do not match the data completely — two points fall outside the 95% confidence intervals of the predictive distribution. Panel (d) shows a model with higher noise and higher length scale than the generative model. The predictive variance is very high, such that the model accounts for most of the variation in the data as noise.

Throughout this thesis, we will apply Gaussian Process regression to touch offset modelling, and will use the following combination of linear and Gaussian covariance terms:

$$k(\mathbf{x}_i, \mathbf{x}_j) = b\left(a\mathbf{x}_i^T\mathbf{x}_j + (1-a)\exp\left\{-l||\mathbf{x}_i - \mathbf{x}_j||_2^2\right\}\right) + \sigma^2\delta_{ij},$$

where δ_{ij} is 1 if $i = j$ and 0 otherwise. This choice was made after trying a number of different covariance functions on data gathered in the next chapter. A discussion of this process is given in Section 4.2.2. Thus we have four hyperparameters to optimise — a, which controls the relative balance of the two terms, b, which scales the overall variance, l, the length scale, and σ^2, the noise variance.

In Chapters 4 and 5, we optimise these parameters using cross validation on the training data. In Chapter 7 we use the maximum marginal likelihood method. Both choices lead to good predictive performance of the offset models.

3.4.3 The RVM as a Gaussian Process

As an interesting aside, we observe that the RVM may be thought of as a special case of a Gaussian Process, with a particular covariance function, though it is not typically presented in this way. The covariance is of the form

$$k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{N} \frac{1}{\alpha_j}\phi_j(\mathbf{x})\phi_j(\mathbf{x}'),$$

where the α_j are the hyperparameters and $\phi_j(\mathbf{x})$ are the basis functions, one for each training point as before.

This covariance function is somewhat unusual in that it depends on the training data. This means that the implied prior over functions of this covariance depends on the data, which is atypical for a Bayesian model. This property leads to some undesirable results.

For a test point $\mathbf{x}_*$ far from any of the relevance vectors, all of the basis functions will have a value close to zero. This means the predictive distribution with a mean close to zero and, crucially, a variance close to zero. That is, the RVM has high certainty for inputs far from the data. This is the opposite behaviour from what would we expect — the certainty should be low since we are extrapolating far from the training data.

Rasmussen and Quinonero-Candela [85] have argued that the root cause of this behaviour is that the covariance function is degenerate (it has finite dimension). They proposed RVM*, a modified algorithm, to solve this problem. However, this algorithm is no longer truly sparse, requiring more data to make good predictions than the RVM.

3.5 Summary and Conclusion

This Chapter has presented an overview of Bayesian regression techniques, with particular focus on the Relevance Vector Machine and Gaussian Process techniques. Both are powerful modelling tools which can model complex, non-linear functions by projecting data into high-dimensional space using a covariance function. The RVM creates solutions which are sparse in the training data, at the cost of introducing some undesirable features of the predictive variance.

In the remainder of this thesis we will use these algorithms to model touch. GPs are used in Chapters 4, 5 and 7 and the RVM is used in Chapter 7. We will make particular use of the fact that the predictive distribution of the GP can be obtained, including the predictive variance. We use this in Chapter 5 to compute the probability that a given touch was meant for each of the keys on a soft keyboard.

There are many aspects of these algorithms which we have not covered in this Chapter since they are beyond the scope of the work in the thesis. For a detailed description of issues surrounding Gaussian Processes, including a broad range of covariance function choices and a comparison with other linear regression models, we refer the interested reader to the work of Rasmussen and Williams [79].

Chapter 4

Modelling Touch Offsets using Gaussian Processes

Summary. When interacting with touch screen devices, the physical contacts of a user's fingers are typically offset from their *intended touch locations* — that is, the points at which they aimed. Existing research has focused on building device specific compensation models for this problem from large datasets. This chapter describes the use of Gaussian Process regression to model touch offsets. The learned offsets are both highly non-linear and highly user specific. We find that personalized models outperform those built from data pooled from many users, echoing results from previous models of touch. The GP approach has the additional benefit of capturing information about the uncertainty in a user's touches. The GP model is shown to outperform techniques from the literature in a target acquisition study.

4.1 Introduction

As discussed in Chapter 2, touch interaction is subject to many sources of error. One of the primary sources is the well-known "fat finger problem": through the act of touching, the user can occlude the targets they are aiming at, and on modern devices the screen resolution is high enough that targets may be smaller than the contact area between finger and screen. Further, Holz and Baudisch [30] framed the problem as one of visual perception. That is, users see only the top of their finger but touch with the underside, and parallax effects can create a mismatch between their perceived and actual touch points. The combination of these factors results in a systematic offset between the user's *intended touch location* and the *recorded touch location* on the device.

As touch devices have become increasingly ubiquitous, there is a clear need for models to correct for this offset and facilitate accurate input. Text entry is a particularly important use

case impacted by offsets, since the keys on a soft keyboard are small and densely packed. The keys on an iPhone 5 portrait keyboard measure 4×6 mm, smaller than the 8-10 mm minimum target size reported as necessary for 95% accuracy in previous studies of touch. Thus, even small offsets can result in touches hitting the wrong keys and causing frustration for users.

As a result, a number of approaches have been proposed to model and correct touch offsets — these were discussed in Section 2.4. These existing models have one or more of the following limitations:

- reliance on custom hardware;
- detailed parametric models of the finger, which may not hold in all usage scenarios;
- derived from data pooled from many users.

The last of these is not obviously a drawback, but research has shown (e.g. [19]) that touch offsets are user specific and that pooled data models perform worse than those tailored to individual users.

This chapter describes the application of Gaussian Process regression to the touch offset problem. GPs suffer from none of the drawbacks above: the algorithm can be implemented on commercial touch hardware, makes no parametric assumptions about the form of the offset function, and can be applied on a user-by-user basis. Further, this Chapter demonstrates the nonlinearity of touch offsets in greater depth than previous work, and investigates the amount of training data required to train an offset model using Gaussian Processes. This is one of the only studies that has treated touch as a machine learning problem, which is perhaps surprising given previous findings about the variability of touch.

Statement of Original Work

The work in this chapter was primarily carried out by the author. The exception is the creation of the Android logging application described in Section 4.4.5, which was implemented by a colleague, Markus Löchtefeld. Markus also assisted in running participants through the user studies mentioned herein.

4.2 Touch as a Machine Learning Problem

The offset modelling problem can be thought of as a regression task where the goal is to learn a mapping between a set of inputs $\mathbf{s}$ and a two dimensonal output vector which is either an

intended position (x', y') or a touch offset (Δ_x, Δ_y). This was visualised in Figure 2.1. This diagram assumes the inputs $\mathbf{s}$ are the touch locations recorded by the sensing algorithm on an existing phone. In this Chapter we also consider an alternative set of inputs: the raw sensor values from the device screen itself. The former is the typical input for offset models in the research literature, since raw capacitive sensor values are seldom available on commercial devices.

Thus there are four possible mappings we can learn:

1. sensor values to intended positions
2. sensor values to offsets
3. device positions to intended positions
4. device positions to offsets

The first and last of these are generally the most useful: sensors to intended positions because it avoids using any intermediate positions from the device, and device positions to offsets since it allows a compensation function to be learned which can be applied on top of a device's own coordinate reporting.

Note that this work refers primarily to capacitive touch screens — resistive screens were once popular, but almost all touch devices in the past few years have moved to capacitive technology. Each phone model has its own algorithm in hardware which converts capacitive sensor readings into touch positions, and those positions are typically the only information reported at an operating system level. However, if the capacitive sensor readings are available directly, they offer a more interesting source of information for computing the intended touch location.

To see why this is, consider Figure 4.1. This shows the sensor values from a Nokia N9 smartphone for a touch aimed at the white circle (the sensor values on this phone are available through a debugging interface). The phone is being held in a landscape orientation, and the touch comes from a user's left thumb. Some low level noise can be seen in many sensors, but by far the highest level of activation is seen in the sensors immediately surrounding the touch area. The device's reported touch location is shown by the blue circle, and is offset from the intended target (the white circle). Now consider that if the device is held in a different orientation or different fingers are used to touch, it may easily be possible to produce a touch with the same *reported* location but a different *intended* location. In this case, using the recorded location as input creates a problem, since we have the same input potentially mapped to two (or perhaps even more) outputs. However, in different orientations and with different fingers, the patterns of activated sensors will be different and thus using the sensor values as input does not cause the same possible model ambiguity.

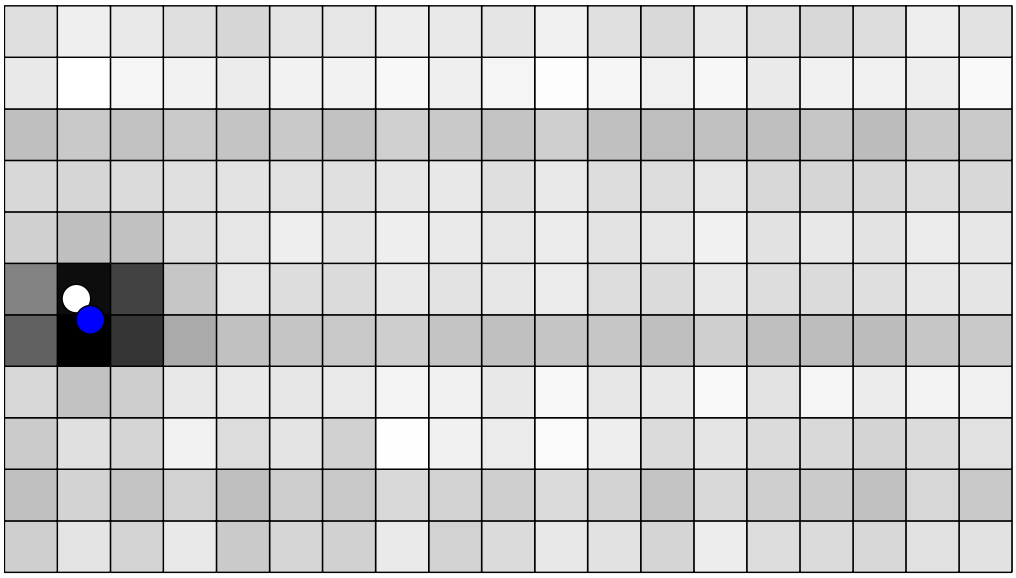


Figure 4.1: Sensor outputs (black = high; white = low) for a touch aimed at the white circle. The device's reported touch location is indicated with a blue circle.

For this reason, we use the N9 as the primary device for the experiments in this Chapter. The availability of the raw sensor values is a significant benefit. However, we will also consider the use of device position as input, since this is all that is available on the vast majority of devices. It is worth noting now that using device position as input leads to a potential explosion in the number of necessary offset models. This issue will be revisited in more detail in Chapter 7.

Given that the goal is to map complex vectors of sensor values to intended touch points, Gaussian Processes are a good choice of modelling tool. This is due to the fact that we need make no parametric assumptions about the form of the mapping function.

4.2.1 Problem Specification

As discussed in Chapter 3, a GP is specified entirely by its mean and covariance functions. In all of the experiments in this chapter, we use a zero mean function. When predicting an offset, this corresponds to predicting no offset in the absence of training data. When predicting touch positions directly, the zero mean corresponds to predicting the center of the screen in the absence of data (all coordinates are transformed to a unit square centered on the origin, without loss of generality.) The latter is a harder problem, since the prior of predicting the center is likely to be less accurate than just predicting the device coordinates, which are already based on some processing. It is thus likely that more training data are required for this task.

Note that the GP is typically used to predict one-dimensional outputs, so we have two

choices. Either we can learn separate models for x- and y-offsets, or we can transform the 2D problem into one dimension. The latter is a more complex problem, since given N training points it involves inversion of a matrix of size $2N \times 2N$, which is more computationally intensive than inverting two $N \times N$ matrices.

The 2D problem can be transformed to one dimension using the following method. Given N training examples, $\mathbf{s}_1, \ldots, \mathbf{s}_N$ and associated target locations, $(x_1, y_1), \ldots, (x_N, y_N)$, we first stack 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). Also, when predicting offsets the vector $\mathbf{z}$ is replaced by a vector of offset values instead of absolute positions.

We now build an $N \times N$ covariance matrix, $\mathbf{C}$, where the n, m-th 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. This is unlikely to be an accurate assumption — work such as [65] has found covariances between x and y in the spread of touches when targeting soft keyboards, for example.

Finally, we assume a small amount of additive Gaussian noise (with variance σ^2). This models the random component of touch — a user does not always have the same offset when targeting the same location multiple times. Additionally, this parameter helps overcome possible numerical 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') locations (or offsets values if they were used for training) 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.

4.2.2 Choice of covariance function

Many different covariance functions (sometimes called kernels) have been used for GP regression. Examples include: the Gaussian kernel

$$C(\mathbf{s}_n, \mathbf{s}_m) = \alpha \exp\left\{-l||\mathbf{s}_n - \mathbf{s}_m||_2^2\right\};$$

the linear kernel

$$C(\mathbf{s}_n, \mathbf{s}_m) = \alpha + \mathbf{s}_n^T \mathbf{s}_m;$$

and the rational quadratic kernel

$$C(\mathbf{s}_n, \mathbf{s}_m) = \left(1 + \frac{l}{2\alpha}||\mathbf{s}_n - \mathbf{s}_m||_2^2\right)^{-\alpha}.$$

There are many other possibilities: the work of Rasmussen and Williams [79] gives a thorough review. It is also possible to create composite covariance functions as linear combinations of the basis kernels.

We chose a covariance function for our GP by training models with a range of different covariance functions, including the Gaussian kernel and linear kernels with and without bias terms, as well as linear combinations of these. We optimised the hyperparameters for these functions on touch data for a single user, then tested the predictive performance on all others.

We achieved best performance using a combination of a linear and Gaussian covariance:

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

where a controls the relative influence of the linear and Gaussian terms and l 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. We shall use it throughout the thesis when we employ GP offset models.

4.3 User Study

We carried out a user study in order to test the performance of GPs in the offset modelling task.

4.3.1 Data Collection

We gathered data using a simple data collector written in Python that ran on the Nokia N9. The software displayed crosshairs on the screen that the users had to touch. The crosshair positions were randomly generated in lower 50% of the screen: this is the area that is normally occupied by the landscape keyboard of the N9 to simulate text entry in landscape mode. We chose a landscape pose since it would require two thumb usage, which we believed would give more complex offsets than one handed usage and thus present a more taxing modelling challenge.

The N9's screen had a display resolution of 480x854, with a physical screen size of 3.9 inches diagonally. Capacitive sensor readings were available through a debug interface at 10 bit resolution, with a sampling rate of approximately 30 Hz.

For each touch on a crosshair, the system recorded the intended location (the on-screen location of the crosshair), the values of the capacitive sensors and the location reported by the N9 for the touch event. We took the touch position at the point the user lifted their finger, rather than the first point of contact. This is standard practice on current phones and has been shown to be more accurate than the touch down position [20]. After the user lifted the finger from the screen another crosshair would be displayed immediately.

Figure 4.2: Experimental setup: participants held the phone in both hands and used their thumbs to touch.

4.3.2 Participants

We obtained data from a total of 8 participants (3 female), aged between 23 and 34. Participants were recruited from our local institution. All but one of our participants owned smartphones and therefore had experience operating a touch-screen device. All participants were computing science students or academics, and thus were expert users of information systems — no novice level users provided data for our study.

4.3.3 Procedure

Participants completed the study while seated in a quiet office environment. Each participant performed 1000 touches while holding the device in both hands in a landscape orientation and using their thumbs to touch (see Figure 4.2). All data was recorded in a single session without breaks, which took between 15 and 20 minutes to complete.

This scenario is representative of a number of potential real cases where users might sit and tap with both thumbs: typing is one common example; playing games another. We did not consider the effect of mobility — walking and tapping — in this study but investigate this further in the next chapter.

4.3.4 Data Preprocessing

Before building offset models, we rescaled all touch and target locations such that they were in the unit square centered on the origin. This rescaling was based on the device size, such that a target at the rightmost edge of the screen would have an x coordinate of 0.5, and a target at the bottom of the screen a y coordinate of -0.5. In the case where we predict intended locations from capacitive sensor values, this make the zero mean function of the GP correspond to the center of the screen. We believe predicting the center of the screen in the absense of data is a slightly more sensible assumption than predicting the upper left corner, which is the zero of the device coordinate system. We transform the data back to a millimeter scale when computing the error on our predictions and recommending target sizes.

4.4 Results

4.4.1 Prediction from raw sensor values

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.

Hyperparameter selection

To choose the covariance hyperparameters, 5-fold cross-validation was performed on the data for subject 1 over a range of values for $a, l, b, \alpha, \sigma^2$ and the optimal values from this analysis were then used across all subjects. We chose subject 1 for this analysis. 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. The individual models are still learned on a user-specific basis.

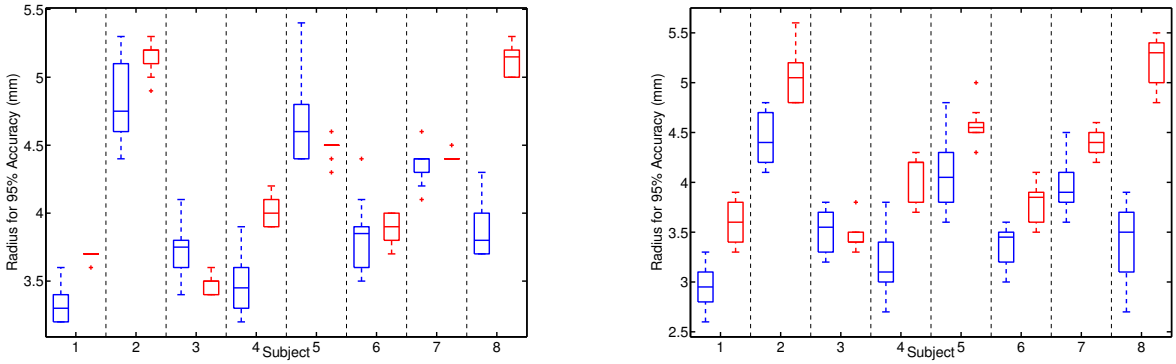
In practice this means that the results for subject 1 are likely to be slightly optimistic, and those for all other users pessimistic. Ideally, a cross-validation would be performed on a per-user basis but the necessary computation is an unreasonable demand in a mobile setting — any deployment of the system would have to use predetermined covariance parameters such as these.

The chosen parameter values were: $l = 0.05, b = 5, a = 0.1, \alpha = 0.9, \sigma^2 = 10^{-3}$. Of particular interest are the values of a and α. a controls the weighting of the linear and

Gaussian terms in the covariance function, and a value of 0.1 means that a much greater weight is assigned to the Gaussian component. This suggests the offsets are highly non-linear, since the linear term contributes a small amount of the overall offset term. α measures the covariance between the x and y components of the model. The high value found here suggests strong covariances between the components, such that the predictive distributions of the GP are not axis-aligned.

Touch accuracy

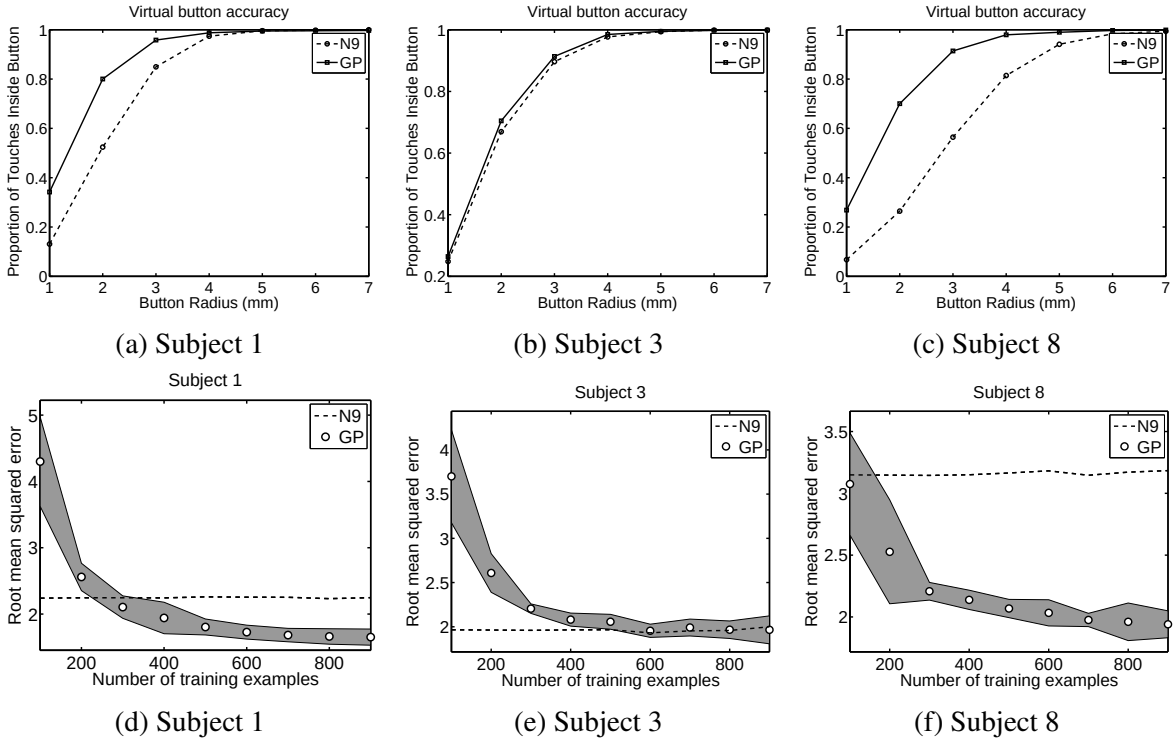
The sensor data is pre-processed prior to use by first setting to zero any values that are less than 150 (we found this was the typical background noise in the absence of touch input) and then normalising each touch so that the sum of the sensor values is equal to 1. We then train a GP mapping using these normalised sensor values as input and the intended target locations as the training targets. The mean of the predictive Gaussian provided by the GP is used as the inferred touch location.



(a) $N = 400$. (b) $N = 800$.

Figure 4.3: Target radii required to achieve 95% accuracy for a user's touches for 400 and 800 training points, when raw sensor values are used as input to the GP. In each case the blue boxplot shows the target sizes with the GP model, and the red plot the sizes for the uncorrected touches. Boxplots show the distribution of target size over 10 repetitions.

We use virtual button accuracy as a first error metric, following [37] and [39]. This is computed by assuming a circular button of a given radius is placed around the centre of the crosshair at which the user was aiming and finding the proportion of touches that would fall within this button. In Figure 4.3 we show the button radii required to achieve 95% selection accuracy for a user's touches. For each user, the blue boxplot shows the target sizes for the GP model and the red boxplot the sizes for the uncorrected touches recorded by the N9. The data were analysed for two different sizes N of the training set (400 and 800 touches). We mentioned in Chapter 2 that previous research has found that 95% accuracy is typically achieved for uncorrected touches with target sizes between 9 and 10 mm. This corresponds to the 4.5 to 5 mm range on our plots.



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

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

Figure 4.4: 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 is an average user in terms of relative improvement, subject 3 a user whose offset behaviour matches the native N9 algorithm, and subject 8 represents a user that doesn't own a touchscreen phone.

For each user, we repeated the experiment 10 times. At 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 boxplots above show the distribution in 95% accuracy target size across these repetitions.

For $N = 400$, we see that for all users except numbers 3 and 5, the GP model performs better than the N9. The differences between N9 and GP distributions is significant for all users (paired t-test with Bonferroni corrections, $p < 0.05$). For the GP, the range in mean across users of target sizes for 95% accuracy is from 3.35mm to 4.83mm, with an overall mean of 4.01mm. For the N9, the range is 3.47mm to 5.3mm, with a mean of 4.28mm.

For $N = 800$, we see that now only user 3 performs worse with the GP than the N9. With added training data, predictive performance has improved for user 5. This suggests that 400 training points is not sufficient in all cases to reach optimal performance when using sensor data as input to the GP. For the GP, mean target sizes range from 2.93mm to 4.4mm, with an overall mean of 3.61mm. The range for the N9 is from 3.45mm to 5.21mm, and the overall mean across all users is 4.28mm. The bigger difference between the means in this

experiment again supports the argument that 400 touches was an insufficient training set size.

The GP achieves 95% accuracy with targets 0.67mm smaller in radius, or 1.34mm in total target size. This is small in absolute terms, but fairly large in the relative sense. Consider that the buttons on the N9 keyboard are only 4.8mm wide — an additional offset of 1.3mm can easily cause touches to miss their intended key.

To place these results in context, in Figure 4.4 we show more detailed results from subjects 1, 3 and 8 (representing a wide range of performances). These users were selected for close analysis as they represent a range of performances. Subject 1 is around the middle of the pack in terms of relative improvement (although they were the most accurate overall), subject 3 is the user for whom our model gives no improvement, and subject 8 is a user with one of the largest improvements.

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 4.4(b), we can see that both models perform very well for this subject — both the N9's 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. This may mean that the other users, who already had smartphone experience, had adapted their touch behaviour in some way based on this experience. If true, this would mean offset models allow novice users to approach expert performance quickly, which is a very attractive property. It is of course not possible to draw a strong conclusion from a sample of one, but investigating the touch behaviour of novice users is an interesting potential avenue for future work.

Training set size

The bottom row in Figure 4.4 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 [1], 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).

Across all users, this plateau in performance improvement is seen consistently. The largest performance gains are found up to 300 touches, and each 100 additional points after that adds a much smaller accuracy increase. We suggest that for a good GP offset model, on the order of 300 training points should be used.

4.4.2 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.

We predict an offset value rather than an absolute position. This approach is similar to the offsets computed in [1] 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.

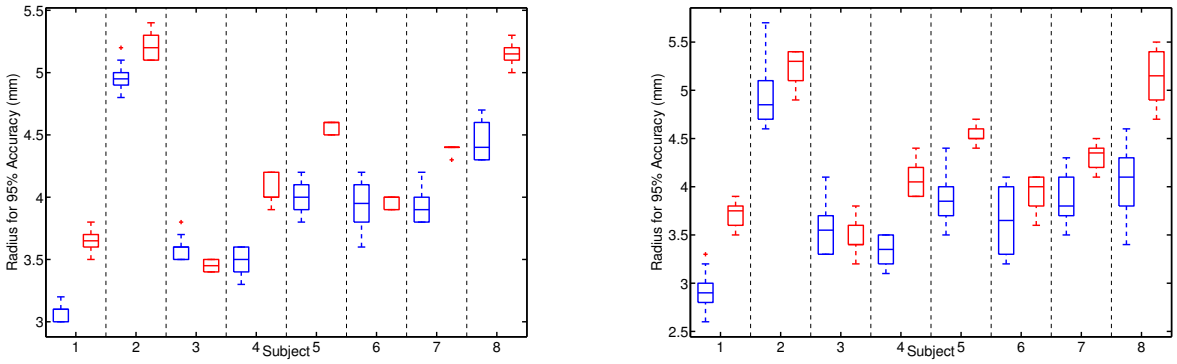
Hyperparameter selection

We once again optimise the GP covariance parameters by performing a cross-validation (in this case using data from user 5; randomly chosen). We use data from a single user to learn hyperparameters which are used for the whole population, following the same logic as above.

The optimal parameters were: $l = 100, b = 1, a = 0.5, \alpha = 0.3, \sigma^2 = 10^{-3}$. The value of $a = 0.5$ shows that in this case the linear and Gaussian terms in the covariance are afforded equal weight. This suggests the functions learned here are more linear than in the previous experiment, which intuitively makes sense — the space of possible device positions is much smaller than the space of possible sensor values, and we can assume that the manufacturer's algorithm will make a prediction close to the intended location, otherwise the phone would be unusable.

Touch accuracy

Figure 4.5 shows distributions of the target sizes required for 95% accuracy across 10 random training sets for both the GP and the N9's recorded touch points. Analysis was performed for two training set sizes, $N = 400$ and $N = 800$.



(a) $N = 400$. (b) $N = 800$.

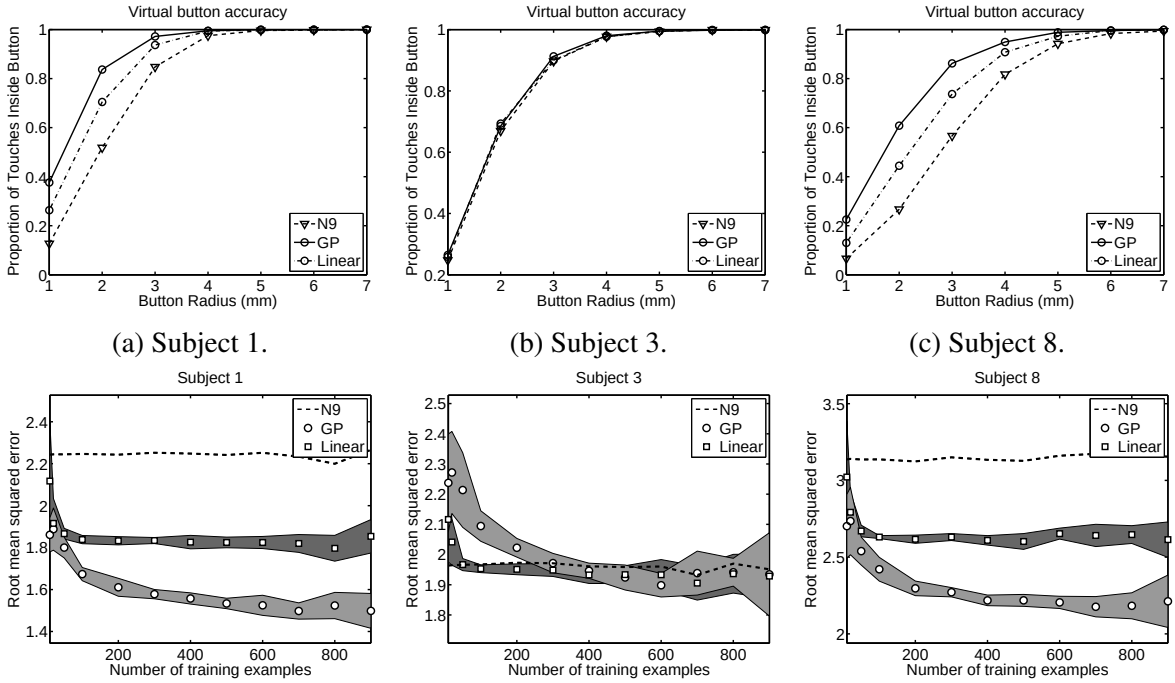
Figure 4.5: Target radii required to achieve 95% accuracy for a user's touches for 400 and 800 training points, when device position is used as input to the GP. In each case the blue boxplot shows the target sizes with the GP model, and the red plot the sizes for the uncorrected touches. Boxplots show the distribution of target size over 10 repetitions.

For $N = 400$, we see that for all users except number 3, the GP model performs better than the N9. Recall that when sensor data was used as input, user 5 also performed worse than the N9 with this training set size. This suggests that using device position as input gives better performance, assuming 400 training examples. The differences between N9 and GP distributions is significant for all users (paired t-test with Bonferroni corrections, $p < 0.05$). For the GP, the range in mean across users of target sizes for 95% accuracy is from 3.09mm to 4.96mm, with an overall mean of 3.90mm. For the N9, the range is 3.45mm to 5.22mm, with a mean of 4.31mm.

For $N = 800$, we see smaller target sizes required for 95% accuracy. As in the sensor input case, it seems that adding more training data still improves the predictive power of the GP after 400 points are used. For the GP, mean target sizes range from 2.92mm to 4.93mm, with an overall mean of 3.75mm. The range for the N9 is from 3.47mm to 5.23mm, and the overall mean across all users is 4.30mm.

Note that the mean for the GP is actually slightly higher in this case. The difference between the distribution over users in the sensor- and position-input cases is significant (paired t-test, $p < 0.05$). This indicates that for $N = 800$, slightly better performance can be obtained by using sensor input. This is an opposite result to the $N = 400$ case. These results suggest that position is better when little training data is available, but sensor data overtakes it in performance as training set size increases. This may be explained by the fact that the space of possible sensor readings is much bigger than that of device positions, meaning it takes more data to make good predictions. However, the larger space is more expressive, so the best possible predictions using this data are better.

The work in [30] 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,



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

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

Figure 4.6: Comparison of performance between the N9 touch events (N9), a simple linear regression model (Linear) (e.g. as in [1]) 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.

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 4.9mm, depending on the user. All but one user has a 95% accuracy radius of less than 4mm. For all subjects bar one, these sizes represents an improvement over the commercially available sensing technology of the N9.

In Figure 4.6 we show more detailed results for subjects 1, 3 and 8. In addition, we also give the performance of a simple linear regression model (see e.g. [86]) 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. Note also that the GP outperforms the linear regression, suggesting that the touch offset is not a linear function of the touch location. This is supported by the covariance function, where the non-linear term was assigned equal importance to the linear kernel.

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 4.4(d) and (f)). This can be explained by the GP

mean function, which is set to zero. When very little training 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 a very large amount from the N9 output. However, when predicting the location rather than the offset, the zero mean predicts the center of the screen, which is likely to be a poor prediction (unless of course the touch was targeting the center). Thus, with any data at all, the offset model can offer an improvement over the N9.

Visualising offsets

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, since the input space can be simply approximated by a grid of (x, y) values. The same is not true for the space of sensor values.

The offsets are visualised for subjects 1, 3 and 8 in Figures 4.7 and 4.8. We divide the input space into a grid, and use a GP trained on 600 of the user's touches to make offset predictions at each point in the grid.

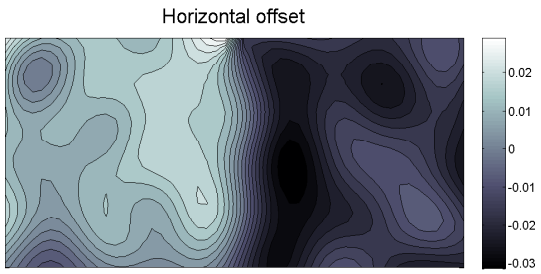
Figure 4.7 shows the continuous horizontal and vertical offset functions. White corresponds to a large positive value (to the right or up for horizontal or vertical offsets respectively) and black a large negative value (to the left or down).

Figure 4.8 shows an approximation of the combined x and y offsets. It shows 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.

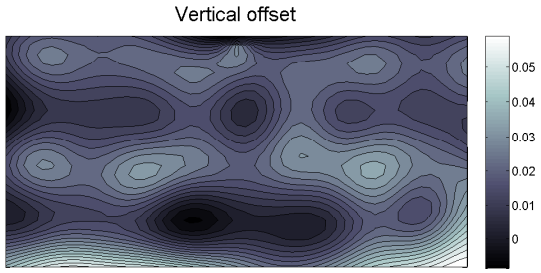
We see very small offsets for subject 3, reflecting the fact that the N9 position is already very close to the intended targets for this user. For users 1 and 8, it is immediately clear that the offset functions are highly non-linear. The horizontal offset functions have two reasonably flat areas separated by a steep ridge. Recall that the user is operating the system with two thumbs (see Figure 4.2) 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.

To show the relative limitations of the linear model, Figure 4.9 shows the linear offsets learned for subject 8. Comparing these with the bottom row in Figure 4.7, 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.

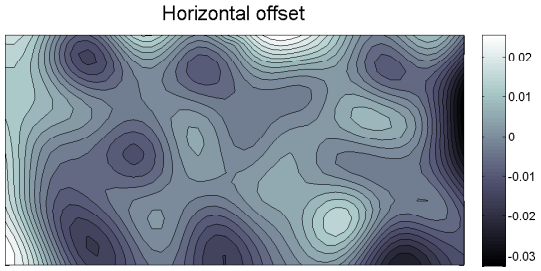
This nonlinearity is not immediately apparent when considering the touch problem, and this highlights a benefit of the GP modelling approach. We were able to determine these nonlin-



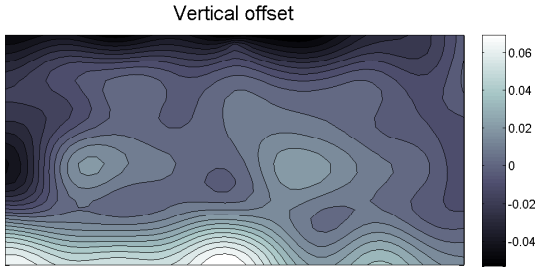
(a) Subject 1 horizontal offset function.



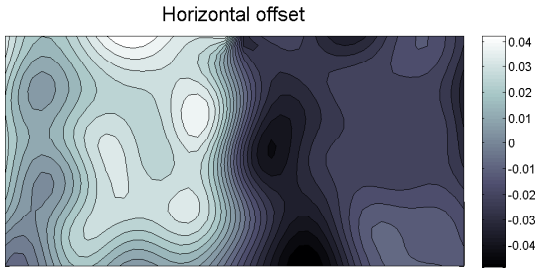
(b) Subject 1 vertical offset function.



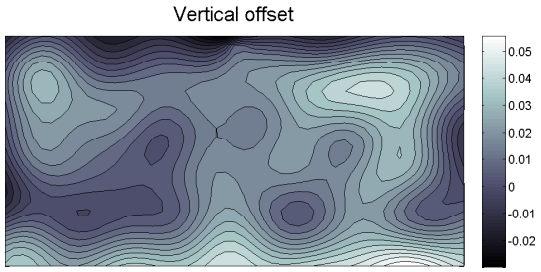
(c) Subject 3 horizontal offset function.



(d) Subject 3 vertical offset function.



(e) Subject 8 horizontal offset function.



(f) Subject 8 vertical offset function.

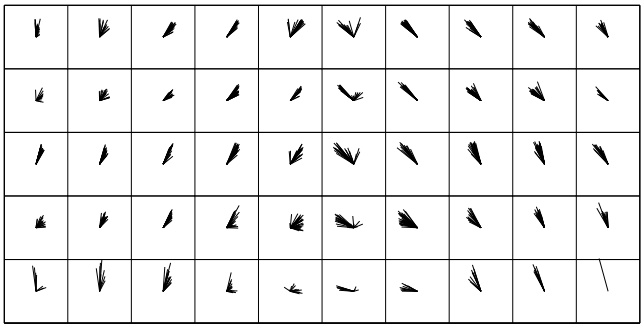
Figure 4.7: Continuous offset functions for three subjects for both x and y. The structure is highly nonlinear, and both components of the device coordinate have an effect on the offsets.

ear offsets automatically, without making prior assumptions about the form of the mapping beyond that it should be smooth.

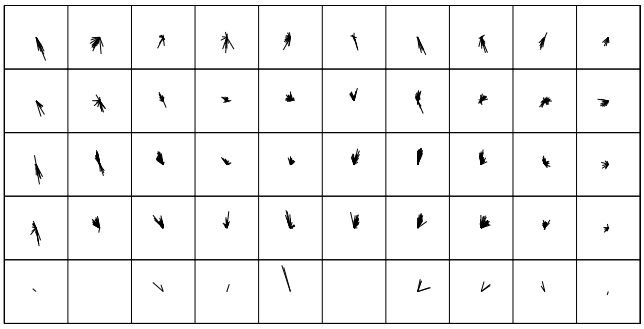
4.4.3 1D vs 2D Regression

As described previously, we have used a two dimensional model up to this point, which means that training our GP requires inversion of a $2N \times 2N$ matrix, where N is the size of the training set. This is quite computationally intensive, and could be a limiting factor if these models were to be implemented on a mobile device.

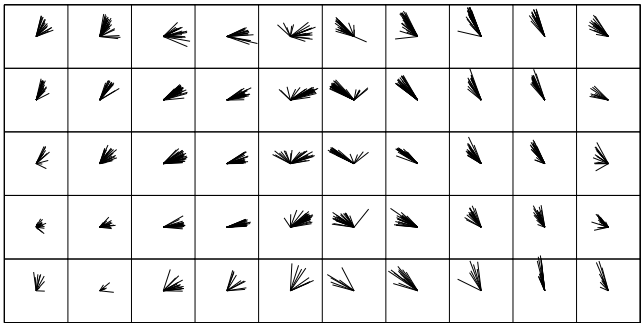
The alternative is to build separate models for the x- and y-offsets. This requires inverting two $N \times N$ matrices, which is computationally easier than the two dimensional case. The tradeoff is that we lose the ability to model covariances between the two components of the



(a) Subject 1.



(b) Subject 3.



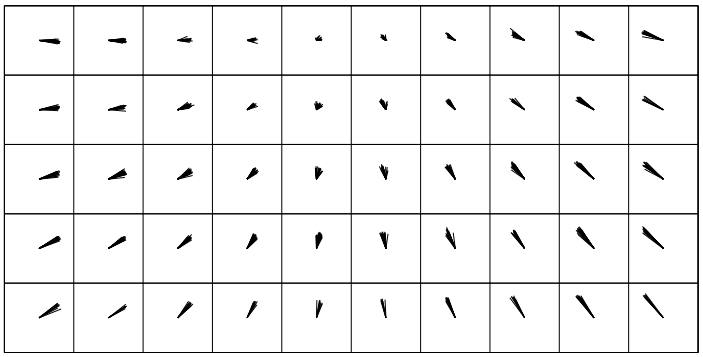
(c) Subject 8.

Figure 4.8: Offset values for subjects 1, 3 and 8. The input space is divided into a grid, and offset values for several points in each cell are drawn at the center of that cell.

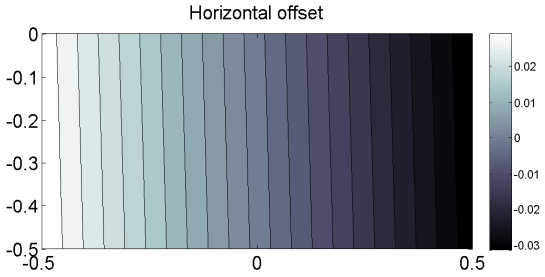
offset, which have been shown to exist [65].

However, in the analysis above we found that for predicting offsets, the covariance between the offset components was only 0.3 times that of the covariance within components. Additionally, in our analyses so far we are only using the mean prediction of the GP, which may be relatively unaffected by the transition from a 2D model to two 1D models. It may be possible to obtain a performance improvement over the N9 using the simpler model.

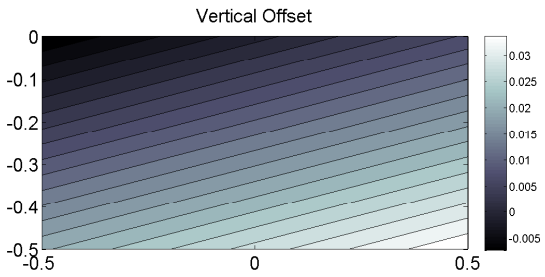
To assess this, we carried out another set of experiments training these one dimensional models on the data gathered in our study. We focus on the problem of predicting offsets given device positions as input. We predict the horizontal and vertical offsets separately, and compute the RMSE between the predicted position and the targets. We compare this to the RMSE for the predictions from the two dimensional model.



(a) Binned offsets.



(b) Horizontal offset function.



(c) Vertical offset function.

Figure 4.9: Offsets learned by linear regression for subject 8.

Figure 4.10 shows our results. For each user, two box plots are shown. The blue plot shows the improvement over the N9 for a two dimensional model, as in our previous analysis, while the red one shows the improvement for a pair of one dimensional models. We measure improvement in terms of the relative difference in selection accuracy for buttons of 2mm radius. An improvement of 0.1 means 10% more touches fell inside this radius for the model than for the N9 position. The boxplots show the distributions of improvement over 10 random restarts, in which we train on 600 of the user's touches and test on the remaining 400.

From this Figure, we can see that the models perform very similarly for all users. The distributions over the 10 restarts are only significantly different for users 1, 4 and 5 under the t-test at a significance level of 0.05. All other users have statistically similar results for both 2D and 1D offset models. Even the models which exhibit significant difference have a mean performance variation of only a few percentage points.

As we have said, this similarity can be attributed to the fact that the benefit of the two dimensional model is that it captures the full structure of the predictive covariance, allowing for predictive distributions which are not axis aligned. Here, however, we are only using the mean of the predictive distribution, and the one dimensional GPs are able to learn this mean offset to a similar degree of accuracy.

Effectively, the 2D model uses x, y and Δ_x to predict Δ_y, or x, y and Δ_y to predict Δ_x. The

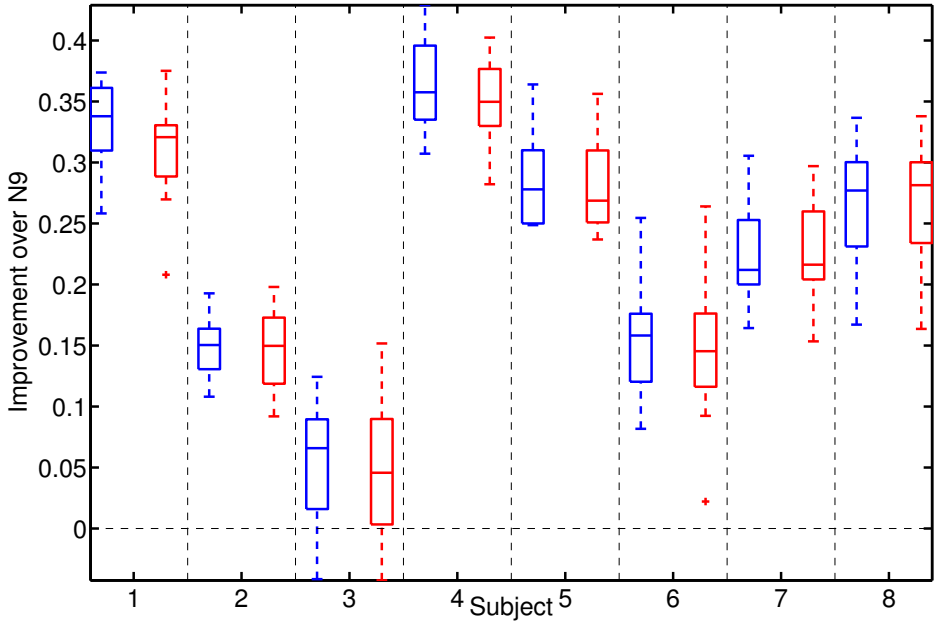


Figure 4.10: Performance comparison between a two dimensional GP model (blue boxplot for each subject) and a pair of one dimensional models (red boxplot). We see that the distributions for both models are very similar.

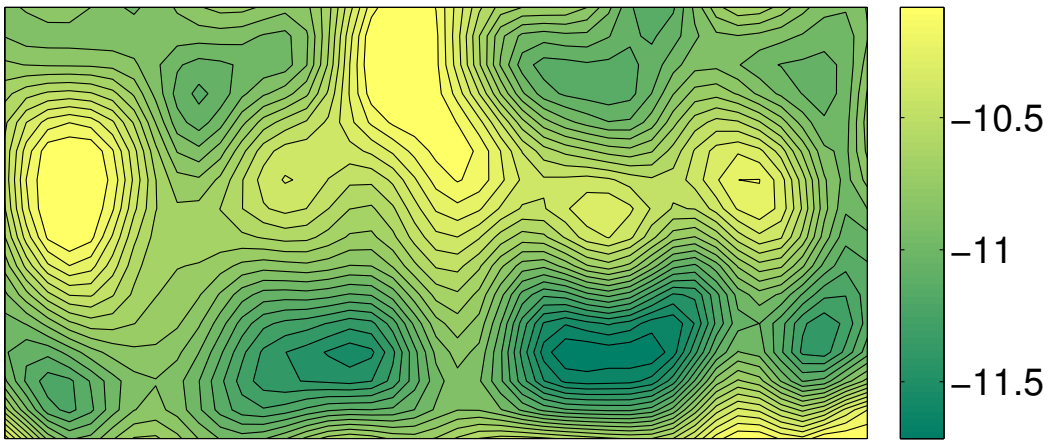
1D models use only x and y as inputs to predict a single offset. It appears that using one offset to help predict the other does not add signficant power to the model.

In general, we conclude that using a two dimensional GP may offer a small improvement in performance over two one dimensional models, but that this improvement can be so small in practical terms that it may not be worth the increased computational complexity. In later Chapters we will use both approaches when constructing offset models, and these results suggest the choice is relatively unimportant if selection accuracy is the only concern.

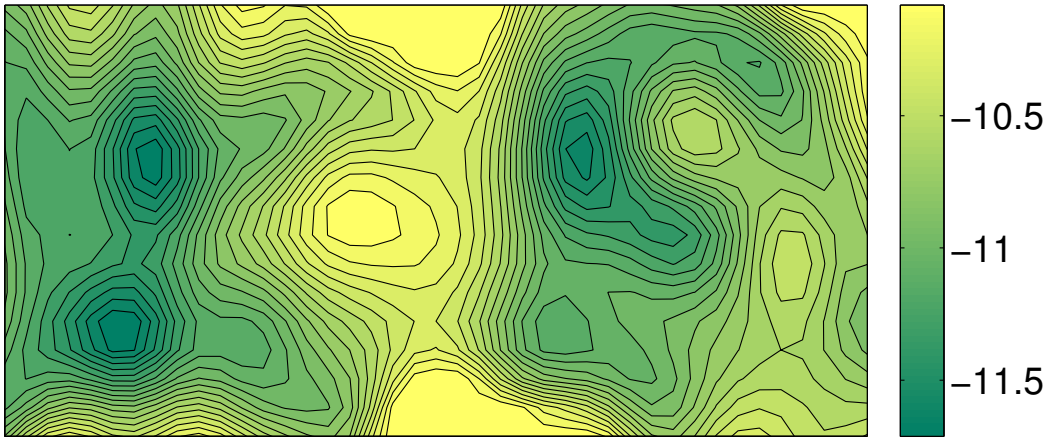
However, if the predictive distribution is to be used, the 2D model has the potential to be more powerful, since touch distributions have been shown to have off-diagonal covariance structure [65]. The 2D GP can predict these elements, while using 1D models restricts us to independent x and y covariance, with no off-diagonal structure.

4.4.4 Predictive Covariance

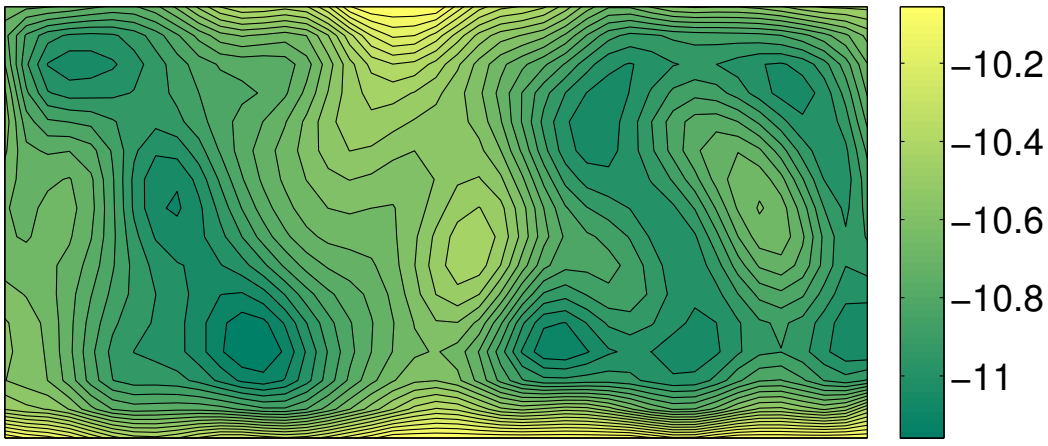
The Gaussian Process prediction consists of a Gaussian distribution. Thus far we have just used the mean of this Gaussian as a point estimate. Since we have used a single GP to infer the complete two dimensional problem, we obtain full covariance structures for each prediction. This means the predictive distribution does not need to be axis aligned. The covariance of this Gaussian is potentially useful as it describes the uncertainty in the prediction and could therefore be used at an application level to obtain additional information about the user's touches.



(a) Subject 1.



(b) Subject 8.



(c) Population Average.

Figure 4.11: Log determinant of the predictive covariance matrix for different parts of the space when the N9's touch location is used as an input. Higher values (more yellow) indicate the GP predictions are more uncertain. Panels show data for Subject 1, Subject 8 and an average for all Subjects.

In Figure 4.11 we visualise the log determinant of the predictive covariance matrix (a higher value of this quantity corresponds to a broader distribution with greater uncertainty) at different points on the screen when the N9 touch location is used as an input. Panel (a) and (b) show these visualisations for the models of Subject 1 and 8 specifically, while panel (c) shows the average across all users. Green regions show relatively low uncertainty regions, while yellow represents higher uncertainty.

The three plots exhibit a number of similarities. There are steep ridges at the edges, representing the rapidly increasing uncertainty limit of the recorded data. There are also areas of higher uncertainty down the center of the screen in each case. This 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. Finally, there are areas of increased uncertainty in the vertical center of the screen towards the horizontal edges. We hypothesise that these are the areas directly beneath the user's thumbs, and the extra movement required to touch these points causes the higher uncertainty observed. This is particularly pronounced for subject 1, who has such regions on each side of the screen. Subject 8 has such a region on the right of the screen, but not on the left. The population average displays vague indications of these center and side regions, but the effect is less pronounced as we might expect from an average. Other users display similar effects.

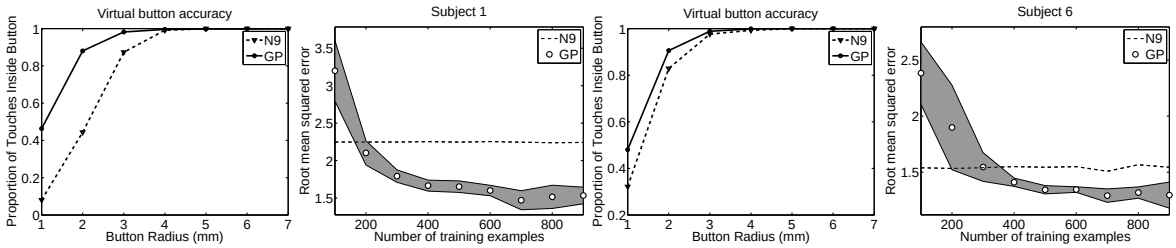
This information about the uncertainty is a further advantage over previous offset models, such as the polynomial function of [1], which can only give a point estimate of the offset. This information suggests some consistent variation in model uncertainty across the space for different users. This has potential implications for interaction design — it is best to avoid placing small or important targets in the regions of higher uncertainty, since it may be harder for users to acquire them reliably.

An example of an application which actually uses the predictive uncertainty is given in Chapter 5. We build a soft keyboard which uses the GP to compute the probabilities for each key given a touch from the user. These probabilities are used in combination with a statistical language model to correct errors in the text the user types.

4.4.5 Generalising Results

To ensure that the results described above were not specific to a particular device and orientation combination, we carried out a number of additional experiments. First, we had two subjects (1 and 6, chosen at random) 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 4.12 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 (4.12(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.



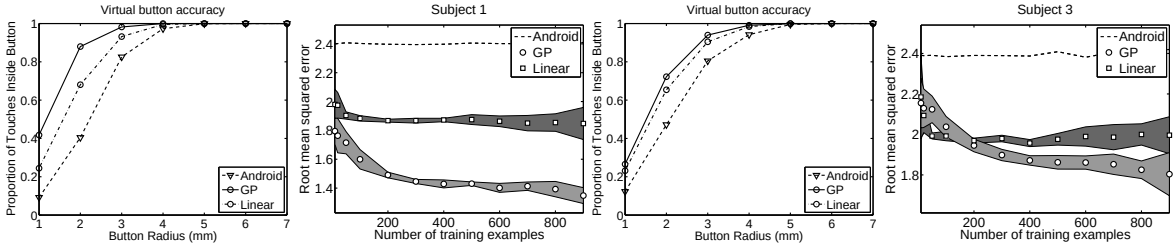
(a) Subject 1 (b) Subject 1 (c) Subject 6 (d) Subject 6

Figure 4.12: Virtual button accuracy and learning curves for portrait touches

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, again chosen randomly) 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 4.13. 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 compared to 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.

These results are included merely as a probe — it is difficult to draw detailed conclusions from a sample of two. However, there is no obvious reason why a different device or orientation should preclude the techniques used here from providing a performance benefit. Further, the work in Chapter 5 uses offset models learned in portrait orientation on an Android device, and performance gains are indeed observed. Chapter 7 also presents GP models trained on data from a range of devices. We hope the reader can be convinced that the GP regression framework is a powerful tool, which can be be used to model offsets on any device and for



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

Figure 4.13: Performance on Android phone

any usage scenario.

4.4.6 Importance of user-specificity

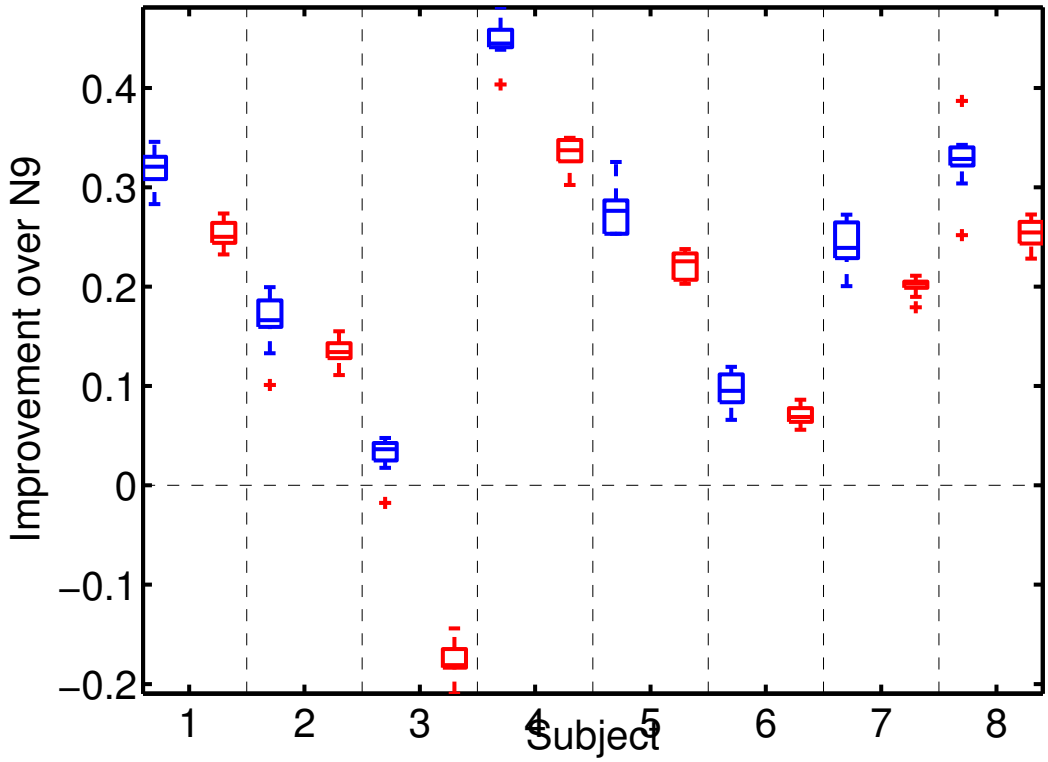


Figure 4.14: Improvements in accuracy rate for 2mm virtual buttons when using a user-specific model (blue boxplot in each case) and a model trained on all users (red boxplot).

In Figure 4.7 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 4.14 shows our results. For each subject, two boxplots are shown. The blue plot shows the improvement in accuracy for a 2mm button over the N9 when training on 600 of that subject's touches. The red plot shows

the performance improvement when the model is trained on 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. This reflects the findings of previous research, and represents an advantage of our model over population-based approaches found in the literature.

4.5 Conclusion

In this chapter, we have presented the use of Gaussian Process regression to model touch. This powerful statistical modelling tool allows to learn flexible mappings between inputs and intended touch positions without making parametric assumptions about the form of these mappings.

In particular, we looked at mapping capacitive sensor values from a Nokia N9 to intended touch positions, and mapping the N9's device position to a set of offset values. Over a set of 8 users, we found that the former mapping allowed a 23.47% reduction in errors compared to the N9 when selecting 2mm radius buttons, 14.20% for 3mm buttons and 4.70% for 4mm buttons. For the second mapping, the error reductions were 23.79% (2mm buttons), 14.79% (3mm) and 5.11% (4mm).

Our models allowed 95% selection accuracy for buttons of radii between 2.8mm and 4mm, depending on the user. The mean was 3.3mm, which is a smaller target size than the recommended minima in previous research. This indicates the GP model allows highly accurate touch input.

We also looked at the number of training points required to train these models. We found that using sensor values as input, we typically needed 300 training examples to be able to make good predictions. Additional performance gains could still be made as more data were added, but the relative improvement became smaller for each training set size above 300. When using device positions as input, we found that the performance gains became smaller after 200 training touches.

An important finding of this work are that the touch offsets observed are both nonlinear and highly user-specific. We found that learning a model on a user's own data resulted in better

predictive performance than a model trained on data pooled from many users. Indeed, for one user the pooled data model was actually worse than not correcting at all.

The GP is a probabilistic model, and allows us to model the uncertainty of a user's touch patterns. Each prediction from the GP is a Gaussian distribution, with a covariance structure that is learned for each user and varies across the phone's screen. However, in the analyses in this chapter, we have not made use of these probabilistic predictions. In the next chapter, we present an application of our GP approach to text entry, a ubiquitous task on mobile devices. We make use of the predictive distributions to compute the probability distribution over letters on the keyboard for each touch.

Chapter 5

Uncertain Text Entry

Summary. This chapter presents an application of our GP touch model which makes use of the predictive uncertainty. In particular, we introduce GPType, a soft keyboard which combines the GP with a statistical language model in order to correct erroneous input from the user. The predictive distribution of the GP is used to obtain probability distributions over the keys on the keyboard for each touch. We show that the system gives performance comparable to a leading commercial product, indicating the value of modelling uncertainty in this use case.

5.1 Introduction

In the previous chapter, the use of Gaussian Process regression for touch offset modelling was introduced. One of the distinguishing features of this approach compared to existing offset models is the fact that the predictions made are Gaussian distributions, with a learned variance structure for each user. Thus, for a given touch we can make claims about the likelihood that touch was intended for each element in an interface (see Figure 5.1 for an illustration of this). In the analysis so far, no use has been made of this attractive property. In this chapter, we present an application of the probabilistic touch model to the problem of typing on a soft keyboard.

Typing is a ubiquitous task on touch devices today. Despite this, the process is still highly error prone, as evidenced by the presence of corrective keyboards in all major smartphone operating systems. As previously discussed, this is due in part to the densely packed keys on the screen being smaller than the human finger, such that even a small touch offsets can cause the wrong key to be hit. We define these errors as substitution errors. Nicolau and Jorge [87] showed that as many as 7% of key presses can be incorrect when users type while walking. However, in text entry there are important additional sources of uncertainty compared to

regular touch. First, the user may be uncertain of the spelling of a particular word, leading to incorrect input. Additionally, users frequently miss out characters (deletetion errors) or add extra characters (insertion errors) when typing quickly. A touch model alone cannot account for such errors.

Smartphones have long had autocorrection systems to catch and correct text entry errors. Some such systems are simple, merely finding the word in a dictionary with the shortest edit distance to the entered text. More commonly in the research literature (and increasingly in commercial products) systems use statistical language modelling to improve their results. This process analyses the entered text and assigns probabilities to candidate words based on a generative statistical model of language, built from large corpora of text. Often, information about previously entered words can be used to change these probabilities, thus taking into account the grammatical structure of the language. This approach has been shown to be powerful: for example, Goodman et al. [65] showed that it was possible to reduce error rates by a factor of 1.67 to 1.87 using a language model to make corrections.

The primary contribution of this chapter is GPType, a text correction system which combines both a GP offset model and a long-span language model. The GP gives a predictive distribution which we can integrate over the keys on the screen to give a set of per-key probabilities for each touch, and the language model assigns probabilities to strings of text. The system incorporates a flexible decoder algorithm that searches for the most likely text given these two sets of probabilities. The decoder also allows for possible missing key presses, and possible extra key presses. The decoder was tested with a variety of correction strategies, such as whether the decoder is free to change previous characters.

Each of the constituent models in GPType is a powerful tool for the text entry task. We will present the results of a user study that shows the combination of the two models allows for additional performance gains, and that GPType gives performance comparable to or better than a leading commercial keyboard.

More broadly, GPType shows the benefits of modelling uncertainty in touch interaction. A conventional offset model such as that of Henze et al. [1] would not be able to generate the key press probabilities used here, which are user specific and vary depending on the learned uncertainty in different parts of the screen. Thus, this chapter is an illustrative example of the potential of probabilistic models when applied to problems in the HCI domain.

Statement of Original Work

Some of the work described in this Chapter was performed together with a number of collaborators. In particular, the language model and decoder algortithm we employ were created by Per Ola Kristensson and Keith Vertanen. The author adapted these models to work together with the GP, ported them to Android and carried out all user studies and subsequent analysis.

5.2 Language Models

Approaches to text entry on touchscreens were described in Chapter 2. An important subgroup of these approaches is those relying on language models, the first of which is described in [65]. At the most basic level, a language model is a statistical tool which assigns a probability to a sequence of characters based on how likely it is to appear in a document. These probabilities are normally learned from a text or speech corpus. Such corpora can be very large, containing millions of words from thousands of documents.

Language models have been extensively studied in the literature. The earliest statistical analysis of text was by Andrei Markov [88], who studied the probabilities of letter sequences in classic Russian literature using what have come to be known as Markov models. This analysis assumes the probability of a letter appearing is conditioned on the letters which immediately precede it. This is an intuitive model — for example, if 'th' has appeared already in a piece of English text, it is much more likely that the next character is 'i' or 'e' than say 'g'.

Another early use of this reasoning came from Claude Shannon [89], who studied both letter and word sequence probabilities using Markov analysis. He used this to illustrate the implications of coding and information theory. The term 'language model' was not coined until the 1980s, when this type of analysis was applied to automatic speech recognition [90].

Formally, a language model aims to assign a probability to a sequence W of m words, $p(w_1^m)$, where w_a^b denotes the sequence of words from position a to position b. In language modelling convention, 'word' is used to denote an arbitrary string — the tokens w are not necessarily words from a dictionary. A good language model should assign a high probability to genuine text from the language it was trained on, and a low probability to a garbled sentence. A popular performance measure is the cross entropy of the language model on the test sequence W, defined as

$$H(W) = \frac{1}{m} p(w_1^m).$$

This quantity can be thought of as the number of bits necessary to encode the test data, and so lower values of cross entropy are desirable. Perplexity, defined as $2^{H(W)}$, is also frequently used to measure performance.

As mentioned above, most language models make the Markov assumption that the probability of a word is conditioned on the previous words. That is, the quantity which we usually wish to know is $p(w_m|w_1^{m-1})$. The probability of a whole sequence is then a product of such probabilities.

As m becomes large, approximating this probability is intractable, so in practice the probabilities are approximated by conditioning on a subset of previous words. A commonly used

example is the *trigram* model:

$$p(w_m|w_1^{m-1}) \sim p(w_m|w_{m-2}^{m-1}),$$

which computes the probabilities based on the previous two words. More generally, a model which makes predictions based on the last n words is called an n-gram model. The trigram probability can be naively approximated by dividing the number of occurences of the sequence $w_{m-2}w_{m-1}w_m$ in the training corpus by the number of occurences of $w_{m-2}w_{m-1}$. This approximation generalises in an obvious way to other values of n.

However, this approximation is quite noisy and has an obvious point of failure — if an exact sequence of words never appears in the training set, it will have zero probability even if it is a reasonable piece of text. Therefore language models have employed a variety of 'smoothing' techniques to assign probabilities to unseen n-grams. The most basic technique is just to assign a count of 1 to all unseen sequences, but clearly this is not representative of language. More advanced techniques combine higher order n-gram models with lower order models to improve probability estimates. This has the effect of reducing the counts of sequences seen in the corpus and increasing the counts for unseen sequences, making the overall probability distribution more uniform — hence the name 'smoothing'.

Two main families of smoothing techniques exist: backoff and interpolated models. Backoff models work by using the n-gram model if the test sequence has a non-zero count in the corpus, and *backing off* to the $(n-1)$-gram model if it has a zero count. If necessary, this process can be carried on recursively, terminating with the 1-gram (commonly called a unigram) probability which expresses how common an individual word is in the corpus. Formally, backoff models set

$$p_{\text{smooth}}(w_m|w_{m-n+1}^{m-1}) = \begin{cases} \tau(w_m|w_{m-n+1}^{m-1}), & \text{if } C(w_{m-n+1}^{m}) > 0 \\ \gamma(w_{m-n+1}^{m-1})p_{\text{smooth}}(w_m|w_{m-n+2}^{m-1}), & \text{if } C(w_{m-n+1}^{m}) = 0. \end{cases}$$

The distribution $\tau(\dots)$ varies across the different techniques in the literature, and the scale factors $\gamma(w_{m-n+1}^{m-1})$ are chosen so that the conditional distributions sum to one. Examples of backoff models include Katz smoothing [91] and Kneser-Ney smoothing [92].

Interpolated models utilise information from lower order models for all probability calculations, not just those for n-grams with zero counts. As the name suggests, they interpolate between the n-th and $(n-1)$-th order models. The distributions are of the form

$$p_{\text{smooth}}(w_m|w_{m-n+1}^{m-1}) = \tau(w_m|w_{m-n+1}^{m-1}) + \gamma(w_{m-n+1}^{m-1})p_{\text{smooth}}(w_m|w_{m-n+2}^{m-1}).$$

Again, this definition is recursive with termination at the unigram level. Commonly used

interpolated models include Witten-Bell smoothing [93] and absolute discounting [94].

A thorough review and comparison of smoothing techniques is given by Chen and Goodman [95]. The authors also derive an extension of Kneser-Ney smoothing which they show outperforms a variety of other methods on training corpora of varying size.

Another important technique in language modelling, particularly for use in a mobile setting, is pruning. For large training sets, the number of n-grams can become high enough that efficient search over the possibilities is intractable. Further, the size of the language model in memory can cause issues. Thus, pruning techniques aim to reduce the size of the language model without significantly affecting performance.

The most basic pruning technique is the use of count cutoffs. Here, all n-grams of a certain length that have fewer than a given threshold of counts in the training set are 'ignored' in some algorithm-specific sense. In a backoff model, n-grams below these counts would typically be assigned probabilities by backing off to a lower order model, even though they might appear in small numbers in the training data. For example, a trigram model might have 0-1-1 count cutoffs, meaning that unigrams with 0 counts are ignored, and bigrams and trigrams with 1 or fewer counts are ignored. Not storing the probability information for these infrequently occuring n-grams can drastically reduce the size of the model, but not adversely affect performance [95].

Entropy pruning, first described by Stolcke [96], is another commonly used technique. This is based on the observation that the relative entropy caused by removing a single n-gram can be computed exactly for backoff models. The size of the model is reduced by removing n-grams which minimise the entropy between the full and pruned models. In experiments, it was shown that a model trained on a large dataset could be reduced in size by 74% without significantly affecting the perplexity.

It has been shown that in some cases pruning and smoothing can conflict. Chelba et al. [97] found that models smoothed using the Kneser-Ney method suffer from performance degradation when aggressive entropy pruning is applied. Models smoothed with other techniques, such as Katz smoothing, suffered much less degradation.

A range of other techniques to reduce cross entropy have been studied, though they are not used in the construction of the language model used in this Chapter. These include caching [98], clustering [99], sentence mixture models [100] and word skipping [101]. A review and comparison of these techniques can be found in the work of Goodman [102].

5.2.1 Language Models for Text Entry

All the major smartphone operating systems include soft keyboards which use language models. These are used in two ways. First, they can be used to complete the word a user is

typing after only a few key presses, or in some cases even based on only previously entered words. Secondly, they can be used to correct mistakes in a user's typing, resulting either from poor spelling or, more relevantly for this work, from problems introduced by the use of the touchscreen. This Chapter focuses on the design and evaluation of GPType, a system which does the latter task. The system combines the GP touch model introduced in the previous Chapter with a language model to improve text entry accuracy.

Other systems have combined touch and language models in the past — these were discussed in Section 2.4.3. In general, such a system aims to take as input a series of touches and infer the intended letter sequence. This can be thought of as finding the maximum over letter sequences of $p(\text{letter sequence}|\text{touch positions})$. GPType uses a novel search algorithm to find this maximum, incorporating probabilities from both the GP and the language model.

5.3 The GPType System

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

5.3.1 Language Model

We adapted a language model that has previously been used for a thumb-typing touchscreen keyboard [103]. The Twitter-based language model was trained based on 778M tweets sent between 12/2010 and 6/2012. Duplicate tweets, retweets, and non-English-language tweets were eliminated via a language-identification module [104] (with a confidence of 95%). Based on a tweet's source string we removed all tweets that did not originate from a mobile device. Tweets were split into sentences and we only kept sentences where all words existed in a list of 330K words drawn from Wiktionary, Webster's dictionary, the CMU pronouncing dictionary, and GNU aspell. The final dataset consisted of 94.6 M sentences, 626 M words, and 2.56 G characters.

The language model was built using the SRILM toolkit [105] (an open source tool created by language modelling researchers) using a vocabulary of A-Z, space, apostrophe, comma, period, exclamation point and question mark. The character-based 7-gram language model was smoothed using Witten-Bell and no count cutoffs. The model was then entropy-pruned to reduce the memory footprint in anticipation of using the model on a mobile device. The final model had 225 K n-grams and a compressed disk size of 2.0 MB.

5.3.2 Touch Model

As in Chapter 4 GP regression is used to model each user's touch offset function. We chose to implement GPType on an Android device, so that we could compare against SwiftKey, a leading keyboard which does language modelling of its own. Such keyboards are not available on the Nokia N9 we used for our previous study. As a result we have no access to capacitive sensor data on our target phone, and so we learn a model which maps the phone's reported touch locations $\mathbf{s} = (x, y)$ to offsets (Δ_x, Δ_y).

As in the previous study, we choose the GP mean function to be zero since in the absence of data we wish to predict no offset. Again, the covariance function

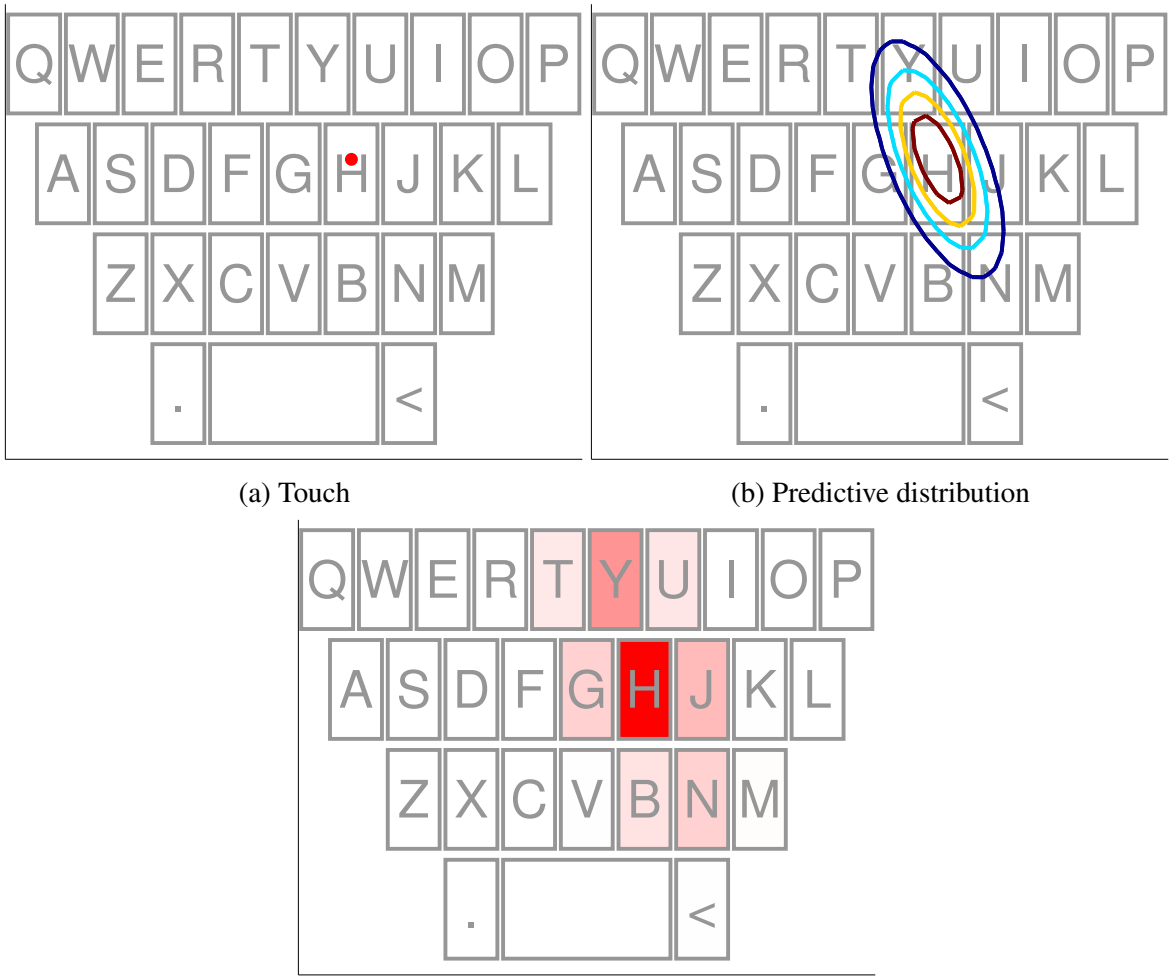
$$C(\mathbf{s}_n, \mathbf{s}_m) = a\mathbf{s}_n^T\mathbf{s}_m + (1-a)\exp\left\{-l||\mathbf{s}_n - \mathbf{s}_m||_2^2\right\},$$

is used. This was chosen because it gave best performance on data from the tapping study in Chapter 4 out of a range of covariance functions studied. As shown previously, this gives excellent predictive performance. We use the technique described in Section 4.2 to model the full 2D predictions task using a single GP.

From the predictive distribution for a touch, the probability of the touch being intended for a given key can be found by integrating the density function over the rectangular area of the key. In practice this is not possible analytically, so we approximate the probability by sampling from the Gaussian, counting which keys the samples fall into, and then normalising these counts to obtain probabilities. During model building, this process was evaluated using 100, 1000, and 10000 samples per touch. The performance difference between 1000 and 10000 samples was not significant, so for speed reasons we used 1000 samples in the on-device deployment of GPType.

Figure 5.1 illustrates the operation of our touch model. In panel (a), the user touches the 'H' key. From this, we make a prediction in the GP — panel (b) shows the predictive distribution. We use the distribution to obtain key press probabilities for each key on the keyboard. Panel (c) shows the keys shaded according to these probabilities, with red keys being more likely. 'H' is still the most likely key, but some probability mass is assigned to the neighbouring keys.

The covariances of the predictive Gaussians, and consequently the probabilities produced by the model, are learned from training data. Thus, in areas of the screen where the offsets are more variable, the Gaussians are larger and the probabilities more diffuse. This user-specificity is an advantage over other probabilistic approaches, such as that proposed by Bi and Zhai [64], which model the touch uncertainty with fixed parameters for all users. The tradeoff is that users must provide calibration touches before using the system.



(a) Touch

(b) Predictive distribution

(c) Key press probabilities

Figure 5.1: Cartoon illustrating the operation of our touch model. (a) shows the user's touch, (b) the predictive distribution over intended targets from the GP, and (c) the keys shaded by probability.

5.3.3 Decoder

We created a decoder which searches for the most probable character sequence given a sequence of touches. Each "touch" is in reality a discrete probability distribution over all of the keyboard characters. These distributions were set according to the GP touch model described above.

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

The decoder searches in the space of all possible character sequences for the non-fixed touches in an observed sequence. During this search, a touch would most likely generate

the actual key hit. But the decoder also allows substitution with all other possible keys according to the provided touch distribution.

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

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

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

5.4 Model Building

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

5.4.1 Participants

We built our models based on data collected from 10 participants (4 female) who volunteered for three 45 minute sessions. Each session was at least 24 hours after the end of the previous one. Participants ranged in age from 19–31 (mean = 25.4, sd = 4.25). All participants were smartphone owners, and rated themselves as intermediate to expert users of touchscreen devices. At the end of the data collection period, participants were paid £10 for their time.

5.4.2 GP Calibration

As in the previous Chapter, we need pairs of target-touch locations to train the GP and predict a user's touch offsets. In a pilot study, we initially collected this training data by asking the user to target a series of crosshairs, as in our previous work. Somewhat surprisingly, we found that the offset model learned for the pilot user on this data did not generalise to the offsets observed when typing on the same device. Thus, for the model building study proper we opted to collect two sets of calibration data to see if this finding was true in general or unique to our pilot user.

We built two data collection applications which ran on a *Samsung Galaxy SIII Mini* smartphone, with Android version 4.0. The first displayed a sequence of crosshairs, placed randomly in the region occupied by the phone's soft keyboard in portrait orientation. Users were asked to hold the phone in both hands and touch these targets as accurately as possible using both thumbs. The application logged the target location and the touch position recorded by the phone for each crosshair press. 300 touches were collected for each user.

The second application gathered data using a modified version of the Android keyboard. The keyboard had a simplified layout consisting only of the alphabetic keys, space bar, period and an enter key. Participants were asked to press a sequence of keys until 10 presses were logged for each key, for a total of 290 training touches. When a key was pressed, the keyboard logged the time and touch location, as well as the location of the center of the requested key.

In both cases, participants repeated the calibration process in three mobility conditions: sitting, standing, and walking. This was motivated by previous research, such as WalkType [33], which found that baseline text entry accuracy on a smartphone was significantly worse when walking than when sitting. We hypothesised that the offset behaviour might therefore be different between the conditions, and wanted condition-specific calibration data. We chose not to use a pace setter for the walking condition, as was used in the WalkType study, instead asking participants to move at whatever speed they were comfortable walking and typing. Participants walked in a set path, and their laps were timed so that walking speed could be determined.

We trained our GPs offline, rather than on device. Before parameters were learned, touches which were more than 3 key widths (or an equivalent distance in the crosshair application) from the intended target were filtered out, as these were deemed likely to be mistakes which would skew the learned offset model. Errors this large are more likely to be cognitive errors (misunderstanding the stimulus) or accidental touches. The covariance function parameters were optimised on a user specific basis using 10-fold cross validation to find the model which minimised the RMS error between the mean corrected touch locations and the centers of the crosshairs/target keys.

For the second application, note that the implicit assumption that users target the key center is without loss of generality — the GP can still learn a model, so long as the user always targets the same key the same way. This assumption is supported by the work of Holz and Baudisch [30], who have shown that users utilise visual features on the tops of their fingers to acquire targets on touchscreens, and that this targeting behaviour is consistent over time.

5.4.3 Typing Data

Each participant also provided typing data. We used phrases from the Enron Mobile Email dataset [106] as stimuli. This phrase set consists of phrases drawn from genuine emails and has been shown to result in similar text entry performance as the MacKenzie and Soukoreff [107] phrase set [108]. Additionally, since the emails were all written on mobile devices, this phrase set contains text representative of things a user of GPType might actually enter. Admittedly these emails were written in a business environment on Blackberries, so the relevance of the content may not be high for the typical mobile device user today. Nevertheless, the phrase set has been shown to be memorable and useful for text entry experiments.

We filtered out any phrases containing characters not on our keyboard, and then removed sentences with fewer than 4 or more than 10 words. This left a set of 427 phrases. Participants were shown a random subset of these and asked to type them as quickly and accurately as possible using our custom keyboard.

As there was no backspace key, participants had to leave any mistakes uncorrected. This was done because the goal was to evaluate the quality of our model's corrections on the text as entered, rather than the ability of users to accurately transcribe text using the keyboard. During each of the three sessions, participants typed for 10 minutes in each mobility condition. The order of mobility conditions was counterbalanced across both participant and session. As in the calibration task, participants were instructed to hold the phone in two hands and type with their thumbs.

5.4.4 Correction Strategies

The stream of touches for each typing session was passed through the best GP for that participant to obtain offset touch locations and key press probabilities. These probabilities were then passed through the decoder to obtain the corrected sentences. We evaluated four correction strategies:

S1 Single-Key Correction—for each touch, we compute the probability of each key according to both the GP and the LM. We hold all previously entered characters as fixed context when computing LM probabilities. The decoder takes the product of these

probabilities for each key and corrects to the key with the highest combined probability.

S2 Modifiable Context Correction—as S1, except that the decoder is also free to change n previously entered characters, where n is a value that can be configured. This includes characters in previous words, potentially up to the beginning of the entered text if n is sufficiently high. Characters outside the correctable window are held as fixed context for the LM.

S3 Word Correction—when the user types a word delimiter (space or period), the system uses the LM and GP to identify the most likely word, holding previously entered words as fixed context. This is essentially S2 with n dynamically chosen to be the number of characters since the previous word delimiter. Note the assumption that word delimiter characters are certain inputs — we assume the user does not touch these by accident. This is a potentially flawed assumption, but is ubiquitous on existing smartphones, which perform corrections at the end of a word.

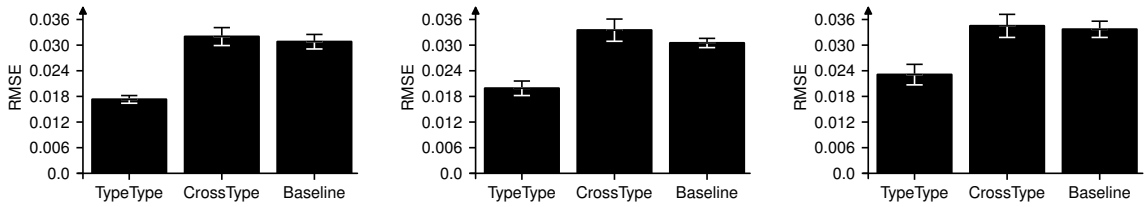
S4 Single-Key + Word Correction—combination of S1 & S3. We input the most likely character according to the combination of the models at each key press, and then correct entire words when space or period is hit.

5.4.5 Results

GP Calibration

To investigate whether the choice of training data — crosshair versus key targets — affected the quality of the GP predictions, we experimented with training on one dataset and testing on the other. That is, we built the GP covariance matrix using crosshair data and then made predictions for the intended locations of each touch from the key target data. This was compared to a model trained and tested using only the key data using 10-fold cross validation. As a first error metric, we considered the root mean square error (RMSE) between the predictions and key centers. We use the RMSE between the device's reported touch location and the key center as a baseline.

Figure 5.2 shows our results. Plots show mean and standard error across the 10 users. The bar labelled *TypeType* shows the results when training and testing on keyboard data, and similarly *CrossType* denotes the model trained on crosshair data and tested on keyboard data. As intended, training and testing on typing data leads to a reduction in RMSE over the baseline — we learn the offsets and the corrective models generalise to new data. However, training on crosshair data results in predictions on typing data that are actually *worse* than the baseline. That is, the predictions are further away on average from the key centers than the locations recorded by the phone.



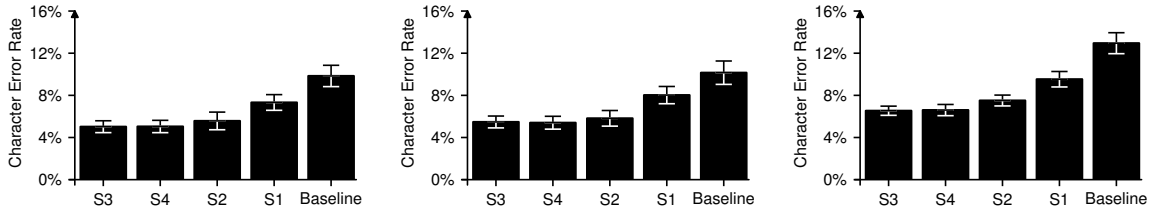
(a) Sitting (b) Standing (c) Walking

Figure 5.2: RMSE between predicted and intended locations for two models: one trained on crosshair data and tested on typing data (crossType), the other trained and tested on only the typing data (TypeType). The offsets learned from the crosshair data do not apply to the typing data. Baseline is the RMSE between the phone's recorded coordinates and the targets.

This result is somewhat troubling, since it suggests that in addition to factors like grip and orientation, the task also has an effect on offset behaviours. If every new task and hand posture needs a new offset model, the potential space of models rapidly becomes very large. Gathering data to train such models would likely become a burden on the user very quickly. A more thorough examination of this problem is given in Chapter 7 of this thesis. For the remainder of this Chapter, however, we will use only GP models trained on typing data for our analysis.

Correction Strategy

We next took the predictive distributions from our optimal GPs and used them to generate key press probability distributions for each letter on the keyboard at each touch. Sequences of these distributions were passed to our decoder algorithm, which applied the various correction strategies described above and compared the output to the stimuli shown to the users.



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

Figure 5.3: Character error rates after applying our four different correction strategies to the typing data gathered in our model building study, separated by mobility condition. Plots show mean and standard error across all participants. The baseline method represents the literal keys touched.

We measured performance in terms of the character error rate (CER) after correction, averaged across all phrases and all users for a given mobility condition. CER is the number of

characters that need to be inserted, substituted, or deleted in order to transform the corrected text into the reference text, divided by the number of characters in the reference text. For each correction strategy, we also tested a range of values for the cost and maximum number of insertions and deletions, the beam width of the decoder's search, and the relative scaling of the probabilities from the LM and GP.

Our results are summarised in Figure 5.3. These plots show the mean CER achieved for each correction strategy, along with the baseline error between the characters as typed and the stimuli.

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

The observed error rates were not significantly different between the sitting and standing mobility conditions, but the error rates for the walking condition were significantly higher (paired t-test, $p < 0.05$). The lowest error rates obtained using the optimal correction strategies were 5.02% for sitting, 5.47% for standing, and 6.5% for walking. The baselines for those conditions were 9.8%, 10.2% and 13.0%. Thus our simulated correction process was able to reduce error rates by almost half in all mobility conditions.

Model Contribution

When performing a search for the most likely correction given a stream of touches, our decoder algorithm assigns each potential candidate character at a given point in the search a probability which is expressed as a weighted combination of the outputs of the GP and language model. Given that the probabilities produced by the language model are typically very small, we work with log probabilities to avoid numerical issues. In particular we take the log probability of character c as

$$\log P(c) = \lambda \log P_{\text{GP}}(c) + (1 - \lambda) \log P_{\text{LM}}(c).$$

The weight parameter λ was learned by cross validation for each user. We used values in 0.05 increments from 0.05 to 0.95. For all users, the value of λ which gave the best performance for correction styles S3 and S4 was in the range $[0.35, 0.5]$ with a mean value of 0.41. This indicates that a slightly higher weight was assigned to probabilities from the language model

compared to those from the GP.

5.5 Evaluating GPType

With the information obtained from the model building study, we evaluated the best correction system in a second study. The goal of this study was to determine whether the benefits shown in our offline simulations were reflected in a live typing task. We also wanted to assess how our system compared to an existing commercial soft keyboard. For this we chose SwiftKey[1], a popular Android keyboard which also uses language modelling to perform intelligent correction. The details of the SwiftKey algorithm are not public, so in this study it is used as a black-box.

We implemented the GP in Java, and ported the language model and decoder to Android. The custom keyboard from the model building study was adapted to compute key press probabilities and make corrections using the GPType algorithm. S3 was chosen as the correction style, since it was as good at correcting as S4 and required fewer LM searches. This means an overall improvement in the speed of operation and usability of the keyboard.

5.5.1 Participants

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

5.5.2 Apparatus

Participants typed using the custom logging application, again running on a Samsung Galaxy SII with Android 4.0. This smartphone has a 4.3 inch screen with a 480×800 pixel resolution. A screenshot is shown in Figure 5.4. The stimulus phrase appears at the top of the screen, and the text entered by the participant is shown below. The time remaining in the current typing task is shown in the upper right. Visually, this keyboard was identical to the one used in the model building study. The backspace key in the logging app was again disabled, as the goal was to to evaluate the quality of the corrections made by GPType system without participants manually backspacing to correct errors.

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

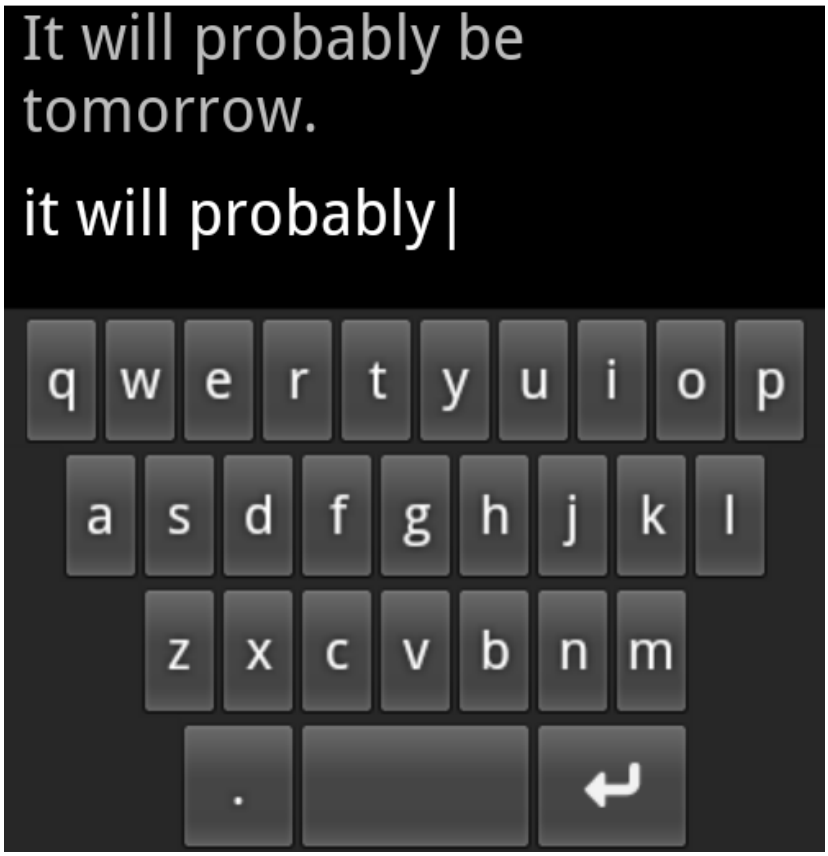


Figure 5.4: A screenshot of our logging application. The target phrase, the currently entered text and our simplified keyboard are shown.

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

5.5.3 Procedure

The procedure for the study consisted of three sessions. In the first session, participants provided keyboard calibration data for the GP in each of the three mobility conditions — Sitting, Standing, and Walking. This calibration was done in the same way as in the model building study, with the exception that only 5 examples of the user's touch per key were collected. This was different from the 10 used in the offline model building simulations, in order to prevent input from slowing down due to large matrix operations when making GP predictions. This was followed by a short break while the GP was trained — the necessary matrix inversion was somewhat slow on the limited processing power of the mobile

device. Afterwards, participants typed for five minutes, while seated, on each of the two study keyboards to familiarise themselves with the layouts. No data was recorded for these familiarisation sessions — they were simply to make sure the participants had experience of both keyboard layouts. As in model building, participants typed phrases from the Enron Mobile Email set [106].

In each of the two remaining sessions, participants performed 10 minutes of typing in each mobility condition. They used GPType in one session, and SwiftKey in the other. All tasks were carried out in a meeting room, with a clear and unobstructed path marked for walking. As before, no pace-setter was used in the walking condition, and participants were free to walk at whatever speed they felt comfortable while typing.

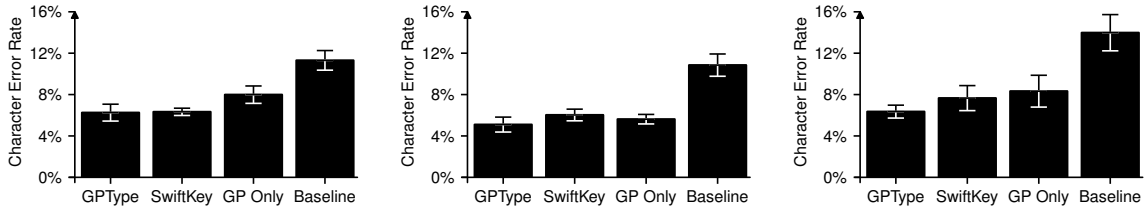
5.5.4 Design

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

5.5.5 Results

We measure performance of the corrections produced by each keyboard in terms of CER between the corrected text and the stimulus phrase. Our results are summarised in Figure 5.5, which shows the mean and standard error across all participants, separated by mobility condition. The baseline is the CER between the keys hit by the participant's touches and the stimulus phrase. Also shown is a *GP Only* condition, in which we apply the mean offset from the GP to the user's touch and see which key was hit by the resulting touch. The GP Only condition does not use the decoder or LM.

Both keyboards offer a significant improvement over the baseline in all mobility conditions (paired t-test, $p < 0.05$). The CER reduction over the baseline for GPType was 4.9% for sitting, 5% for standing, and 7.6% for walking. Further, GPType offers a small but significant improvement over SwiftKey in the standing and walking mobility conditions (approximately 1% reduced CER for the standing condition and 1.3% for walking). No significant difference between the keyboards was observed in the sitting condition. This is likely an effect of the participants' touch offsets becoming more pronounced when standing or moving. Interestingly, the standing condition had the lowest CER for both baseline and the evaluated keyboards. It is unclear why this should be the case.



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

Figure 5.5: Character error rates for the two keyboards evaluated, separated by mobility condition. Plots show mean and standard error across all participants. The baseline method represents the literal keys touched, while GP Only shows the keys hit after the mean GP offset is applied.

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

Effect of User-Specific Variance

We have argued in this thesis that modelling the uncertainty of touch interaction adds value. However, it is unclear from the results described above whether the user specific variance learned from the GP is useful. It may be the case that a model with key press probabilities from a fixed variance Gaussian would perform equally well. This could be useful as it might allow the creation of a system with the advantages of GPType without the need for a potentially lengthy training phase for each new user.

To investigate this, we carried out some offline simulations on the data gathered from the study. For each touch from a user, we recalculated all the key press probabilities by placing a Gaussian distribution with a fixed variance on top of their offset touch location. These probabilities were then passed through the decoder offline to generate corrections, which were compared to the original stimuli to obtain CER values.

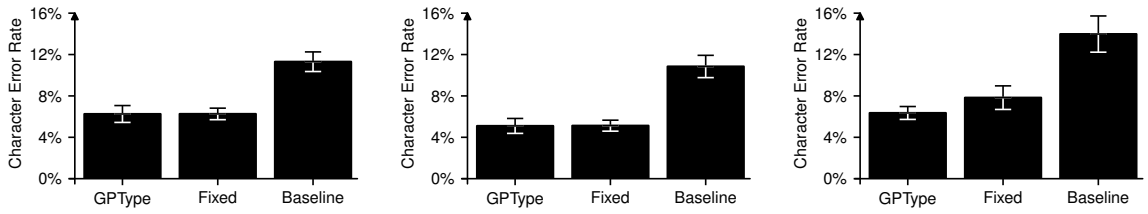
We used the offset touch location in order to separate effects from modelling the offset and the variance. In practice of course, these offsets are optimised on a user specific basis so the results of the simulation will be somewhat optimistic in comparison to what could be obtained on a real device without training.

We used Gaussian distributions aligned to the axes of the keyboard, with covariance matrices of the form

$$\begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}.$$

In contrast, the GP predictions can potentially have non-zero elements off the diagonal of the covariance matrix, resulting in distributions which are not axis aligned. However, in the absence of the knowledge gained from using the GP, it is unclear how the distributions might look so we believe axis alignment is a reasonable assumption to make. We repeated the simulation process for a range of values of σ_x and σ_y, and found the values which gave the largest average reduction in CER between the baseline and the predictions across all users.

Figure 5.6 summarises the results of this process. Shown are the lowest character error rates for GPType, the best performing fixed variance model for each mobility condition, and the uncorrected baseline, averaged across all users. Error bars represent standard error.



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

Figure 5.6: Character error rates for GPType and a related model using fixed rather than user specific variances. Plots show mean and standard error across all participants. The baseline method represents the literal keys touched. GPType provides lower error rates in the Walking condition.

We found no significant difference between GPType and the fixed variance model in the Sitting and Standing mobility conditions. For the Walking condition, we found that GPType offered a 1.5% lower average CER than the fixed variance model. This, along with the earlier observation that this condition had the highest baseline error rate, suggests that there is a more significant noise effect when walking and typing than when sitting or standing.

Further, this increase in noise is not uniform — otherwise a scaling of σ_x and σ_y would be able to account for the effect, which is not what these results indicate. There are a number of possible factors which might cause this noise increase. First, the variance across users might become more pronounced when walking, so that a single set of parameters in the fixed variance model does not adequately capture the variation. Alternatively, the level of variance in different areas of the screen might be more extreme in the walking condition, so that the fixed variance model does reflect the differences across the space.

To evaluate the effect of the first of these factors, we look at the learned hyperparameters for our participants. Figure 5.7 shows the mean and standard deviation across all users of the noise parameter σ in the GP covariance function. The values of σ are not significantly different between Sitting and Standing, but the noise parameter is indeed higher on average for the Walking condition than for the other two. Further, the standard deviation is around

50% higher in this condition, indicating more pronounced variations between users as they type. This indicates why the fixed variance models performs worse than GPType for walking users.

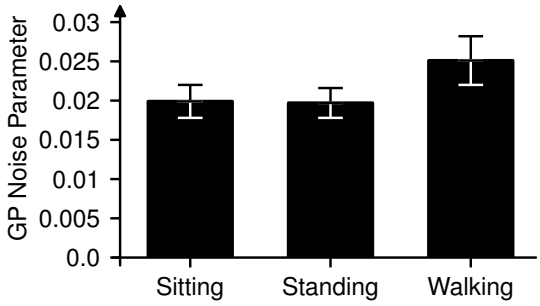


Figure 5.7: Mean and standard deviation across all users of the noise parameter σ learned by the GP. The noise level is higher in the Walking condition than in either of the other two. The standard deviation is also higher for the Walking condition, but is still small relative to the mean.

This result shows that taking into account the variability of individual users is useful when modelling touch. In particular, it provides a measurable benefit for users typing as they walk. Given that smartphones are often used in settings outside the home or office when users are on the move, this is an attractive property of the system.

Entry Rate

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

Further, we found no significant variation in entry rate between the mobility conditions. Across all participants, the mean entry rate for both keyboards was approximately 30 words per minute, where a word is taken as any five character string. One might expect a decrease in entry rate when walking as the user's attention is divided, but this was not the case in our study. This can potentially be explained by the lack of a pace setter — users were free to choose a walking speed at which they felt comfortable typing, and it seems they chose speeds which did not affect their text entry rates.

5.6 Conclusions

This Chapter has presented the design, implementation and evaluation of GPType, a text correction system which exploits the probabilistic nature of our GP offset model to improve text entry accuracy. GPType is a novel combination of an uncertain touch model and a long-span language model. In our user study, we showed that the system could reduce absolute character error rates by 4.9% while sitting, 5% while standing, and 7.6% while walking. The increased reduction seen for walking users highlights the ability of our model to account for the greater inherent noise present while walking and typing.

GPType's performance in the study was as good or slightly better than SwiftKey, a leading commercial keyboard for Android. However, it is worth noting that SwiftKey performs longitudinal adaptation to a user's typing patterns, learning their commonly used words and phrases over time. Since our participants used it for only a single 45 minute session, SwiftKey was not given a chance to do this adaptation. In an extended study, it may be the case that SwiftKey eventually outperforms GPType, but nevertheless we contend that our system is an effective solution, offering commensurate performance to the state of the art.

An important auxiliary finding of the development of GPType was that an offset model trained by asking users to touch a series of crosshairs did not work well when predicting a user's intentions in touching a soft keyboard. This suggests that *task* is a relevant feature to consider when thinking about touch offsets, in addition to physical considerations such as hand posture or angle of the device with respect to the user. Does every task require a different offset model? If so, collecting training data may become an issue. Our users spent approximately five minutes calibrating in each of three mobility conditions, and for some users that might be an unreasonable requirement for each new interface they use. The effect of task on offset model performance, and an alternative offset model which can be trained much quicker than the GP, will be discussed in Chapter 7 of this work.

More than just being a text entry system, GPType acts as an example of a model which models the uncertainty in an interaction in order to improve that interaction. However, this model is hidden from the user. In the next Chapter, we explore the use of a system in which the uncertainty model is *explicit* — the user can directly convey their certainty about inputs to the system. We use this model to answer the question: what happens when autocorrect fails?

Chapter 6

Explicit Uncertainty Control

Summary. This chapter introduces an alternative approach to uncertainty modelling, in which the user can explicitly control the level of uncertainty they express to the system with their touches. Again, we focus on a text entry use case, which we motivate by observing that there are cases where an autocorrect system changes input against the user's intention. We show that users have a good mental model of how autocorrect works, and introduce ForceType, a pressure sensitive keyboard which allows users to change the probability distribution over keys. By pressing hard on a key, they make the touched letter more certain and prevent unwanted autocorrection. We show that this gives a performance improvement when entering out of vocabulary words.

6.1 Introduction

The GPType system presented in Chapter 5 is a flexible autocorrect system which combines information from multiple probabilistic sources in order to improve text entry accuracy. It serves as an example of the value of a system which models the uncertainty in a user's input, and propagates that uncertainty until a prediction needs to be made. Predictions in GPType are corrections at the end of words, and the uncertainty information for each key press in the characters of the word is used to make these predictions. Clearly this approach has merit, since it leads to performance slightly better than the state-of-the-art. However, like all existing autocorrect systems its operation is opaque to the user — the uncertainty model is hidden.

In this Chapter, we propose another model of uncertain touch interaction, in which the user can *explicitly* control the level of uncertainty. Again, this is implemented in the form of a text entry solution, since that task is ubiquitous on mobile devices and error prone. While autocorrection can account for many errors, there are situations where it can produce unwanted

results. For example, proper nouns, abbreviations, slang, and words from local dialects are all examples of words the user might want to type that are not in dictionaries or language models used by autocorrect systems. These words might be corrected by the device when the user doesn't want this.

The method presented here allows users to prevent these by empowering the user to communicate *certainty* about their inputs to the system. In particular, we present ForceType, a keyboard which allows autocorrection to be dynamically turned on and off using touch pressure. By typing harder on a word they do not wish to be changed, the user can prevent unwanted corrections. This model of explicit uncertainty control complements the GPType model from the last Chapter, since users need only engage it when they want to prevent a correction. For regular typing without unusual words, GPType or another standard autocorrecting keyboard is sufficient.

Statement of Original Work

The work in this chapter was performed in collaboration with Henning Pohl, a PhD student at the University of Hannover. The work was done during his internship at Glasgow and subsequently during a visit by the author to Hannover. Henning developed the hardware side of ForceType and created the logging software. The author handled the interface with the language model and performed the subsequent offline analyses of the data.

6.2 The Autocorrect Trap

Current generation phones employ a range of autocorrection techniques (e.g., adaptive but clamped target resizing [72]). These techniques, as well as GPType, implicitly model uncertainty: the user has little influence on the way autocorrection works. In any such system, users might be able to reject a proposed correction, delete a character and retype with autocorrection switched off, or select the corrected word and pick from the originally typed version or other suggestions. These implementations can themselves introduce potential for error. On iOS, for example, corrections are rejected by tapping a small cross in a popup bubble over the word. This target is even smaller than the keyboard keys, and reliably acquiring it can be challenging. Nevertheless, these measures theoretically allows users to control text correction according to their needs, but there are still widespread frustrations[1] and situations where text prediction falls short. When typing proper nouns or other words not in standard dictionaries, using regional dialects, or mixing languages, autocorrection is often not flexible enough to adapt.

[1] `http://www.damnyouautocorrect.com/`

We thus propose giving users control over how their phones correct text. Empowering users to vary the level of text correction per word allows them to fall back to text correction for phrases they deem correctable while being able to 'tighten the reins' during phrases they feel their phones cannot handle.

Before developing such a system, the question of whether this is a useful and usable system or not must be considered. To successfully use this concept, users must have a good mental model of the way autocorrect works, and the sorts of words and phrases that will be corrected. To get an idea of how the capabilities of autocorrection are viewed by users, we conducted an online survey.

We designed a questionnaire listing 20 phrases of varying levels of difficulty—some consisting only of common English words and others with proper nouns, slang, and/or words or phrases borrowed from other languages (e.g., *summa cum laude*). Participants were recruited using mailing lists and social media. For each phrase, participants were asked to rate on a 5-point Likert scale whether they thought autocorrect would change it or not when entered on a smartphone, based on their own experiences with autocorrect. A rating of 1 indicated that participants were certain a phrase would be corrected, and a rating of 5 indicated they were certain it would not.

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

To establish a ground truth of whether the phrases would be autocorrected or not, we entered them into three smartphones—an iPhone 5 running iOS 6.1, an LG E700 running Windows Phone 7.5, and a Samsung Galaxy S3 Mini using the SwiftKey keyboard on Android 4.1. Care was taken to ensure that the phrases were typed without transposition errors, so that all autocorrections were the result of the phones not recognising one of the entered words rather than spelling errors. We noted for each phone which phrases were autocorrected. In cases of disagreement, where some phones corrected and others did not, majority voting was used to produce a singe binary value for each phrase. Such disagreement occured on 5 of the 20 phrases, with Windows Phone and iOS disagreeing with the majority for 2 phrases each and Android for one phrase.

We next split the user ratings for corrected and uncorrected phrases and produced histograms showing the counts for each rating. By normalising these histograms we obtain discrete probability distributions over the ratings for the two groups of phrases. Our results are shown in Figure 6.1. For phrases that would actually be autocorrected, the most probable rating is 1 (certain correction) and the least probable is 5 (no correction). The monotonic decrease across the other ratings shows a clear trend, indicating our respondents had a good sense of

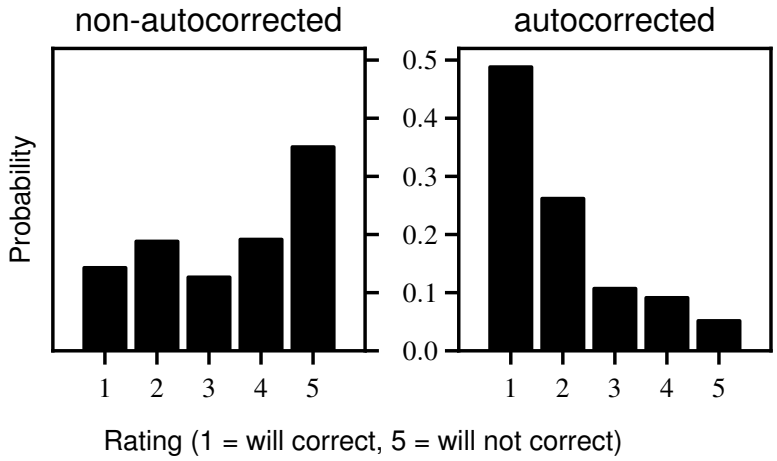


Figure 6.1: For a set of 20 phrases, we asked 28 people whether they thought their phone's autocorrect would (a) change it when entered, or (b) leave it unchanged. Participants gave a rating between 1 (will definitely be changed) to 5 (will definitely not be changed). We classified each phrase by testing it for autocorrection on several different phones. This figure shows the probability of each point on the rating scale for both classes (changed and unchanged). Note how users' understanding of when autocorrect would be active aligns well with when autocorrect actually was active (right side). Understanding of when it would not be active is less pronounced but follows a similar trend.

phrases that would be autocorrected.

For phrases that were not corrected, the trend is less pronounced. The most likely rating is for no correction and there is a remarkably high probability that users rate these uncorrectable sentences as certain or near certain corrections. In essence, they overestimate how likely autocorrect systems are to take action. This is not necessarily an issue—if a user takes action to prevent autocorrection when none would have occurred, there is no difference to their final input.

The distributions over ratings are significantly different (Wilcoxon test, $p \ll 0.01$) for the two groups of phrases. This suggests that users have a functional mental model of the way autocorrect operates.

The results suggest that users could make use of a system that affords finer control over text correction behaviour. Respondents were able to identify when a phone would make an unwanted correction. As an example, the phrase containing *summa cum laude* was corrected on two out of the three phones tested, and respondents correctly identified this as a phrase which would be corrected. However, this Latin term is in reasonably common usage in universities which award Latin honours and these autocorrections are unwanted. This pattern was repeated for other phrases, indicating users are able to identify situations where preventing autocorrection is desirable. This motivates the work presented in the remainder of this Chapter.

6.3 ForceType: Pressure as Certainty

Given that users already have a mental model of which words are likely to be autocorrected, we desire an input modality which enables them to express certainty about words they do not want corrected. In essence, we wish to allow users to negotiate control between themselves and the decoder. For normal typing, the autocorrect functions as it does on current phones, but when the user suspects the word they are entering is unknown to the phone's dictionary they can smoothly limit the autocorrect behaviour.

We choose pressure as the factor to control this negotiation. When the user wishes to indicate certainty, they simply press harder on the keys. It has been shown that users can reliably select between different targets by varying their input pressure [109], and the idea of pressing hard for a different input behaviour is simple to grasp. We are interested in an increase in pressure after the user has acquired a key by touch, rather than a momentary high pressure contact from a high impact velocity. Controlling the latter was shown to be more difficult for users.

Other research has considered negotiated uncertainty through user input. In [62], the authors use the height of the finger above the screen as a proxy for uncertainty when browsing a map — as the height increases, the system increasingly attracts the view to nearby points of interest. Fine control is restored when the user brings the finger back to the screen. Although 'hover' input of this form is now being introduced on commercial phones such as the *Samsung Galaxy S4*, it is not an appropriate choice for a typing application. [110] shows users display unintentional drift when varying hover height, so controlling the uncertainty via this modality might be suboptimal given that text entry requires some degree of accuracy - hence our choice of the pressure modality.

Another potential input modality we considered was the use of time dependent key presses. A user could press and hold on a key to increase their certainty about that input instead of pressing harder. This has the advantage that it could be implemented on a regular smartphone without significant difficulty. However, it adds time to typing over and above any time penalty added by the need for the user to recognise that a correction might need to be prevented, which we consider an undesirable property. Further, on current smartphones pressing and holding on certain keys is used to select accented versions (e.g. 'é' instead of 'e'). Overloading the semantic meaning of this input modality might be an irritant for some users.

As a minor nomenclature note, we observe that 'pressure' in this context is not technically a correct usage. Pressure is a measure of force per unit area, whereas HCI systems that use 'pressure' as a modality do not take area into account and thus are technically force-sensitive, not pressure-sensitive. Despite this, we use the pressure terminology here for consistency

with the literature. We do however name our system ForceType, both as an acknowledgement of this observation and to avoid any confusion with PressureText, a previously published pressure sensitive text system.

6.3.1 Pressure and Touch

Pressure has been used as an input modality in a wide range of interactive systems. *Pressure Widgets* by Ramos et al. [111] was an early system that looked at users' ability to control touch pressure in a pen based interface. The hardware used had a space of 1024 possible pressure values, and the authors divided this space into 'levels' of equal size. Users in their study had to select from between 4 and 12 pressure levels using one of four selection techniques. They found that only 6 levels could reliably be distinguished by users — any finer subdivision of the pressure space resulted in error rates of 10% or more. They also found that lifting off once the desired pressure level was reached allowed faster selections than 'dwelling' on a target for a given time without significantly increasing the error rate. Finally, they noted the importance of visual feedback when selecting between pressure levels — participants were unable to reliably make selections eyes-free after an hour of training, but with feedback achieved good performance very rapidly. This is not necessarily an issue for a system like ForceType, where no specific level of pressure must be reached to prevent autocorrection. Rather, increasing pressure smoothly changes the probability distributions in favour of making the pressed key more certain. However, future versions of the system might benefit from implementation on a real smartphone rather than our custom hardware, where visualisation would allow for feedback about the applied pressure and the corrections (or lack thereof) that will be applied given the current input.

With *Force Gestures* [112] Heo and Lee explored augmenting tapping and dragging gestures with varying input pressure. They evaluated their expanded gesture set on a web browser and an e-reader application. Users were able to perform the five gestures available in the application with around detection 95% accuracy after only a short demonstration and a few minutes of practice. They observed that users preferred gestures where visual feedback on their pressure was given, and that dragging gestures with high pressure were seen as fatiguing by users.

A more general study on the performance of pressure in touch interaction was carried out by Stewart et al. [113], who investigated the effect of hand posture on pressure control. They compared selection between pressure level using fingers on the screen, using the fingers at the back of the device, and using a set of 'grip gestures' where pressure is applied to both sides of the device at once. They found that the grip gestures offered a small improvement in selection accuracy over either of the one-side methods, and were preferred by users. Further, they studied whether audio and vibrotacticle feedback could facilitate eyes-free pressure

based interaction, and found that while selection times were not affected compared to the visual feedback condition, selection accuracy decreased by 10 to 30%.

Pressure in Text Entry

A number of systems have explored the use of pressure to augment text entry on touch devices. *PressureText* [114] uses a pressure sensitive keypad to perform text entry. This uses the paradigm of mapping multiple letters to a single numeric key, as found on older mobile phones ('abc' mapped to the 2 key, 'def' to 3, and so on). Rather than pressing the key multiple times to cycle the letters ('multitap'), this system uses input pressure to select between the letters. This was shown to be slower than multitap for novice users, but with training users attained higher maximum typing speeds with pressure than with multitap.

Brewster and Hughes [115] used pressure as a mode switch on a touchscreen keyboard. Users were able to automatically capitalise letters by increasing their touch pressure. They compared two pressure selection stategies — quick release at the desired pressure level and dwelling on the key to select — to typing using a standard soft keyboard with a shift key. They found the quick release keyboard was faster to use than the standard one, but caused more errors. The dwell keyboard had the opposite properties, causing fewer errors than the standard keyboard but slightly increasing the mean time per key.

One-Press Control applied a similar mode switch paradigm to physical PC keyboards augmented with pressure sensors. Once a key is pressed, users can perform a range of pressure variations before releasing the key to take a number of contextual actions. Capitalisation was one example, as was cycling through a list of proposed suggestions in a search engine dropdown. These pressure control techniques were learnable in under 15 minutes of use with around 80% accuracy.

Clarkson et al. [116] studied potential applications of pressure sensitive buttons on a mobile phone with a physical keypad. They present the use of automatically tagging emails typed with higher input pressure as high importance messages. This is similar in a sense to our ideas, since it involves assigning 'weight' to a piece of text based on how hard the user types when entering it.

Another system which shares some elements with our idea is TypeRight [117]. This is based on a physical desktop computer keyboard where the resistance of a key to being pressed can be dynamically control. Using a language model, the system makes it harder to press keys which do not create words in the system dictionary, with the goal of reducing spelling errors as users type. The keyboard effectively makes it harder to type out-of-vocabulary (OOV) words. Our approach takes the reverse approach to combining pressure with a language model — we assume that autocorrection is sufficient to fix most spelling errors, and design a system which makes it *easier* to enter OOV words when a user is sure they wish to do this.

Pressure on Smartphones

All of the systems described above use custom hardware to determine input pressure, whether that takes the form of physical buttons with built in pressure sensors, pressure sensitive graphics tablets, or custom extensions built around smartphones. This makes these systems difficult to implement on commercial hardware. The Android operating system reports a pressure value for touches which is based on the detected contact area of the finger, but this is subject to significant noise since finger posture and the overall size of the user's hands also affect this area.

Recently, systems such as GripSense [118] and VibPress [119] have looked at approximating input pressure on commercial smartphones using accelerometer and gyroscope data. These systems involve pulsing the phone's vibration motor when a user touches and measuring the dampening of the response as a proxy for the pressure. This uses the observation that for harder presses, more of the vibration is absorbed by the user's hand. GripSense could distinguish three pressure levels at 95% accuracy, and VibPress four levels at 90% accuracy. These techniques might allow our ideas to be tested on a current device, but constantly pulsing the motor as the user types might prove very distracting. Additionally, our system relies on smooth variation of pressure to control uncertainty negotiation, so these approximated techniques might not be sufficient for our needs.

6.3.2 The ForceType System

We designed and implemented a prototype pressure sensitive text entry system, which we call ForceType. The system uses the same language model and decoding algorithm as for GPType, but rather than generating key press probabilities from a GP we use a pressure dependent touch model. In theory this technique could actually be used in conjunction with a GP offset model, but in this study we wanted to clearly identify the performance gains available by giving users control of their uncertainty. Adding a GP gives another factor in the touch model which could potentially explain some improvements, and we wanted to be sure to avoid any such effect.

In the new touch model, we take the likelihood $p(K|T)$ of a key given a touch is computed as a Gaussian:

$$p(K|T) = \mathcal{N}(K_C|\mu_T, \sigma_T^2),$$

where K_C is the centre of the key, μ_T is the touch location and σ_T^2 is the variance. This assumes a circular distribution, which is not entirely in line with the rectangular shape of the keys, but this simplifying assumption reduces the number of parameters which must be learned and was found to work quite well in practice.

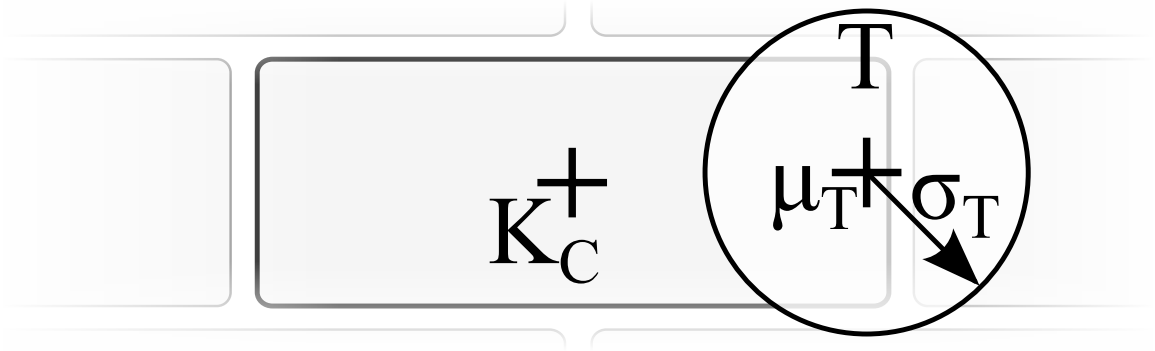


Figure 6.2: The touch model used by our correction system. For each key K, we evaluate the likelihood of its centre K_C under a Gaussian on the touch point μ_T. The standard deviation σ_T is controlled by pressure. Higher pressure causes a narrower distribution.

These likelihoods are computed for each key on the keyboard and normalised to give a discrete probability distribution. It is here that we introduce pressure sensitivity, by changing the variance of this Gaussian dependent on the pressure. In particular, we take the variance

$$\sigma_T = \frac{C}{\omega_T},$$

where C is a constant and ω_T is the pressure for touch T. Stewart et al. [113] showed that although many researchers have used non-linear mappings between input pressure and sensor response, with well conditioned sensors this relationship can be shown to be linear. In initial pilots of the system, we found that this simple inverse relationship between pressure and variance worked well, so we did not experiment with alternative models.

With this model, for high pressure touches the variance is small and the probability mass is concentrated in the pressed key, whereas for lower pressure touches the probability mass is spread over the keyboard, allowing correction to take place. The touch model is illustrated in Figure 6.2. Note that we evaluate the Gaussians at the key centers to produce likelihoods, rather than sampling as in GPType, since we are using simple circular Gaussians with the same variance in all directions. If the system were extended to incorporate a more advanced touch model with offset compensation, sampling to obtain key press probabilities would likely be a better strategy.

The constant C in our likelihood term needs to be calibrated appropriately. In particular, a value such that the distribution for a 'typical' touch has a standard deviation equal to half a key width is desirable. This means that predictions for such a touch assign the majority of the probability mass to the pressed key and its neighbors, and a smaller probability for more distant keys.

To determine an appropriate value for C, we collected data in a short and informal preliminary study. We asked 10 participants to repeatedly write *the quick brown fox jumps over the lazy dog*, an English pangram, on our input device, a pressure sensitive touch pad called the

ForcePad (more information on this hardware is given in the next section). This was done without presenting any feedback about what was typed, and in the absence of any correction scheme. This provided us with pressure information for typing with every key on the keyboard. From a histogram of the logged pressure information, it was the clear the data were not normally distributed and had high positive skew. We chose to fit a gamma distribution to the data. Next, from the 2333 collected datapoints we removed 144 outliers ($>3\sigma$ away from the mean — these were primarily points where the hardware reached its upper sensing limit). We then refitted a gamma distribution to the remaining data. The mean of this distribution was $168.28\,\mathrm{g/cm^2}$ (σ = $106.20\,\mathrm{g/cm^2}$). We then chose C such that the Gaussian distribution for a touch with this mean pressure had a standard deviation equal to half a key width.

6.3.3 Comparison to Offset Modelling

An interesting observation is that the pressure sensitive model presented here can accomplish an increase in text entry accuracy in a different manner to an offset model such as GPType. The offset model accounts for the user's own offsets and for noise in different parts of the screen and assigns probability mass to keys in a way that can often correct for touches hitting the wrong key. The pressure sensitive model allows users to type more softly when they are uncertain of a word's spelling, spreading their probability of more keys and giving the language model more influence in the corrections made. Alternatively they can type harder when they are certain of a spelling and don't want the language model to have any ability to act.

Both approaches offer value, and it is possible that a combination of offset modelling and pressure control could offer further improvements to the text entry experience. This is an interesting avenue for future work, particularly as it becomes more feasible to measure or approximate pressure on touch screen devices.

6.4 Evaluating ForceType

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

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

H2 Users are faster when typing words unknown to text correction when utilising pressure adaptation

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

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

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

6.4.1 Participants

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

6.4.2 Apparatus

For pressure sensing and finger tracking we use a *Synaptics ForcePad* sensor. The device weights $526\,\mathrm{g}$ and measures $14.2 \times 12.2 \times 1.1\,\mathrm{cm}$. The touch-sensitive area covers $10.9 \times 6.9\,\mathrm{cm}$ on the device's surface. The sensor's diagonal thus is $12.9\,\mathrm{cm}$—within roughly $6\,\%$ of a *Samsung Galaxy SIII's* $12.19\,\mathrm{cm}$. Overall, the Forcepad is slightly larger but considerably heavier (an SIII only weighs $133\,\mathrm{g}$) than current mobile devices. However, it does provide a comparable experience when holding the device in landscape mode and typing. Thus, we feel that the Forcepad is a valid placeholder device for evaluating how future mobile devices incorporating pressure sensing touch could be used. Overall we expect that entry rates will be lower than on a comparable phone because of the additional weight and slightly bulky form factor.

The ForcePad provides capacitive tracking with pressure information for up to 5 fingers. Force readings are provided with 6 bit resolution at 67Hz. We take the maximum value in the last 10 frames before the finger lifts off as the pressure value for a touch. We used maximum rather than average values as there were occasional issues where the reported pressure would become large and negative immediately after finger liftoff. The force sensitivity of the ForcePad made us choose this device instead of experimenting with approaches using accelerometers to infer typing pressure (e.g. [120, 119]). Such approaches are typically limited or inaccurate, and in the first instance we wanted accurate values to assess how users controlled the system.

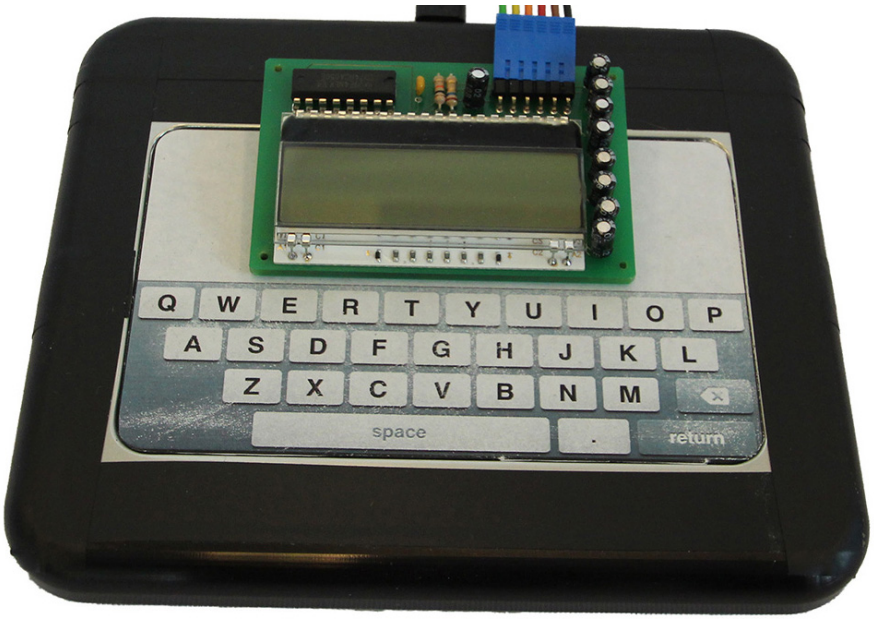


Figure 6.3: We modified a Synaptics ForcePad to allow for graphical feedback to users while typing by attaching an LCD display on top of the device. The display is controlled via an Arduino (not shown). A modified version of the iOS landscape keyboard is glued to the device. Shift, voice-recognition, and mode switch buttons were removed and the space key extended to encompass the @-key.

The ForcePad is an input-only device and cannot display visual feedback or instructions directly on its surface. Using an external monitor for feedback, however, would not allow us to evaluate normal typing behaviour. Users would be forced to repeatedly change focus between the monitor and the ForcePad. We thus modified the ForcePad by attaching a 132×32 px LCD to the device. With a medium sized font this can display three lines of text. The LCD was glued to the top of the ForcePad which brings the overall weight up to 651 g but does not impede hands holding the device. An Arduino is used for control and allows the connected PC to set display content via the serial port. This combination of ForcePad and LCD enables us to simulate future mobile devices with pressure sensitive touch. One limitation of this hardware is there is no visual feedback on corrections before they happen. Again, we expect that this difference from a conventional phone is likely to reduce the overall

entry rates for the study — this is another reason that exploring pressure proxies on current smartphones is a focus for future work.

Figure 6.4: We use a modified version of the iOS landscape keyboard. Shift, voice-recognition, and mode switch buttons were removed and the space key extended to encompass the @-key.

To simulate an onscreen keyboard, we glued a keyboard overlay on the ForcePad. We used a modified version of the iOS landscape keyboard with all buttons triggering mode switches removed (as not to confuse participants). This keyboard is shown in Figure 6.4. The glued on keyboard did not introduce new haptic cues and thus provides an experience comparable to current smartphone keyboards.

6.4.3 Procedure

We used a subset of the English *NUS SMS Corpus* (version 2012.04.30) as phrase set [121]. This dataset contains 41537 text messages, sent primarily by users in Singapore, India, and the USA. Text messages often contain slang and shorthand, making them a good example of the a text where autocorrect fails. However, not all messages are equally appropriate for our evaluation and we removed all messages that:

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

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

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

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

Uncorrectable phrases, therefore, are those which could contain text unknown to the language model and might result in undesired autocorrections. For every participant, 20 correctable and 20 uncorrectable phrases were picked at random. While phrases were chosen randomly, a post-hoc analysis showed that there was no significant difference across users in the number of out of vocabulary (OOV) words per sentence ($p > 0.4$). Participants were not provided with an indication of whether a phrase is correctable or not. They were left to decide whether a given phrase had any words which might be autocorrected when they shouldn't be.

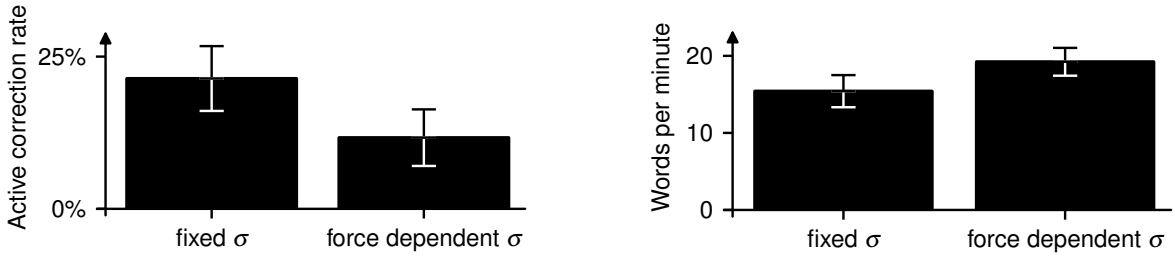
At the beginning of the study, we gave participants the chance to familiarize themselves with our prototype. Participants then had to complete 40 trials. In each trial, they were shown the complete phrase on a PC monitor in front of them and then had to copy that phrase. The PC monitor was used because the full phrases would often not fit on the LCD screen and we wanted participants to know the full sentence before typing. Once they began typing, the phrase on the monitor was hidden. During text input, the phrase (clipped to the display size and current position) is shown alongside the input text on the device. We asked participants to copy the phrases accurately—if they noticed a mistake, whether from a language model correction or their own typing, they were instructed to correct it. After using backspace to delete part of a word, autocorrect was disabled until the next word. This is equivalent to the behaviour on current phones and prevents infinite correction loops. Participants had to submit each phrase with the return key.

6.5 Results

Prior to performing detailed analysis, we performed a 'sanity check' of sorts in which we looked at the baseline text entry accuracy of each user. As a result, we removed data from one participant from the subsequent analysis, as his touch offsets were so large, he rarely hit the right key. His uncorrected error rate in terms of the keys hit was over 70%, and even after language model correction his error rate remained above 50%.

6.5.1 Text Entry Errors

Our first concern is to assess whether ForceType allows users to enter text more efficiently or not. As our performance measure, we use active correction rate (ACR), defined as the



(a) ACR (b) WPM

Figure 6.5: ForceType requires significantly fewer active corrections from users when entering text. Required corrections dropped by ≈ 10 percentage points. Additionally, ForceType enabled users to enter phrases $> 20\%$ faster. Errors bars are 1 std dev in both plots.

proportion of words which the user has to actively correct by backspacing and retyping. This is similar to the metric used in [117], where they looked at the number of times backspace needed to be pressed. If **H1** is accepted, we should see a decrease in ACR since users can use pressure to prevent autocorrection of non-standard words. We chose this metric over CER, since in this study users were able to manually change unwanted corrections. In GPType, corrections made by the system could not be changed, since we wanted to evaluate the quality of the corrections, not the ability of users to control the system. This choice is supported by the results — the mean CERs for the two study groups were not significantly different ($\sim 3\%$ in each case).

Our results are shown in Figure 6.5a. Using pressure adaptation, the average user had an ACR of 10.86%, compared to a value of 19.48% for users without pressure adaptation. This is a significant decrease compared to the baseline condition (independent two-sample *t*-test: $t_{12} = -3.48, p < 0.005$). We can thus reject the null hypothesis and accept **H1**.

It is worth noting that the ACR values for both groups are quite high. This is explainable by the nature of the phrase set used — since they come from an SMS corpus, many of the phrases contain words not found in common English. Again, this choice was made to ensure that participants actually encountered phrases where the language model might make an unwanted correction. This has the potential of magnifying the apparent effectiveness of ForceType compared to typical usage, but we would argue that SMS makes up a significant portion of the usage of mobile phones, and some people are attached to 'text speak' even though their phones might try to correct them. Unfamiliarity with the form factor of our apparatus may have also increased the number of errors made.

6.5.2 Typing Speed

We also looked at how using pressure adaptation changes the text entry speed. As a metric we used words per minute (WPM), with 5 letter words. Our results are shown in Figure 6.5b.

With pressure adaptation active, users typed 19.23 WPM, while without they only typed 15.42 WPM. This is a significant increase in typing speed (independent two-sample *t*-test: $t_{12} = 3.5002, p < 0.005$). We can reject the null hypothesis and also accept **H2**. We note that these entry speeds are quite low in comparison to other keyboards — as disussed before, we attribute this to the bulky form factor and limited visual feedback afforded by our hardware. We expect that if the system were implemented on a conventional smartphone, the overall speed would be higher.

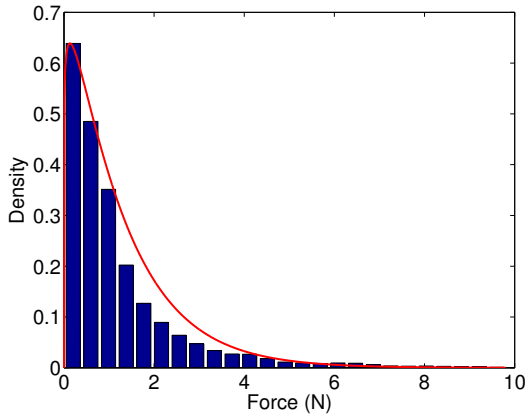
We also looked at the impact of uncorrectable phrases on typing speed. A small drop in speed could be expected, as users have to invest more mental effort in processing those phrases and deciding to use ForceType. For ForceType, we saw WPM go down by 3.97 for uncorrectable sentences compared to correctable ones. For the control condition, the speed reduction was 2.61WPM. Both changes are significant ($p < 0.05$) while the difference between the two changes is not ($p > 0.13$). That is to say, ForceType introduces cognitive load and reduces the typing speed more in absolute terms, but since it offers a higher baseline speed when entering unusual phrases, the proportionate change is not significant. In this sense, ForceType and the control are both equally affected by uncorrectable phrases, resulting in a small performance drop. Overall, ForceType resulted in faster typing speed.

6.5.3 Use of Pressure

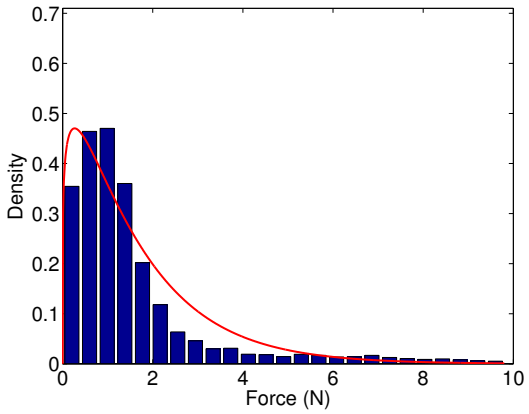
An analysis of how our participants used pressure was also performed. This is important because it may affect the design of the system. If, for example, the typing pressure when the user wishes to suppress the language model is always above some threshold that is never otherwise reached, the system can be reduced to a binary mode switch and the full range of pressure values need not be modeled. Alternatively, it might be the case that users in the pressure dependent group simply typed with increased pressure at all times.

We began by comparing the pressure values for all touches in the study, looking for differences between those users in the pressure adaptation condition and those in the control. As in the calibration study, we observed that the distribution of observed values had high positive skew, and so we fitted a gamma distribution to each participant group. These distributions are shown in Figure 6.6. The distributions are fitted to all recorded touches, including those for both correctable and uncorrectable sentences.

Somewhat surprisingly, we see that the average touch pressure for the users in the pressure adaptation condition is lower than in the control. We applied the two-sample Kolmogorov-Smirnov (KS) test, and the distributions are significantly different with $p < 0.01$. It seems that when given the knowledge that the system was pressure sensitive, their default behaviour was to type more softly than they otherwise would. It may be that participants had uncer-



(a) Pressure adaptation.

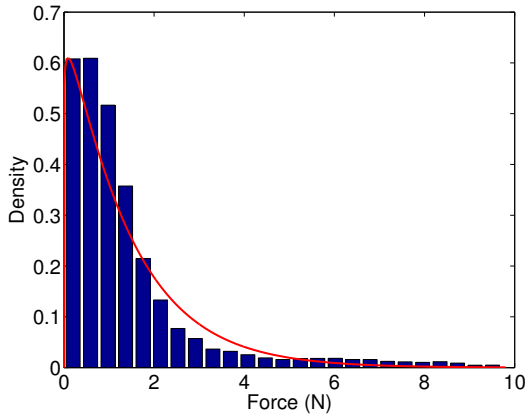


(b) No pressure adaptation.

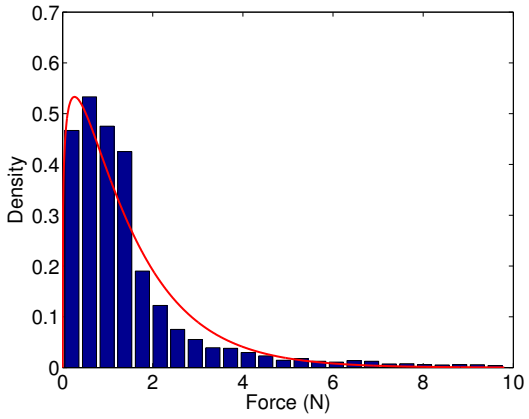
Figure 6.6: Distributions of observed pressure values for participants in the two study groups. Those in the pressure adaptation condition had a lower typical pressure.

tainty about how much force was required to prevent a correction, and typed lightly to avoid accidentally doing this.

Next, we considered the differences between pressure values for correctable and uncorrectable sentences. We would expect that if participants actually make use of ForceType, the pressure values for uncorrectable sentences should be higher since users will take action to prevent corrections in these sentences. This analysis is only presented here for users in the pressure adaptation condition — a similar analysis of the control showed no significant differences between the sentence types. This is to be expected, given users in the control had no idea of the force sensitivity of the hardware.



(a) Correctable sentences.



(b) Uncorrectable sentences.

Figure 6.7: Distributions of observed pressure values for correctable and uncorrectable sentences. The former theoretically can be entered without any ForceType usage. Pressure values were higher for uncorrectable sentences, indicating participants made use of ForceType.

Figure 6.7 shows the fitted distributions for the two sentence types. The most likely pressure

for uncorrectable sentences is higher than that for correctable sentences. The distributions are significantly different (two-sample KS test, $p < 0.01$). This indicates that participants were able to identify situations where an OOV word was present and invoke ForceType to prevent any corrections. This supports our belief that users have a good mental model of text entry, and are able to control a system with an explicit uncertainty model.

Note however that the although the distributions are different, the peaks are still relatively close together ($0.098\,\text{N}$ and $0.279\,\text{N}$). The distributions are also not very sharply peaked — pressure values in the 2–$6\,\text{N}$ range are quite likely for both sentence types. These properties make it difficult to set a single pressure threshold at which to start preventing corrections. A threshold between the two peaks would cause many failed or accidental invocations of ForceType, and one higher than the second peak would still allow corrections in some cases where users were trying to prevent it. This validates the decision to use a continuous model of pressure when computing key press probabilities.

Finally, we look at differences between pairs of users. Touch offsets have been shown to be highly user specific, so we might expect that the pressure distribution will also vary between users. To minimise errors from multiple testing, we applied both the two-sample KS test and the Mann-Whitney U-test, with Bonferroni corrections, to the data from each pair of users. This tests the null hypothesis that two samples come from the same underlying distribution.

For all pairs of users, the null hypothesis is rejected by the KS test with $p < 0.01$. Under Mann-Whitney, the null hypothesis is rejected for all but 3 of the 16 users with $p < 0.01$. The remaining 3 users are each similar to the other two, but not to the rest of the population. Taking the more conservative of these two test results, the pressure distribution for the majority of users was significantly different from all other users. There may be groupings of users with similar pressure profiles, but in general there is substantial variation across users. This again implies the difficulty of a thresholding approach to pressure typing, and shows the value of a full pressure model for each user. Given this result, it may be possible to further reduce the ACR by tailoring the weighting of the language and touch models to each user.

Additionally, this result may suggest why the control condition had such a high ACR. The fixed variance touch distributions were based on a ‘typical’ touch as defined by pooled data from 10 users. Had a fixed variance been defined on a user specific basis, it might have been possible to achieve better results. However, in practice we wanted to compare against a model which did not require training, similar to existing text entry systems.

6.6 Conclusions

This chapter presented ForceType, a system designed to tackle cases where traditional autocorrect systems fail. By allowing users to convey certainty in their input using pressure,

autocorrect can be dynamically turned off to allow entry of words not in standard word lists, such as acronyms or words from local dialects.

ForceType was successful in increasing typing speed and reducing the number of cases where users had to manually correct their entered text. Our results also show that there is significant variation in the use of pressure across users. As with touch offsets, modelling this variation may raise the potential ceiling for text entry accuracy.

In ForceType, our model of the uncertainty is *explicitly* controlled by the user, allowing finer control over the system when the user can anticipate undesired autocorrection behaviour. This stands in contrast to the GPType system presented in Chapter 5, where the uncertainty model is hidden from the user. Such hidden models are applicable in the majority of situations, but the layer of control offered by ForceType allows users to smoothly take back control from a system when they need to. We found that people have good models of words which need correction, but are often overly concerned about the potential for interference from autocorrect. In situations where they are unsure, increased pressure can give them peace of mind that they are in control. When typing normally, nothing changes for them, but if they see a need they can step in and retain control. This puts the work here in line with the Focused-Casual Interaction Spectrum proposed by Pohl and Murray-Smith [78]. We believe the results here show that giving users control over their uncertainty is a viable interaction paradigm.

Chapter 7

Training Data Requirements

Summary. The touch models we have reported on in the thesis so far require a substantial training set — on the order of hundreds of points — to make good predictions. This Chapter considers whether we can avoid this limitation. This is motivated by a study into the differences in touch behaviour across tasks. These differences are significant, and this suggests the need for many models to be trained. Given this, we investigate the use of the RVM to train sparse offset models with small training sets. We show that these models give good performance on a range of datasets representing different tasks and hand postures.

7.1 Introduction

In previous chapters, we have built Gaussian Process offset models using calibration data gathered from a set of known targets. However, we have always trained and tested our models using data from the same task — for example, touching crosshair targets. However, in Chapter 5 we experimented with an offset model trained on crosshair targets and tested on typing data, and found that this model gave worse predictions than the baseline. This suggests that the offsets were quite different between these two tasks.

It is therefore worthwhile to look in more detail at how users' offset behaviour changes based on the touch task they are performing. It may be that each new task requires a completely different offset model. This chapter presents the results of a user study into the cross-compatability of offset models based on a number of target acquisition tasks and a typing task.

We find that users exhibit significantly different offset behaviour when typing than when pressing buttons or aiming for crosshair targets — in particular, the offset functions we learn for typing are more complex than those for the Fitts' Law style pointing tasks, perhaps due to the fact that offsets may change depending on previously hit keys when typing. We conclude

that the choice of calibration task is an important consideration when developing an offset model.

There has already been research that shows that a range of factors affect touch offsets — hand of use, thumb vs index finger input, even effects of the particular device. Our results suggest the necessary space of models is even larger than previously realised, since task-specific models are needed for each of the conditions mentioned above.

Given this, if offset models are to be widely adopted there is a need for a mechanism to train them with as little data as possible, since some users may be unwilling to perform substantial amounts of calibration for each new condition. We thus seek to find an offset modelling framework which is less intensive in terms of training data requirements than the GP, which required hundreds of training examples to reach full effectiveness.

In particular, we use Relevance Vector Machines to build sparse offset models using fewer than 10 training points. We find that these sparse models outperform population based polynomial models, and are better than user-specific polynomials when the number of training points is fewer than 20. The locations of the points used to make predictions in these models are highly conserved across users, indicating some parts of the screen are more important than others when collecting training data. These results hold for both one- and two-handed usage.

Statement of Original Work

One of the datasets analysed in this chapter — the one thumb RVM dataset — was gathered by a student, Daniel Buschek, during an internship at Glasgow. However, all analyses reported here were produced by the author, as were the other datasets and software.

7.2 Touch Variability Across Tasks

The impact of task on touch offset behaviour has not been extensively studied in the literature. Beringer and Peterson [19] gave an early indication that it might be a factor in their 1985 work on desktop touchscreens. They found differences in the horizontal offsets seen between a study where users had to select a single target from a grid of nine, and another study where users tapped a single target on an otherwise blank background.

Essentially all other studies into offset behaviour have focused on a single task for training and testing. Henze et al. published two large studies on touch behaviour, one based on hitting round targets [1] and the other on typing [46]. In each case they learned a polynomial offset correction function, but they did not compare these models. However, looking at the plots of

the predicted offsets for the Galaxy S phone in the two papers, different offsets can be seen for the models in some parts of the screen.

We have also seen some evidence in this thesis that task has an impact on touch offsets. Recall that in Chapter 5, we collected two sets of calibration data for our GP model for use in a soft keyboard. One set consisted of presses on the keys of the keyboard, and the other of touches aimed at crosshair targets. We found that for all users, training the GP on the crosshair data resulted in predictive performance on the key press data that was worse than performing no correction at all. Training and testing on the key press data gave a performance improvement over the uncorrected baseline, however.

Given this observation that offset models from crosshair data did not generalise to typing, it is a natural progression to ask how offsets vary across other tasks. In particular, we wish to collect touch data for a range of tasks and then assess whether offset models trained on one task produce accuracy improvements when they are tested using data from other tasks. Further, we wish to investigate the causes of any observed differences between tasks. The GP framework gives a way to do this in terms of the covariance function hyperparameters — these provide a measure of the relative complexity and variability of the learned offset models.

There are two hyperparameters of interest — the noise variance σ^2 and the covariance length scale l. The former measures the noise found in the training data — that is, it quantifies the random component of a user's offsets. Variation in this across the tasks might explain the incompatability of offset models trained on different tasks.

The length scale hyperparameter l controls how smooth the offset functions can be. A lower value of l means the offset function can change more rapidly between two targets at a given separation. That is, lower values of l result in potentially more complex offsets. If one or more tasks result in more complex functions, then applying these functions to simpler data will result in poor predictions.

7.2.1 User Study

To explore these issues, we collected data in a user study. We created a web app to collect touch data. This was implemented using the HTML 5 Canvas element for drawing, with JavaScript and PHP for logging functionality. This choice was made in favour of developing a platform-specific app so that a wider range of devices and participants could be targeted. We recruited participants using university mailing lists and social media.

Since the application was hosted online, the study was carried out remotely rather than in the lab. Participants accessed the application through the web browser on their own phones. They were instructed to complete the touch tasks while seated in a quiet area, but obviously

we cannot guarantee that this was the case as we did not have control over the participants. This reduces the internal validity of the study, but increases the external validity of our results by capturing touch data in a wider range of settings. This tradeoff between external and internal validity for remote touch studies is discussed in [1].

Design

We chose to collect data for a total of six tasks, as follows:

T1 Full typing — Participants transcribed phrases using a custom soft keyboard.

T2 Key press data — Participants were shown the same soft keyboard as in T1, and given instructions of the form 'Press E'. This is the calibration task used in Chapter 5.

T3 Random crosshairs — Participants had to touch crosshairs that were randomly placed on the screen.

T4 Fixed crosshairs — Participants had to touch crosshairs placed at the locations of the center of our keyboard keys. Only one crosshair was shown at a time however, rather than the full grid.

T5 Random buttons — Participants had to touch rectangular buttons, the same size as the keys on the keyboard, placed randomly on the screen. One button was shown at a time.

T6 Fixed buttons — Participants had to touch rectangular buttons in a identical layout to the keyboard, but with no labels. The target button was highlighted.

These tasks allow us to investigate a range of different aspects of this problem. T1, T2 and T6 all use the same button layout, but allow us to test whether the offset behaviour is different with different stimuli. T1 and T2 split the user's attention between the instruction and the keyboard, while in T6 the buttons *are* the stimuli.

Comparing T3 and T5 allow us to assess any impact of different target shapes on the offset behaviour for arbitrary target locations. T4 and T6, meanwhile, test whether using crosshairs instead of buttons has any effect for a fixed set of target locations.

The tasks fall into three broad categories. T3–T6 are all variants on Fitts' Law style target acquisition tasks, and can be grouped together. T2 has the additional contextual requirement of identifying the location of the target key on the keyboard, and is a category of its own. T1 is related to T2, but as a typing task it has a different associated mental load. In T2, users are given one instruction at a time, whereas in T1 they have a whole phrase to type and can plan ahead to future touches.

Given our earlier observations in Chapter 5, we hypothesise that the observed offset behaviour between T3–T6 and T1, and between T3–T6 and T2, will be different. It is not clear in advance whether T1 and T2 are sufficiently different to induce different offset behaviour, but we hypothesise that they are.

Note that in designing this set of tasks, we have assumed a standard English QWERTY layout for the keyboard. We could equally have picked targets in some other layout — say, the split layout of the KALQ keyboard [103] — and this would likely have led to different offset behaviours. However, from a modelling standpoint this is not an issue. The GP framework is flexible to any layout of interface elements, and this is a key strength of our approach. Some knowledge of the size of targets is needed if we wish to calculate key press probabilities, but there are no restrictions on what those sizes can be (other than the limitations of screen size). This modelling flexibility is a very convenient property.

Participants

We recruited 16 participants (5 female). Participants ranged in age from 18 to 33 (mean = 26.0, sd = 4.5). 14 participants were right handed, and the remainder left handed. The majority of participants (13) had 3 or more years of smartphone experience, and no participant had less than 1 year. Participants were not paid for their time, but were able to opt-in to a prize draw for an online retail voucher.

Procedure

Participants accessed the logging application online, and after providing demographic information and informed consent, were led through the tasks. Order of presentation was counterbalanced across participants. Prior to each task, a short instruction page was shown. For T3 and T5, a sequence of 300 targets were shown. For T2, T4 and T5, we collected 10 touches per target, for a total of 290 touches. For T1, participants had to transcribe 13 phrases, taken from the dataset of Vertanen and Kristensson [106]. The first 3 phrases were not considered in our analysis, to allow participants time to become familiar with the keyboard. The total length of the remaining 10 phrases varied, but all participants provided at least 300 touches in this task. Thus, for each participant we collected at least $(300 \times 3 + 290 \times 3) = 1770$ touches. The maximum number of touches collected for any one participant was 2054.

In all tasks, participants were asked to hold their phone in portrait orientation and touch targets with both thumbs. For each touch we recorded the touch down and up positions and times, along with the position of the intended targets. For tasks T2–T6, no feedback was given about the touches.

Figure 7.1: The interface used for the typing task in our study. The stimulus is shown at the top of the screen and the currently typed characters are shown as asterisks and spaces below. Users were asked to ensure the asterisks lined up with the stimuli.

For T1, participants were shown only a series of asterisks and spaces to ensure that their input lined up with the stimulus phrase. A backspace key was added to the keyboard for this task, to allow users to correct when they believed they had made a mistake. This was done in order to ensure we had knowledge of the intended keys for the touches — we do not mind if a user hits an adjacent key, as that factors into our offset model, but if a character is skipped or an extra one typed, the sequence of touches and intended targets gets out of sync and it is difficult to correct this without hand annotation of the data. Thus users were asked to make sure the asterisks lined up with the stimulus and each word had the correct number of characters. Figure 7.1 illustrates this interface.

Analysis

We began by filtering from our dataset all touches that were more than three target widths away from the center of their intended target. This is a standard filtering, used for example in [122], which defines these touches as ‘blunders’. Such touches are likely to be errors and could affect the performance of the offset models.

For each user-task combination we trained GP regression models using 70% of the available data. For that task, we evaluated the predictive performance on the remaining 30%. As an error metric, we used the root mean square (RMS) error between the model predictions

and the intended locations. We compared this against the baseline RMS error between the recorded touch up locations and the targets.

Given the large number of models to be trained, we learned separate one-dimensional GPs for the horizontal and vertical offsets, which is faster than a single two-dimensional model as used in previous Chapters. We optimised the hyperparameters for each user and task using type-II maximum likelihood maximisation of the marginal likelihood. As discussed in Chapter 4, a user specific optimisation of these parameters is not necessarily realistic in a wide deployment but was important here in order to quantify differences in the models across tasks.

Then, for the touch data for each other task we made offset predictions using the trained model and evaluated the RMS error between those predictions and the targets for those other tasks. In effect, we use the offsets learned on one task to make predictions of the intended targets for each other task, and see whether those predictions are good.

7.2.2 Results

Figure 7.2 summarises our results in a Hinton diagram. The i, j-th box shows the performance when training on task i and testing on task j. Black boxes indicate a performance improvement over the baseline, while white boxes indicate a predictive performance worse than the baseline. Larger boxes indicate a larger difference from the baseline. These boxes represent the average percentage differences in RMSE across all users. For cases where no box is shown (e.g. training on T4, testing on T2) the relative change compared to the baseline was less than 1%, too small to visualise.

To provide scale for the diagram, the rightmost column contains a single box which shows the size of a 100% increase or decrease relative to the baseline. Note that this almost the same size as the box for training on typing data and making predictions on the random crosshair data. This is the largest negative performance effect of any pair of tasks. This box corresponds to a decrease in performance of 101% — that is, the predictions were on average just over twice as far from the target as the uncorrected baseline. This is a significant problem if we wish to deploy offset models on real devices.

A number of interesting features are present. On the diagonal, we can see that training and testing on the same task always gives the largest possible improvement. This is to be expected — the GP is a flexible regression tool and can fit the offset models for a given task easily. However, strikingly, the first row shows that training on typing data and testing on any other tasks results in predictions that are significantly worse than the baseline.

This indicates that the offset functions for the typing task are quite different from those learned from other tasks. Similarly, training on any other task and testing on the typing

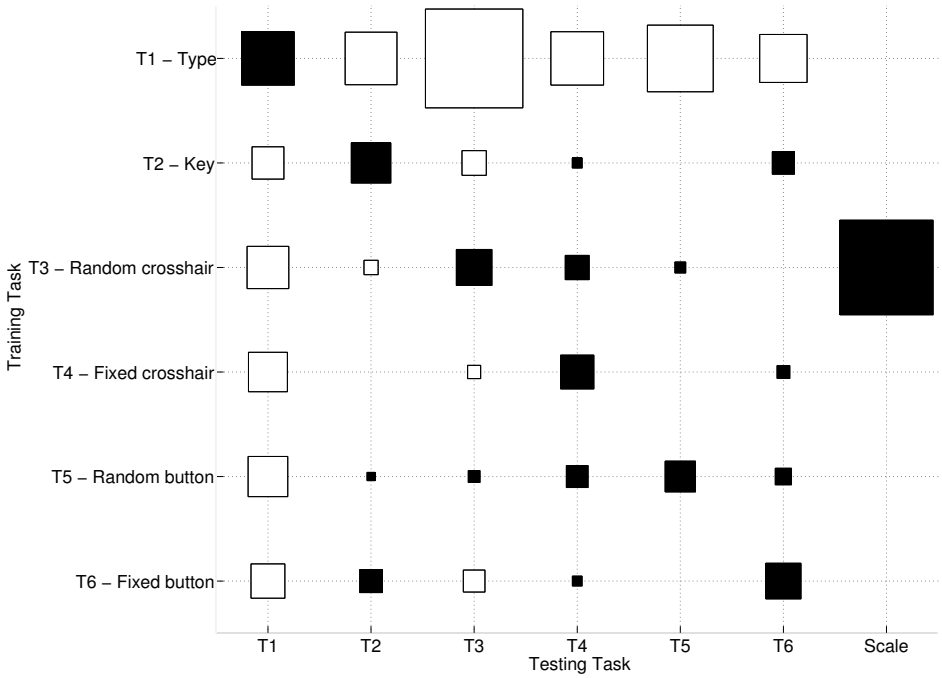


Figure 7.2: Hinton diagram indicating the performance difference over the baseline when training and testing on different tasks. Black boxes represent improved performance, white boxes show decreased performance. The size of the boxes shows the relative size of the increase or decrease. The rightmost box is for scale and represents a 100% improvement relative to the baseline.

data (the first column) results in worse performance than the baseline. Thus, we suggest that where possible, offset models for typing should be learned on typing data and not on pointing tasks.

For T2–T6, the performance when using a model from a different task is often better than the baseline, but worse than training on the specific task. In cases where the performance is worse than the baseline (e.g. train on key presses, test on random crosshairs), the difference is quite small. This suggests that the touch behaviour for different pointing tasks is broadly similar, but varies slightly based on target type and placement. We can see that training on T5 *always* results in a performance improvement over the baseline, for non-typing data. This is the task using randomly distributed buttons — therefore, in cases where it is not possible to collect task-specific data to build an offset model, we make the design recommendation that calibration data is gathered from button targets.

Explaining Differences

To explain why the observed offset behaviour is different, we turn to the learned covariance parameters for each task. For each user, we pairwise compared the distributions of values for σ^2 over the 10 restarts for each task. There is no significant difference in the noise variance

across the tasks (t-test, $p > 0.4$) for all users but one. The exceptional user had significantly higher variance for the typing data model compared to all others. This means that, in the majority of cases, the random component of a user's touches is independent of the task — instead, the observed performance differences come from differences in the systematic offsets across tasks.

We also examined how consistent the variation across user and task for σ^2 was. Does the user with the highest variance in one task have the highest variance for the other tasks? To quantify this, we took the set of σ^2 values learned for each user and task, and sorted it by user for each task, generating six ordered lists. For each pair of these lists, we computed the Kendall tau rank distance between them. This counts the number of discordant pairs in the list, where user i is below user j in one list and above user j in the other, or vice versa. These counts are normalised to give the proportion of the orderings which disagree.

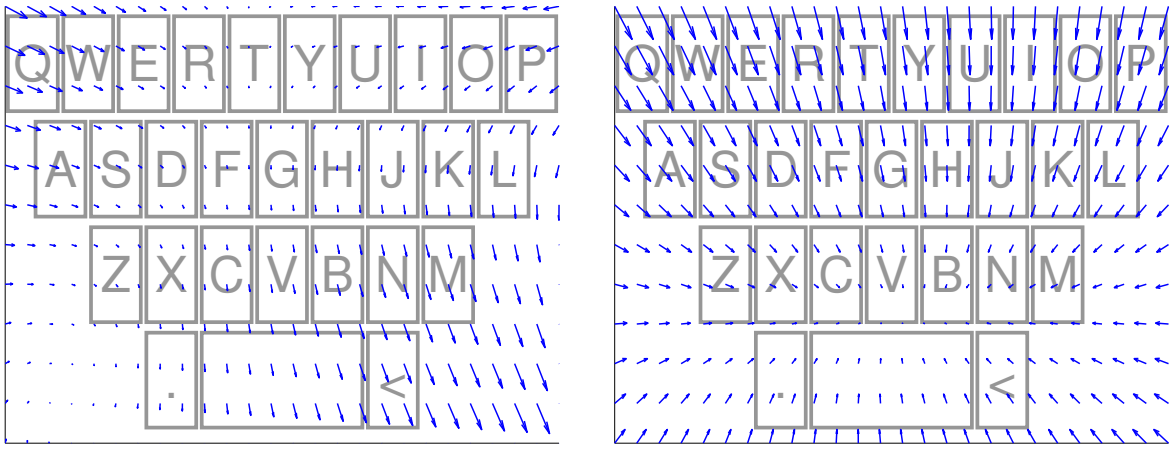
The values obtained vary significantly, from a minimum disagreement of 0.2857 between the orderings for the fixed and random buttons tasks, to a maximum of 0.6286 between typing data and random crosshairs. In general the orderings vary significantly across the tasks, reflecting a lack of consistent structure here. The noise variances are all fairly closely grouped in the 0.01–0.02 range, and the fact that a user is more variant than others for a given task tells us little about their relative variance in other tasks. The exceptional user described above was notable in that for T1 they were the most variable user but for T2–T6 they were consistently in the bottom two.

We now consider the values of the length scale l. We perform a similar analysis, comparing the values for given tasks pairwise for a given user. We find values of l that are significantly lower for T1 and T2 than for any other task (t-test, $p < 0.01$). Additionally, the values for T1 are significantly lower than for T2. These findings are true for all users. This confirms the initial hypothesis that the offset behaviours for the three task categories are different.

A smaller length scale means that the offset function can change more rapidly between two touch points which are relatively close — in general this means the offset behaviour for these tasks is more complex. T1 and T2 are the tasks which involve the soft keyboard, indicating users touch in a more complex fashion given the additional cognitive load of these tasks. For the pointing tasks T3–T6, l is larger, meaning the offset function is simpler. Trying to apply a complex offset function to simple touch data, or vice versa, leads to the observed worsening in performance.

As a visual example, Figure 7.3 shows a visualisation of the offsets learned for T1 and T3 for one of our users. In each case, we evaluated the offset functions in a grid over the area covered by the soft keyboard in T1. The keyboard is shown for clarity. The two surfaces are quite different, with the T3 trained model tending to have offsets towards the bottom right of the screen, while the T1 trained model has offsets towards the bottom row of letter

keys. This is perhaps an indication that when typing the user's thumbs rested around this row between key presses. In the bottom right corner of the screen, the two models predict offsets in entirely opposite directions Other users also display differences between offset surfaces for the different tasks, and further there are different surfaces for the same task across users. As our previous work, and that of others, has shown, touch offsets are highly individualised.



(a) Random Crosshairs (b) Typing

Figure 7.3: Offset surfaces for a single user trained on the Random Crosshairs and Typing tasks. The location of buttons on our soft keyboard are shown for reference. The offset patterns are distinctively different, to the point that in the bottom right corner they point in opposite directions.

The observed additional complexity in T1 compared to T2 may be a result of the fact that typing is a more serial process — offsets may change based on the previously hit key, for example. Using the previous touch point(s) as additional input to the GP may be an interesting direction for future work on offset modelling. Also, all of our participants had at least 1 year of smartphone experience. It is possible that their typing behaviour has come to depend to some degree on autocorrect systems and knowledge of the QWERTY layout. It would be interesting to repeat our study with a set of novice users and see if the same conclusions about offset behaviour hold.

7.2.3 Discussion

The results here are quite striking. There is clear separation in touch behaviour between tasks involving acquiring button or crosshair targets and tasks involving typing phrases or pressing keys on a soft keyboard. This is a novel finding, and has implications on previous research.

Henze et al. [1, 46] developed offset functions at a device specific level by collecting large amounts of calibration data using games, one using circular targets and one where users had to type words. Our results suggest that these models would likely not be cross compatible — a universal touch model is difficult, if not impossible, to construct.

Buschek et al. [47] looked at the creation of functions to adapt a user's offset model from one device to another, or to adapt a pooled data model for a given phone to a specific user. These models were all based on crosshair data. Our results add an additional layer of complexity to this work, since task is also a factor which must be considered. An interesting avenue for future work is to take a similar approach to these authors and try to adapt models trained on one task to suit another task. The observed changes in model complexity may be systematic in some way.

An interesting observation is that here we found training on key press data and testing on typing data resulted in a small decrease in performance over the baseline. This is equivalent to the GP Only condition in our text entry study in Chapter 5, but there we found this gave an improvement in text entry accuracy. The differences in results can perhaps be attributed to the different feedback given in the two studies for typing data. Here, only asterisks were shown, while in the GPType study users could see which keys they were hitting and may have adjusted their touch behaviour on the fly in response to this. The effect of visual feedback on touch behaviour is another area which should be explored in future work.

7.3 Sparse Offset Models

Given that our results here suggest a number of different offset models are necessary to adequately deal with different tasks, it becomes important to be able to train these models using as little data as possible. Gathering data for these models is not trivial. Data from typing, for example, is not typically acceptable as we must be sure where the user was aiming. When typing users are most often composing new text rather than transcribing something known to the system, so the intended target of each key press is not known in advance. One could assume that the final text submitted was representative of these intended targets, but the capability of autocorrection to fix transposition errors in text means this is not necessarily the case.

To get round this problem, researchers have typically included a calibration phase in their studies. Henze et al. [1] used a touch-based game, whilst we have used bespoke calibration applications to collect training data. The problem is particularly acute for individualised models as we cannot rely on data from other people – all training data must come from the current user. Given that the phone may be used in multiple modes (portrait, landscape, one hand, two hand, thumb, finger, etc), and that each mode-task combination may require its own set of training data, the burden on the user becomes far too onerous.

In this Section, we investigate whether it is possible to train individual offset models with very small quantities of data (i.e. fewer than 10 calibration touches). We are not interested, in this study, in finding out how many touches users might tolerate, although this is clearly

important. Rather, we are looking to push the number required to train touch models down as low as possible. As well as answering the question 'how many points are required', we will also compare different touch models to see which models perform best in this small data setting. In particular, we propose that it is very important that any touch model, when presented with a very small set of calibration touches shouldn't make the touch problem worse.

The work here has two strands. In the first, we use recorded touch data to investigate, in a perfect scenario, how few calibration touches can be used to define a model. For this, we use the Relevance Vector Machine (RVM), discussed in Chapter 3, as it is a regression algorithm that produces a solution that is sparse in basis vectors (corresponding to training points). The solution is therefore defined with respect to a (typically) small subset of the total training data. This represents an ideal scenario as we perform the analysis on a large touch dataset. However, it provides an indication of the number of touches required to produce improvements in touch accuracy of the baseline of no offset model. The results suggest that good performance can be achieved with fewer than 10 training points and that the location of the chosen calibration points is highly conserved across different users. However, this analysis is entirely idealised and gives us only a lower bound on the number of training examples required.

The second strand involves investigating how rapidly it is possible to train models starting from no data. In other words, if we sequentially add more data points to the model, how rapidly does the error drop? This is perhaps a more useful line of investigation, since a real world deployment of an offset model would start from this point. In Chapter 4, our results showed improvement was almost immediate when a GP regression model was trained using the recorded touch position as an input and the intended touch position as an output. Here, we perform a similar analysis but focus in on the very small dataset sizes and look at data from different devices and different input algorithms. We discover that the RVM can get excellent performance even with datasets so small that it would be impossible to train parametric models.

7.3.1 RVMs for the Offset Problem

Recall from Chapter 3 that given a set of observed inputs $\mathbf{x}$, the RVM makes predictions of the form

$$\delta(\mathbf{x}) = \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}).$$

For the touch offset problem, $\mathbf{x}$ are the sensed touch locations, $\delta(\mathbf{x})$ is the approximation to the offset between the sensed and intended location, $\phi_m(\mathbf{x})$ are the basis functions and w_m are the weights. If we have a set of N training points $\{\mathbf{x}_n, t_n\}_{n=1}^{N}$, the RVM algorithm finds

values for the weights such that $\delta(\mathbf{x})$ generalises well to test data but relatively few of the weights are non-zero. In this way, only a small proportion of the basis functions are used.

7.3.2 Dataset

We evaluated the RVM on an existing set of touch data [47]. This data was gathered in a user study where users were shown a series of 300 crosshair targets on a smartphone and asked to touch them as accurately as possible. Participants were seated and held the phone in one hand, touching using the thumb on that hand. The dataset contains touches from 30 subjects on 13 different smartphones. All subjects repeated the study on multiple phones, but not all subjects used all phones. In this chapter, we analyse 43200 touches from the 3 phones for which most data was collected — the iPhone 4 (25 subjects), Nokia N9 (24) and Nokia Lumia 900 (22).

7.3.3 Making Predictions

To train an RVM using this data, we chose the basis functions $\phi_m(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}_m)$, where $K(\mathbf{x}, \mathbf{x}_m)$ is the radial basis function kernel between the test point $\mathbf{x}$ and the training example $\mathbf{x}_m$. In this way, we have a number of basis functions equal to the number of training points — 300 for each session. The basis functions given non-zero weights by the RVM (the 'relevance vectors') are those points which are important for making predictions of the intended target for new test touches.

The predictive function from the RVM is one-dimensional — that is, although (x, y) training pairs are used to compute the basis functions, the RVM can only predict either x or y. Thus, we need to train two models for each session, each of which will have different relevance vectors. This is potentially interesting, as it allows us to to find which areas of the screen we need to collect data from to predict each of the dimensions, and also which points are used to predict both x and y. Note that in the remainder of the Chapter, we will discuss models in terms of requiring n training points for making predictions — unless otherwise stated, this refers to the sum of the number of points needed for the two RVMs.

7.3.4 Results

Predictive Performance

For each session, we train the two RVMs and evaluate their predictive performance. We perform 5-fold cross validation, holding out 60 touches for testing at each step. Our performance metric is the RMS error between the predicted location of the RVM and the intended

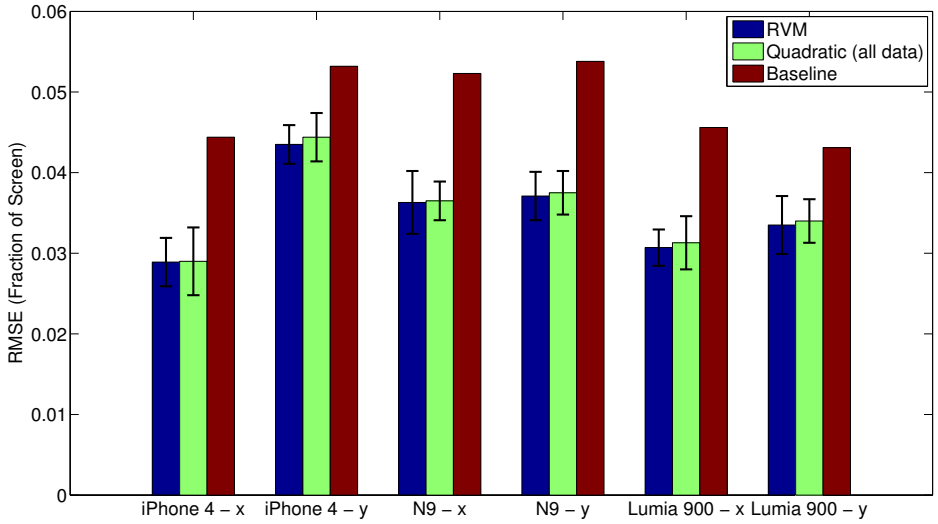


Figure 7.4: Mean and standard deviation in RMS error across all sessions for several phones and predictive models. The baseline is the error between the orginally recorded touches and the targets.

touch location (position of the crosshair in the trial). As a baseline, we compare against a second order polynomial of the form $\delta = w_0 + w_1 x + w_2 x^2 + w_3 y + w_4 y^2$, fitted using least squares regression. This allows us to investigate how a sparse solution performs in comparison to a model learned on all the available data.

In [1] the authors learn an offset function using a fifth order polynomial. However, we found that the typical number of relevance vectors was small enough that it was not possible to train a well conditioned fifth order model (such a model would require 11 datapoints at least, due to the 11 features required). The number of relevance vectors was typically fewer than 5, so we desire a baseline model which can be trained with a similar amount of data. A linear model is likely to be too simple, so we opted to use a quadratic baseline, which can be trained with 5 points if necessary.

Our results are summarised in Figure 7.4. We present results for the three most common devices in the dataset. For each device, the plot shows the mean and standard deviation in RMS error across all sessions for the RVM and the quadratic model. We also show the RMS error between the recorded and intended locations in the absence of any correction as a baseline.

We see that the RVM and the polynomial both perform better than the baseline. These differences are both statistically significant (paired t-test, $p < 0.05$). We also see that the RVM provides a small performance improvement over the polynomial model. For all but the x conditions on the iPhone and N9, this improvement is also significant. That is, in four out of six conditions the RVM performs as well as or better than a model with many times more training data. This makes it an attractive algorithm for building offset models given the previously identified need for a new model for each new task.

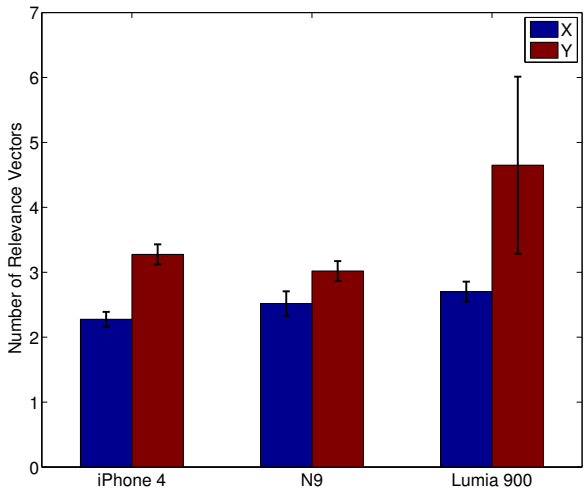


Figure 7.5: Mean and standard error across sessions in the number of relevance vectors used to predict each dimension for three different phones.

Number of Points Required – the Ideal Case

Figure 7.5 shows the number of relevance vectors used to predict each dimension for each of the three most common phones. We can see that for each device, the RVM typically selects between 2 and 5 relevance vectors for each dimension. That is, on average we need to evaluate the kernel function at fewer than 10 training points from the original 240 to make good predictions on test data. Across all sessions, the number of relevance vectors used to predict x is always between 1 and 9. For y, the range is 1 to 13. Given that our basis functions are relatively simple r.b.f kernels, the fact that y predictions required more points on average suggests that vertical offsets are more complex than horizontal ones.

The variability for the vertical offsets on the Lumia 900 is by far the highest. This may be because this is the largest device of those studied by a large margin, meaning that some users could not fully extend their thumbs to the distant parts of the screen. Thus, grip adjustments were sometimes necessary, so that the offset patterns potentially changed. This means the RVM needs more relevance vectors to model these cases.

Additionally, the fact that in some cases the RVM retains only one training point per dimension as informative is quite interesting. This means that the offset at a single point is representative of the whole screen, suggesting quite a simple offset pattern. This may be because the dataset analysed consists of one thumb touches, so there are no effects of changing hand that lead to more complicated offsets.

Note that this minimum number of points is something that we may not be able to meet in practice as the RVM analysis has many points available to start with and finds which of those are important for prediction. In reality, when a new user picks up a device we have no data and want to build a model as fast as possible.

Distribution of Relevance Vectors

The RVM analysis also allows us to explore where these points are positioned in the input space. This can give insight into which areas of the screen are important when learning the offset function. In order to do this, we use a kernel density estimate to produce a probability density function of the point location for each phone type.

For a given phone, we collect all relevance vectors extracted for all users and sessions on that phone, and place a narrow Gaussian at each of these points. To estimate the density at a new point $\mathbf{x}'$, we take the average of all of these training point densities evaluated at $\mathbf{x}'$. This is known as a kernel density estimate [123]. By evaluating the density estimate at each point in a grid, we can visualise the surface over the screen. Figure 7.6 shows the density of the points used to make predictions on each phone for both horizontal and vertical co-ordinate predictions.

There is a clear structure evident, showing that some areas of the screen are more important than others for making predictions. Additionally, it is interesting to note that the points which are most important for predicting x are distributed quite differently from those for y. The distributions are somewhat different for the different phones as well.

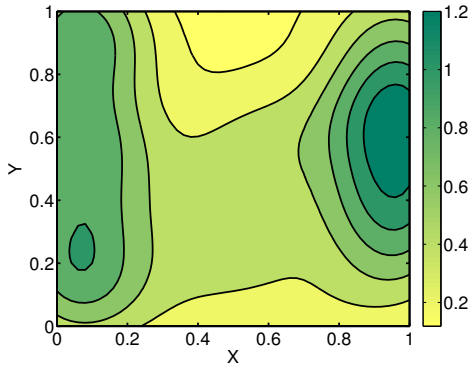
For predicting x, the most important areas are at the right and left edges of the screen. The exact positioning of the important areas on the left edge varies by phone — for the iPhone, the whole edge has a similar importance, whereas models on the N9 relied more on points in the bottom left corner, and the Lumia 900 used points in the top left.

To predict y offsets, the most important points are in the corners of the screen. In particular, the N9 puts a high importance on training points from the bottom left corner of the screen for predicting this offset. The Lumia 900 assigns much higher weight to the left corners than the right. This phone had the largest screen of any device in the dataset, and reaching targets on the left side (when using the phone with the right hand) was difficult for users with smaller hands. This makes the offsets at this side of the screen more pronounced, and thus training data in that area is more important.

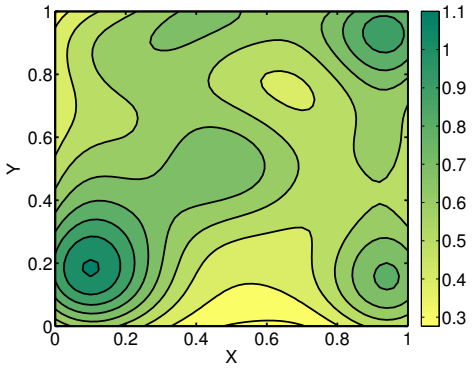
Since all participants in the study which generated this data held the phones in their right hands and touched with the right thumb, these distributions are specific to that usage mode. For left handed usage, we would expect a horizontal mirroring. Two handed usage is more complex, and requires more data.

These relevance vector distributions are potentially of interest for interaction design. The points where the distributions are peaked are important precisely because they are harder to model, and so this could be used to inform the placement of interface elements.

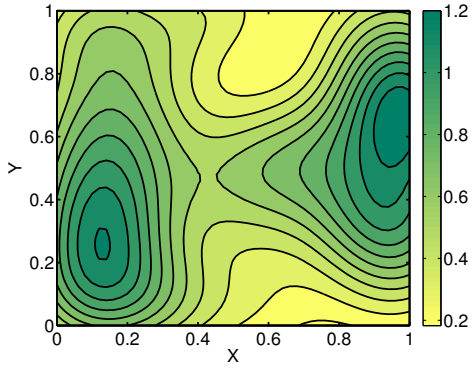
Further, these distributions may be helpful in developing future offset models. They enable us to identify which parts of the screen are most important in determining the offset function,



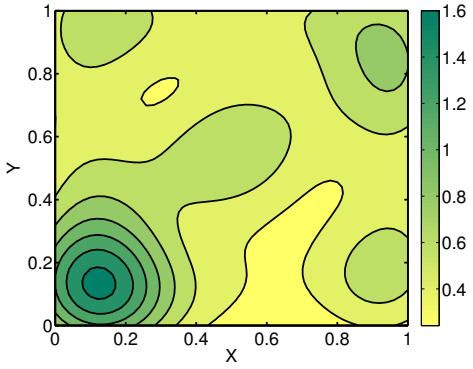
(a) iPhone — x Relevance Vectors



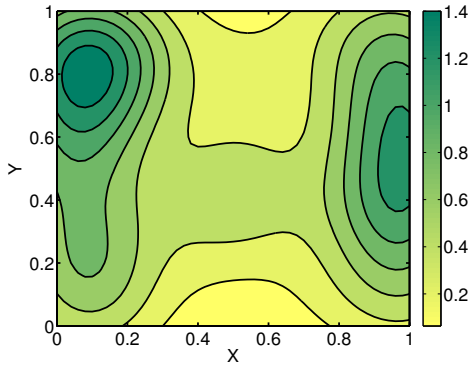
(b) iPhone — y Relevance Vectors



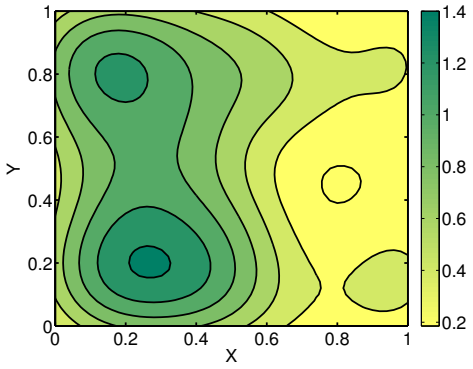
(c) N9 — x Relevance Vectors



(d) N9 — y Relevance Vectors



(e) Lumia 900 — x Relevance Vectors



(f) Lumia 900 — y Relevance Vectors

Figure 7.6: Kernel density estimates for the distribution of relevance vectors on three phones in portrait orientation. Peaks of the distribution indicate the important training point locations for constructing touch offset surfaces. The important points for predicting X and Y offsets are quite different, and there is variation across phone model.

so target locations for training examples could be sampled from these distributions rather than randomly placed on the screen. This is an interesting avenue for future work.

Generalising to New Users

The analysis above represents an 'ideal world' situation, in which we start with a fairly large dataset of touches for a user and use the RVM to extract the relevant training points. This is not the desired use case in general — we wish to learn a model for a new user which requires only a small number of training points.

An algorithm designed to work with a small training set needs to satisfy two properties. First, it must be stable. That is, when there is very little data (fewer than 5 examples) the predictions should not be worse than the baseline where no offset correction is performed. Secondly, the performance should improve quickly as training data is added. We have seen that the RVM produces good predictive performance using a small set of basis vectors, so we now investigate the performance of the algorithm when starting from no data.

We assume no knowledge of the relevance vector distributions produced in the previous Section, and instead seek to learn how quickly performance improvements can be obtained given some number of randomly located training points. To conduct the analysis, we take the training examples for a user, permute them into a random order, and add them to the training set in small increments. We train an RVM at each increment and compute the predictive performance on a held out test set. In addition, we also train a quadratic model using the same points and a least squares regression based on the kernel matrix passed to the RVM. We evaluate the performance of the models using 2, 5, 10, 15, 20, 50 and 100 training points. Our performance measure is the root mean square error between the model predictions (x', y') and the true positions (x, y).

Figure 7.7 shows the results averaged across all sessions on all three phones. We rescaled device coordinates to a unit square to allow cross-device comparison. The performance value for each session was obtained by averaging the RMSE over 20 random restarts of the process described above. This helps ensure we use a variety of subsets of the training data.

The plotted curves show the mean test RMS and standard error across all sessions, and thus represent a mean of means. To provide a comparison to a more traditional population approach, we also fitted a quadratic predictor based on all of the collected training data. The performance of this model is shown by the black dashed line. We can see that the RVM error is equivalent to that of the population quadratic model with only 5 training points, and better with 10. The population model was trained with thousands of data points, so this is an impressive result.

The parametric models, by contrast, have higher error for the smallest values of n. The

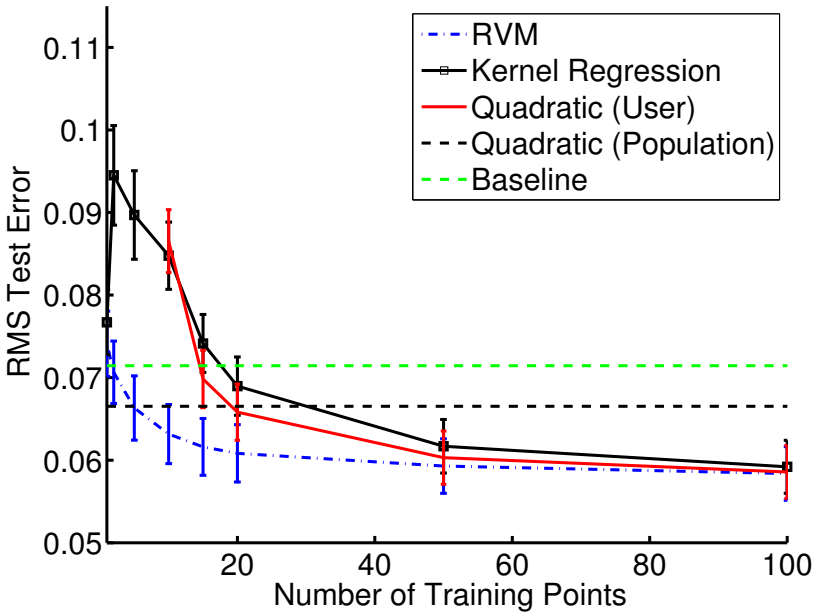


Figure 7.7: RMS error as a function of training set size for four prediction models. Results show mean and standard error across all subjects over 20 random restarts. The baseline is the RMSE between the targets and original touches.

results for the individualised quadratic model are only shown for $n = 10$ and above. This is because these models require at least 5 training points, and with exactly 5 they suffer from significant overfitting and make very poor predictions on the test data.

For the RVM, the average improvement over the population model at 10 training points was small but statistically significant (paired t-test, $p < 0.05$). This analysis shows the power of the RVM for very quickly building offset models that offer an improvement over commercial hardware and existing correction strategies.

It is interesting to note that the user specific quadratic model exceeds the performance of the population model for 20 or more training points. This is in line with our earlier findings, which suggested that user specific offset models performed better than population based techniques.

When 100 or more training points are used, the RVM, kernel regression and individual quadratic models all give similar performance. However, the RVM approaches this performance limit more quickly, with little improvement after 20 training points, whereas the other models improve steadily as data are added up to 100 points.

Also shown is an uncorrected baseline, indicated by the green dashed line. This shows the RMSE between the recorded touches and the target locations, without any offset correction. We can see that as soon as any training data are added, the RVM immediately outperforms this baseline. The same is not true for the individual quadratic model and the kernel regression, which only consistently outperform the baseline when 20 or more training points are

used. This is an attractive quality of the RVM — it does not make things worse when little data is available, and thus can begin offering improvements to the user's touch accuracy very quickly.

Note that we have assumed a random sampling of points when gathering data to train RVM models. This may not be the best approach — it may be possible to reach the best possible performance with even less data than we used here. For example, we could sample training points according to the kernel density estimates of the relevance vector distributions. Alternatively, we could employ an active learning approach, where we place training points in areas of high variance under the models obtained with the current data. Investigating this is an area for future work.

7.3.5 RVMs using Two-Thumb Data

The small number of training points required by our model in the above analyses suggests the offset surfaces are quite simple on this dataset. This may be because the dataset used consists of touches from only one thumb. Two-thumb touch behaviour is likely to be more complex, and so we turn now to an RVM analysis of some of our previously gathered data.

In particular, we analyse the data gathered in the first part of this Chapter, in which users provided data for a variety of tasks. We will focus on the data for the typing task and the task using randomly positioned crosshairs as targets. The latter is chosen for comparison with the one thumb crosshair data from the previous Section, while the former had quite a different offset pattern to the other tasks.

If the offset behaviour for two-thumb usage is indeed more complex, we would expect that the average number of relevance vectors required to predict the offsets in each dimension would be higher. Further, in our GP analysis of this data we found that the typing offsets were typically more complex than those for pointing tasks. We would therefore expect that an RVM trained on typing data would retain more relevance vectors than one for pointing data.

Method

We again perform an 'ideal world' analysis, starting with a full set of training data and running the RVM algorithm to see how many relevance vectors are obtained, and their locations. We use the RVM to predict offset values given device locations as inputs. As before, we train two separate models for each session, one to predict x offsets and one to predict y. We hold out 20% of the data from each session for testing, evaluating model performance based on the RMSE between the target locations and RVM predictions. All coordinates are rescaled

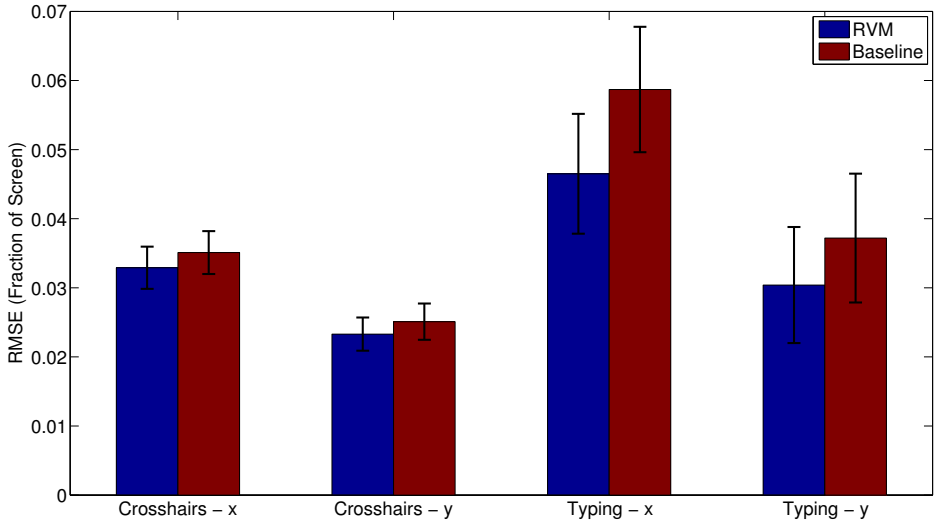


Figure 7.8: Mean and standard deviation in RMS error across all users for the Random Crosshairs and Typing tasks. The baseline is the error between the orginally recorded touches and the targets.

to a unit square, since as before our dataset contains data from multiple phones which we wish to compare.

Results

Our results are shown in Figures 7.8 and 7.9. For each of the Random Crosshairs task (T3 in the earlier study) and the Typing task (T1), Figure 7.8 shows the mean RMSE between the target and RVM prediction while in Figure 7.9 we see the average number of relevance vectors n needed to predict offsets in each dimension. The baseline condition is the RMSE between the phone's touch coordinate and the target center. Results are averaged across multiple users and not separated by phone, since most of the devices in the study appeared only once.

We see that the RVM outperforms the baseline for each dimension in both tasks. All of these differences are significant (t-test, $p < 0.05$). Thus, it is still possible to get a performance improvement for two-thumb usage when using the sparse model. The improvements seen for the typing data are larger than for the Random Crosshairs task. The Typing task also has a higher baseline error, suggesting users are less accurate when typing than when hitting crosshairs.

In terms of the number of relevance vectors required, several interesting results are present. First, for both tasks we see that the mean value of n for predicting y (vertical) offsets is quite low. In fact, it is lower than the values found for this dimension for one handed data. This is contrary to our supposition that offsets for two-thumb usage would be more complex and thus require more relevance vectors. However, on reflection this is potentially reasonable

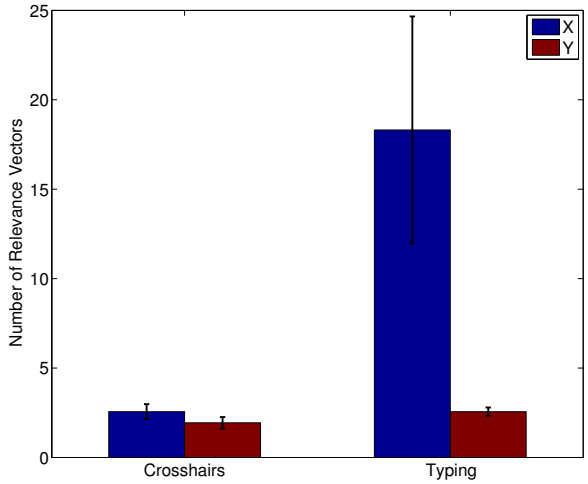


Figure 7.9: Mean and standard error across sessions in the number of relevance vectors used to predict each dimension for the Random Crosshairs and Typing tasks. The x dimension of the Typing data requires by far the most relevance vectors.

to expect. In one-handed usage, the thumb has to reach across the entire screen, possibly extending beyond the user's comfortable range and thus the difference in observed offsets between the two edges of the phone might be quite pronounced. In two-handed use, however, each thumb has only to stretch over approximately half of the screen, resulting in fewer extreme movements and more consistent vertical offsets.

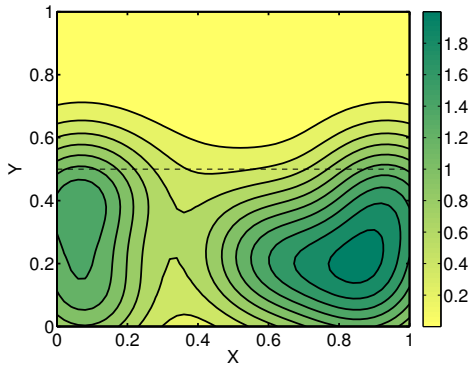
The next observation is that the number of relevance vectors required to model the x offsets in the typing data is very high — around 18 on average. This is substantially more than the values of 2–3 seen in the previous study and for the Random Buttons task. This suggests that these offsets are much more complex than those in the other tasks, in line with the findings of the first part of this Chapter. This observation is in line with expectation.

As described previously, these complex offsets are likely the result of the rapid changing between thumbs as the user types. The exact point that this transfer occurs on the screen will vary depending on the previous keys hit, and so the area in the center of the keyboard is quite uncertain.

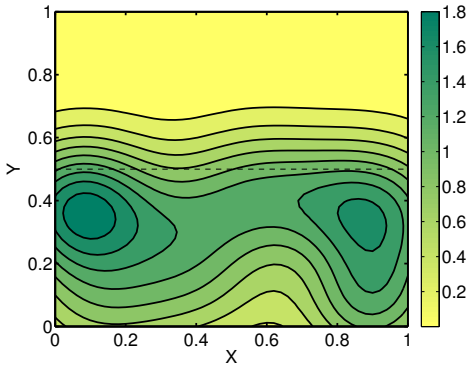
Interestingly, the same complexity in horizontal offsets is not seen for data from the Random Crosshairs task, which require a similar number of relevance vectors to model as the one-handed use case. We attribute this to the slower nature of these selections — each time a new target appears, the user must identify its location and then touch. While typing, they may be planning ahead about which keys the fingers will move to next and generally operating more quickly, leading to the some of the observed complexity increase. All of our users had a year or more of smartphone experience, so there is also likely to be some learning effect there.

Relevance Vector Distribution

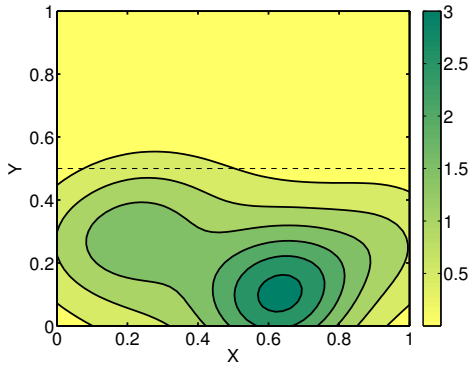
As before, we analyse the distribution of relevance vectors across the screen for each dimension and each task, to see whether this holds useful information. We compute a kernel density estimate of the spatial distribution and plot it over a grid. For comparison between these plots and those for the one thumb data, we show offsets over the whole screen but note that data was only collected in the bottom half, in the region corresponding to the keyboard. This region is indicated with a dashed line — while the relevance vector distributions extend into the top half of the screen, no points are actually located here.



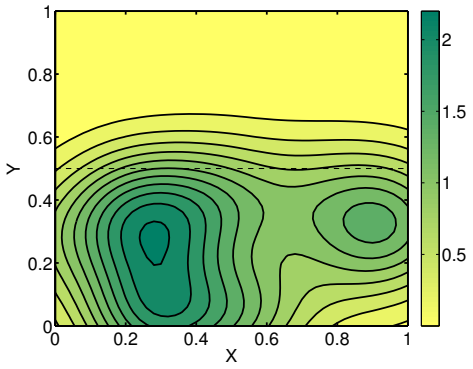
(a) Crosshairs — x Relevance Vectors



(b) Crosshairs — y Relevance Vectors



(c) Typing — x Relevance Vectors



(d) Typing — y Relevance Vectors

Figure 7.10: Kernel density estimates for the distribution of relevance vectors for two-thumb usage for targeting randomly placed crosshairs and typing. Peaks of the distribution indicate the important training point locations for constructing touch offset surfaces. The distributions for the Random Crosshairs task are smoother.

The distributions are shown in Figure 7.10. As expected, we can immediately see a sharp ridge in the horizontal center of the space for the distribution of x relevance vectors for the typing task. This supports the belief that the change in offset caused by using one thumb or the other near the center of the screen is an important factor in this task.

For the vertical offsets in the typing task, the relevance vectors are concentrated on the left side of the screen. It is unclear why this should be the case. Perhaps, as theorised above, the

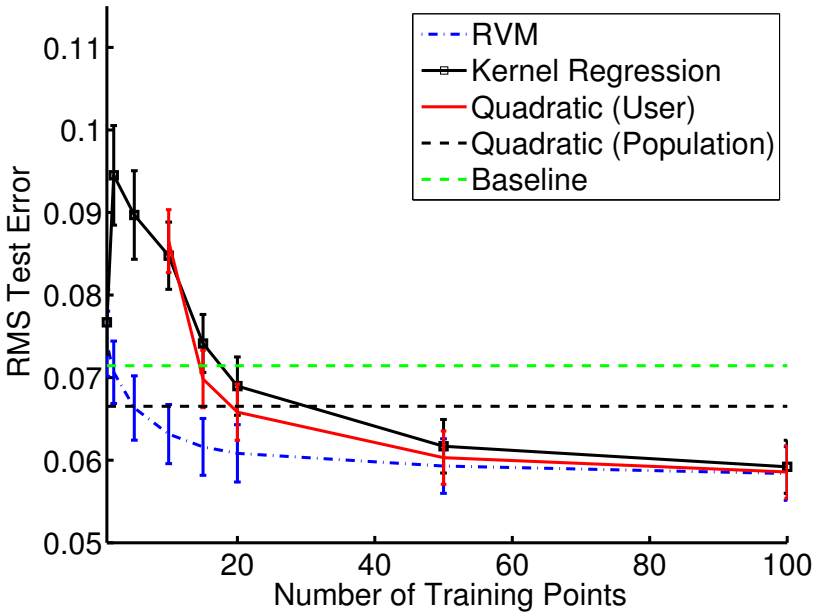


Figure 7.11: RMS error as a function of training set size for four prediction models applied to two thumb data from the Random Crosshairs task. Results show mean and standard error across all subjects over 20 random restarts. The baseline is the RMSE between the targets and original touches.

offsets for each thumb are quite similar and thus the offsets for the right side of the screen can be inferred from relevance vectors on the left.

The distributions for the Random Crosshairs task are less sharply peaked than those for the Typing task — the important locations for determining these offsets are more spread over the screen. In general, the center of the screen seems to have a lower concentration of relevance vectors than the edges for this task. Since these distributions are drawn from the relevance vectors of all users, it may be that this indicates a lack of a consistent targeting behavior across users — different parts of the screen are important for predicting the offsets of different users, so that the aggregate distribution is less well defined than for the Typing task. It may be that the added semantics of the Typing task lead to consistent behaviour over users, or that the set of possible targets for the Crosshair condition is bigger so we see more variation in the observed data.

Performance for Small Training Sets

We also analysed the performance of our model as a function of training set size, as we did for the one thumb data. We employed an identical protocol, randomly permuting the data and adding them to the training set in small increments, training an RVM at each increment. Figure 7.11 shows our results on the Random Crosshairs data. This data is plotted for comparison with Figure 7.7, which shows one thumb data for the same task.

We see very similar results to those observed in the one thumb case. The RVM is better than the baseline for any amount of data, and outperforms a pooled data quadratic model for 20 or more training points. The user specific quadratic and kernel regression models are not better than the pooled data model until 50 training points are used. Performances for the three user specific models converge at 100 training points. Interestingly, the baseline accuracy on this data is lower than in the one thumb case. This may be because when interacting with one thumb, the user sometimes has to stretch across the entire width of the screen to touch, leading to a large offset. This problem is avoided when touching with both thumbs.

7.4 Conclusion

In this Chapter, we considered the training data requirements for offset models. Existing research has shown differences in touch behaviour based on factors such as grip, mobility condition, angle of the screen with the respect to the user, and others. Here, we extend this by considering the relationship between touch task and user offset behaviour.

We collected data in a remote online study for 16 participants performing 6 tasks, including a range of pointing tasks, pressing keys on a soft keyboard, and a text transcription task. We find that training on a given task always gives higher performance when testing on that task than training on data from any other. Further, using models trained on typing data to predict offsets for pointing tasks, or vice versa, actually leads to worse performance than not correcting.

We were able to explain the observed differences between tasks in terms of the complexity of the offset functions for the various tasks — in general, typing offsets are the most complex, followed by keyboard button presses, and finally pointing offsets are simplest. Using a model trained on one class of data to predict on another class with different complexity tends to make things worse, or at least reduce the size of the potential improvement. Clearly, calibration task is an important consideration when developing offset models, and should be tailored to the intended task.

Given that we now know task is an important factor in addition to physical considerations like hand posture and mobility condition, it becomes apparent that a general touch offset model is almost impossible to construct. Instead, we need many models for the different tasks and usage conditions. Existing offset correction systems have used hundreds or thousands of training points. To reduce the burden on the user of providing training data for many models, we also studied the use of the Relevance Vector Machine as an alternative probabilistic algorithm for touch. The RVM is a powerful tool because it is sparse in terms of training data — models can be trained with far less data than other techniques.

We found that for one-handed usage, models with good predictive performance could be learned with 10 or fewer training points, and that best performance was reached by the time 50 training points were used. For two-handed use, we found that good predictive performance was still possible with very few training points. Additionally, we found more points were needed to accurately model typing data than a pointing task, confirming the increased complexity in typing offsets discussed in the first part of the Chapter. These results make the RVM a potentially useful to train these models in almost real time as a new user interacts with a device, which is not necessarily possible with existing offset models.

We also found that the locations of points used by the RVM to train offset models were conserved across users and devices. This could be used to inform the design of interfaces, or to aid in selecting training data locations for future offset models.

While the work in this Chapter was motivated by a desire to investigate the practicalities of a wide deployment of offset models, we believe that the results found are independently interesting. The variation in touch behaviour across tasks in particular is a novel result, with broad implications as touch technology continues to grow in popularity and appear in more devices which users encounter daily.

Chapter 8

Conclusions

Touch is a ubiquitous input modality on modern smartphones and is increasingly popular on devices such as laptops and smartwatches. Despite how common this modality is, the underlying technology is still quite error prone. As discussed in Chapter 2, there are a number of factors which contribute to this. These include the ambiguous contact area between the user's fingers and the screen, and the visual occlusion of small targets by the hand. Touch errors manifest themselves as an offset between the user's intended target and the location recorded by the device. A number of models have been proposed to try and compensate for these offsets.

However, touch errors are not entirely systematic. They also exhibit a random component. If a user tries to acquire the same target multiple times, their touches will define a distribution. This means that touch is subject to significant uncertainty.

There may be other forms of uncertainty in play for touch interaction, such as uncertainty of intent — for example, when typing a user might be uncertain about the spelling of a word, leading to bad input. Existing models of touch tend to ignore the uncertainty, and those that do take into account (e.g. [64]) do so at a global, rather than user-specific, level. However, it has been shown that there is significant variation across users in terms of touch behaviour [19].

In this thesis, we have proposed these use of machine learning techniques to model touch offsets and uncertainty on a per user basis. We looked at Gaussian Process regression, a flexible non-parametric algorithm, and the Relevance Vector Machine, a related technique which requires very little training data. Using this we built offset models which allowed accurate selection of targets smaller than the unaided finger can reliably acquire.

Our goal in using these algorithms was to show that by modelling the uncertainty, we can provide measurable benefits to the user. We consider two complementary approaches, one in which the uncertainty model is hidden from the user and one that gives the user control over

the uncertainty. In this thesis we have shown that these models can be used to improve text entry accuracy, but more generally we believe our work stands as an example of the value of modelling uncertainty in touch.

8.1 Summary of Contributions

This work makes a number of contributions, which were motivated by our thesis statement in Section 1.1. We restate this again here to remind the reader:

> Touch interaction is ubiquitous on modern mobile devices, but is still prone to inaccuracy and error. This is due in large part to the inherent uncertainty related to touch, which stems from a number of sources: ambiguity from the soft finger touching small targets; uncertainty introduced by the hardware; and uncertainty of intent (e.g. the user being unsure of a word's spelling while typing). However, this uncertainty is typically neglected in current touch systems. The core assertion of this work is that by modelling this uncertainty, the quality of touch interactions can be improved.
>
> We support this assertion through a number of experiments. In particular, we propose that probabilistic modelling of the offsets between a users intended and recorded touch locations can be used to allow more accurate selection of small targets. Further, this technique can also improve the accuracy of typing on a soft keyboard. This improvement is shown through the results of a number of user studies, highlighting the benefit of implicit uncertainty modelling. Additionally, this approach is contrasted with an explicit uncertainty model, in which the user can control the ability of the system to correct their input. We show that both approaches have merit and can lead to increased performance in the text entry task. Finally, we present the results of several studies into the training data requirements of the models described in the rest of the thesis, and a sparse solution which potentially allows these models to be deployed in the real world with small training sets.

In support of the assertion that uncertainty modelling improves touch, we have made the following contributions:

- Chapter 4 presented a flexible offset model using Gaussian Process regression which describes the non-linear, user-specific behaviour seen in touch in a fully probabilistic way. This was shown to allow accurate selection of small targets. We also demonstrated that user specific touch models outperform those trained on data from many users.

- Chapter 5 presented a text entry system which combines our GP touch model with a long span statistical language model. This makes use of the probabilistic nature of the GP predictions to obtain probability distributions over keyboard keys. Using this system, we obtained text correction performance comparable with the state of the art. This highlights a particular example of the benefits of uncertainty models.

- Chapter 6 presented a study of the use of input pressure as a mechanism for negotiating uncertainty with a text entry system, in order to prevent autocorrections. This is a model which gives explicit control of the uncertainty to the user, and our results show that not only are users able to control such a model, they can use it to give performance improvements when entering text unknown to a correction system.

- Chapter 7 presents a thorough examination of the differences in touch behaviour across different tasks. We show that offsets are different for a typing task than for a range of pointing tasks, and explain this by demonstrating that the offset surfaces for the typing task are more complex. This chapter also gave an exploration of the use of sparse machine learning techniques to learn touch models with very small amounts of training data. In particular, we used the Relevance Vector Machine model to analyse a number of datasets, showing that in the best case models with good predictive power could be learned with 10 or fewer training examples. We further showed that the locations of points which are used by these sparse models are highly conserved across users and devices, which has potential implications for interface design.

8.2 Future Work

There are a number of interesting directions for future work which could follow from the results presented in this thesis. This Section will describe these.

8.2.1 Combining GPType and ForceType

In this thesis, we presented implicit and explicit uncertainty models separately. GPType in Chapter 5 implicitly modelled the uncertainty in the user's touches to obtain predictive distributions over keys on a soft keyboard, and used these to correct text. ForceType in Chapter 6 gave users explicit control over their uncertainty so that they could prevent the system from correcting words that were not in its vocabulary.

These approaches, however, are not at odds with one another, but rather could be used together. The robust offset model of GPType corrects for many errors and allows more interesting baseline touch distributions than the simple ones used in ForceType. ForceType

could thus be applied on top of GPType, with pressure allowing the GP distributions to be modified to prevent any corrections. The correction algorithms are very similar, so the combination would be easy to implement from an algorithmic standpoint. The system would just apply the mean GP offset for the recorded touch position, and the predictive covariance from the GP would correspond to an 'average' touch. The predictive distribution could then be broadened or narrowed depending on pressure.

Implementing this combination of models and measuring how its performance compares to the individual techniques is a useful area for future work. To do this, it would be useful to implement ForceType on an existing smartphone using one of the pressure approximations found in the literature, or alternatively using a pressure proxy like touch time or area.

This is not such a straightforward proposition, as the practicalities of this remain to be resolved — for example, existing pressure sensing methods on phones have operated by pulsing the vibration motor and measuring the dampening of the response. Having the phone pulse while typing, for example, may be unpleasant or distracting to the user. An alternative approach would be to augment a phone with force sensors, although this means that the solution would not be widely deployable. Commercial entities including Apple and Qualcomm have pending patent applications for capacitive screens with built in force-sensors: if these products reach the market, then the task of implementing ForceType on a regular device will become significantly easier.

8.2.2 Offset Models for Novice Users

We found in Chapter 4 that the user in our study who had never used a smartphone before had high baseline error and received a large improvement from applying the offset model. It would be interesting to investigate whether this is true for novice users in general. If so, this has potential benefits for rapidly improving touch performance for people who have not used touch devices — for example, elderly users.

Related to this, the use of ForceType is conditioned on having some kind of mental model of which words will and won't be corrected, which novice users typically won't have. Therefore it would be interesting to see how ForceType is used by novices, and if there are any performance benefits for these users.

8.2.3 Further Analysis of Sensor Data

In Chapter 4, we compared touch models which predicted intended positions from capacitive sensor data to those which predicted offsets from device positions. The latter class of model was shown to require less data to make good predictions, and was used throughout the rest

of the thesis because capacitive sensor values are not directly available on the majority of phones.

However, an advantage of sensor data based models is that they avoid issues of mapping different inputs to the same output, since the sensor space is much larger and more expressive than the device location space. Given our later observations about the variation of offsets over task, and existing research about the effect of hand posture on touch behaviour, it would be interesting to see whether a single model based on sensor data would be able to handle these variations. It may be that the sensors are expressive enough that we could sidestep the need for multiple different offset models for a range of contexts.

8.2.4 Effect of Visual Feedback on Touch Behaviour

One interesting observation from the work in Chapter 7 was that offset models trained on key press data (gathered with instructions like 'Press E') did not make good predictions on typing data. However, in Chapter 5 we trained such a model and found that it gave an improvement over the uncorrected baseline for typing data.

The primary difference between the conditions for these two studies was that in the former, the user did not see which keys they had typed while in the latter they did. It may be that this difference in visual feedback caused a change in the observed touch behaviour, even over the short sessions in the studies.

Therefore an interesting line for future work is a comparison between touch tasks with different levels of visual feedback, to see whether this feedback has a systematic effect on touch behaviour.

8.2.5 Online RVM Training

A final area for potential future work is a real world implementation of our RVM offset model. In Chapter 7, we showed the existence of consistent distributions over relevance vector locations. It would be interesting to compare an approach sampling touch points from these distributions to train the model versus one which takes training points at random. Further, it would be interesting to see in a real setting how iteratively training these models improves touch accuracy. This could be done by presenting a task where the user has to select one from a grid of closely packed targets, and training RVMs on the fly as more and more data are collected. The touch accuracy over time could be monitored, giving an indication of how rapidly we see an improvement using the RVM.

8.3 Summary and Conclusions

Touch is a complicated process, and subject to a great deal of uncertainty. In this thesis we have shown how a robust model of this uncertainty can be used to improve touch accuracy. We have built probabilistic offset models and used their predictions to improve text entry accuracy. Further, we have shown that users can gain performance benefits from a system in which they can control the level of certainty they express to the system.

While there are still outstanding issues to consider, we believe that the work herein presents a compelling case for the power of probabilistic techniques in solving problems in this domain. Uncertainty is often neglected in interaction, and our results show that this results in a loss of useful information which can be used to improve the quality of interaction with mobile devices.

Bibliography

[1] N. Henze, E. Rukzio, and S. Boll, "100,000,000 taps: Analysis and Improvement of Touch Performance in the Large," in *MobileHCI '11*. ACM Press, 2011, pp. 133–142. [Online]. Available: http://dl.acm.org/citation.cfm?id=2037373.2037395

[2] E. A. Johnson, "Touch Displays: A Programmed Man-Machine Interface," *Ergonomics*, vol. 10, no. 2, pp. 271–277, Mar. 1967. [Online]. Available: http://dx.doi.org/10.1080/00140136708930868

[3] F. Beck and B. Stumpe, "Two devices for operator interaction in the central control of the new CERN accelerator," CERN, Geneva, Switzerland, Tech. Rep., 1973. [Online]. Available: http://cds.cern.ch/record/186242?ln=en

[4] F. Ebeling, R. Johnson, and R. Goldhor, "Infrared Light Beam X-Y Position Encoder for Display Devices," 1973.

[5] W. C. Colwell Jr and G. S. Hurst, "Discriminating contact sensor," Oct. 1975. [Online]. Available: http://www.google.com/patents/US3911215

[6] N. Mehta, "A Flexible Machine Interface," M.A.Sc. Thesis, University of Toronto, 1982.

[7] M. W. Krueger, T. Gionfriddo, and K. Hinrichsen, "VIDEOPLACE—an artificial reality," in *CHI '85*, vol. 16, no. 4. ACM, Apr. 1985, pp. 35–40. [Online]. Available: http://dl.acm.org/citation.cfm?id=1165385.317463

[8] P. Wellner, "The DigitalDesk calculator," in *Proceedings of the 4th annual ACM symposium on User interface software and technology - UIST '91*. New York, New York, USA: ACM Press, Nov. 1991, pp. 27–33. [Online]. Available: http://dl.acm.org/citation.cfm?id=120782.120785

[9] W. Buxton, "Living in Augmented Reality: Ubiquitous Media and Reactive Environments," in *Video Mediated Communication*, K. Finn, A. Sellen, and S. Wilber, Eds. Hillsdale, NJ: Erlbaum, 1997, pp. 363–384.

[10] P. Dietz and D. Leigh, "DiamondTouch," in *Proceedings of the 14th annual ACM symposium on User interface software and technology - UIST '01*. New York, New York, USA: ACM Press, Nov. 2001, pp. 219–226. [Online]. Available: http://dl.acm.org/citation.cfm?id=502348.502389

[11] M. Wu and R. Balakrishnan, "Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays," in *Proceedings of the 16th annual ACM symposium on User interface software and technology - UIST '03*. New York, New York, USA: ACM Press, Nov. 2003, pp. 193–202. [Online]. Available: http://dl.acm.org/citation.cfm?id=964696.964718

[12] S. Hodges, S. Izadi, A. Butler, A. Rrustemi, and B. Buxton, "ThinSight: versatile multi-touch sensing for thin form-factor displays," in *Proceedings of the 20th annual ACM symposium on User interface software and technology - UIST '07*. New York, New York, USA: ACM Press, Oct. 2007, pp. 259—-268. [Online]. Available: http://dl.acm.org/citation.cfm?id=1294211.1294258

[13] Y.-J. Hsieh, C.-W. Chen, H.-E. Tsai, and J.-L. Hsu, "Multi-touch method for resistive touch panel," Nov. 2010. [Online]. Available: http://www.google.com/patents/US20100283748

[14] S. Lee, "A Fast Multiple-Touch-Sensitive Input Device," Master's Thesis, University of Toronto, 1984.

[15] S. Lee, W. Buxton, and K. C. Smith, "A multi-touch three dimensional touch-sensitive tablet," in *CHI '85*, vol. 16, no. 4. ACM, Apr. 1985, pp. 21–25. [Online]. Available: http://dl.acm.org/citation.cfm?id=1165385.317461

[16] W. Westerman, "Hand Tracking, Finger Identification, and Chordic Manipulation on a Multi-Touch Surface," PhD Thesis, University of Delaware, 1999.

[17] J. Rekimoto, "SmartSkin: an infrastructure for freehand manipulation on interactive surfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI '02*. New York, New York, USA: ACM Press, Apr. 2002, pp. 113–120. [Online]. Available: http://dl.acm.org/citation.cfm?id=503376.503397

[18] A. Sears, C. Plaisant, and B. Shneiderman, "A New Era for High-Precision Touchscreens," in *Advances in Human-Computer Interaction (Volume 3)*, 1992, vol. 3, pp. 1–33. [Online]. Available: http://www.cs.umd.edu/local-cgi-bin/hcil/rr.pl?number=90-01

[19] D. B. Beringer and J. G. Peterson, "Underlying Behavioral Parameters of the Operation of Touch-Input Devices: Biases, Models, and Feedback," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 27, no. 4, pp. 445–458, Aug. 1985. [Online]. Available: http://hfs.sagepub.com/content/27/4/445

[20] A. Sears and B. Shneiderman, "High precision touchscreens: design strategies and comparisons with a mouse," *International Journal of Man-Machine Studies*, vol. 34, no. 4, pp. 593–613, Apr. 1991. [Online]. Available: http://dl.acm.org/citation.cfm?id=105291.105301

[21] A. Sears, "Improving Touchscreen Keyboards: Design issues and a comparison with other devices," *Interacting with Computers*, vol. 3, no. 3, pp. 252–269, 1991. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.1352

[22] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement." *Journal of Experimental Psychology*, vol. 47, no. 6, pp. 381–391, 1954.

[23] A. Sears and Y. Zha, "Data Entry for Mobile Devices Using Soft Keyboards: Understanding the Effects of Keyboard Size and User Tasks," *International Journal of Human-Computer Interaction*, vol. 16, no. 2, pp. 163–184, 2003.

[24] S. Mizobuchi, M. Chignell, and D. Newton, "Mobile text entry: Relationship between Walking Speed and Text Input Task Difficulty," in *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services - MobileHCI '05*. New York, New York, USA: ACM Press, Sep. 2005, pp. 122–128. [Online]. Available: http://dl.acm.org/citation.cfm?id=1085777.1085798

[25] S. Brewster, "Overcoming the Lack of Screen Space on Mobile Computers," *Personal and Ubiquitous Computing*, vol. 6, no. 3, pp. 188–205, May 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=594351.594356

[26] I. S. MacKenzie and S. X. Zhang, "An empirical investigation of the novice experience with soft keyboards," *Behaviour & Information Technology*, vol. 20, pp. 411–418, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.7771

[27] P. Parhi, A. K. Karlson, and B. B. Bederson, "Target size study for one-handed thumb use on small touchscreen devices," in *MobileHCI '06*, Sep. 2006, pp. 203–210.

[28] A. Roudaut, S. Huot, and E. Lecolinet, "TapTap and MagStick," in *Proceedings of the working conference on Advanced visual interfaces - AVI '08*. New

York, New York, USA: ACM Press, May 2008, p. 146. [Online]. Available: http://dl.acm.org/citation.cfm?id=1385569.1385594

[29] D. Vogel and P. Baudisch, "Shift: a technique for operating pen-based interfaces using touch," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*. New York, New York, USA: ACM Press, Apr. 2007, p. 657. [Online]. Available: http://dl.acm.org/citation.cfm?id=1240624.1240727

[30] C. Holz and P. Baudisch, "Understanding touch," in *CHI '11*, 2011, pp. 2501–2510.

[31] F. Wang and X. Ren, "Empirical evaluation for finger input properties in multi-touch interaction," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, New York, USA: ACM Press, Apr. 2009, p. 1063. [Online]. Available: http://dl.acm.org/citation.cfm?id=1518701.1518864

[32] S. Azenkot and S. Zhai, "Touch behavior with different postures on soft smartphone keyboards," in *MobileHCI '12*, pp. 251–260. [Online]. Available: http://dl.acm.org/citation.cfm?id=2371574.2371612

[33] M. Goel, L. Findlater, and J. Wobbrock, "WalkType: Using Accelerometer Data to Accomodate Situational Impairments in Mobile Touch Screen Text Entry," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. New York, New York, USA: ACM Press, 2012, pp. 2687–2696. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2207676.2208662

[34] K. A. Siek, Y. Rogers, and K. H. Connelly, "Fat Finger Worries: How Older and Younger Users Physically Interact with PDAs," in *Proceedings of Human-Computer Interaction - INTERACT'05*, 2005, pp. 267–280. [Online]. Available: http://www.springerlink.com/index/aphl5wkd04yk3hmq.pdf

[35] DTI, "Consumer and Competition Policy Directorate: Specific Anthropometric and strength data for people with dexterity disability," Department of Trade and Industry, London, UK, Tech. Rep., 2002.

[36] D. Wigdor, C. Forlines, P. Baudisch, J. Barnwell, and C. Shen, "Lucid touch: a see-through mobile device," in *Proceedings of the 20th annual ACM symposium on User interface software and technology - UIST '07*. New York, New York, USA: ACM Press, Oct. 2007, p. 269. [Online]. Available: http://dl.acm.org/citation.cfm?id=1294211.1294259

[37] S. Rogers, J. Williamson, C. Stewart, and R. Murray-Smith, "AnglePose: robust, precise capacitive touch tracking via 3d orientation estimation," in *CHI '11*. ACM, pp. 2575–2584. [Online]. Available: http://portal.acm.org/citation.cfm?id=1979318

[38] M. R. Lee, C. M. Hammer, and D. R. Seguine, "Temperature compensation method for capacitive sensors," 2008. [Online]. Available: http://www.google.com/patents/US20080047764

[39] C. Holz and P. Baudisch, "The Generalized Perceived Input Point Model and How to Double Touch Accuracy by Extracting Fingerprints," in *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. New York, New York, USA: ACM Press, 2010, pp. 581–590. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1753326.1753413

[40] X. Bi, Y. Li, and S. Zhai, "FFitts law: Modeling Finger Touch with Fitts' Law," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. New York, New York, USA: ACM Press, Apr. 2013, p. 1363. [Online]. Available: http://dl.acm.org/citation.cfm?id=2470654.2466180

[41] T. Grossman and R. Balakrishnan, "The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*. New York, New York, USA: ACM Press, Apr. 2005, pp. 281–290. [Online]. Available: http://dl.acm.org/citation.cfm?id=1054972.1055012

[42] R. L. Potter, L. J. Weldon, and B. Shneiderman, "Improving the accuracy of touch screens: an experimental evaluation of three strategies," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '88*. New York, New York, USA: ACM Press, May 1988, pp. 27–32. [Online]. Available: http://dl.acm.org/citation.cfm?id=57167.57171

[43] Y. S. Park, S. H. Han, J. Park, and Y. Cho, "Touch key design for target selection on a mobile phone," in *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services - MobileHCI '08*. New York, New York, USA: ACM Press, Sep. 2008, p. 423. [Online]. Available: http://dl.acm.org/citation.cfm?id=1409240.1409304

[44] A. Roudaut, H. Pohl, and P. Baudisch, "Touch input on curved surfaces," in *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*. New York, New York, USA: ACM Press, May 2011, p. 1011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1978942.1979094

[45] H. Benko, A. D. Wilson, and R. Balakrishnan, "Sphere," in *Proceedings of the 21st annual ACM symposium on User interface software and technology - UIST '08*. New York, New York, USA: ACM Press, Oct. 2008, p. 77. [Online]. Available: http://dl.acm.org/citation.cfm?id=1449715.1449729

[46] N. Henze, E. Rukzio, and S. Boll, "Observational and experimental investigation of typing behaviour using virtual keyboards for mobile devices," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. ACM, 2012, pp. 2659–2668. [Online]. Available: http://dl.acm.org/citation.cfm?id=2208636.2208658

[47] D. Buschek, S. Rogers, and R. Murray-Smith, "User-Specific Touch Models in a Cross-Device Context," in *MobileHCI '13*, 2013, pp. 382–391.

[48] P.-A. Albinsson and S. Zhai, "High precision touch screen interaction," in *Proceedings of the conference on Human factors in computing systems - CHI '03*. New York, New York, USA: ACM Press, Apr. 2003, p. 105. [Online]. Available: http://dl.acm.org/citation.cfm?id=642611.642631

[49] A. Olwal and S. Feiner, "Rubbing the Fisheye: Precise Touch-Screen Interaction with Gestures and Fisheye Views," in *Conference Supplement of UIST '03*, 2003, pp. 83–84. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.2582

[50] X. Ren and S. Moriya, "Improving selection performance on pen-based systems: a study of pen-based interaction for selection tasks," *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 3, pp. 384–416, Sep. 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=355324.355328

[51] A. Esenther and K. Ryall, "Fluid DTMouse," in *Proceedings of the working conference on Advanced visual interfaces - AVI '06*. New York, New York, USA: ACM Press, May 2006, p. 112. [Online]. Available: http://dl.acm.org/citation.cfm?id=1133265.1133289

[52] H. Benko, A. D. Wilson, and P. Baudisch, "Precise selection techniques for multi-touch screens," in *Proceedings of the SIGCHI conference on Human Factors in computing systems - CHI '06*. New York, New York, USA: ACM Press, Apr. 2006, p. 1263. [Online]. Available: http://dl.acm.org/citation.cfm?id=1124772.1124963

[53] P. Baudisch and G. Chu, "Back-of-device interaction allows creating very small touch devices," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, New York, USA: ACM Press, Apr. 2009, p. 1923. [Online]. Available: http://dl.acm.org/citation.cfm?id=1518701.1518995

[54] K. A. Li, P. Baudisch, and K. Hinckley, "Blindsight: eyes-free access to mobile phones," in *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*. New York, New

York, USA: ACM Press, Apr. 2008, pp. 1389–1398. [Online]. Available: http://dl.acm.org/citation.cfm?id=1357054.1357273

[55] M. Sugimoto and K. Hiroki, "HybridTouch: an intuitive manipulation technique for PDAs using their front and rear surfaces," in *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services - MobileHCI '06*. New York, New York, USA: ACM Press, Sep. 2006, p. 137. [Online]. Available: http://dl.acm.org/citation.cfm?id=1152215.1152243

[56] D. Wigdor, D. Leigh, C. Forlines, S. Shipman, J. Barnwell, R. Balakrishnan, and C. Shen, "Under the table interaction," in *Proceedings of the 19th annual ACM symposium on User interface software and technology - UIST '06*. New York, New York, USA: ACM Press, Oct. 2006, p. 259. [Online]. Available: http://dl.acm.org/citation.cfm?id=1166253.1166294

[57] T. Ohtani, T. Hashida, Y. Kakehi, and T. Naemura, "Comparison of front touch and back touch while using transparent double-sided touch display," in *ACM SIGGRAPH 2011 Posters on - SIGGRAPH '11*. New York, New York, USA: ACM Press, Aug. 2011, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2037715.2037764

[58] M. F. Mohd Noor, A. Ramsay, S. Hughes, S. Rogers, J. Williamson, and R. Murray-Smith, "28 frames later: predicting screen touches from back-of-device grip changes," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, Apr. 2014, pp. 2005–2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=2556288.2557148

[59] O. Schoenleben and A. Oulasvirta, "Sandwich keyboard: fast ten-finger typing on a mobile device with adaptive touch sensing on the back side," in *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services - MobileHCI '13*. New York, New York, USA: ACM Press, Aug. 2013, pp. 175–178. [Online]. Available: http://dl.acm.org/citation.cfm?id=2493190.2493233

[60] D. Buschek, O. Schoenleben, and A. Oulasvirta, "Improving accuracy in back-of-device multitouch typing: a clustering-based approach to keyboard updating," in *Proceedings of the 19th international conference on Intelligent User Interfaces - IUI '14*. New York, New York, USA: ACM Press, Feb. 2014, pp. 57–66. [Online]. Available: http://dl.acm.org/citation.cfm?id=2557500.2557501

[61] A. Doucet, N. De Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice*. Springer Verlag, 2001.

[62] S. Rogers, J. Williamson, C. Stewart, and R. Murray-Smith, "FingerCloud: Uncertainty and Autonomy Handover in Capacitive Sensing," in *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. New York, New York, USA: ACM Press, 2010, pp. 577–580. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753412

[63] K. Go and Y. Endo, "CATKey: customizable and adaptable touchscreen keyboard with bubble cursor-like visual feedback," in *Interact '07*. Springer-Verlag, Sep. 2007, pp. 493–496. [Online]. Available: http://dl.acm.org/citation.cfm?id=1776994.1777055

[64] X. Bi and S. Zhai, "Bayesian touch: a statistical criterion of target selection with finger touch," in *UIST '13*, Oct. 2013, pp. 51–60.

[65] J. Goodman, G. Venolia, K. Steury, and C. Parker, "Language modeling for soft keyboards," in *Proceedings of the 7th international conference on Intelligent user interfaces - IUI '02*. New York, New York, USA: ACM Press, 2002, pp. 194–195. [Online]. Available: http://portal.acm.org/citation.cfm?doid=502716.502753

[66] S. Zhai and P.-O. Kristensson, "Shorthand writing on stylus keyboard," in *Proceedings of the conference on Human factors in computing systems - CHI '03*. New York, New York, USA: ACM Press, Apr. 2003, pp. 97–104. [Online]. Available: http://dl.acm.org/citation.cfm?id=642611.642630

[67] P.-O. Kristensson and S. Zhai, "SHARK: A Large Vocabulary Shorthand Writing System for Pen-Based Computers," in *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*. New York, New York, USA: ACM Press, 2004, pp. 43–52. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1029632.1029640

[68] ——, "Relaxing Stylus Typing Precision by Geometric Pattern Matching," in *Proceedings of the 10th international conference on Intelligent user interfaces - IUI '05*. New York, New York, USA: ACM Press, 2005, pp. 151–158. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1040830.1040867

[69] J. Himberg, J. Häkkilä, P. Kangas, and J. Mäntyjärvi, "On-line personalization of a touch screen based keyboard," in *Proceedings of the 8th international conference on Intelligent user interfaces - IUI '03*. New York, New York, USA: ACM Press, Jan. 2003, p. 77. [Online]. Available: http://dl.acm.org/citation.cfm?id=604045.604061

[70] L. Findlater and J. O. Wobbrock, "Personalized Input: Improving Ten-Finger Touchscreen Typing through Automatic Adaptation," in *CHI '12*, pp. 815–824. [Online]. Available: http://terpconnect.umd.edu/~leahkf/pubs/CHI2012-findlater-PersonalizedTyping.pdf

[71] K. Al Faraj, M. Mojahid, and N. Vigouroux, "BigKey: A Virtual Keyboard for Mobile Devices," in *HCI '09 Part III: Ubiquitous and Intelligent Interaction*, ser. Lecture Notes in Computer Science, J. A. Jacko, Ed., vol. 5612. Berlin, Heidelberg: Springer Berlin Heidelberg, Jul. 2009, pp. 3–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1601265.1601267

[72] A. Gunawardana, T. Paek, and C. Meek, "Usability Guided Key-Target Resizing for Soft Keyboards," in *Proceedings of the 15th international conference on Intelligent user interfaces - IUI '10*. New York, New York, USA: ACM Press, 2010, pp. 111–118. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1719970.1719986

[73] D. Rudchenko, T. Paek, and E. Badger, "Text Text Revolution: A Game That Improves Text Entry on Mobile Touchscreen Keyboards," in *Proceedings of the 9th International Conference on Pervasive Computing - Pervasive '11*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 206–213. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-642-21726-5_13

[74] F. O. Flemisch, C. A. Adams, S. R. Conway, K. H. Goodrich, M. T. Palmer, and P. C. Schutte, "The H-Metaphor as a Guideline for Vehicle Automation and Interaction," NASA, Tech. Rep., Jan. 2003. [Online]. Available: http://ntrs.nasa.gov/search.jsp?R=20040031835

[75] J. Williamson, "Continuous Uncertain Interaction," PhD Thesis, University of Glasgow, 2006. [Online]. Available: http://www.dcs.gla.ac.uk/~jhw/thesis/index.html

[76] J. Schwarz, S. Hudson, J. Mankoff, and A. D. Wilson, "A framework for robust and flexible handling of inputs with uncertainty," in *Proceedings of the 23nd annual ACM symposium on User interface software and technology (UIST '10)*. ACM, 2010, pp. 47–56. [Online]. Available: http://portal.acm.org/citation.cfm?id=1866039

[77] J. Schwarz, J. Mankoff, and S. Hudson, "Monte carlo methods for managing interactive state, action and feedback under uncertainty," in *UIST '11*. ACM Press, p. 235. [Online]. Available: http://dl.acm.org/citation.cfm?id=2047196.2047227

[78] H. Pohl and R. Murray-Smith, "Focused and Casual Interactions: Allowing Users to Vary Their Level of Engagement," in *Proceedings of the 31th international conference on Human factors in computing systems - CHI'13*, 2013.

[79] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006. [Online]. Available: http://www.amazon.co.uk/Gaussian-Processes-Machine-Learning-Rasmussen/dp/026218253X

[80] M. E. Tipping, "Sparse Bayesian Learning and the Relevance Vector Machine," *Journal of Machine Learning Research*, vol. 1, pp. 211–244, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.1089

[81] J. Quinonero Candela, "Learning with Uncertainty Gaussian Processes and Relevance Vector Machines," PhD Thesis, Technical University of Denmark, 2004.

[82] M. E. Tipping and A. Faul, "Fast Marginal Likelihood Maximisation for Sparse Bayesian Models," in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003, pp. 3–6. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1199

[83] A. C. Faul and M. E. Tipping, "Analysis of Sparse Bayesian Learning," in *Advances in Neural Information Processing Systems 14 - NIPS '01*. MIT Press, 2001, pp. 383–389. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.9801

[84] C. K. I. Williams, "Learning in Graphical Models," in *Learning in Graphical Models*, M. I. Jordan, Ed. Dordrecht: Springer Netherlands, 1998, pp. 599–621. [Online]. Available: http://link.springer.com/10.1007/978-94-011-5014-9

[85] C. E. Rasmussen and J. Quinonero Candela, "Healing the relevance vector machine through augmentation," in *Proceedings of the 22nd International Conference on Machine Learning*, 2005, pp. 150–176. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.78.1127

[86] S. Rogers and M. Girolami, *A First Course in Machine Learning*. CRC Press, 2011.

[87] H. Nicolau and J. Jorge, "Touch typing using thumbs," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. New York, New York, USA: ACM Press, May 2012, p. 2683. [Online]. Available: http://dl.acm.org/citation.cfm?id=2207676.2208661

[88] G. P. Basharin, A. N. Langville, and V. A. Naumov, "The life and work of A.A. Markov," *Linear Algebra and its Applications*, vol. 386, pp. 3–26, Jul. 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0024379504000357

[89] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, Jan. 1948. [Online]. Available: http://dl.acm.org/citation.cfm?id=584091.584093

[90] F. Jelinek, *Statistical Methods for Speech Recognition*. MIT Press, 1997. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=1C9dzcJTWowC&pgis=1

[91] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 400–401, Mar. 1987. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1165125

[92] R. Kneser and H. Ney, "Improved backing-off for M-gram language modeling," in *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 1995, pp. 181–184. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=479394

[93] I. Witten and T. Bell, "The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression," *IEEE Transactions on Information Theory*, vol. 37, no. 4, pp. 1085–1094, Jul. 1991. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=87000

[94] H. Ney and U. Essen, "On smoothing techniques for bigram-based natural language modelling," in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1991, pp. 825–828 vol.2. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=150464

[95] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–393, Oct. 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0885230899901286

[96] A. Stolcke, "Entropy-based pruning of backoff language models," in *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, 2000, pp. 8—-11. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.157.819

[97] C. Chelba, T. Brants, W. Neveitt, and P. Xu, "Study on Interaction between Entropy Pruning and Kneser-Ney Smoothing," in *Interspeech 2010*, 2010, pp. 2242—-2245. [Online]. Available: http://research.google.com/pubs/pub36472.html

[98] R. Kuhn, "Speech recognition and the frequency of recently used words," in *Proceedings of the 12th conference on Computational linguistics* -, vol. 1. Morristown, NJ, USA: Association for Computational Linguistics, Aug. 1988, pp. 348–350. [Online]. Available: http://dl.acm.org/citation.cfm?id=991635.991706

[99] P. F. Brown, P. V. DeSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n -gram models of natural language," *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, Dec. 1992. [Online]. Available: http://dl.acm.org/citation.cfm?id=176313.176316

[100] R. Iyer and M. Ostendorf, "Modeling long distance dependence in language: topic mixtures versus dynamic cache models," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 1, pp. 30–39, 1999. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=736328

[101] H. Ney, U. Essen, and R. Kneser, "On Structuring Probabilistic Dependences in Stochastic Language Modeling - Microsoft Research," *Computer Speech & Language*, vol. 8, pp. 1—-38, 1994. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=64973

[102] J. Goodman, "Putting it all together: language model combination," in *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. ICASSP '00*. IEEE, 2000, pp. 1647—-1650.

[103] A. Oulasvirta, A. Reichel, W. Li, Y. Zhang, M. Bachynskzi, K. Vertanen, and P. O. Kristensson, "Improving Two-Thumb Text Entry on Touchscreen Devices," in *Proceedings of the 31st ACM Conference on Human Factors in Computing Systems - CHI '13*, 2013.

[104] M. Lui and T. Baldwin, "langid.py: an off-the-shelf language identification tool," pp. 25–30, Jul. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2390470.2390475

[105] A. Stolcke, "SRILM- an extensible language modeling toolkit," in *Interspeech 2002*, 2002.

[106] K. Vertanen and P. O. Kristensson, "A versatile dataset for text entry evaluations based on genuine mobile emails," in *MobileHCI '11*, Aug. 2011, pp. 295–298. [Online]. Available: http://dl.acm.org/citation.cfm?id=2037373.2037418

[107] I. S. MacKenzie and R. W. Soukoreff, "Phrase sets for evaluating text entry techniques," in *CHI '03 extended abstracts on Human factors in computing systems - CHI '03*. New York, New York, USA: ACM Press, Apr. 2003, p. 754. [Online]. Available: http://dl.acm.org/citation.cfm?id=765891.765971

[108] P. O. Kristensson and K. Vertanen, "Performance Comparisons of Phrase Sets and Presentation Styles for Text Entry Evaluations," in *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces - IUI '12*. New York, New York, USA: ACM Press, 2012, pp. 29–32. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2166966.2166972

[109] M. A. Srinivasan and J.-s. Chen, "Human Performance in Controlling Normal Forces of Contact with Rigid Objects," in *Proc. Advances in Robotics, Mechatronics, and Haptic Interfaces - ASME'93*, vol. 49, 1993, pp. 119–125.

[110] S. Subramanian, D. Aliakseyeu, and A. Lucero, "Multi-layer interaction for digital tables," in *UIST '06*, Oct. 2006, pp. 269–272. [Online]. Available: http://dl.acm.org/citation.cfm?id=1166253.1166295

[111] G. Ramos, M. Boulos, and R. Balakrishnan, "Pressure Widgets," in *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*. New York, New York, USA: ACM Press, 2004, pp. 487–494. [Online]. Available: http://portal.acm.org/citation.cfm?doid=985692.985754

[112] S. Heo and G. Lee, "Force Gestures: Augmenting Touch Screen Gestures with Normal and Tangential Forces," in *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*. New York, New York, USA: ACM Press, 2011, pp. 621–626. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2047196.2047278

[113] C. Stewart, M. Rohs, S. Kratz, and G. Essl, "Characteristics of Pressure-Based Input for Mobile Devices," in *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. New York, New York, USA: ACM Press, 2010, pp. 801–810. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753444

[114] D. C. McCallum, E. Mak, P. Irani, and S. Subramanian, "PressureText: Pressure Input for Mobile Phone Text Entry," in *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems - CHI EA '09*. New York, New York, USA: ACM Press, 2009, pp. 4519–4524. [Online]. Available: http://dl.acm.org/citation.cfm?id=1520693http://portal.acm.org/citation.cfm?doid=1520340.1520693

[115] S. A. Brewster and M. Hughes, "Pressure-Based Text Entry for Mobile Devices," in *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services - MobileHCI '09*. New York, New York, USA: ACM Press, 2009, pp. 9:1–9:4. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1613858.1613870

[116] E. C. Clarkson, S. N. Patel, J. S. Pierce, and G. D. Abowd, "Exploring Continuous Pressure Input for Mobile Phones," Georgia Institute of Technology, Tech. Rep. GIT-GVU-06-20, 2006.

[117] A. Hoffmann, D. Spelmezan, and J. Borchers, "TypeRight: A Keyboard with Tactile Error Prevention," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, New York, USA: ACM Press, 2009, pp. 2265–2268. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1518701.1519048

[118] M. Goel, J. Wobbrock, and S. Patel, "GripSense: : using built-in sensors to detect hand posture and pressure on commodity mobile phones," in *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, Oct. 2012, pp. 545—-554. [Online]. Available: http://dl.acm.org/citation.cfm?id=2380116.2380184

[119] S. Hwang, A. Bianchi, and K.-y. Wohn, "VibPress: : estimating pressure input using vibration absorption on mobile devices," in *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services - MobileHCI '13*. New York, New York, USA: ACM Press, Aug. 2013, pp. 31—-34. [Online]. Available: http://dl.acm.org/citation.cfm?id=2493190.2493193

[120] K. Iwasaki, T. Miyaki, and J. Rekimoto, "Expressive Typing: A New Way to Sense Typing Pressure and Its Applications," in *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems - CHI EA '09*. New York, New York, USA: ACM Press, 2009, pp. 4369–4374. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1520340.1520668

[121] T. Chen and M.-Y. Kan, "Creating a live, public short message service corpus: the NUS SMS corpus," *Language Resources and Evaluation*, pp. 1–37, Aug. 2012. [Online]. Available: http://www.springerlink.com/index/10.1007/s10579-012-9197-9http://wing.comp.nus.edu.sg:8080/SMSCorpus/xml.jsp

[122] D. B. Beringer, "Target Size, Location, Sampling Point and Instructional Set: More Effects on Touch Panel Operation," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 34, no. 4, pp. 375–379, Oct. 1990. [Online]. Available: http://pro.sagepub.com/content/34/4/375.abstract

[123] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832—-837, Sep. 1956. [Online]. Available: http://projecteuclid.org/euclid.aoms/1177728190