Perceptual and Context Aware Interfaces on Mobile Devices

Jingtao Wang

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-64
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-64.html

May 12, 2010

Perceptual and Context Aware Interfaces on Mobile Devices

by

Jingtao Wang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John F. Canny, Chair
Professor Maneesh Agrawala
Professor Ray R. Larson

Spring 2010

The dissertation of Jingtao Wang, titled Perceptual and Context Aware Interfaces on Mobile Devices, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Perceptual and Context Aware Interfaces on Mobile Devices

Abstract

Perceptual and Context Aware Interfaces on Mobile Devices

by

Jingtao Wang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor John F. Canny, Chair

With an estimated 4.6 billion units in use, mobile phones have already become the most popular computing device in human history. Their portability and communication capabilities may revolutionize how people do their daily work and interact with other people in ways PCs have done during the past 30 years. Despite decades of experiences in creating modern WIMP (windows, icons, mouse, pointer) interfaces, our knowledge in building effective mobile interfaces is still limited, especially for emerging interaction modalities that are only available on mobile devices.

This dissertation explores how emerging sensors on a mobile phone, such as the built-in camera, the microphone, the touch sensor and the GPS module can be leveraged to make everyday interactions easier and more efficient. We present studies and models to quantify the capabilities of these sensing channels, and show how effective interfaces in text entry, gaming, and CSCW can be built on mobile phones.

The first such technology is *TinyMotion*. *TinyMotion* detects the movements of a mobile phone in real time by analyzing image sequences captured by its built-in camera, providing a usable analog pointing channel to existing mobile phone users. We quantified *TinyMotion*'s human performance as a basic input control sensor. We found target acquisition tasks via *TinyMotion* follow Fitts' law, and Fitts' law parameters can be used for *TinyMotion*-based pointing performance measurements. We show that using camera phone as a handwriting capture device and performing large vocabulary, multilingual real time handwriting recognition on the mobile phone are feasible. Based on experiences and lessons learned from *TinyMotion*, this dissertation also introduces*SHRIMP* (**S**mall **H**andheld **R**apid **I**nput with **M**otion and **P**rediction), a predictive mobile text input method runs on camera phones equipped with a standard 12-key keypad. *SHRIMP* maintains the speed advantage of Dictionary-Based Disambiguation (DBD) driven predictive text input while enabling the user to overcome collision and OOV problems seamlessly without explicit mode switching. Then, *FingerSense* is presented as another example of perceptual interface to enhance the expressiveness of physical buttons on space-constrained mobile devices. This dissertation also introduces a context-aware system named *GLAZE* (**G**eneralized **L**ocation **A**ware Model**Z** for **E**nd-users). *GLAZE* allows average user without any programming experiences, to create everyday location-aware applications directly on their mobile phones. Last, this

thesis describes the design, implementation and evaluation of *Event Maps*, a web-based calendaring system targeted at improving the experience of attending and organizing large, multi-track conferences on both desktop computers and mobile devices. *Event Maps* has been successfully deployed in multiple large, real world conferences.

To my parents, Xizhe Wang and Qiaozhen Li

and my dear wife, Jia Li

Contents

List of Figures

List of Tables

Acknowledgments

Getting a Ph.D. degree is a long and challenging journey, it would be impossible for me to finish this work without the help and support from many people. I would like to take this opportunity to express my gratitude towards them.

First and foremost, I would like to thank my advisor, John Canny, for guiding me through the research. It's so fortunate to have John as my advisor, both John and I started our research in artificial intelligence and expanded to human-computer interaction later; John also has broad and deep knowledge in many areas, including: theory, graphics, hardware design, education, and qualitative methods. John's vast knowledge always helps me to get insightful guidance for whatever crazy ideas I come up with. John also strongly influences the direction I think research should take for meaning in the real world. Much of John's seminal work has been included text books and will remain in the history of scientific research forever; I truly admire this and wish to contribute to influential research like John did in the future. His trust and continuous support allow me to explore the research problems that intrigue me most and see past the trying times of my life, like when I was struggling with physical illness and the loss of family member.

I also appreciate the efforts of Maneesh Agrawala and Ray Larson who served on my dissertation committee. Maneesh not only gave constructive suggestions to guide my research, but also invited me to paper reviewing activities and guest lectures, from which I benefitted greatly. Ray is very supportive to my research, gave me timely guidance and provided helpful suggestions on my career planning. Additionally, Stuart Russell provided thoughtful feedback while serving on my proposal committee. Special thanks to Jennifer Mankoff, for acting as my temporary advisor when I first came to Berkeley and for her continuous support throughout the years. Thanks are also owed to Ken Goldberg, Björn Hartmann, John Kubiatowicz and Alan Smith, who provided useful advice and prompt support many times about my research, teaching and job search.

I would like to thank members (as well as ex-members) in the Berkeley Institute of Design (BiD) and its precursor, Group for User Interface Research(GUIR), for their support: Ryan Aipperspach, Chris Beckmann, Andy Carle, Scott Carter, Ana Ramírez Chang, Keng-hao Chang, Irene Chien, Corie Cobb, Yitao Duan, Jeff Heer, Jono Hey, Jason Hong, Seth Horrigan, Matthew Kam, Omar Khan, Scott Klemmer, Nicholas Kong, Scott Lederer, Yang Li, Jimmy Lin, Tara Matthews, Reza Naima, Alan Newberger, David Nguyen, Divya Ramachandran, Tye Rattenbury, Jeremy Risner, Celeste Roschuni, Daniela Rosner, Lora Oehlberg, David Sun, Simon Tan, Anuj Tewari, and Wesley Willett. They have been a constant source of help and entertainment. Being a graduate student in computer science, I feel very lucky to have the opportunity to engage in an interdisciplinary environment everyday.

I'm grateful to Shumin Zhai, my long-term mentor, colleague and friend. We started collaborations before I came to Berkeley and worked on multiple, successful projects throughout the years. I benefit tremendously from discussions with him. His rigors thinking skills and the belief in the importance of a fundamental understanding of human performance is always a source of inspiration for me.

I am so fortunate to have Hui Su as my immediate manager at IBM China Research Lab. Hui first inspired my love of research, taught me how to think creatively and collaborate

effectively. I still remember that he even provided hands-on help on mundane things at the beginning, such as how to write emails (that's correct, I didn't know how to write emails professionally when I first joined CRL), and how to initiate a conference call. He sparked my confidence in doing research and let me experience, for the first time, the joy of completing a successful project. I'm always proud of you.

I am particularly indebted to Chandra Narayanaswami, who provided me the opportunity to spend two summers at IBM T.J. Watson Research Center. He treated me with great trust and gave me problems that were truly attractive in research and challenging in technical details. I worked closely with Danny Soroker, Dan Coffman, and Jon Munson. Thank you Danny, I really like the joy of learning through the intensive daily discussions with you. During my internship, I was also fortunate enough to receive help and support from a lot of IBM researchers, including Sam Adams, Tom Erickson, Ramesh Gopinath, Dan Gruen, Ravi Konuru, Jennifer Lai, Jeff Nichols, Marcel Rosu, Alpana Tiwari and Mercan Topkara for the Event Maps project.

I'd like to thank all the friends and colleagues with whom I spent my time as a graduate student at Berkeley, including Evan Chang, Jike Chong, Nathaniel Good, Lianpeng Guo, RJ Honicky, Ling Huang, Gunho Lee, Zeyu Li, AJ Shankar, Alex Smolen, Manu Sridharan, Kai Wei, Zile Wei, Wei Xu, Hao Zhang, Feng Zhou, Qi Zhu, Li Zhuang and many others. The people at Berkeley are the greatest asset I've had during my Ph.D. study.

Special thanks to Elaine Judkins and Anna Horn for spending numerous Saturday mornings helping me to develop my English accent. I'd like to thank Martha Haller, Jill Hodges, Evan and Charisse Hong, Lorraine Thiebaud and Qianying Wang for their continuous help to our family.

I would like to express my earnest gratitude to my parents and my sister for their love and support, without which any of my achievements would not have been possible. My father motivated my early interests in science and engineering. He taught me to always treat things positively and be tenacious regardless of failures. His unconditional confidence gave me the strength and courage to overcome any difficulties. I still remember that his last word in the hospital was not about his own health condition, but to tell the nurse how proud he was to be my father. Dad, I have tried my best and kept my promise, may you have peace in heaven. Thanks to my mother for her love, support and countless sacrifices to raise me and to give me the best possible education. Special thanks to my sister for taking care of our parents during the past few years, for her encouragement when I'm in bad mood, and for sharing the joy with me.

Last but not least, I am deeply indebted to my dear wife Jia for her love, encouragement and understanding during all the years we spent in Berkeley. She was always behind me and gave her unconditional support even in the toughest times. She not only takes over most of the duties for taking care of our daughter, but also supports the family financially. Her expertise in music and literature, as well as her sense of humor, makes our everyday life exciting. Without her support, it would be impossible for me to concentrate on my research. Finally, thanks to our daughter Justina for the joy and the happiness she brings to me, she always reminds me there is another exciting world outside the Ivory Tower.

Chapter 1

Introduction

"Unlike the steady, predictable hardware price/performance improvement described by Moore's law, there has been no such year-to-year progress in user interface design. Instead, the history of user interfaces can be characterized using evolutionary biologist Steven Jay Gould's notion of punctuated equilibrium: long periods of stability interrupted by rapid change."

—Andries van Dam, 1997 [130]

With an estimated 4.6 billion units in use [91], mobile phones have already become the most popular computing device in human history. Their portability and communication capabilities may revolutionize how people do their daily work and interact with other people in ways PCs have done during the past 30 years. Further, Gartner Inc predicts that mobile phones will overtake PCs as the most common Web access device worldwide by 2013 [16]. In 1991, Mark Weiser envisioned "*The technology required for ubiquitous computing comes in three parts: cheap, low-power computers that include equally convenient displays, software for ubiquitous applications and a network that ties them all together.*"[144] 19 years later, it is clear that his vision about ubiquitous computing is becoming a reality with the popularity of mobile phones.

Unfortunately, despite decades of experience in creating modern WIMP (windows, icons, mouse, pointer) interfaces, our knowledge in building effective mobile interfaces is still limited, this is especially true for emerging interaction modalities that are only available on mobile devices. On one hand, we have accumulated substantial knowledge in building GUI applications for desktop computers. All major commercial desktop operating systems, such as Windows, Mac OS and Linux, look similar five feet away. This is in-part, due to the fact that generic WIMP based GUI design for desktop computers is considered a mature area and the corresponding design '*know-how*' has converged in many topics. On the other hand, the user interfaces on mobile devices went through multiple rounds of redesigns and overhauls since the release of Apple Newton PDA (Personal Data Assistant) in 1992. Even with the redesigns and overhauls of the Palm Pilot, Windows CE, Symbian, iPhone, We-

bOS, Android and Windows Phone 7 series, just to name a few, there is still no common agreement on what is the most effective UI design guideline for mobile devices. The highly diversified mobile market implies that we are still in the very early stage towards a full exploration of the mobile design space.

Due to Moore's law, the computing power of mobile devices increases steadily over the years. The state-of-the-art mobile devices are as fast as a desktop computer was ten years ago and have no problem in running a major portion of desktop applications from a decade ago after minor adaptations. However, in this dissertation, we argue that *porting or existing applications from desktops directly to PCs will most likely lead to inferior user experiences*, due to the limitation on size, input/output channels of mobile devices and the fragmented nature of attentional resources [96]. Instead, we show with concrete examples that *by leveraging the unique capabilities on mobile devices (coupled with careful design to hide the corresponding weaknesses), it is possible to support activities on mobile that are hard to achieve on traditional desktop computers.*

What are the unique capabilities of mobile devices? First, portable computers that not only support instant-on, but are also always connected to the internet as well. The rich, local interactions on mobile phones, combined with back-end services in the cloud, will be a paradigm shift that changes how we build and use computers. Second, most mobile phones are equipped with sensors that are not commonly available on desktop computers, such as the built-in camera, touch screen, microphone, GPS and accelerometers. These sensors are new opportunities for effective mobile interactions.

This thesis summarizes research work on understanding how emerging sensors on a mobile phone can be leveraged to make everyday interactions easier and more efficient. We present studies and models to quantify the capabilities of these sensing channels. We also illustrate how effective perceptual and context aware interfaces in text entry, gaming, and computer supported cooperative working (CSCW) can be built on mobile phones. Most of the interfaces we introduce in this dissertation fall in the definition of perceptual and context-aware interfaces [19].

1.1 Dissertation Outline

This chapter has illustrated the potential value of designing interfaces that leverage the unique capability of mobile devices. The rest of the thesis is organized as follows:

Chapter 2 presents *TinyMotion*, a pure software approach for detecting a mobile phone user's hand movement in real time by analyzing image sequences captured by the built-in camera. In this chapter, we describe in detail the design and implementation of *TinyMotion* and several interactive applications based on *TinyMotion*. Through an informal evaluation and a formal 17-participant user study, we found: (1), *TinyMotion* can detect camera movement reliably under most background and illumination conditions. (2), Target acquisition tasks based on *TinyMotion* follow Fitts' law and Fitts' law parameters can be used for *TinyMotion* based pointing performance measurement. (3), The users can use Vision TiltText, a *TinyMotion* enabled input method, to enter sentences faster than MultiTap with a few minutes of practice. (4), Using camera phone as a handwriting capturing device

and performing large vocabulary, multilingual real time handwriting recognition on the cell phone are feasible. (5), *TinyMotion* based gaming is enjoyable and immediately available for the current generation camera phones. We also report user experiences and problems with *TinyMotion* based interaction as resources

Based on experiences we get in designing and using *TinyMotion*, we advance the state-of-the-art in mobile input in chapter 3 - Dictionary-based disambiguation (DBD) is a very popular solution for text entry on mobile phone keypads but suffers from two problems: (1), the resolution of complete ambiguity or collision (two or more words sharing the same numeric key sequence) and (2), entering out-of-vocabulary (OOV) words. We present *SHRIMP*, a system and method that solves these two problems by integrating DBD with camera-based motion sensing that enables the user to express preference through a tilting or movement gesture. *SHRIMP* (**S**mall **H**andheld **R**apid **I**nput with **M**otion and **P**rediction) runs on camera phones equipped with a standard 12-key keypad. *SHRIMP* maintains the speed advantage of DBD driven predictive text input while enabling the user to overcome DBD collision and OOV problems seamlessly without even a mode switch. In a novice user study, we found the text entry speed of *SHRIMP* (12.1 wpm) was significantly faster than that of MultiTap (7.64 wpm), with less than ten minutes of practice.

Chapter 4 proposes a novel method, *FingerSense* to enhance the expressiveness of physical buttons. In a *FingerSense* enabled input device, a pressing action is differentiated according to the finger involved. We modeled the human performance of *FingerSense* interfaces via a GOMS style analysis and derived related parameters from a preliminary usability study.

Chapter 5 presents a system for creating location-aware mini-applications, *GLAZE* (**G**eneralized *L*ocation **A**ware model**Z** for **E**nd-users). This system enables average end-users to create everyday location-aware applications by themselves and on their cell phones. The *GLAZE* system provides a set of location-aware primitives named *REWIND* (**RE**AD, **W**RITE and F**IND**) to help end-users model and generate their intended applications though the help of a set of form-style smart templates. The *GLAZE* system is designed to lower the threshold of location-aware application creation and encourage both expertise sharing and group interactions in a community.

Chapter 6 describes *Event Maps*, a web-based collaborative calendaring system targeted at improving the experience of attending and organizing large, multi-track conferences. *Event Maps* runs on both desktop computers and mobile devices. Through its zoomable *Tabular Timeline*, users can navigate conference schedules, seamlessly moving between global and local views. Through a compact decoration widget named *Active Corners*, *Event Maps* enables contextual asynchronous collaboration before, during, and after a conference. Organizers can easily create or import conference schedules via a backend interface, and also use the provided analytic toolkits to get insights from visiting patterns and statistics.

We conclude in chapter 7 with a summary of findings and lessons learned.

Chapter 2

TinyMotion : Camera Phone Based Motion Sensing

"Everything is best for something and worst for something else. The trick is knowing what is what, for what, when, for whom, where, and most importantly, why."

—Bill Buxton, 2007 [17]

2.1 Motivation

Mobile phones have become an indispensable part of our daily life. Their compact form has the advantage of portability, but also imposes limitations on the interaction methods that can be used. While the computing and display capabilities of mobile phones have increased significantly in recent years, the input methods on phones largely remain button based. For basic voice functions, a button press based keypad is quite adequate. But mobile phones are rapidly moving beyond voice calls into domains such as gaming, web browsing, personal information management, location services, and image and video browsing. Many of these functions can greatly benefit from a usable analog input device. Mobile phones may take on even more computing functions in the future if higher performance user interfaces can be developed. We show in this chapter that the built-in camera in mobile phones can be utilized as an input sensor enabling many types of user interactions.

Various technologies have been proposed and tested to improve interaction on mobile devices by enhancing expressiveness [46, 53, 105, 98, 146], or sensing contextual features of the surrounding environment [53]. Accelerometers [53, 105, 98, 146], touch sensors [46, 53] and proximity sensors [53] have been used. While some of these technologies may eventually make their way inside the phone, they are rarely seen in phones today.

On the other hand, camera phones are already popular and pervasive. The global cell phone shipment in 2005 was 795 million units, 57% of which (about 455 million units) were camera phones. At the same time, 85% of the mobile phones were camera phones in 2008

Figure 2.1: Using *TinyMotion* enabled applications out-doors (left) and in-doors (right)

with a shipment of 800 million units [18].

We have developed a technique called *TinyMotion* (figure 2.1) for camera phones. *TinyMotion* detects the movements of a cell phone in real time by analyzing image sequences captured by its built-in camera. Typical movements that *TinyMotion* detects include horizontal and vertical movements, rotational movements and tilt movements. In contrast to earlier work, *TinyMotion* does not require additional sensors, special scenes or backgrounds. A key contribution of this chapter is an experimental validation of the approach on a wide variety of background scenes, and a quantitative performance study of *TinyMotion* on standard target acquisition tasks and in real world applications.

2.2 Related Work

Related work fall into three categories: emerging camera phone applications, new interaction techniques for mobile devices and computer vision based interaction systems.

2.2.1 Emerging Camera Phone Applications

Inspired by the success of CyberCode [106] from SONY, several researchers [109] and companies have designed customized 2D barcodes that can be recognized easily by camera phones. Most of these systems measure the size, position and angle of the barcode relative to the cameras optical axis, which can be used to infer the camera's 3D position relative to the barcode, and provide an alternative spatial input channel. SemaCode [111] is positioned

as a general purpose tagging solution, SpotCode (a.k.a. ShotCode [114]) has a feature that maps 2D bar codes to online URLs. On top of their barcode system Visual Codes [109] have also built UI widgets to achieve desktop-level interaction metaphors. Visual Codes is also used to manipulate objects on large public displays interactively [5].

Hansen and colleagues [44] proposed the idea of a "mixed interaction space" to augment camera phone interaction. Their method relies on camera imaging of a light uniform background with a circular marker. This image needs to be laid out on a suitable flat surface. 2D bar codes [5], Orthogonal axis tags [94], high gradient still objects [31] and human faces [45] can all be used as markers to facilitate the tracking task. In contrast, *TinyMotion* provides general motion sensing from whatever scene the camera is pointed at, near or far.

If we make assumptions on the texture/gradient distribution of surrounding environments, other vision based approach such as Projection Shift Analysis [28] or gradient feature matching [43] could also provide real time motion detection on mobile devices. However, these approaches will fail when these strong assumptions do not hold. E.g., image projection based approach won't work on background with repeating patterns [28]. Many everyday backgrounds with less gradient, e.g. floors, carpets and the sky, will make gradient feature detection infeasible.

2.2.2 New Interaction Techniques for Mobile Devices

Previous work has proposed many compelling interaction techniques based on physical manipulation of a small screen device, including contact, pressure, tilt, motion and implicit biometric information. Specifically with regard to navigation, Rekimoto [105] used tilt input for navigating menus, maps, and 3-D scenes, and Harrison et al. [46] and Hinckley et al. [53] have used tilt for scrolling through documents and lists. Peephole Displays [152] explored the pen interactions on spatially aware displays on a PDA. Earlier before the current generation of phones and PDAs, Fitzmaurice et al. [33] explored spatially aware 'Palmtop VR' on a miniature handheld TV monitor.

2.2.3 Computer Vision in Interactive Systems

Considering the amount of information captured by human eyes, using computer vision in interactive systems has long been a popular topic [35]. Much previous research in this category covers multimodal interaction [34], gesture recognition, face tracking, body tracking [93] etc. There are also numerous systems that map certain types of user's movements, for example, body, gesture, finger, face, and mouth movements into computer inputs. Please refer to [34, 93], which include some extensive survey in the related directions, and [35] for some commonly used basic algorithms. However, most of those applications are built on powerful desktop computers in controlled lab environments.

2.3 The *TinyMotion* Algorithm

Computer vision techniques such as edge detection [43], region detection [35] and optical flow [56] can be used for motion sensing. Ballagas et al [5] had implemented an optical flow based interaction method - "sweep" on camera phones. However, optical flow [56] is a gradient based approach and it uses local gradient information and the assumption that the brightness pattern varies smoothly to detect a dense motion field with vectors at each pixel. Due to the additional assumptions on gradient distribution and the smoothness of illumination, they are usually less robust than direct methods based on correlation or image difference. The latter are used in optical mice, video codecs etc, and we follow suit. *TinyMotion* has used both image differencing and correlation of blocks [37, 72] for motion estimation.

The *TinyMotion* algorithm consists of four major steps:

1. Color space conversion.
2. Grid sampling.
3. Motion estimation.
4. Post processing.

All of these steps are realized efficiently by integer only operations. To use *TinyMotion*, the camera is set in preview mode, capturing color images at a resolution of $176 * 112$ pixels, at a rate of 12 frames/sec [1].

2.3.1 Color Space Conversion

After a captured image arrives, we use a bit shifting method (equation 2.2, an arithmetic approximation of equation 2.1) to convert the 24-bit[2] RGB color to an 8-bit gray scale image.

$$Y = 0.299 * R + 0.587G + 0.114B \tag{2.1}$$

$$Y = (R >> 2) + (G >> 1) + (G >> 3) + (B >> 3) \tag{2.2}$$

2.3.2 Grid Sampling

Grid Sampling, a common multi-resolution sampling technique [72], is then applied on the gray scale image to reduce the computation complexity and memory bandwidth for the follow-up calculations. We use $8 * 8$ sampling window in our current implementation after much experimentation.

[1] Without displaying the captured image and additional computation, the camera phones in our experiments can capture images at the maximal rate of 15.2 frames/sec.

[2] There are only 20 effective bits in each pixel for our specific camera phone used.

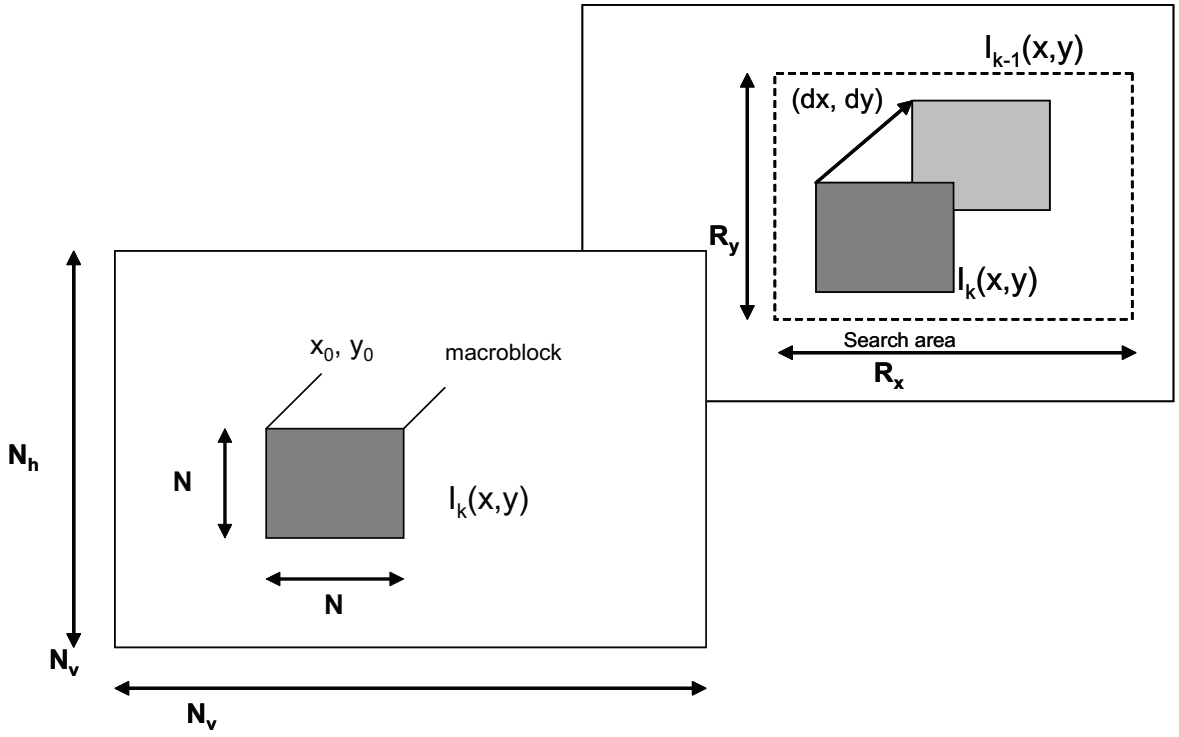

Figure 2.2: Estimate the relative motion of a macro-block between frame I_{k-1} and I_k

2.3.3 Motion Estimation

The motion estimation technique we use is similar to those commonly used by video encoders (MPEG2, MPEG4 etc). We denote the result of grid sampling as a macro-block (MB) and apply Full-search Block Matching algorithm (FBMA) [72, 37] on temporally adjacent fames. Let I_k represent the current frame and I_{k-1} represent the previous frame. In any frame I, $I(x, y)$ is the pixel value at location (x, y). For FBMA, the MB in current frame I_k is shifted and compared with corresponding pixels in previous frame I_{k-1}. The shifting range is represented as R_x and R_y respectively (Figure 2.2). In our current implementation, $R_x = (-3, 3)$, $R_y = (-3, 3)$. Common distance measurements include Mean Square Error (MSE, equation 2.3 [72]), Sum of Absolute Difference (SAD) and Cross-Correlation Function (CCF). After block matching the motion vector MV is chose as the corresponding block shifting distance (equation 2.4).

$$MSE(dx, dy) = \frac{1}{MN} \sum_{m=x}^{x+M-1} \sum_{n=y}^{y+N-1} (I_k(m, n) - I_{k-1}(m + dx, n + dy))^2 \qquad (2.3)$$

$$\overrightarrow{MV} = (MV_x, MV_y) = min_{(dx,dy)\in \mathbf{R}^2} MSE(dx, dy) \tag{2.4}$$

The motion vector $\overrightarrow{MV} = (MV_x, MV_y)$ represents the displacement of the block with the best result for the distance criterion after the search procedure is finished. According to the output of the motion estimation, tilting left is equivalent to moving the phone left, tilting the upper part of the phone towards the user is equivalent to moving the phone upwards, and so on. To detect camera rotation, we split each global MB into $2 * 2 = 4$ sub MBs and estimate their relative motions respectively.

2.3.4 Post Processing

The relative movements detected in the motion estimation step are distance changes in the x and y directions. These relative changes are also accumulated to provide an absolute measurement from a starting position.

In the current implementation, *TinyMotion* generates 12 movement estimations per second, and takes 19 - 22 ms to process each image frame on a Motorola v710 phone. The memory needed is around 300KB.

2.4 Implementation

Our primary implementation platform is the Motorola v710 (a CDMA Phone from Verizon Wireless), a common off-the-shelf camera phone at the time our implementation. The v710 has an ARM9 processor, 4M RAM, $176 * 220$ pixel color display. Our application is written in C++ for BREW (the Binary Runtime Environment for Wireless [13]) 2.11. We use Realview ARM Compiler 1.2 for BREW [104] to cross-compile the target application. BREW is an efficient binary format that can be downloaded over the air, so there is a rapid distribution path for commercial applications built using *TinyMotion*. We believe *TinyMotion* can also be ported easily to other platforms such as Windows Mobile and Symbian. To test the efficacy of *TinyMotion* as an input control sensor, we wrote four applications (Motion Menu, Vision TiltText, Image/Map Viewer, Mobile Gesture) and three games (Camera Tetris, Camera Snake and Camera BreakOut). All these prototypes can be operated by moving and tilting the camera phone. Figure 2.3 shows some screen shots of the *TinyMotion*-enabled prototype applications. Not including the recognizer used in Mobile Gesture, the current *TinyMotion* package includes a total of $23,271$ lines of source code in C++. We now discuss the Mobile Gesture and Vision TiltText applications in greater detail.

2.4.1 Mobile Gesture

The Mobile Gesture application was inspired by the idea of using the camera sensor on the cell phone as a stylus for handwriting recognition and gesture based command and control. In the current implementation of Mobile Gesture, the user presses the “OK” button

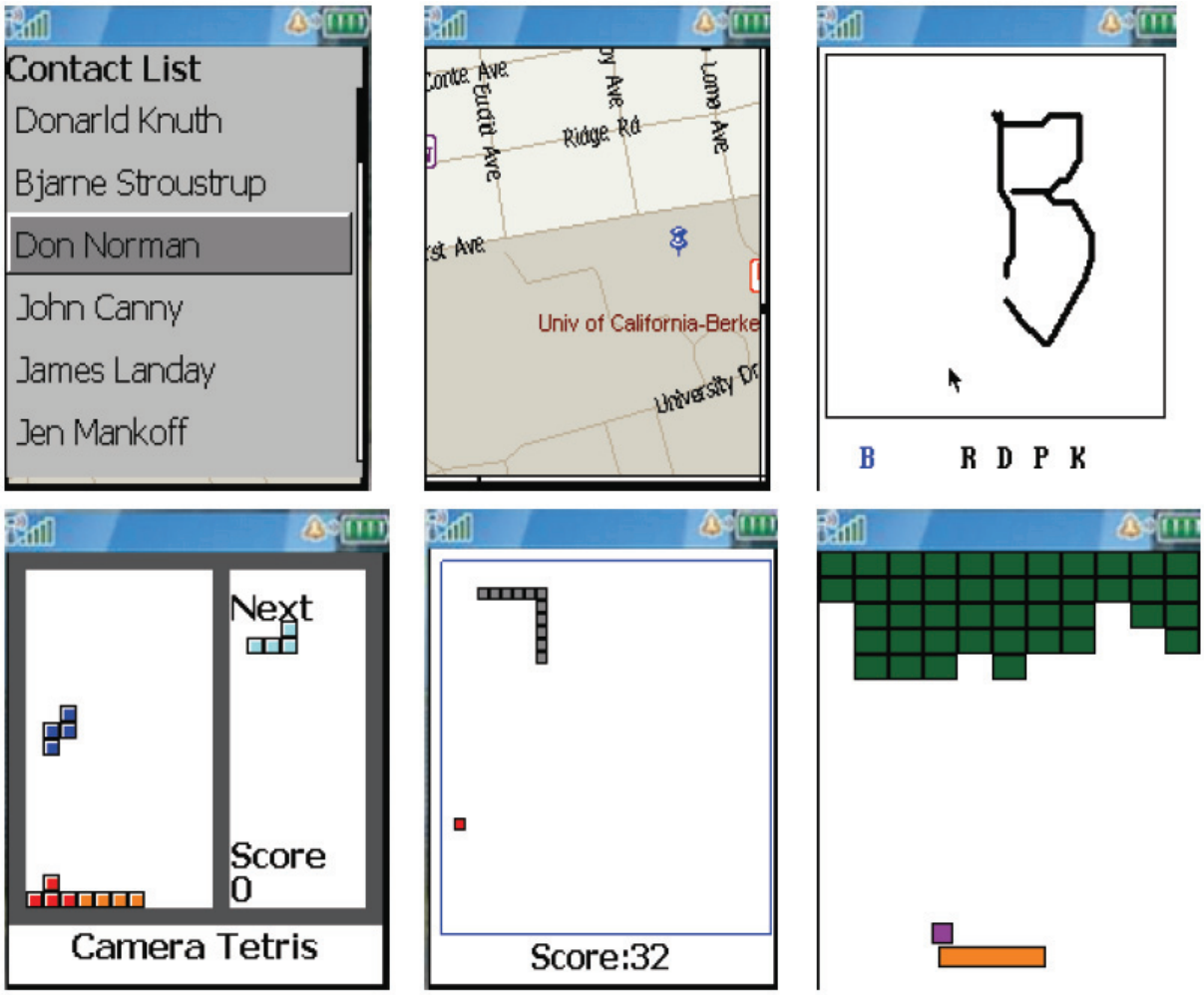

Figure 2.3: Sample *TinyMotion* applications and games. From left to right, top to bottom C Motion Menu, Image/Map Viewer, Mobile Gesture, Camera Tetris, Camera Snake and Camera BreakOut

on the phone to trigger the "pen down" operation on the phone. Instead of restricting gesture/handwriting to be single stroke or setting a timeout threshold to start the recognition, the user presses the POUND ("#") key to signal the end of a character.

The recognizer used by the Mobile Gesture application is a commercial product designed by one of the authors. The original recognizer was designed for handheld devices running Palm Pilot, Windows CE or Linux etc. It supports multilingual input of western character and East Asian double byte characters (e.g. Chinese, Japanese characters). By default, we use a recognition library which supports all the English characters, punctuation symbols and around 8000 Chinese and Japanese characters (6763 simplified Chinese characters defined by the GB2312 national standard and around 1300 Hiragana, Katakana and Kanji characters in Japanese). The size of the recognizer is around 800KB including both the code and the recognition library. If we remove the support for Asian double byte characters, the size of the recognizer and related library can be reduced to around 350kb. On the Motorola v710 phone, it takes 15-20ms to recognize a handwritten Roman character, and 35-40ms to recognize one of the 8000 supported double byte characters. As a reference, it takes the same recognizer around 700ms to recognize the same double byte character on a Palm V PDA with 2 Mb memories, which implies an off-the-shelf cell phone in year 2005 is about 20 times faster than a common PDA in the year 1999 for this specific recognition task. We made one modification on the recognizer after porting it to BREW by adding a four-pixel wide smoothing window filter on the handwriting traces before starting the actual recognition process. This is designed to reduce the hand shaking noise captured by *TinyMotion*. The handwriting traces displayed on the user's screen are not smoothed.

2.4.2 Vision TiltText

We use the following configuration in our Vision TiltText text input method, which is a remake of the accelerometer based mobile input method by Wigdor and colleagues [146]. To input character '**A**', a user need to press keypad button '**2**', hold it, tilt or move the phone to the left, release button '**2**'. To input character '**B**', press and release button '**2**' without movement, to input character '**C**', press button '**2**', hold it and tilt or move the phone to the right, then release the button. This definition is based on the convention that the alphabet characters displayed on telephone button '**2**' is ordered as '**A**','**B**','**C**' from left to right respectively. To input numeric characters, one presses the corresponding numeric key, move the phone up, then release it. To input the fourth character '**S**' or '**Z**' on button '**7**' and '**9**', the user can press the related button, move down, then release. To avoid noisy movement generated by hand shaking, we set a movement threshold for all the characters that need tilting to enter. When the movement in one direction exceeds the corresponding threshold, the phone will vibrate for 70ms to signal that the input state had changed so the button can be safely released.

	Left	Right	Up	Down
Outdoor direct sunshine	97%	100%	96%	97%
Outdoor in the shadow	100%	99%	99%	100%
In-door ambient light	100%	100%	100%	100%
In-door fluorescent lamp	100%	100%	99%	100%

Table 2.1: Movement benchmarking results in four typical environments (shifting and tilting movements in the same direction are not differentiated)

2.5 First Informal Evaluation

We evaluated the reliability of *TinyMotion* by two methods. First, we benchmarked the detection rate of camera movements in four typical conditions and four different directions. To measure each direction 50 shift action and 50 tilt actions were performed. In total, $1,600$ actions were recorded. A shift action required at least a half inch of movement, while a tilt action had to exceed 15 degrees. If the accumulated movements value in a certain direction exceeded the threshold value 5, our system will output that direction as the detected movement direction. The summarized detection rates in each condition are listed in table 1. Most of the errors in the outdoor direct sunshine condition were caused by unexpected objects (mostly vehicles or people) moving into/out of the camera view during the testing process.

We also conducted an informal usability test by distributing camera phones installed with *TinyMotion*-enabled applications/games to 13 users, most of them students or faculty members in a local university. We asked them to play with the Motion Menu application, the Camera Tetris game and the Camera Snake game (Some applications such as Mobile Gesture and Camera BreakOut were not ready at the time of the informal evaluation) and encouraged them to challenge *TinyMotion* in any background and illumination conditions that they could think of or had access to.

All of the users reported success against backgrounds such as an outdoor building, piles of garbage, different types of floors indoor and outdoor, grass in a garden, cloth, and a bus stop at night, areas with low illumination or colored illumination, different areas in pubs, etc. Most were surprised to learn that the motion sensing was based on camera input. One participant was shocked when he found that *TinyMotion* still worked when he pointed the camera at a blue sky and moved the phone (even motion of smooth gradient images can be detected). Figure 2.4 shows some difficult situations where traditional edge detection based methods may fail but *TinyMotion* can still work. The environments in which *TinyMotion* won't work include completely dark rooms, extremely uniform background without any pattern (e.g. the glass surface when an LCD monitor is turned off) and pointing the camera to the outside of a window in a moving vehicle.

The participants in our informal study were clearly amazed with *TinyMotion* and interested in its use. Comments include

"Cool, I didn't expect the tracking can work that well."

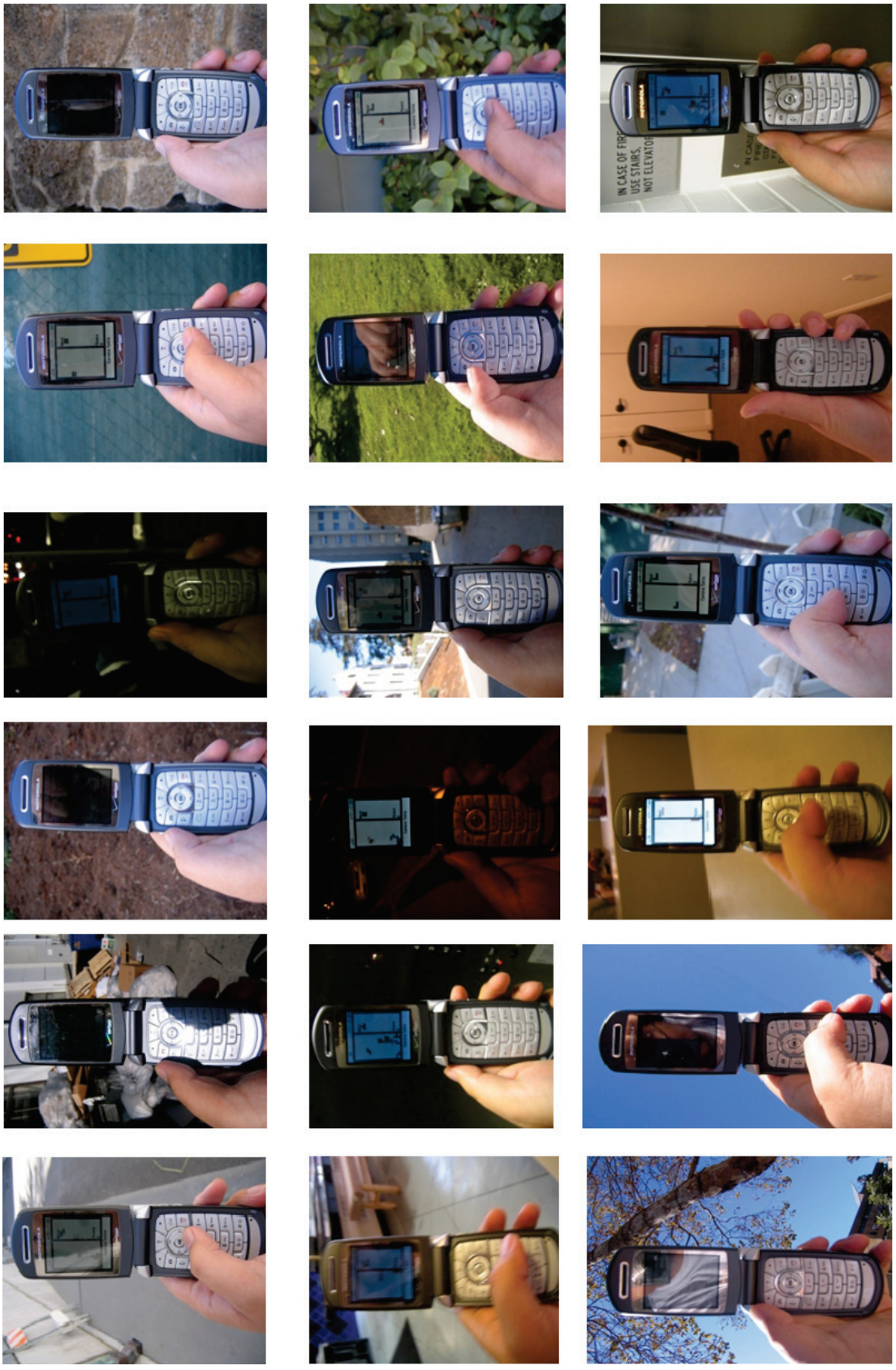

Figure 2.4: Environment/Backgrounds in which *TinyMotion* work properly

"Using this (motion menu) makes [operating] cell phones a lot more fun."

"it will be an ideal method to play the monkey ball game on a cell phone."

One user quickly realized that instead of moving the camera phone directly, he can put his other hand in front of the camera lens and control the *TinyMotion* games by moving that hand. Another user initially felt *TinyMotion* was not very sensitive, only to find that his extended index finger covered the camera lens.

2.6 Formal Evaluation

Although the results of the first informal user study were very encouraging, a formal study was necessary for understanding the capabilities and limitations of *TinyMotion* as an input sensing mechanism. We had two basic goals for the formal study. One was to quantify human performance using *TinyMotion* as a basic input control sensor. In particular we measured the performance of pointing, menu selection, and text entry by tap-tilt action. The second goal was to evaluate the scope of applications that can be built on the *TinyMotion* sensor, for example using *TinyMotion* to play games and do handwriting / gesture recognition.

2.6.1 Experimental Design

The experiment consisted of six parts:

Overview In this session we gave a brief overview of the *TinyMotion* project, and demonstrated some of the *TinyMotion* applications to the participant. We also answered their questions and let them play with *TinyMotion* applications freely.

Target Acquisition/Pointing This session was designed to quantify the human performance of the *TinyMotion* based pointing tasks. The section started with a warm up practice session to allow the participants to become familiar with the pointing task. Pressing UP button started a trial from an information screen indicating the number of trials left to be completed. Each trial involved moving the cell phone UP, DOWN, LEFT or RIGHT to drive the on screen cursor (a slim line) from an initial position to the target and then pressing OK button (Figure 2.5, left). If the user hit the target, the target acquisition screen disappeared and the information screen returned. If the user missed the target, the cursor returned to the initial position and the trial repeated until the user successfully hit the target. The users were encouraged to hit the target as fast as possible and as accurately as possible during the target acquisition stage, but could rest as long as needed when the information screen was displayed. We encouraged the users to practice as long as they wanted before the actual test, most of the users practiced for 3 to 5 minutes.

There were 4 different target sizes (20, 30, 40, 50 pixels), 4 different distances (30, 50, 70, 90 pixels) and 4 different movement directions (left, right, up, down) in this task. Each participant completed 160 randomized trials.

Menu Selection In each trial of this task, a participant was required to select a target name from a contact list. After reading the target name from an information screen, the participant could press the STAR ("*") button to switch to the contact list and start the actual trial. The contact list included 30 alphabetically sorted names and the cell phone could display 6 names per screen. After highlighting the intended name, the user can press the "OK" button to complete the trial and switch back to the information screen (Figure 2.5, middle).

There were three conditions in this task: cursor button based selection, *TinyMotion* based selection, and *TinyMotion* based selection with tactile feedback. In the tactile feedback condition every time when the highlighted menu item changed as a result of moving the phone, the phone vibrated for around 100ms, providing a non-visual cue to the user about her progress on the menu item movements. We added this condition to check the potential influences of tactile feedback on menu selection.

Each participant was asked to complete 16 selections in each condition. The order of the three conditions was randomized to counter balance the learning effects.

Text Input In this task, we compared the performance of the most popular mobile text entry method C MultiTap with our Vision TiltText input method. We followed configurations similar to those used in Wigdor et al's original TiltText study [146]. The short phrases of text were selected from MacKenzie's text entry test phrase set [83, 112]. The timeout for the MultiTap method was 2 seconds. Due to time constraint of our study, each participant entered only 8 sentences with each input method. Note that in studies like [146], each participant entered at least 320 sentences for each input method tested. As a result, our study was not intended to reveal the learning curve and eventual performance of the input methods tested. Instead, we only measured users' initial performance without much practice. This task started with a warm up practice session, the users could practice with the two methods tested as long as they wanted. Most of them choose to practice for 2-5 minutes before the actual test. The order of the two methods tested was randomized.

More Complex Applications After completing the three basic quantitative performance tasks described above, we asked the participants to play with the games we created (Camera Tetris, Camera BreakOut, and Camera Snake) and the handwriting recognition application (Mobile Gesture).

After demonstrating the games and the handwriting recognition application to the users, we let the users play with these applications by themselves. They were encouraged to play the games as long as they wanted and enter at least 3 different characters/gestures in our handwriting recognition application.

Collecting qualitative feedback We conducted a final survey immediately after a user completed all the tasks. In the survey the user completed a questionnaire and commented on the applications they tested and on the idea of *TinyMotion* in general.

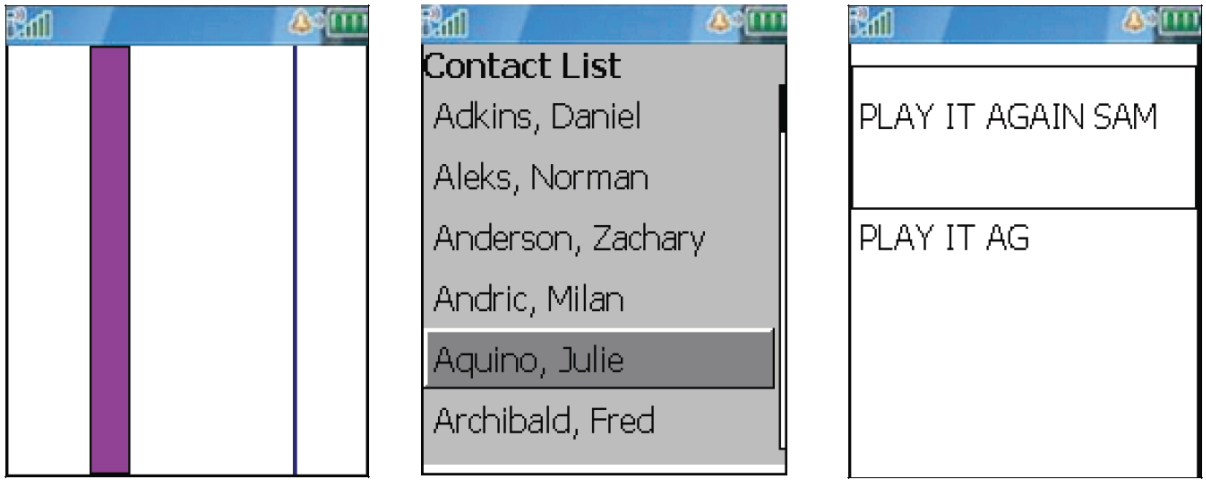

Figure 2.5: From left to right, screen shots of the target acquisition task, menu selection task and text input task.

To simulate the real world situations of cell phone usage, we did not control the environment used for the study. The participants were encouraged to choose their desired locations to complete the study. Most of the studies were completed in the participants' own chair or at a public discussion area in a lab. Figure 2.6 shows some of the actual environments used during the study.

2.6.2 Test Participants

17 people participated in our study. 15 of them were undergraduate or graduate students in a university, the other two were staff members of the university. 6 of the participants were female and 11 male. Five of them owned a PDA and all of them owned a cell phone at the time of the study. 12 of the 17 cell phones were camera phones. Four of the participants sent text messages daily, six weekly, three monthly and four never sent text messages. Interestingly, no user in our study use the camera function of their cell phone on a daily basis, three of them use the camera function weekly, four monthly, four yearly and one of them never uses the camera function.

Two participants didn't complete the menu selection task and one participant didn't complete the text input task due to time constraints. One of the studies was interrupted by a false fire alarm for around 25 minutes. All of the participants completed the target acquisition task, played with all the applications we created and completed our survey and questionnaire.

Figure 2.6: Some sample pictures taken from our user study.

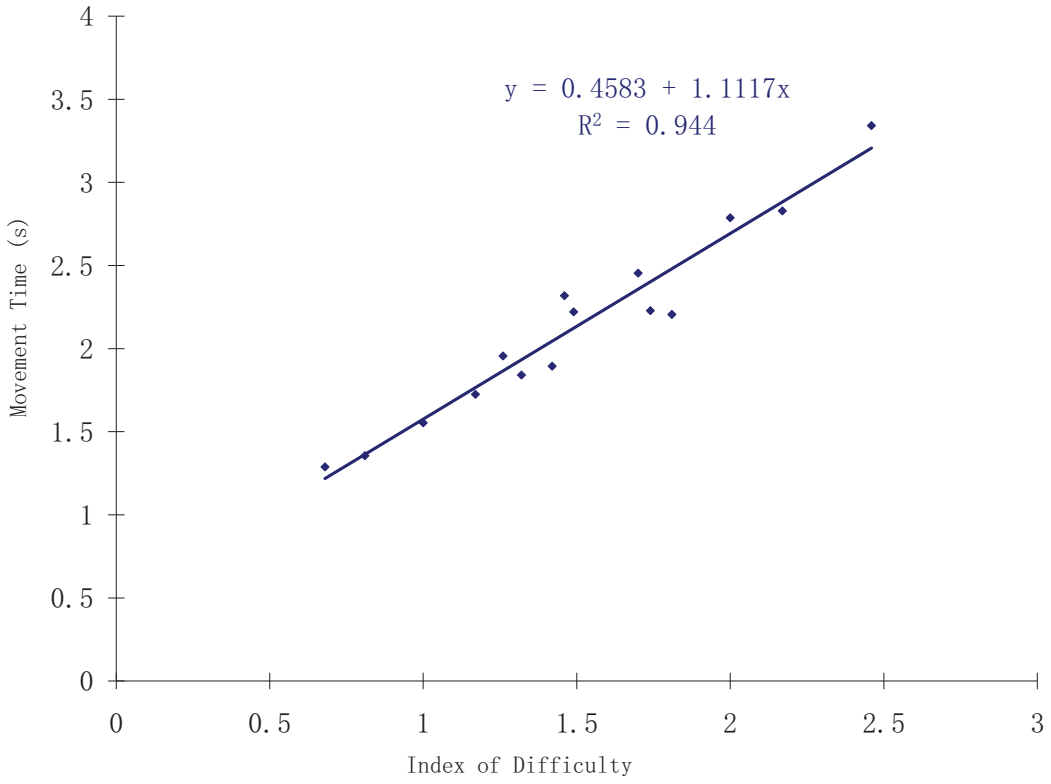

Figure 2.7: Scatter-plot of the Movement Time (MT) vs. the Fitts' Law Index of Difficulty (ID) for the overall target acquisition task.

2.7 Evaluation Results

2.7.1 Target Acquisition/Pointing

2842 target acquisition trials were recorded. Despite the low sampling rate of *TinyMotion* and the novel experience of using it, 2720 of the pointing trials were successful, resulting in an error rate of 4.3%, which is common in Fitts' law [32] studies. This means that it is safe to say that it is already possible to use *TinyMotion* as a pointing control sensor. While there is a vast literature showing hand movements involving various joints and muscle groups follow Fitts' law[4], it is still informative to test whether Fitts' law holds given the particular way a *TinyMotion* instrumented cell phone is held and operated, and the current sampling rate limitation of the cameras in phones.

Linear regression between movement time (MT) and Fitts' index of difficulty (ID) shows (Figure 2.7):

$$MT = 0.4583 + 1.1117 \log_2 (\frac{A}{W} + 1), (sec) \tag{2.5}$$

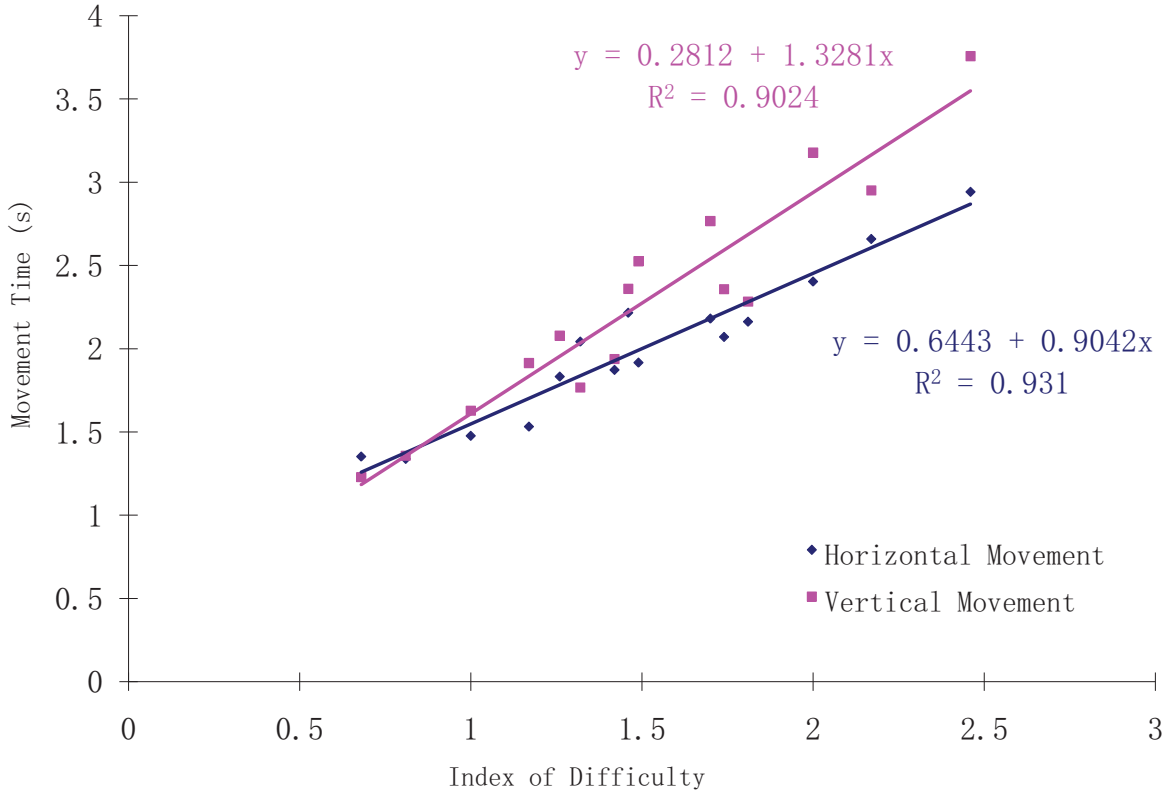

Figure 2.8: Scatter-plot of the Movement Time (MT) vs. the Fitts' Law Index of Difficulty (ID), separately grouped by horizontal and vertical directions.

In equation 2.5, A is the target distance and W is the target size. While the empirical relationship between movement time (MT) and index of difficulty ($ID = \log(A/W + 1)$) followed Fitts' law quite well (with $r^2 = 0.94$, see Figure 2.7), both of the two Fitts law parameters (time constant $a = 0.458$ sec and information transmission rate $1/b = 1/1.1117 = 0.9$ bits/sec) indicated relatively low performance of pointing. This is not surprising given the low sampling rate of the camera (12 frames per second as opposed to 40+ frames per second in a typical computer mouse). However since we now know *TinyMotion* based pointing follows Fitts' law, these parameters can serve as an informative benchmark for future improvement in hardware (e.g. image frame rate, image capturing quality) or the software (detection algorithms and related parameters).

An interesting finding is the difference in manipulation direction of *TinyMotion* instrumented mobile phones. The error rates for four different moving directions (left, right, down, up) were 3.3%, 2.6%, 5.4% and 5.9% respectively. Analysis of variance showed that there was a significant main effect ($p < 0.05$) between horizontal movements and vertical movements in error rate. There was no significant main effect ($p = 0.29$) between hor-

izontal and vertical movements in movement time, although on average it took longer to accomplish vertical target acquisitions than horizontal acquisitions under the same ID value, particularly when ID was high (Figure 2.8). Participants also subjectively felt that it was more difficult to acquire vertical targets. This result could have implications to *TinyMotion* application design.

2.7.2 Menu Selection

The menu selection results show that it is also possible to use *TinyMotion* as a menu selection mechanism. Of the 755 menu selection actions recorded in our study, 720 of them were successful selections. The overall error rate was 4.6%. The error rates for the three experimental conditions - cursor key selection, *TinyMotion* selection and *TinyMotion* selection with tactile feedback (referred as *TinyForce* later) were 3.2%, 4.8%, and 5.9% respectively. Analysis of variance did not show a significant difference among these error rates. As shown in Figure 2.9, the average menu selection time was 3.57s, 5.92s, 4.97s for the Cursor Key, *TinyMotion* and *TinyForce* condition respectively. Analysis of variance results showed that there was a significant difference ($p < 0.001$) in completion time. Pairwise mean comparison (t-tests) showed that completion time between the cursor key and *TinyMotion* methods, and the cursor key and *TinyForce* methods were significantly different from each other ($p < 0.01$), but not between the *TinyMotion* and *TinyForce* conditions. While on average the tactile feedback did reduce menu selection time, the difference was not significant due to the large performance variance.

There were several factors that lead to the low observed performance of *TinyMotion*-based menu selection techniques. First, the contact list used in our study was relatively long (30 names or 5 screens). Although the names were sorted alphabetically, it was still hard for the participants to estimate the approximate location of the name in the contact list given that the name distribution was unlikely to be uniform. As a result, if the desired item was not on the first screen, the participants had to scroll down the contact list slowly to locate the name, which proved to be a difficult task based on our observation. Second, our current Motion Menu was based on the menu behavior provided by the BREW platform, when the highlighted item reached the bottom (the sixth) row and a move down command was received, all the currently on-screen items would move up one row and a new item appeared on the sixth row and was highlighted. This feature was not a problem for cursor key based selection, because the user can simply pay only attention to the sixth row, while holding the phone steadily. However, this feature became troublesome when the users use camera movement for menu selection, most users felt it was difficult to keep track of the last row while keep the phone moving.

2.7.3 Text Input

In total 6150 characters were entered (including editing characters) in this experiment.

Consistent with Wigdor et al's finding [146], the overall speed of Vision TiltText was higher than that of MultiTap (Figure 2.10) and the error rate of Vision TiltText (13.7%)

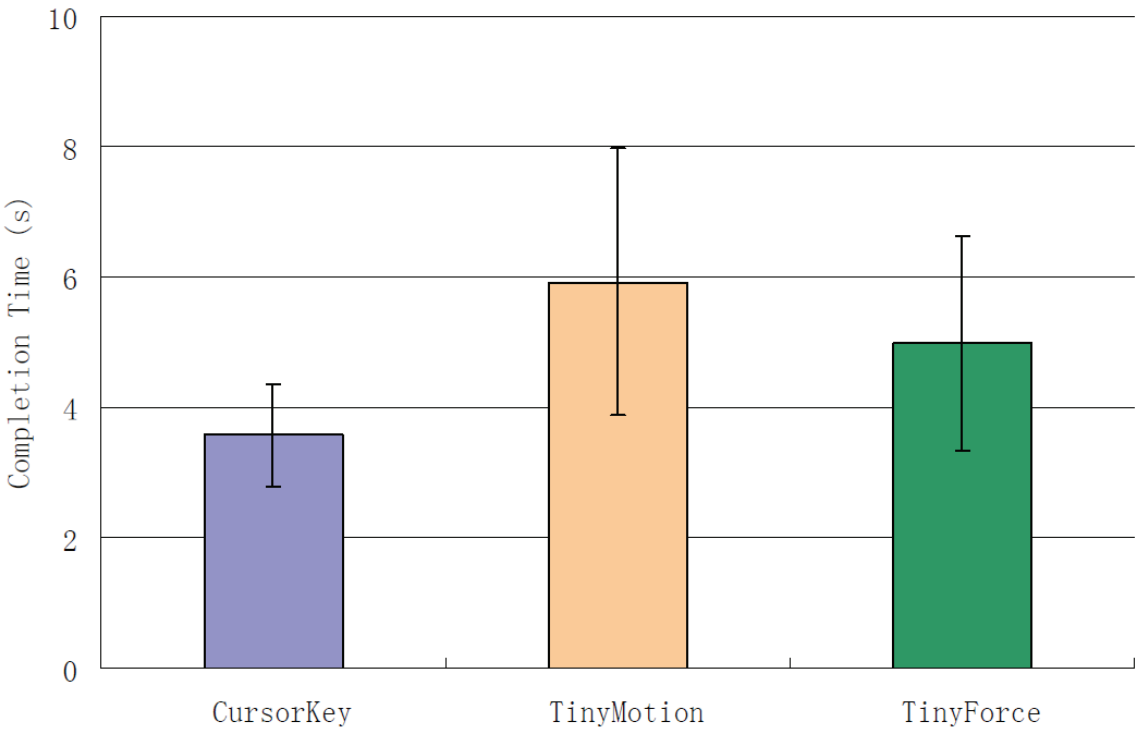

Figure 2.9: Menu selection time from the experiment.

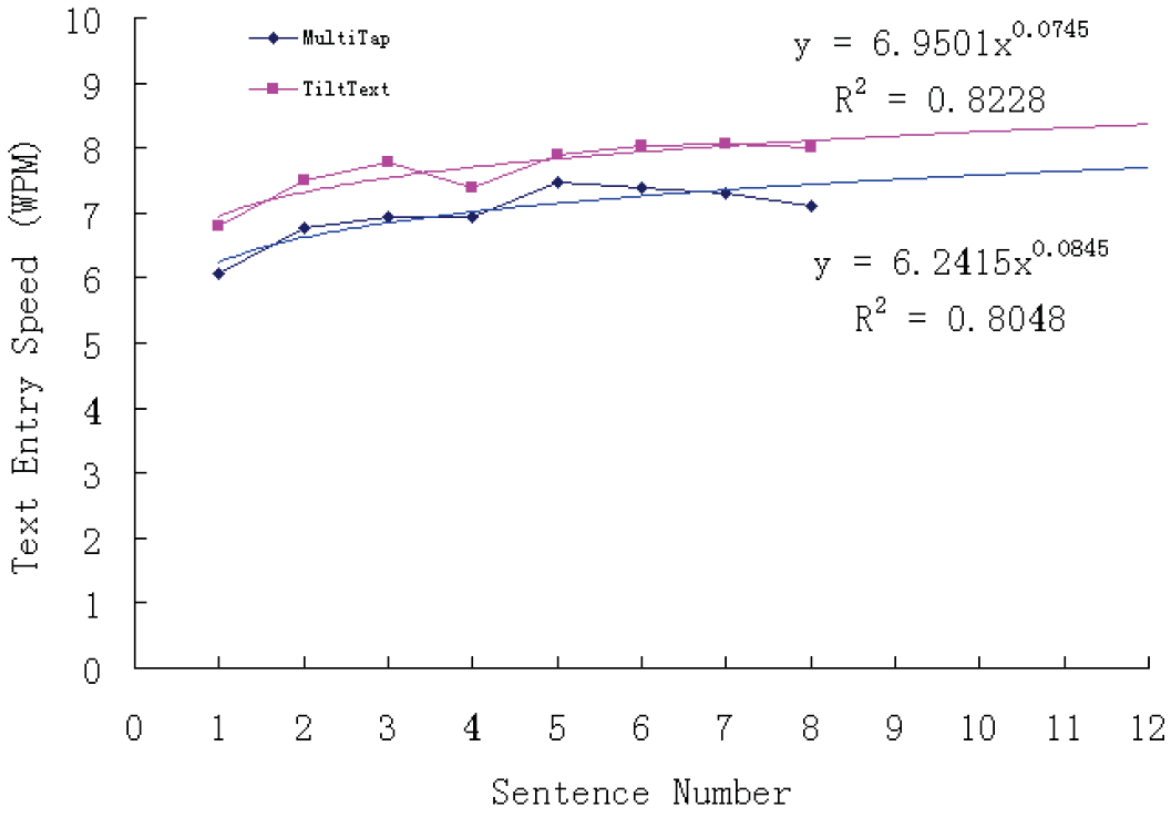

Figure 2.10: Entry speed (wpm) by technique and sentence for the entire experiment.

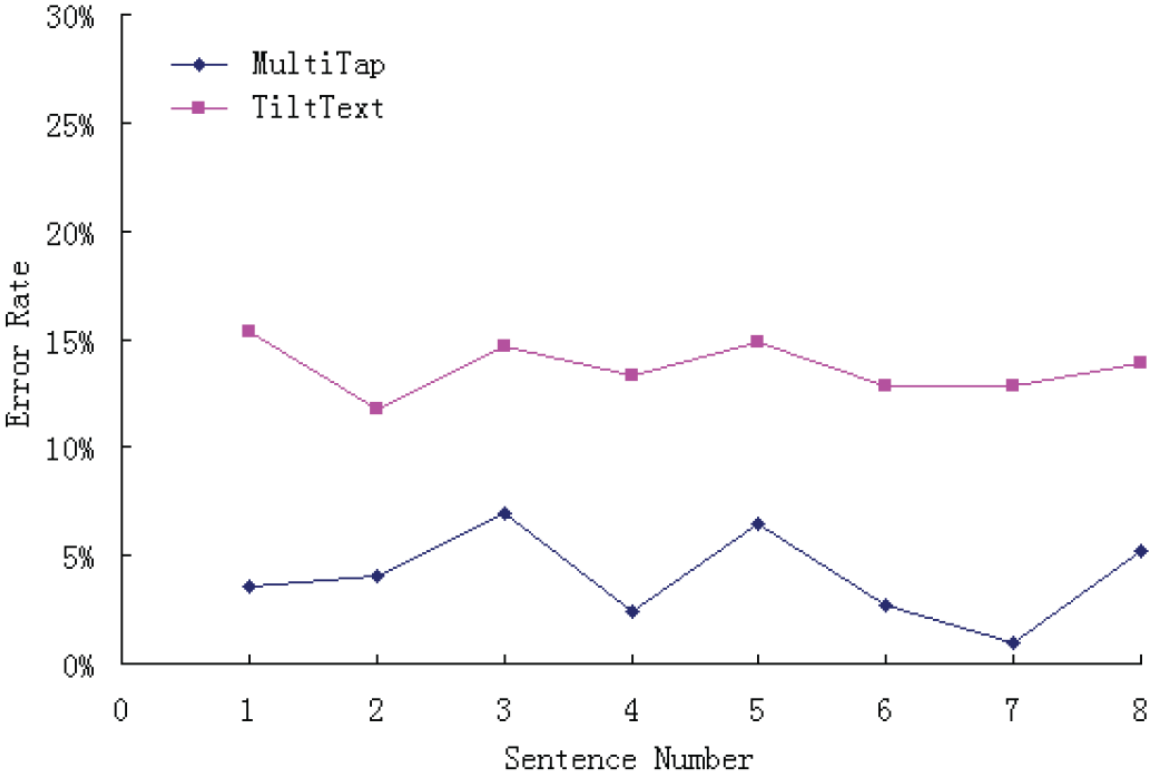

Figure 2.11: Error rate (%) by technique and sentence for the entire experiment.

was much higher than for MultiTap (4.0%) (Figure 2.11). The difference in error rate was statistically significant ($p < 0.001$), but the difference in input speed was not. Overall, the results of vision based TiltText were similar to or slightly better than the accelerometer based TiltText reported by Wigdor et al [146] at the same (initial) learning stage. This shows that as an input sensor *TinyMotion* is at least as effective as an accelerometer for tap-tilt action based text input.

Nearly all the users believed that Vision TiltText is an efficient text entry method (average rating 4.2, $SD = 0.8$, on a 5 point Likert scale [78], 5 means most efficient, 3 means neutral, 1 means least efficient, no one rated Vision TiltText "less efficient" or "least efficient") and is easy to learn (average rating 4.3, $SD = 0.7$, on a 5 point Likert scale scale, 5 means extremely easy to learn, 3 means neutral, 1 means extremely difficult to learn, no one rated Vision TiltText difficult or extremely difficult to learn). 13 users commented explicitly that they would like to use Vision TiltText in their daily life immediately.

Subjectively, participants liked the vision based TiltText over MultiTap.

> *"[For Vision TiltText,] doing a gesture can really speed things up, it was very intuitive."*
>
> *"[Vision] TiltText [is] faster once you learn it, fewer clicks."*
>
> *"[VisionTiltText based] text entry is truly useful because the multitap for names is annoying, the T9 helps with words, but not names."*

2.7.4 More Complex Applications

The participants were excited about their experience of using camera phone movement for gaming. They played the provided games for around 5 - 12 minutes. One user rated use of motion sensing for games as "extremely useful", 10 rated "useful", 6 rated "neutral". No one rated these games "not useful" or "extremely not useful". As a comparison, 9 participants reported that they never played games on their cell phones before, 4 played games yearly and 4 played monthly. In the closing questions, 7 users commented explicitly that these *TinyMotion* games were very fun and they would like to play these games on their own phones frequently.

The user study also revealed several usability problems related with gaming. A lot of participants pointed out that the "conceptual model" or "control" is inconsistent across the current games. e.g. in the Camera Tetris game, when a user moves the cell phone to the left, the block under control will move to the right and vice versa (assuming we are moving the frame, or the background of the game, the block is still). On the other hand, in the Camera BreakOut game, moving the camera phone to the left will move the paddle to the left (assuming we are moving the paddle, the background is still). Around two third of the users believe the "moving background" model is more intuitive while the other one third of users believe the "moving foreground object" model is more intuitive. All of them agree that such game settings should be consistent across all games and it is better to let the user decide which control model to use.

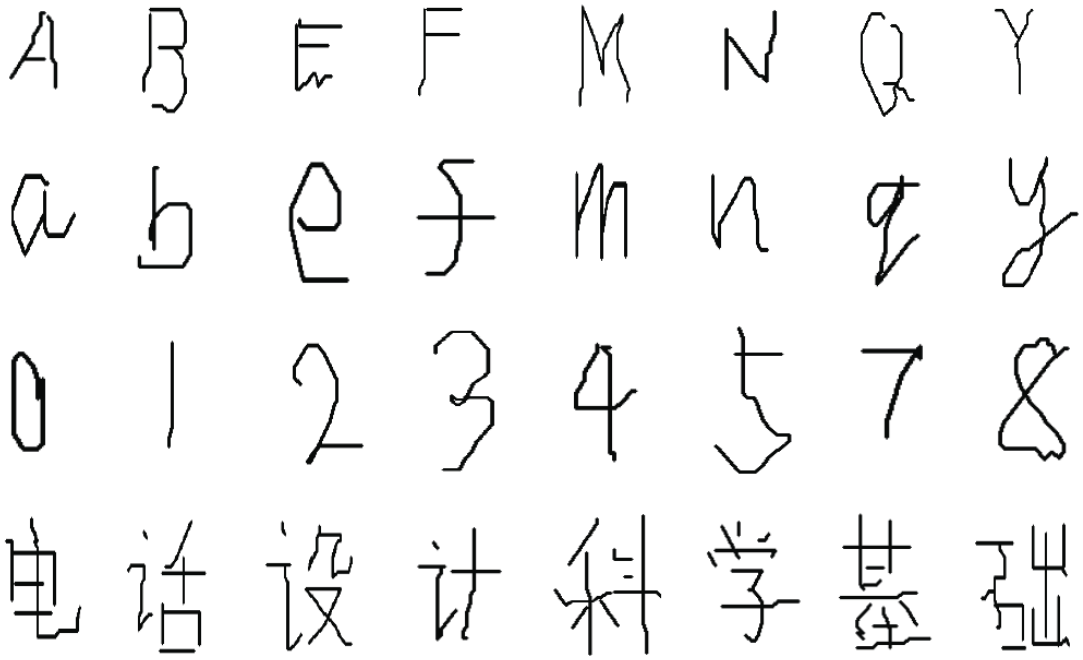

Figure 2.12: Some handwriting samples collected by mobile gesture that had been successfully recognized. The last row is a list of four Chinese words (with two Chinese characters in each word) meaning telephone, design, science, and foundation respectively. No smoothing operation was applied on any of the handwriting samples.

The users also had diverse opinions on the sensitivity of the game control; three users felt the games control should be more sensitive than the current setting while other two users want the game control to be less sensitive, which seem to suggest games controlled by arm/wrist movements should provide adjustable sensitivity setting for each game (the current game control sensitivity was decided by the authors subjectively).

Most of the users were surprised by the ability of using camera phones to do handwriting and receiving recognition results in real time. After a brief demonstration on how to use the mobile gesture application, all the participants successfully entered some alphabetical characters/numeric characters after 1 - 2 minutes of practice (some handwriting samples shown in Figure 2.12). One of the test users, whose native language is Chinese, even tested for more than ten Chinese characters after knowing that the recognizer also supports Chinese and Japanese characters. Based on our observation, it took a participant around 5 - 15 seconds to write an alphabet character and around 15 - 30 seconds to write a Chinese character by using the mobile gesture application. Although from text input speed point of view, Mobile Gesture was obviously slower than most of the keypad input method, most of the users felt really excited when their handwritings (sometimes distorted) got recognized by the cell phone correctly. Indeed most of the users did not believe Mobile Gesture was an efficient text input method (average rating 3.2, $SD = 1.3$, on a 5 point Likert scale,

	TinyMotion	Accelerometer
Working Mechanism	Movement Sensing	Acceleration Sensing
Refresh Rate	12 frames/sec	10 - 500 updates/sec
Computing Power	High	Low
Illumination condition	Low influence	No influence
Holding Position	No influence	Some influence
Inside Moving Vehicles	Still Works	No longer works
Acceleration Estimates	Less accurate	Accurate
Motion Estimates	Accurate	Less Accurate

Table 2.2: A comparision of TinyMotion vs. Accelerometer

5 means most efficient, 3 means neutral, 1 means least efficient). But they also thought Mobile Gesture was intuitive to use (average rating 4.0, $SD = 1.0$). 4 users suggested the ideas of using Mobile Gesture for sentence level input rather than character level input, i.e. predefine some frequent sentences by arm gestures and use Mobile Gesture to trigger those sentences by directly writing the corresponding gesture.

One participant suggested the idea of using Mobile Gesture for authentication tasks. i.e. distinguishing whether a user using the phone is the actual owner by measuring the movement characteristics of predefined trajectories.

2.8 Discussions and Future Work

2.8.1 Battery Life

One major concern related with the popularization of *TinyMotion* could be the battery life. We measured the battery life of *TinyMotion* in two conditions: a power saving situation and an exhaustive usage situation. For the power saving condition, we tested *TinyMotion* by continuously running the Camera Tetris game with the backlight and the vibration function turned off. Our Motorola v710 cell phone ran 8 hours and 7 minutes after a full charge. For the exhaustive usage situation, we measured battery life while *TinyMotion* was constantly running in the background, with the backlight of the camera phone always on, and the vibration function activated around 40% of the total time, and keypad frequently used. In these conditions the same cell phone lasted around 3 hours and 20 minutes to 3 hours and 50 minutes. In more moderate use (less than 5% continuous use each time) the phone battery should last for several days. On most modern phones, the biggest power drain results from screen backlight, and radio use.

2.8.2 *TinyMotion* vs. Accelerometer

The most relevant movement sensing technology for mobile devices might be accelerometers. Much mobile device related research [53, 105, 98, 146] uses one or more accelerometers to sense device movements, for fall protection, user context detection [53], UI navigation

and text input [105, 98, 146]. Due to its small size and relatively low manufacturing cost, we feel accelerometers also have the potential to become pervasive on mobile devices. Hence we feel a fair comparison of *TinyMotion* and accelerometer will be helpful to mobile interface designers.

Table 2.2 highlights the major differences between TinyMotion and Accelerometers. Accelerometers do not require the computing power of host mobile devices in the sensing process and do not depend on illumination condition or view background. In contrast, *TinyMotion* requires certain amount of computing power from the device to generate movement estimates and may not work well in extreme illumination and background conditions. The working mechanisms in accelerometers and *TinyMotion* are very different. The piezoelectric or MEMS sensors in accelerometers are actually sensing movement accelerations and the magnitude of gravitational field. In contrast, *TinyMotion* is detecting deviation/shifting of backgrounds. Double integral operations are needed to estimate position from the raw output of acceleration sensors, which cause accumulate drift errors and make distance estimation less reliable than acceleration. Similarly, acceleration estimations from *TinyMotion*, derived by differential operations, are less reliable than the original deviation estimations.

2.8.3 Future Work

There are many interesting questions worth exploring in the near future. For one example, we feel it might be important to carefully consider the use a "clutch" that can engage and disengage motion sensing from screen action. For another example, the traditional one dimensional linear menu is obviously not the most effective method for camera movement based navigation. We are exploring the possibility of applying a marking menu [73] approach using gesture angles rather than movement distance for menu selection. We feel that the Vision TiltText input method is quite promising and can be further improved, for example, by adding visual feedback to guide the user and speed up the error correction process. Many more applications, such as those involving panning and zooming, should also be explored, particularly in the context of domain specific applications.

2.9 Conclusion

In this chapter, we proposed a method called *TinyMotion* that measures cell phone movements in real time by analyzing images captured by the built-in camera. Through an informal evaluation and a formal 17-participant user study we found that:

1. *TinyMotion* can detect camera movement reliably under most background and illumination conditions.

2. Task acquisition tasks based on *TinyMotion* follows Fitts' law and the Fitts' law parameters can be used to benchmark *TinyMotion* based pointing tasks.

3. The users can use Vision TiltText, a *TinyMotion* enabled input method, to input sentences faster than MultiTap with a few minutes of practice.

4. Using camera phone as a handwriting capture device and performing large vocabulary, multilingual real time handwriting recognition on the cell phone are feasible.

5. *TinyMotion* based gaming is fun and immediately available for the current generation camera phones.

Overall, we conclude that by adding software the use of the built-in camera in phones can go beyond taking pictures into the interaction domain. It is already possible to use the motion sensing result for basic input actions such as pointing, menu selection and text input, and the performance of these tasks can be further improved as hardware performance (in particular the camera frame rate) in phones advances. We showed that it is also possible to build higher level interactive applications, such as gaming and gesture recognition, based on our sensing method and we expect broader and more creative use of camera motion in the future. A major port of the results reported in this chapter previously appeared in [140, 136].

TinyMotion is a pure software project. We choose not to make any hardware changes to the standard phone so results of our research are immediately available for download and use. *TinyMotion* is open source software released under BSD license. The current implementation can be downloaded freely from URL `http://tinymotion.org` [128].

Chapter 3

SHRIMP: Predictive Mobile Input via Motion Gestures

"the dictionary method is not a sufficient input method on its own but having two modes is likely to lead to great confusion."

—Mark Dunlop, 2000 [29]

3.1 Motivation

With an estimated 4.6 billion units in use in December 2009 [91] [1], mobile phones have already become the most popular computing device in human history. Their portability and communication capabilities have revolutionized how people interact with each other. However, despite the rapid growth of mobile phones, text entry on small devices remains a major challenge. Due to trade-offs in both size and compatibility, most mobile phones today are equipped with a 12-key keypad (Figure 3.1). This keypad is effective for dialing phone numbers but not for editing contact lists, composing SMS messages or writing emails. Other input devices, such as mini QWERTY keyboards and touch screens are on the rise, but the 12-button keypad-based mobile phones are still, and will likely to be for years to come, the majority in the market.

One fundamental difficulty in text entry using a 12-key keypad is that the mapping of 26 alphabet characters to the 12 keys is inherently ambiguous. In the ITU E.161 standard [59], one numeric button corresponds to 3 or 4 alphabet characters on the keypad (Figure 3.1). All mobile text input techniques relying on the 12-key keypad have to resolve the ambiguity that arises from this one-to-many mapping.

Most disambiguation methods can be categorized into the following two categories:

Action Based Disambiguation Users rely on multiple key presses (e.g. MultiTap, TNT [58]), concurrent chording [147], tilting [146] or motion [140] to select one character from the multiple alphabetical characters on each key.

[1] The number was 4.0 billion in December 2008 [90]

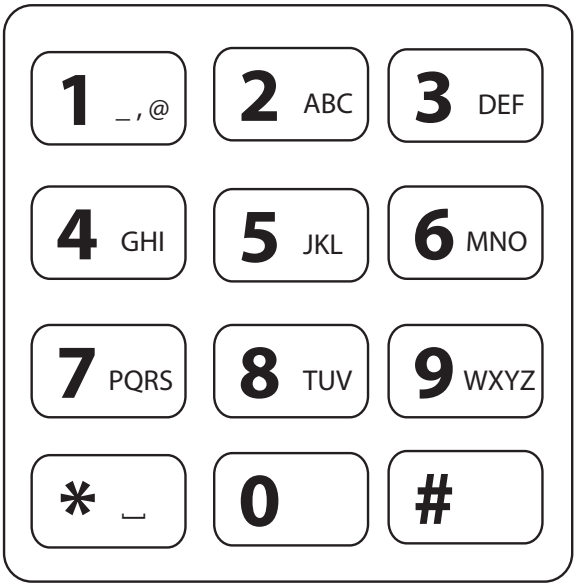

Figure 3.1: The standard 12-key telephone keypad, character layout follows the ITU E.161 standard [59]

Linguistic Disambiguation Also known as predictive input, these methods use redundant information in language to disambiguate users' input when entering standard English words. Linguistic knowledge can be leveraged either through Dictionary-based Disambiguation (DBD e.g. T9 [42]), character level N-gram models (e.g. LetterWise [81]), or a combination of both. Besides disambiguating uncertain input strings, linguistic knowledge can also be used for predicting users' future intention (a.k.a. word completion [150, 148]).

Methods in both categories have their unique strengths and weaknesses. Action based disambiguation allows users to enter any character deterministically, but requires additional sequential or concurrent actions. DBD input techniques such as T9 can achieve approximately 1.007 KSPC (Key Stroke Per Character) for words that are in the dictionary [81]. However, they depend on an alternative input method to enter words that are not in the dictionary known as out-of-vocabulary (OOV) words and suffer from the encoding collision problem (to be detailed in the next section). Character level n-gram model based disambiguation, such as LetterWise [81], can achieve a KSPC that is close to DBD and works for OOV words; however, continuous visual attention is required to confirm suggested characters after each key press.

In this chapter, we present a novel method called *SHRIMP* [2](**S**mall **H**andheld **R**apid **I**nput with **M**otion and **P**rediction) which enable the user to handle the collision and OOV

[2]The method presented in this chapter, *SHRIMP*, is named in the tradition of SHARK [155, 70] and Fisch [150]. SHARK requires a relatively large touch screen for the virtual keyboard overlay; Fisch is originally designed for a small touch screen and can be extended to a touch-ball or a joystick [148], *SHRIMP* works

problems of DBD input more easily. *SHRIMP* is a predictive text input method based on Vision TiltText [140] and runs on camera phones equipped with a standard 12-button keypad. *SHRIMP* is as effective as conventional DBD when entering unambiguous in-dictionary words. *SHRIMP* uses seamlessly integrated concurrent motion gestures to handle ambiguous dictionary words or OOV words without mode switching.

3.2 Related Work

3.2.1 MultiTap

MultiTap is perhaps the most popular text entry method for mobile phones. It requires the user to press the key labeled with the desired character repeatedly until the correct character appears on the screen. MultiTap is simple, unambiguous but tedious. It has an average KSPC of 2.03 [116].

3.2.2 Vision TiltText

Vision TiltText by Wang et al [140] is a remake of the TiltText input method by Wigdor and Balakrishnan [146]. Instead of using an accelerometer, Vision TiltText uses the built-in camera on a phone to detect motion so that it can run on unmodified mainstream camera phones. Implementation details can be found in [146] and [140]. With the help of a concurrent gesture movement, Vision TiltText can achieve 1 KSPC on any character. However, moving a cell phone left and right for entering about 60% of characters is an overhead which can be further reduced. In this chapter, we present *SHRIMP* that combines Vision TiltText with DBD. *SHRIMP* can minimize the concurrent movement requirement of Vision TiltText when entering in-dictionary words but leverage vision TiltText when entering OOV words.

3.2.3 Predictive Input

Dictionary based disambiguation (DBD) has been well-researched since the 1970s [117]. A popular commercial implementation of this kind is marketed as T9 by Tegic Communications, a former subsidiary of AOL and now Nuance Communications Inc [123]. DBD uses a dictionary to detect all the possible words that match users' numeric keypad input. For example, the numeric sequence `2-6-6-7-8-8-3-7` will result in "`computer`" because that is the only English word in the dictionary that meets the constraints defined by the input string, When multiple words in the dictionary map to the same numeric string (encoding collision), manual selection is needed if the intended word is not displayed as the first choice. For example, the sequence 6-3 may mean either "`of`" or "`me`", while the sequence `2-5-6-8-3` could mean "`cloud`", "`aloud`" or "`clove`". In an extreme situation, this encoding collision problem has caused the "SMS generation" to accept "`book`" as "`cool`" since the former is more frequent in formal English hence presented as the first choice and many users don't bother to change it to the latter [10]. Language models [63] can be used to predict the more

on unmodified camera phones equipped with a standard 12-key keypad.

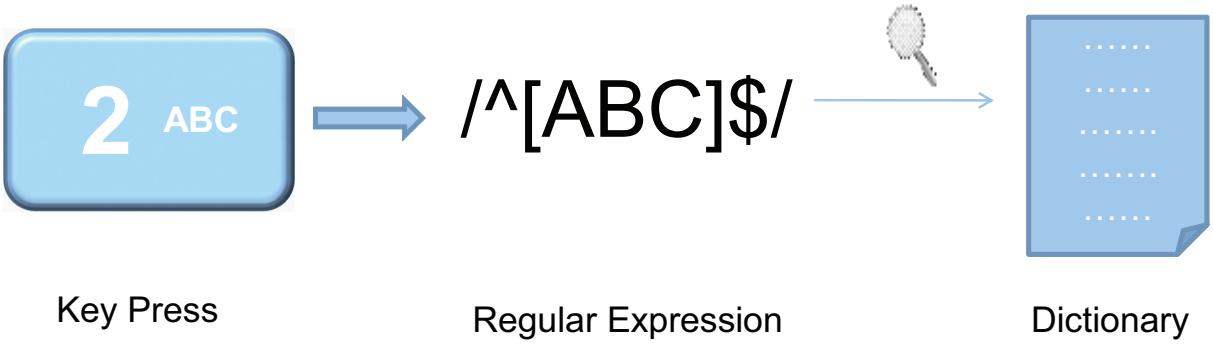

Figure 3.2: Treating DBD as a Regular Expression Matching process

likely one from the colliding candidates [103], but they cannot totally eliminate uncertainty. Another problem is that DBD only works when the user enters English words that are in the dictionary. Both Dunlop and Crossan [29] and Mackenzie et al. [81] questioned the flexibility of such a dictionary based approach. Person names, place names, company names, new product names, abbreviations, acronyms, or combinations of letters and numbers are frequently used in mobile environment but are not likely in the dictionary. According to Jansen et al. [62], 20% of Twitter posts mention specific product and brand names. Dunlop summarized the dilemma well: "*the dictionary method is not a sufficient input method on its own but having two modes is likely to lead to great confusion*" [29].

OOV words can be handled with additional time consuming work around steps. For example, the following approach is usually implemented in commercial products running DBD input such as T9 (used in mobile phones from Nokia, Samsung LG and others). For an OOV word, the user can use the "UP" and "DOWN" arrow key to navigate though the character candidates from each key press and use the "RIGHT" arrow button to confirm character after character. Similarly, collision words are handled by navigating through the multiple matching words via the "UP" and "DOWN" arrow if the intended word is not the most frequent one. In this chapter, we show that *SHRIMP* is a dictionary based predictive input method that supports the entry of any word efficiently without mode switching.

3.3 The Design of SHRIMP

3.3.1 DBD as Regular Expression Matching

Predictive text entry via DBD can be considered a regular expression [2] matching problem and *SHRIMP* can be defined as a natural upgrade of DBD under the regular expression matching framework.

When a user starts to press button '2' in DBD, he/she tells the system that the intended word starts with either '`a`', '`b`', or '`c`'. This kind of constraint can be captured by the regular

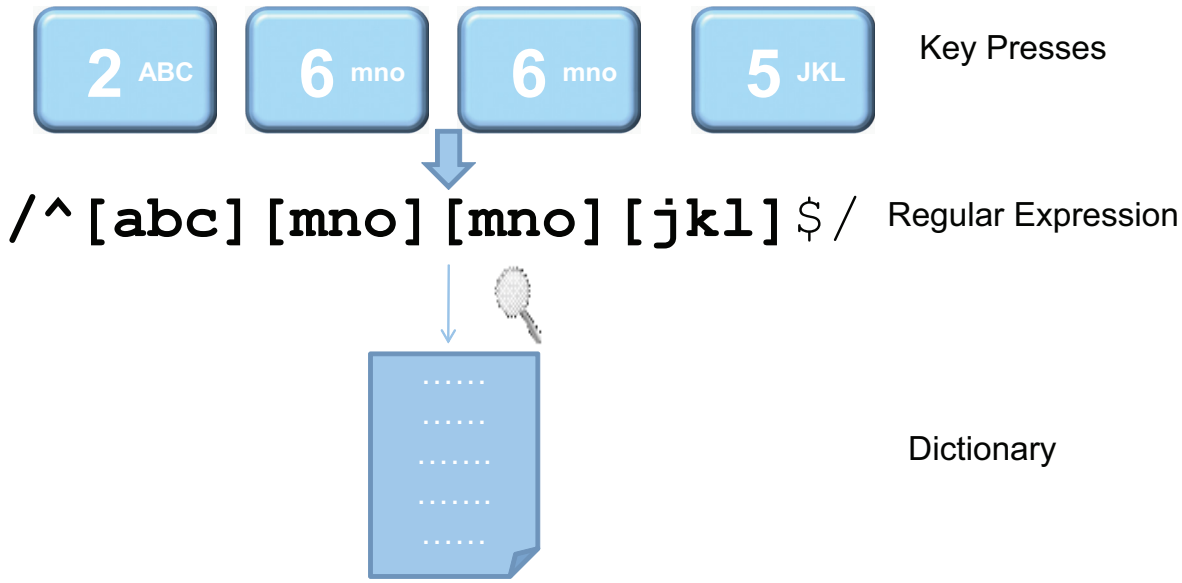

Figure 3.3: Treating DBD as a Regular Expression Matching process

expression `/^[abc]$/` [3] (Figure 3.2); if the user follows with a key press '6', the regular expression is extended as `/^[abc][mno]$/`. If the user finishes the whole word with two more characters '6' and '5', the regular expression becomes `/^[abc][mno][mno][jkl]$/`. As a result, DBD input can be considered as a query to the dictionary to retrieve all words that match the given regular expression (Figure 3.3). In this example, the words "`book`" and "`cool`" are the candidates that match the regular expression and would be the output of DBD.

3.3.2 SHRIMP as an Extension of DBD

Pressing a numeric button in DBD can be considered as adding a `[wxyz]` style constraint to the existing regular expression. However, it is not the only constraint we can add from a regular expression perspective. If we use a Vision TiltText gesture to enter character '`c`' at the beginning, i.e. press and hold button '2', move right, release. This action will tell the computer that the intended word starts with '`c`' rather than `[abc]`, the corresponding regular expression for this action will be `/^c$/`. If we continue the input without motion by typing '`6-6-5`', the final regular expression will become `/^c[mno][mno][jkl]$/` and '`cool`' will be the only word that matches the given regular expression (Figure 3.4). Vision TiltText can be incorporated at any stage of the word entry. Any time a motion gesture

[3] Symbol ^ and **$** in the regular expression mark the beginning and end of a character string. **[abc]** means that either **a**, **b** or **c** could be a valid match.

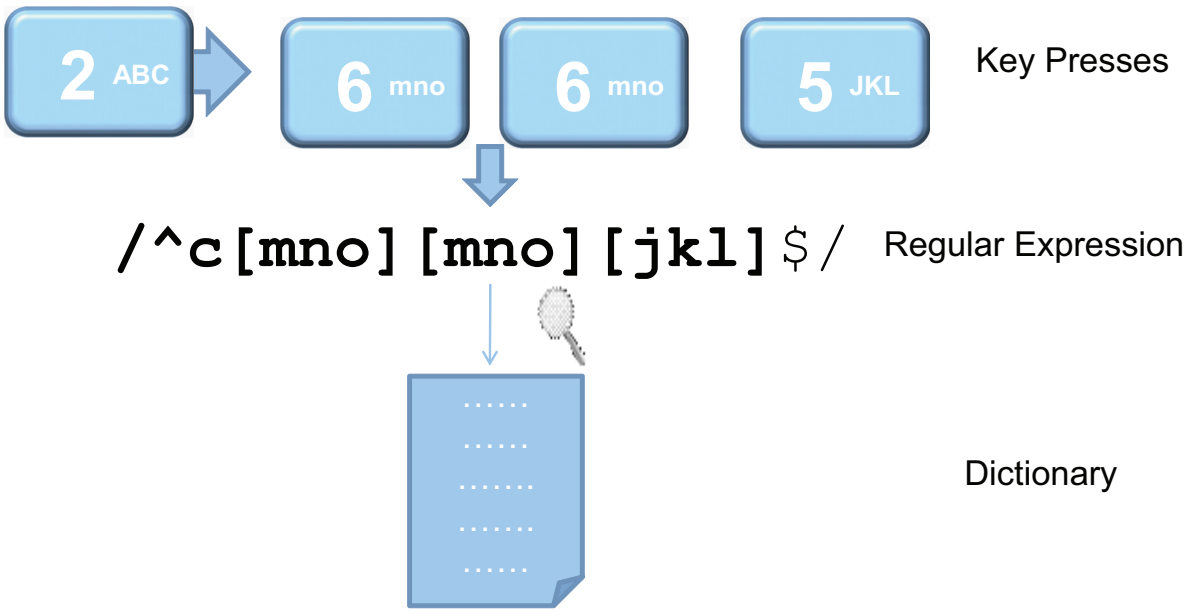

Figure 3.4: Treating *SHRIMP* as a natural upgrade of DBD under the regular expression matching framework

is used, it is telling the system that only one character, determined by Vistion TiltText, should appear at the current character location and the specific character, rather than the bracket enclosed character set will be used to construct the corresponding component in the regular expression. Figure 3.5 shows the steps and corresponding screen output to enter word '`cool`' by using MultiTap, DBD and *SHRIMP* (here motion based constraint is used to enter the last character '`l`').

DBD and Vision TiltText can be considered two extreme cases of *SHRIMP* (Figure 3.6) - if we type every character without motion, then the output of *SHRIMP* will be no different from DBD. If we type each character with motion constraints, then *SHRIMP* becomes Vision TiltText so OOV words can be entered easily. Figure 3.7 shows how DBD and *SHRIMP* could be used to enter an OOV word - "`hci`". *SHRIMP* can save four out of the seven key strokes required by DBD with concurrent motion. Note that the character level confirmation approach that appeared in commercial products is used in DBD to enter OOV word in this example. If the user chooses to switch to MultiTap, the total number of key presses would become - 1(switch to MultiTap) + 2 (`h`) + 3(`c`) + 3(`i`) + 1(switch back to DBD) = 10.

There is one more case that needs additional consideration in *SHRIMP* - the action of typing a key without movement. It could mean the user did not bother to express his/her preference so he/she may intend any of the characters shown on that key. Alternatively, it could mean the user intended to type the middle character on the button via Vision TiltText (by default, no motion means entering the middle character in Vision TiltText). Such ambiguity can be addressed with two different approaches. First, we can redefine

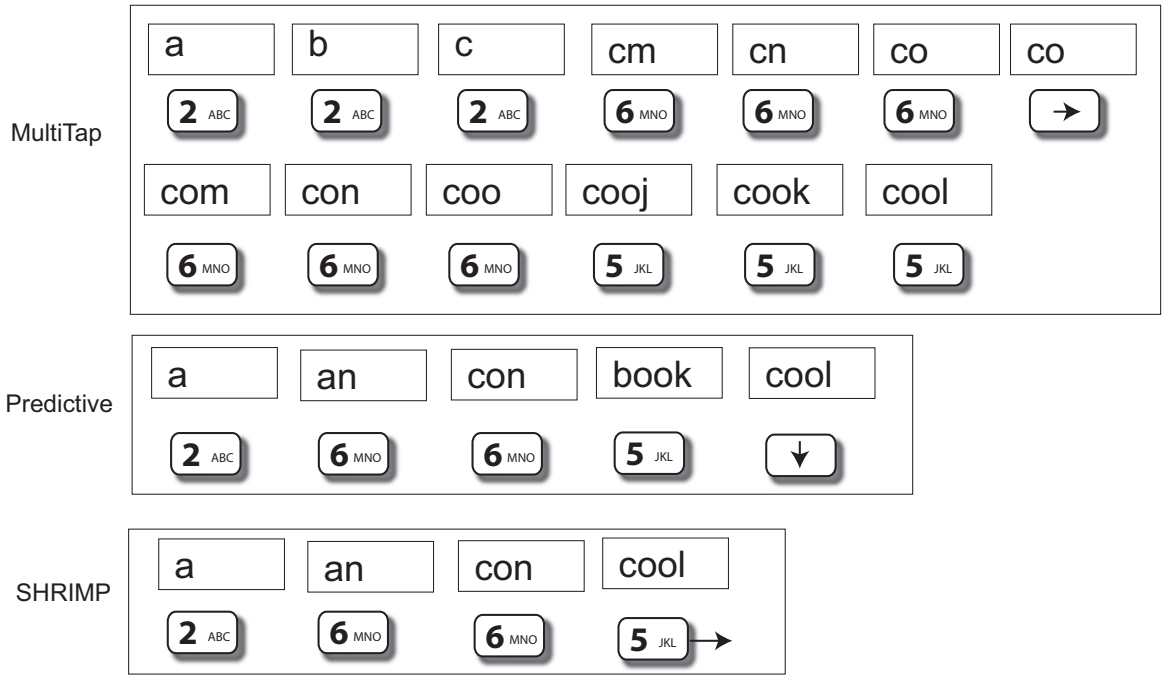

Figure 3.5: Using MultiTap, Predictive Input (DBD) and *SHRIMP* to enter a word 'cool' with encoding collision

Figure 3.6: SHRIMP is a seamless integration of Vision TiltText and DBD(T9)

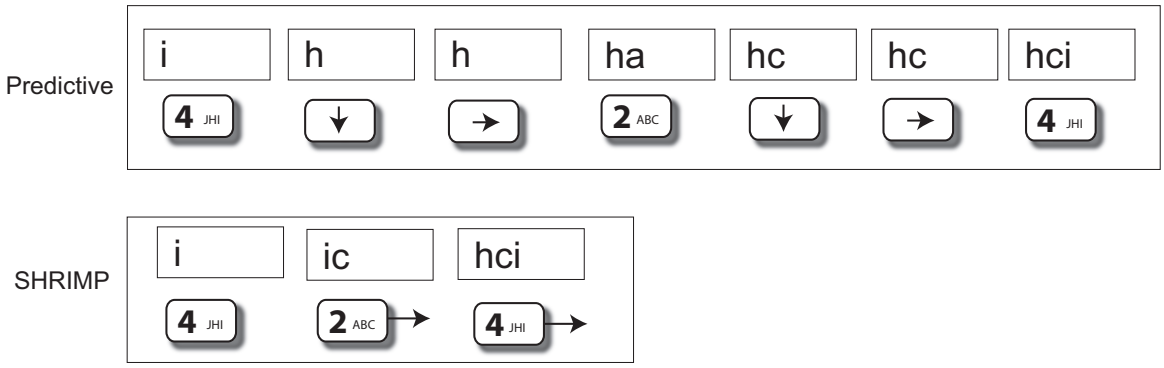

Figure 3.7: Using DBD and SHRIMP to enter out of the dictionary words

typing the middle character in Vision TiltText as a "press"-"move up"-"release" action to eliminate such kind of ambiguity completely. Second, the following heuristics can be used - we first assume the no movement situation as DBD style ambiguous input to construct the regular expression. If some words in the dictionary can match the given regular expression, and the Vision TiltText style unambiguous interpretation corresponds to an OOV word, then the OOV word will be added as the last candidate just in case the user intends to type the OOV word rather than the word within the dictionary. If no word matches the given regular expression, then we can interpret every no-movement key press as typing the middle character via Vision TiltText. We have implemented both approaches in our prototype and have found that the second approach more convenient in actual usage. This is because when entering an OOV word, the users have to type every character in Vision TiltText and the chance that this constrained input will match existing words in the dictionary is small (please refer to the next section for details).

SHRIMP has three major advantages when compared with other linguistic based disambiguation or word completion methods. First, *SHRIMP* can enter both in-dictionary words and OOV words, without mode switching. Second, searching for correct candidates visually after each key press is not required; users only need to visually confirm the result at the end of a word. Third, *SHRIMP* provides a smooth learning curve for beginners - it is not necessary to remember when and where to add motion constraints and adding more motion constraints than necessary won't change the input result (as illustrated in the next section - adding one, at most two motion constraints will disambiguate encoding collisions in the worst case). Even if a user completely ignores to use motion constraints, *SHRIMP* is equivalent to DBD in this worst case. Different from DBD, *SHRIMP* provides users an opportunity to enter "troublesome" words more effectively next time. If the same problematic word shows up frequently (e.g. "`book`" for "`cool`"), a user is more likely to use the motion based gesture to deal with it next time. In short, *SHRIMP* allows the user to be more "expressive" than traditional DBD, but to a completely voluntary degree. There is no downgrade from DBD for not using the additional expressive power through motion gesture.

In the next section, we show the feasibility and power of *SHRIMP* via corpus analysis.

3.4 Analysis

Analyzing text entry performance realistically is a challenging task. For the current topic, we need to consider three different types of words:

1. Words in the dictionary without encoding collision
2. Words in the dictionary with encoding collision
3. OOV Words

Previous studies tend to focus on words in categories 1 and 2 [30].

As discussed in the previous section, *SHRIMP* is similar to DBD when typing words in the dictionary without encoding collisions (Type 1 words). When typing words with encoding collisions (Type 2 words), some of the characters can be entered with motion sensing to reduce ambiguity. When typing OOV words (Type 3 words), users need to provide a motion gesture for each character, and the experience of *SHRIMP* is no different from Vision TiltText. As a result, the actual performance of *SHRIMP* vs. DBD will depend on the distribution of the three types of words in the user's writing.

In order to understand the distributions of these types of words, we performed quantitative analyses on two text corpora. The first is the Brown Corpus [71] from which we extracted the top 17,805 words based on frequency rank, stripped punctuations and normalized them to lower case. This corpus is similar to the one used in [40]. We used this corpus to study the encoding collision problem in DBD. The second corpus is the NUS SMS Corpus [57] [4]; it has 7,189 distinct words representing text messaging vocabulary. This corpus gives an opportunity to study the OOV problem in mobile environment.

Two types of encoding collision analysis were conducted - raw (treating each distinct word with the same weight) and frequency weighted (each word weighted by its frequency of occurrence calculated from the corpus). The raw analysis gives a sense of proportion to all unique words including rare words. The weighted frequency analysis is proportional to natural occurrence in real use (according to the word frequency distribution in the corpus used).

3.4.1 Encoding Collision Analysis

Figure 3.8 shows the distribution of words encoding collisions on the standard 12-key keypad (Figure 3.1) in the Brown Corpus. The horizontal axis is the category of collisions (namely the number of words sharing the same key press sequence). For example, the percentage shown in the first two bars (raw and frequency weighted percentage respectively)

[4] Available for download at `http://www.comp.nus.edu.sg/~rpnlpir/downloads/corpora/smsCorpus/`

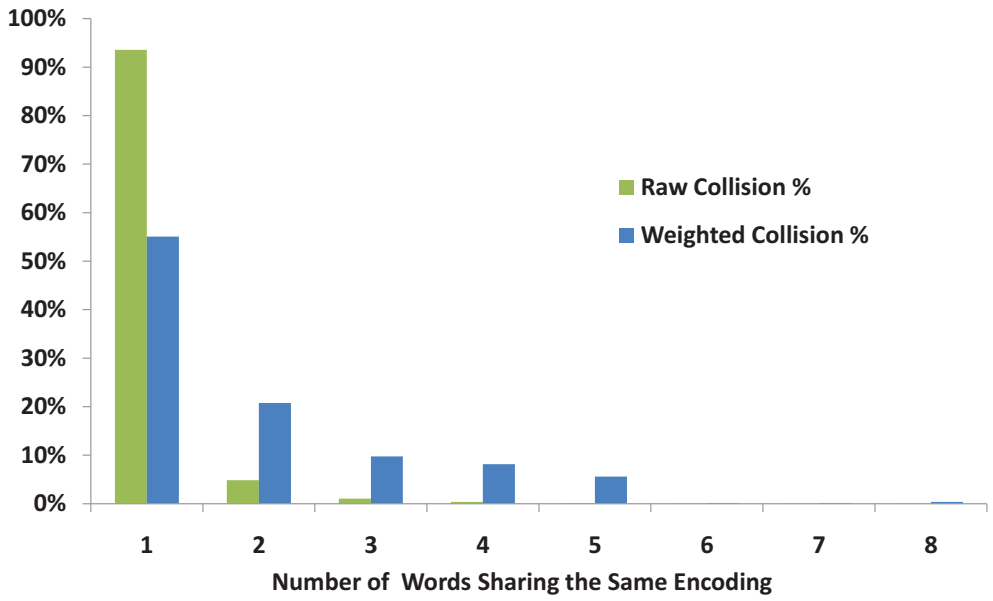

Figure 3.8: Word distribution in different collision categories according to the number of words sharing the same key code.

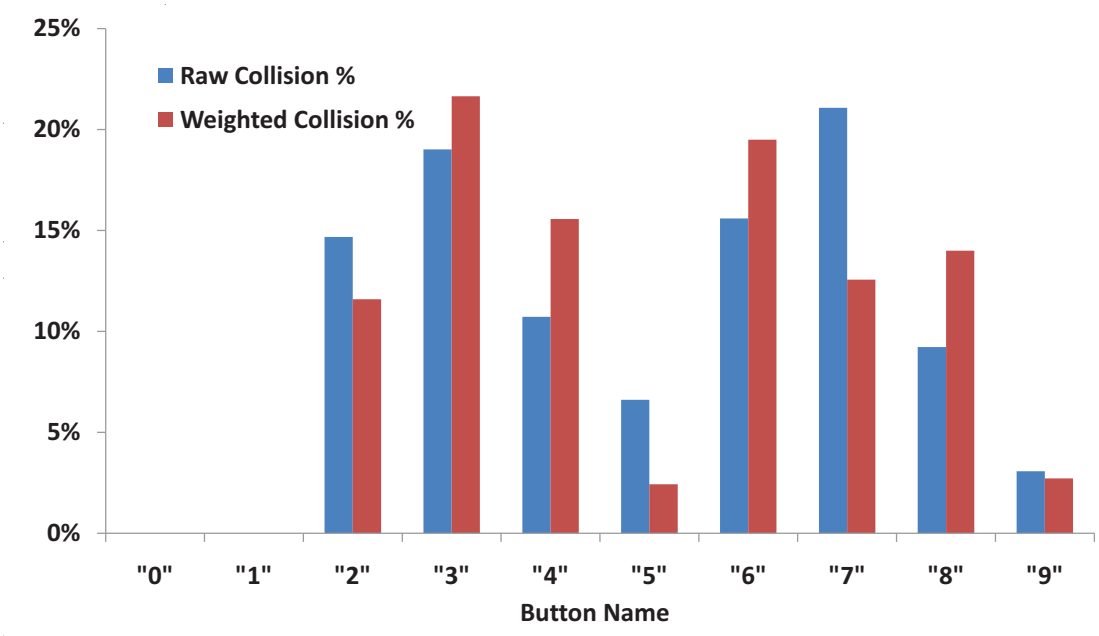

Figure 3.9: Collision distribution by Numeric Button.

where the collision number is 1 are the percentage of words without encoding collision. From Figure 3.8 we see that there can be as many as eight words that collide on the same numeric encoding. For DBD, 6.4% of words have encoding collisions in the raw analysis. In the weighted analysis, the collision rate raises to 44.9%. Not all of these 45% collisions would require additional action of the DBD users. For example, when two (and only two) words collide on the same code, outputting the word with higher frequency will be correct in more than 50% of the time by definition. More complicated prediction methods based on for example a word level N-gram statistical model may further increase the chance of success when multiple words are in collision. However, not only these methods may require the amount of CPU and memory still not available on mobile phones, statistical models based on large and formal corpus analyses may not apply well to actual mobile use at all. The previously mentioned book vs. cool is one example.

Which buttons do words tend to collide on? The analysis summarized in Figure 3.9[5]

[5]Button "**0**" and button "**1**" are not used for encoding alphabet characters in the ITU E.161 standard [59].

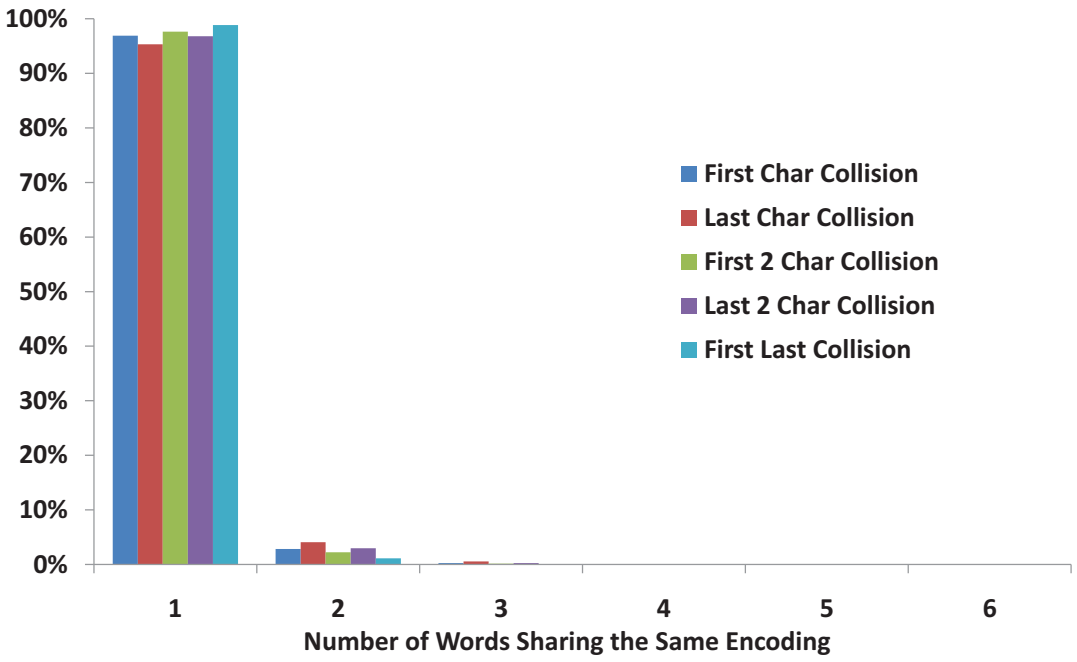

Figure 3.10: Raw SHRIMP Word Collision Frequency by Typing Strategy and Number of Words Sharing the Same Encoding.

shows that based on the weighted analysis most ambiguities appear on button "3", "6", "4", "8", corresponding to the character groups "[def]", "[mno]", "[ghi]" and "[tuv]". So these buttons can benefit the most from *SHRIMP*'s motion gesture enhancement. In contrast, words rarely collide on buttons "5[jkl]" and "9[wxyz]". A possible design implication is to graphically mark the collision prone buttons in a way that encourages the use of motion gesture on them.

As previously stated, if motion gesture is used for every button press, then *SHRIMP* becomes Vision Tilt Text and the output will be completely unambiguous. If motion gesture is only used on some of the button presses, then the frequency of word collisions can still be reduced although not completely eliminated. Figure 3.10 and 3.11 show the result of using hypothetical strategies of partial motion gesture use: on first letter only, on last letter only, on both first and last letter, on the first two letters, and on the last two letters of a word. Compare them with Figure 3.8 one can see the dramatic reduction in collision with these partial use strategies. For example if we only use motion gesture to disambiguate the first

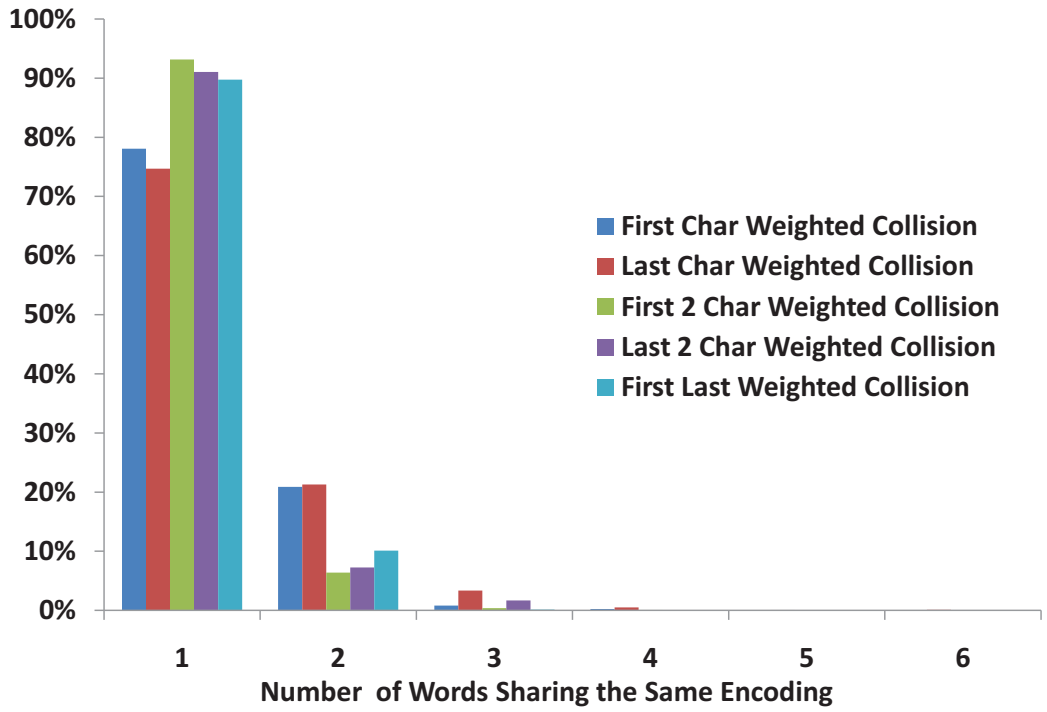

Figure 3.11: Weighted SHRIMP Word Collision Frequency by Typing Strategy and Number of Words Sharing the Same Encoding.

key press, the encoding collision will drop from 6.4% in DBD to 3.1% in the raw analysis (a 52% drop) and from 40.4% to 21.9% in the weighted analysis.

When motion gesture is used with *SHRIMP*, even only some of the times (e.g. on first and last only), the maximum number of words colliding when collision do occur can be reduced to less than three. Three words can be disambiguated easily selected through a space button press coupled with a motion gesture (left, none, right).

3.4.2 OOV Analysis

It is well known that the users' vocabulary in mobile text entry is different from formal written English collected in more formal text corpora [29, 81]. However, due to the lack of large scale, publically available text corpora from mobile users, it is difficult to quantify the difference between formal written English and the language people use in mobile text entry. To our knowledge, the only publically available mobile text corpus is the NUS SMS Corpus [57]. Despite its relatively small size (121k words with 7,189 distinct words), it has been adopted by HCI researchers to simulate word frequencies in mobile applications [40]. The corpus is a collection of self-submitted SMS messages from students at National University of Singapore (NUS). Because of the limited sample size, it is difficult to estimate how representative this corpus is when compared with the whole mobile population. As a result, the NUS SMS corpus could only serve as a case study on the OOV problem.

Based on our analysis, the NUS SMS word frequency is quite different from that in traditional written English. For example, the top 8 words in the Brown Corpus - "`the`, `of`, `and`, `to`, `a`, `in`, `that`, `is`" have little overlap with the top 8 words in the NUS SMS Corpus - "`i`, `u`, `to`, `me`, `at`, `my`, `go`, `you`". Only the word "`to`" appears in both lists. The top three words in formal English - "`the`, `of`, `and`" did not even make it into the top 8 list of NUS SMS corpus.

Our analysis shows that only 49% of the words in the SMS Corpus can be found in the traditional word corpus by raw frequency. In the frequency weighted calculation, only 36% of the words in SMS can be found in the traditional word corpus. Sample out-of-the-dictionary words include "`lor`, `liao`, `okie`, `slackin`, `thkin`". This means that if we create the DBD dictionary by parsing a traditional corpus based on written English and use a DBD method based on that dictionary to enter the NUS SMS Corpus, 64% of the intended entries will be OOV.

As mentioned earlier, a fully 20% of Twitter micro blogs contain product and brand names [62]. A very high percentage of these names are likely to be OOV. Twitter imposes a 140 character limit per posting. This limit, plus the relatively slow process of entering text on mobile phones in general, are likely to encourage users to create ad hoc abbreviations that are OOV.

In conclusion, although we have only one quantitative case study available, it is safe to say a large percentage of words in SMS and Twitter like applications are OOV. The problems of collision and OOV with the traditional DBD method are frequent enough to warrant a *SHRIMP* or *SHRIMP* like solution.

A follow up question is when a user enters an OOV word, would he/she realize the word

is OOV therefore engages *SHRIMP*'s motion gesture to preempt DBD's failure or the need to switch to a different mode? The question is important to both DBD and *SHRIMP* and will be addressed in the user study reported later in this chapter.

3.5 Implementation

Although lacking in some mobile HCI research, it is important to test mobile interface concepts and methods with real mobile device for both engineering feasibility and interaction form factor reasons. We have made a complete implementation of *SHRIMP* on a Motorola v710 phone (a CDMA Phone from Verizon Wireless). This was a common off-the-shelf camera phone at the time of our implementation. The v710 has an ARM9 processor, 4M RAM and a $176 * 220$ pixel color display. Our application is written in C++ in BREW 2.11 (the Binary Runtime Environment for Wireless [13]). We used the Realview ARM Compiler 1.2 for BREW [104] to cross-compile the target application. We used the TinyMotion library [128, 140] for real time camera phone based motion sensing. The compiled target application, including the implementation of MultiTap, DBD, Vision TiltText, *SHRIMP*, as well as a 15k words dictionary and the corresponding index data is around 1.3MB in size[6]. The required runtime memory is about 250k. At this time, we have also ported *SHRIMP* to a Motorola RAZR V3 cell phone that runs BREW 3.12. We believe that porting our code to other platforms, such as Windows Mobile, Symbian and Android, would be straightforward.

3.5.1 DBD

The programming of DBD is straightforward. Particularly worth noting is how collision and OOV are handled in its UI. When there is a word collision, our implementation of DBD allows the user to use the "UP" and "DOWN" arrow buttons to navigate through the word candidate list and use the space key (i.e. the star button in Motorola V710[7]) or the "RIGHT" arrow button to confirm. Basic DBD input methods [61] cannot handle OOV words. In our implementation, in addition to allowing the user to switch to another input method such as MultiTap, we also implemented an extension to DBD that is available in most Nokia, Samsung and LG phones. When entering an OOV word, the "UP" and "DOWN" arrow button can be used to navigate though the candidates of each character and use the "RIGHT" arrow button to confirm character after character. Most users in our user study prefer this extension rather than switching to and from MultiTap to enter OOV words.

3.5.2 SHRIMP

We implemented *SHRIMP* as a natural extension of both Vision TiltText and DBD, so most of the operation conventions in Vision TiltText and DBD are kept intact. For instance,

[6] In the current implementation, DBD and *SHRIMP* shares the same dictionary and a major portion of the index data, the input method code plus the motion sensing code alone, is about 150k in size.

[7] The space button could be assigned to the pound ("#") key or the zero ("**0**") key in other cell phones.

users can press button "2", hold it, move/tilt the phone to the left until a vibration is felt and then release the button to indicate that character "a" is desired. The user can also use "UP" and "DOWN" buttons like in DBD to correct an ambiguous word. Of course, the candidate list is much smaller than that of DBD if additional motion constraints are used. The same 70 ms vibrato-tactile feedback mechanism [140] is also used in *SHRIMP* to signal that the critical movement amount has been reached.

3.6 An Initial Empirical Study

The quantitative analyses presented have shown the following.

1. Collision and OOV are both quite frequent problems for DBD;
2. *SHRIMP* can significantly reduce collision even if motion gestures are only used sparingly (on first letter or only when a word has been known to be troublesome from previous experience);
3. *SHRIMP* can handle OOV more effectively than previous DBD workaround methods.

To complement these analytical findings, we also conducted an empirical study to do an initial but holistic test of *SHRIMP* as an everyday text entry method. We had two basic goals for the study. One was to figure out whether or not the idea behind *SHRIMP* is easy to understand and if the current *SHRIMP* implementation is easy to learn. The second goal was to evaluate the initial performance of *SHRIMP* in comparison with existing text entry methods such as MultiTap, DBD and Vision TiltText. A successful mobile text entry method should not have a steep learning curve. The users should be able to pick up the method in a few minutes, and users should gain immediate benefits. Otherwise, many users may give up and switch back to older methods they were accustomed to. As part of the study we also measured users' ability to recognize OOV words. Analyzing the longitudinal performance of *SHRIMP*, which may reveal its advantage in handling collision, and a systematic empirical comparison across different word categories (unambiguous, collision, and OOV words) are beyond the scope of this chapter and will be deferred to future work.

3.6.1 Study Design

Our study consisted of five parts.

Overview We gave a brief overview of the tests to be conducted and demonstrated the four text entry methods, MultiTap, DBD, Vision TiltText and *SHRIMP*, to the participants. To help them better understand the motion sensing idea behind Vision TiltText and *SHRIMP*, we also gave a brief introduction of TinyMotion and demonstrated other motion sensing applications to the user. We let the users play with the demo applications and answered their questions. This session lasted 10 to 15 minutes.

Pre-test Questionnaire In this session we collected basic information of the study participants including gender, educational background, current occupation, experiences with cell phones and camera phones, frequency of mobile tasks such as voice communication, SMS usage, picture taking, use of other mobile phone applications etc.

Out of Vocabulary Word Recognition In this session we tested participants' ability in identifying OOV words. We told users that, similar to a spell checker, many mobile text entry methods maintain a list of words to predict users' intended words based on the keypad presses. We told the users that the input methods to be tested used a word list of 15K of the most popular words and showed them samples of words in the list as well as words not in the word list. We then gave the participants a list of 21 words and let them identify each of them as in-dictionary or OOV (Table 3.1). It's worth noting that none of the OOV words in this section appeared in the follow-up text entry tasks and the subjects didn't know their performance on the OOV recognition task throughout the study. So the confounding effect between the OOV recognition task and the actual text entry task was minimal.

Text Input In this session, we compared the performance of four mobile text entry methods - MultiTap, Vision TiltText, DBD and *SHRIMP*. The testing phrases were selected randomly from MacKenzie's text entry test phrase set. The timeout for the MultiTap method was 2 seconds. DBD and *SHRIMP* share the same dictionary, which has 15k words sorted by word frequency. Due to time constraints, each participant entered 12 sentences with a total of 52 words for each input method. Although a popular source of testing phrases used in recent studies on text entry such as [146], MacKenzie's phrase set [83, 112] has a limitation to our study - most of the sentences were formal and "correct" English and the word frequency in this phrase set were designed to simulate the corpus of formal written English [83, 89], not those used in SMS or other mobile applications. However, we felt these phrases would still serve the purpose of this initial pilot study - to test if users can understand and use *SHRIMP* without much practice. A Motorola V710 mobile phone loaded with our application was used in the experiment.

This session started with a warm up practice phase in which the users could practice with the four methods tested for as long as they desired. Most of them choose to practice for 2 to 10 minutes before the actual test. The order of the four input methods was balanced via the order four Latin square patterns.

Collecting Qualitative Feedback We conducted a final survey immediately after a participant completed all the sessions. In the survey the participant completed a questionnaire and commented on the input methods they tested.

To simulate the real world situations of cell phone usage, we did not control the environment used for the study. The participants were encouraged to choose their desired locations to complete the study. Most of the studies were completed in the participants' own chair or at a public discussion area in a lab. Figure 3.12 shows some of the actual environments used during the study.

Sample in-dictionary words	the I use cab fox looks priority computers computing computation jeopardize Israel hodges smell smells smoked accommodation accommodations California England diplomacy amazon Arnold Michael Tom Smith
Sample OOV words	iPhone leet Ramirez Emeryville Ohlone NYPD facebook Chang twitter gmail Google Obama thanx eBay Jingtao Cerrito Canny
Testing words	Shaharyar compelling FastTrak Sillers Costco Yvette learning forgot effectiveness babies Carlson deadline please StarBucks know craigslist room airport Michigan organizations advisor

Table 3.1: Sample in dictionary, out of vocabulary words and testing words used in the OOV words recognition task.

Figure 3.12: Sample pictures taken from the user study.

All our input methods run on real, unmodified cell phone hardware and confounding factors such as dictionary size, user interface and screen resolution have been controlled. We will release the source code of all four text entry methods, data collection application and log processing application under the BSD license. We hope it can establish a standard and realistic testing platform for mobile text entry research and provide a baseline for future studies.

3.6.2 Participants

Twelve people participated in the study. Nine of them were undergraduate or graduate students in a university, two were visiting researchers of the university and one was an instructor at a local high school. Three of the participants were female and nine were male. All of them owned a cell phone at the time of the study and 11 of the 12 cell phones were camera phones. On average they had 7.5 years of experience in using cell phones (stddev = 2.2). 10 out of 12 cell phones were equipped with the standard 12-key keypad. One cell phone had a mini-QWERTY keyboard (Palm Centro) and one cell phone was touch screen only (Apple iPhone). 9 participants reported that MultiTap was their primary text entry method on cell phones; The other three participants used DBD, mini-QWERITY keyboard, or a virtual on-screen keyboard for mobile text entry. All of the participants completed all the five sessions in our user study.

3.7 Empirical Results

3.7.1 Out of Vocabulary Word Recognition

Participants correctly identified OOV words 97.6% of the time. No in-dictionary word in the test was incorrectly recognized as OOV. Among the 6 OOV recognition misses, the person name "`Carlson`" was incorrectly categorized as "in the dictionary" four times. Similarly, "`Yvette`" was incorrectly categorized as "in the dictionary" two times. Both participants who mislabeled "`Yvette`" also mislabeled "`Carlson`". From this preliminary test, it seemed that people were overall fairly proficient at estimating whether a word was in the vocabulary even if they only had a few samples from the dictionary. In our experiment, the participants had no problem in identifying OOV business names, place names, and abbreviations. It was more difficult estimating the popularity of person names. People's social networks may have a huge impact on their perception of "frequent names".

3.7.2 Text Input

In total 14,281 characters were entered (including white space and editing characters). There were 41 unique words in the test sentences. 17 of them had no encoding collisions and 21 of them had encoding collisions. Among the 21 words that had encoding collisions, 16 of them do not require explicit action of the user if we output the words with the highest word frequency from the candidate list. 2 words were OOV words.

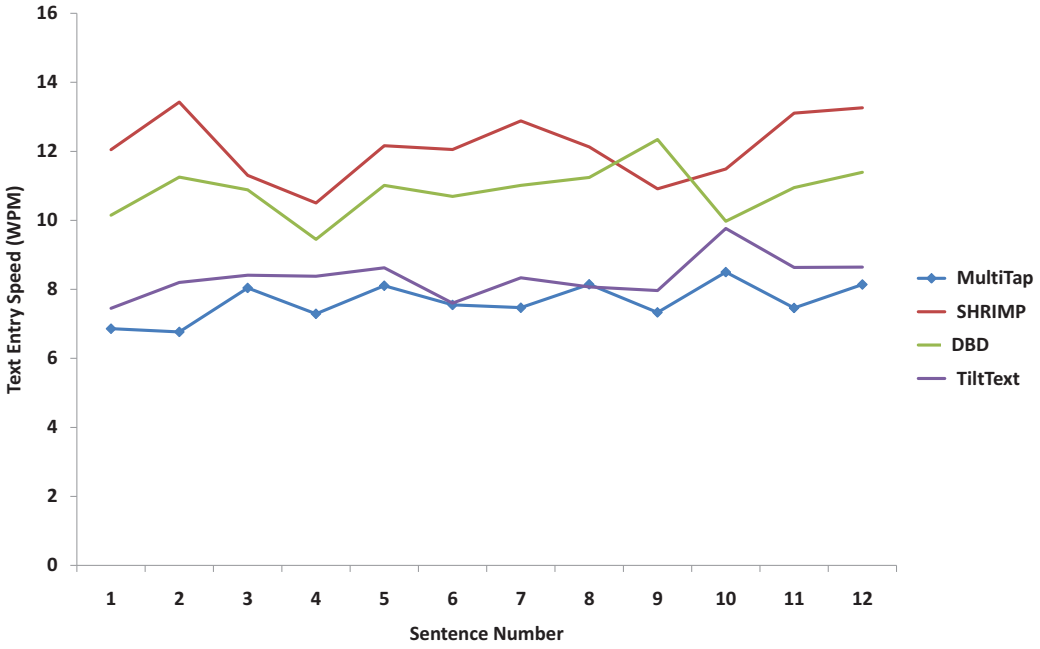

Figure 3.13: Text entry speed (WPM) by technique and sentence number for the entire experiment.

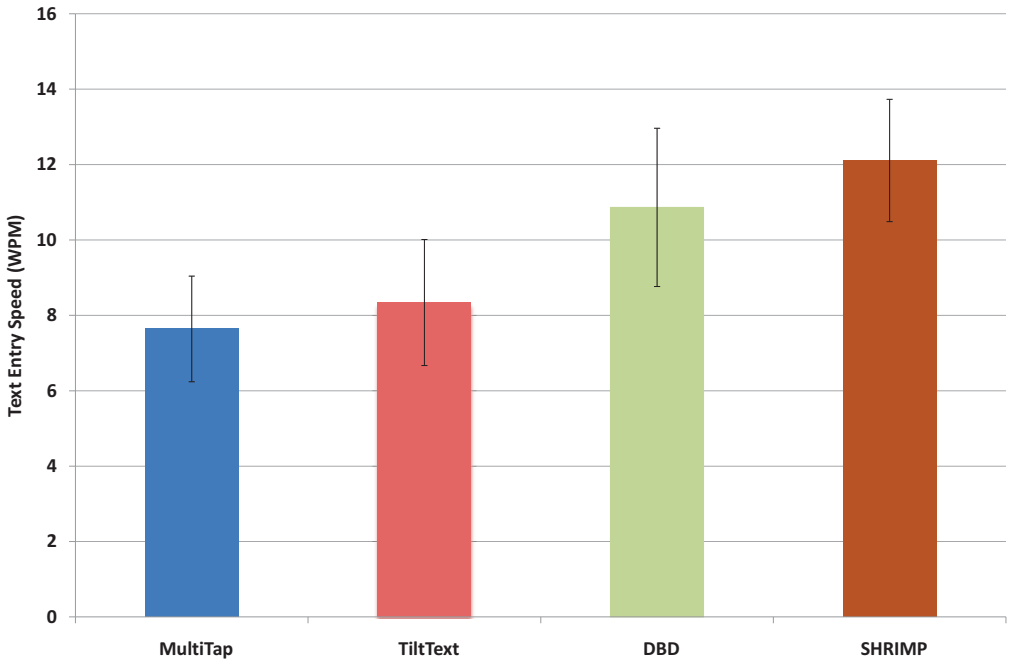

Figure 3.14: Text entry speed from the experiment.

Figure 3.13 shows the speed of the four different text entry methods we have tests in the pilot user study. As stated in the experimental design section, users started the tests only after 2 to 10 minutes of practicing. All of the users had previously used MultiTap (with an average of 5.3 years of experience) while only one user had used DBD frequently before our study. So the results on Vision TiltText, DBD and *SHRIMP* can be viewed as users' initial text entry speed without much practicing. A longitudinal study is needed to understand the expert performance of these methods. From Figure 3.13, we can see that *SHRIMP* and DBD are already faster than MutiTap and Vision TiltText when typing the first three sentences.

As shown in Figure 3.14, input speed varied from one method to another. Repeated measure variance analysis showed significant difference due to input method: $F_{(3,33)} = 110.9, p < .0001$. Fisher's post hoc tests showed that the speed of *SHRIMP* was significantly higher than the speed of MultiTap ($p < 0.001$) or the speed of Vision TiltText. The average speed of *SHRIMP* (12.1 wpm) was higher than that of DBD (10.86 wpm), the difference was significant in paired-sample t-Test ($p < 0.001$) but not significant in two sample t-Test ($p = 0.12$). DBD was significantly faster than MultiTap ($p < 0.001$). Vision TiltText (8.34 wpm) was faster than MultiTap (7.64 wpm) in average, but the difference was not

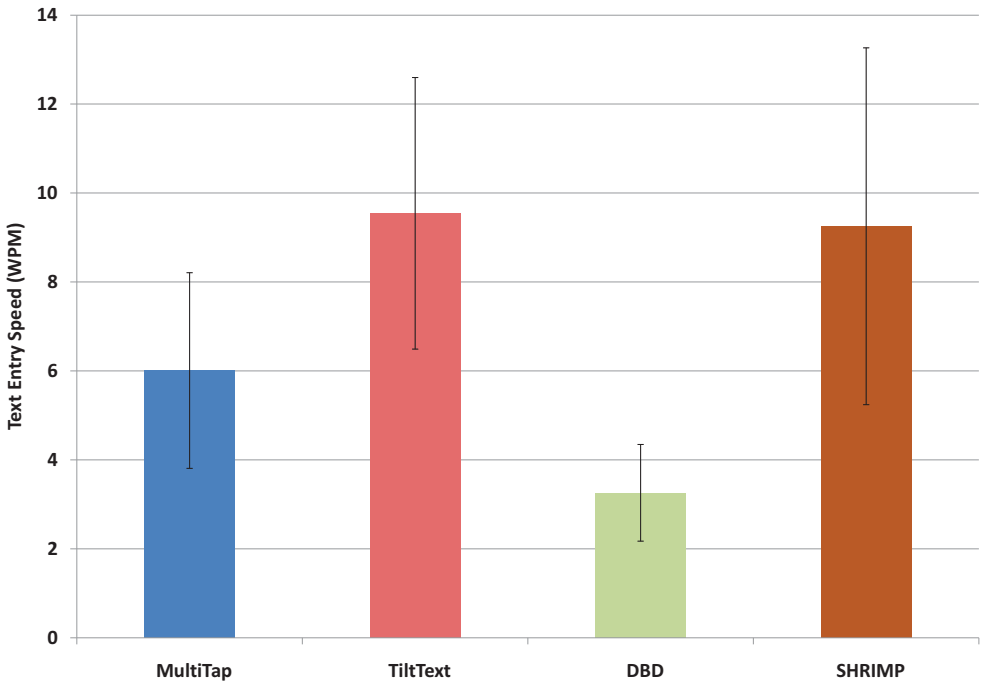

Figure 3.15: Text entry speed for the OOV words.

statistically significant ($p = 0.07$).

As we discussed earlier, the testing phrases in the study resembled formal English. As a result there were only two OOV words in the 12 sentences. The close performance of DBD and *SHRIMP* in Figure 3.14 was in part caused by the relative low percentage of OOV words in our test sentences. As a follow up analysis, we isolated logs for OOV words. As shown in Figure 3.15, the text entry speed of DBD (3.3 wpm) dropped drastically when entering OOV words. In fact DBD became the slowest input method among all four methods. The speed of DBD for OOV words was significantly lower than MultiTap ($p < 0.001$), Vision TiltText ($p < 0.001$) and *SHRIMP* ($p < 0.001$). The speeds of *SHRIMP* (9.3 wpm) and Vision TiltText (9.5 wpm) in handling OOV were not significant ($p = 0.059$).

The uncorrected error rate was less than 0.5% for each method. The average error rates for MultiTap, Vision TiltText, DBD, *SHRIMP* were 7.2%, 14.1%, 2.8% and 2.4% respectively (Figure 3.16. The overall error rate [149] of *SHRIMP* was significantly lower than that of MultiTap ($p = 0.017$). There was no significant difference in error rate between *SHRIMP* and DBD ($p = 0.76$). The error rate difference between *SHRIMP* and Vision TiltText was also significant ($p < 0.001$). The error rate difference between Vision TiltText and MultiTap was also significant ($p = 0.04$). This result agrees with previously reported

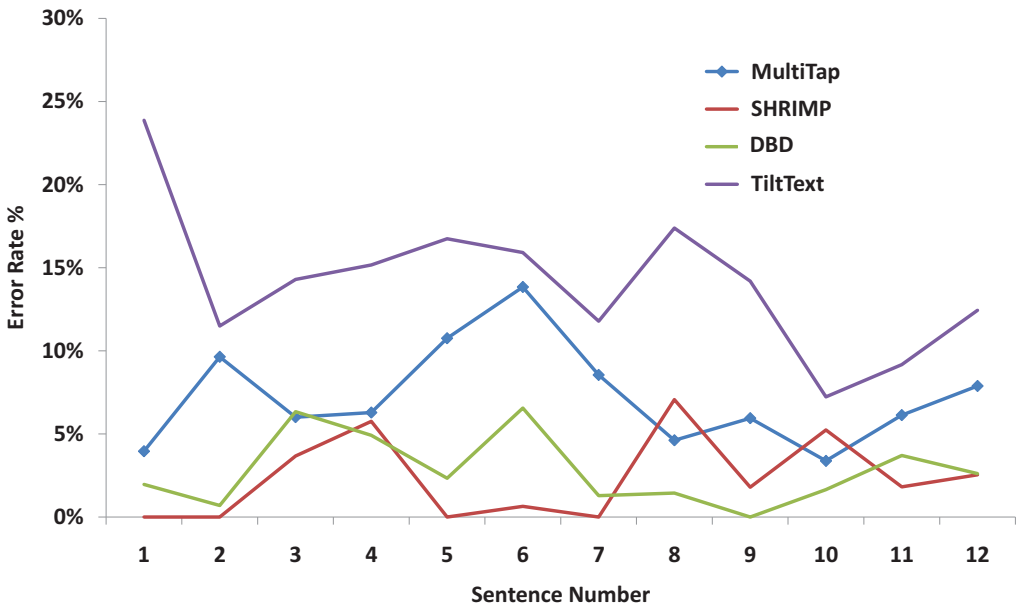

Figure 3.16: Overall text entry error rate by technique and sentence number for the entire experiment.

error rates in similar tasks [146, 140].

Users in general have no trouble understanding the working mechanism of *SHRIMP*. Initially, many users tended to add more motion gestures than necessary, trying to enter a few characters with motion in each word. "*I just want to verify whether the system works exactly as you described*" one user explained. After they confirmed that adding motion gestures truly worked in *SHRIMP*, they started to trust the system. They typed familiar words without motion and used motion gestures when they were not sure whether a word, such as "`jedi`", was in the dictionary.

After finishing the study, the users generally felt positive about *SHRIMP*. They expressed their desire to switch from MultiTap to alternative input methods such as *SHRIMP* if they were available on their cell phone. Sample comments included -

> *"I found [*SHRIMP *based] predictive text input [effective] due to the reduced number of button presses."*
>
> *"T9 + Tilt was pretty handy, if just b/c it's more interactive yet efficient with common words."*
>
> *"I found it convenient to have dictionary based input methods."*
>
> *"SHRIMP is easy to learn, an improvement on T9, and resolve the problem of T9 when facing names or other special words which do not exist in the T9 lib."*
>
> *"It's more efficient to remember the spatial location pattern in T9 and SHRIMP, 'the' is a big left arrow and 'you' is a small up arrow [on the keypad]."*

Users also discovered some usability problems in the current *SHRIMP* prototype. One user complained that sometimes *SHRIMP* was not fast enough to follow her typing. She had to wait about half a second for the screen to refresh while she was typing a long word. Users also suggested that in addition to tactile feedback, some kind of visual feedback should be added when using the motion gesture.

It is worth noting that the text entry speed of DBD reported in our study was slower than studies such as [30]. We suspect that two major reasons caused the difference. First, the subjects in our study might be less experienced in DBD than those in other studies. Second, the experiment settings were also different e.g. in [30], the bigram model used by the predictive input method was trained from 500 sentences and testing sentences were selected from those sentences, so users won't meet OOV words in the study and incorrect predictions for words with encoding collisions would be minimal in such a setting.

3.8 Future Work

Our current study is only an initial step towards having a full understanding of *SHRIMP*. Enhancements such as adding new words to dictionary and richer visual/audio feedback can be made to the current prototype. A longitudinal study can be conducted to figure out the

learning curve and expert performance of *SHRIMP*. A human performance model can be built to estimate its theoretical performance; of course, such a model will depend on a more accurate estimation of the distribution of words with encoding collision and OOV words in mobile environments.

3.9 Conclusion

Dictionary-based disambiguation (DBD) is a popular solution for text entry on mobile phone keypad, but it suffers from two problems: the resolution of collision (two or more words sharing the same key code) and entering out-of-vocabulary (OOV) words. Our analysis shows that both types of problems are quite frequently encountered. *SHRIMP* (**S**mall **H**andheld **R**apid **I**nput with **M**otion and **P**rediction) is a system and method that address these two problems by integrating DBD with camera based motion sensing. It enables the user to express preference through a tilt or move gesture. *SHRIMP* runs on camera phones equipped with a standard 12-key keypad. *SHRIMP* maintains the speed advantage of DBD driven predictive text input while overcoming the collision and OOV problems seamlessly without mode switching. By coupling a motion gesture with the action of typing the first character, *SHRIMP* can reduce encoding collision by more than 50%. By coupling two motion gestures, one with typing the first character and the other with typing the last character, *SHRIMP* can eliminate almost all encoding collisions.

An initial empirical user study showed that users can easily understand and learn *SHRIMP* with less than 10 minutes of practice. The text entry speed of *SHRIMP* (12.1 wpm) was significantly faster than that of MultiTap (7.64 wpm). The study also showed that *SHRIMP* can handle OOV words much faster than a traditional DBD method.

The *SHRIMP* concept is not limited to camera phone based motion sensing - this chapter also contributes to the understanding of text entry based on ambiguous input in general. We unified the representation of action-based disambiguation and dictionary based disambiguation under the regular expression matching framework. The paradigm for *SHRIMP* can be applied to other multi-model input systems such as chording [147], accelerometer based tilting [146] and Nintendo Wiimote to achieve faster speed with a shorter learning curve. A major port of the results reported in this chapter previously appeared in [141].

SHRIMP has been implemented on unmodified, off the shelf Motorola V710 and Motorola RAZR V3 camera phones. *SHRIMP* is open source software released under BSD license. The current implementation can be downloaded from [115]. We hope *SHRIMP* can inspire commercial implementations that change how people enter text on mobile phones in everyday life.

Chapter 4

FingerSense : Augmenting Physical Buttons by Fingertip Identification

"Information at your fingertips."

—Bill Gates, Fall Comdex, 1990

4.1 Motivation

Tapping physical buttons is one of the most frequent tasks in computer-human interaction. In a button-based input device, e.g. the QWERTY keyboard, 1/2/3-button mouse or the telephone keypad, the user's fingers act as triggers for executing commands. Although alternative input modalities such as speech and handwriting are available, button-based interfaces, especially the keyboard, are still the most widely used input device.

The emergence of handheld, cell phone and other forms of mobile computing devices, however, present unique challenges to traditional button interfaces - due to the size of human fingers and the corresponding motor control accuracy, buttons can not be made too small. It becomes increasingly difficult for a full QWERTY keyboard to fit into the ever smaller mobile devices.

In this chapter, we propose an alternative method, *FingerSense*, to improve the expressiveness of pushing buttons without the cost of minimizing the button size or adding additional key strokes[1]. In a *FingerSense* enabled input device, a button will respond differently when it is pressed by different fingers. As illustrated in figure 4.1, when the thumb finger taps the given button, the action can be interpreted as event `A`. If index finger is used, the system will interpret this action as event `B`, similarly the middle finger will correspond to event `C`, etc. As a result, a single pressing action could generate as many events as the number of user's fingers. We define *FingerSense* as the method of multiplexing a physical button according to the actual finger selected in taping, despite the underlining sensing/recognition technology used to distinguish fingers.

[1]Here additional keystrokes also mean pressing multiple buttons at the same time.

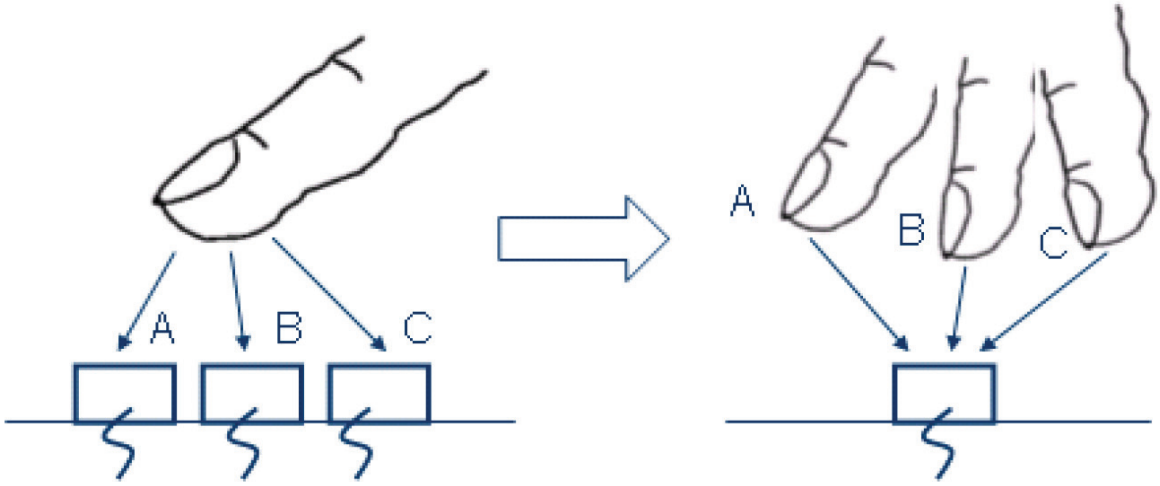

Figure 4.1: From classic buttons to FingerSense button.

To verify the effectiveness of *FingerSense*, we investigate the follow three questions in this chapter:

1. Is *FingerSense* technologically feasible? i.e. is it possible to classify the finger tapping at a button in real time and in a cost effective manner?

2. To use *FingerSense*, the user must select and switch to the correct finger before tapping the intended button; is this procedure a cognitive workload too high to be adopted by most of the users?

3. Is there any speed advantage for the *FingerSense* enabled text input when it is compared with the state-of-the-art?

In the next section, we give a survey of projects and sensing technology related with *FingerSense*, and then we describe the implementation of a computer-vision based prototype, which aims to demonstrate the feasibility of *FingerSense*. In the follow-on section, we present a theoretical model of *FingerSense* and quantitatively calculate the parameters in this model through a preliminary usability study.

4.2 Related Work

The key idea behind *FingerSense* is to detect and use the information implicitly encoded in specific fingers. To acquire and use such "*information at your fingertips*", many potential sensing technologies are possible. In this short review, we focus on projects and methods related with detecting and identifying human fingers and hand gestures. Visual Panel [156] uses a camera to track finger movements and translates the user's virtual typing on a

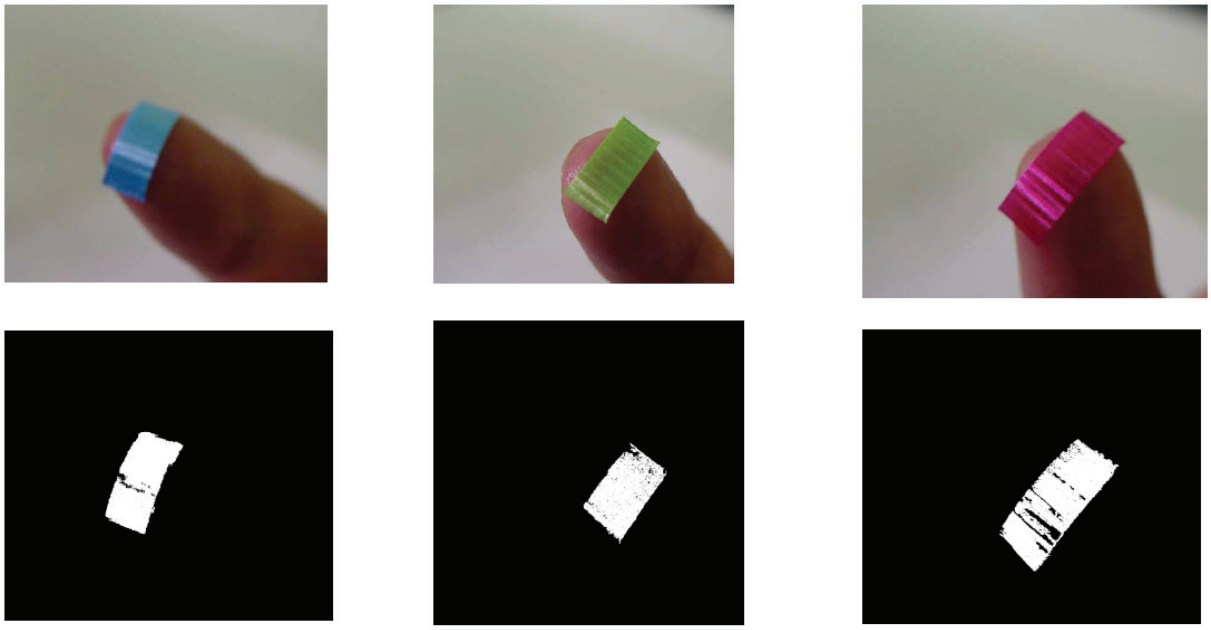

Figure 4.2: Threshold in H-S space to segment predefined color tags. Color tags are (from left to right) C blue (index finger), light green (middle finger), pink (the third finger)..

paper to characters. In this project, only finger movements, not the specific finger involved is detected. Additional techniques can be applied to make the corresponding computer vision algorithms easier C cameras could be mounted under the forearm to simplify the captured images [131]; LED tags could be attached to finger joints and wrists to facilitate the segmentation of finger images[120].

It is possible to identify fingers by capturing and recognizing the associated fingerprints. Sugiura et al [121] uses fingerprint scanner and an out-of-the-box fingerprint recognizer to create a "finger aware" user interfaces. Different fingers are used as shortcuts to launch common applications or invoke commands in that project. It's also possible to attach sensors to fingers to capture and identify finger movements. Actually, most of the methods used in building digital gloves could be applied to identify fingers. See [120] for a comprehensive survey of different sensing technologies behind digital gloves. Sensors which could be attached to fingers or palms include Infra-red sensor, pressure sensor, acceleration sensor[36], or Electromyographic (EMG) sensor [145] etc.

4.3 Prototype

In the initial stage, we build a computer vision based prototype as a proof-of-concept. In this prototype, we attach color tags on different fingertips and use images captured from a CMOS camera to detect colors, so as to identify the corresponding finger.

We used a PrimaScan iCam 320 camera with USB interface to capture still images

at a resolution of $320 * 240$. Five color tags - (pink, blue, green, yellow, purple) were attached to the corresponding fingers on a hand. In Figure 4.2, the first row shows sample images captured by our camera. Thanks to the usage of color tags, the image segmentation algorithm becomes very straightforward, we first convert the captured image from RGB color space to the HSV color space. In addition, we only use the Hue and Saturation information and ignore V(brightness) in order to minimize the influence of shadow and uneven lighting. The second row of figure 4.2 are the image segmentation results after applying global thresholds on H-S space for the 5 preregistered color tags. We can see that after attaching color tags to fingertips, it's computationally efficient to identify the finger used in tapping buttons.

Although the camera-based prototype is relatively large and not comfortable to use due to the additional color tags, it provides a starting point for us to identify potential usability problems related with *FingerSense*. We plan to build the second prototype by mounting *Frustrated Total Internal Reflection* (FTIR) fingerprint sensors on buttons and detect finger used by matching the partial fingerprint collected in the tapping process through a modified minutiae-matching algorithm [60].

4.4 Usability Study

As mentioned in the first section, several concerns exist with *FingerSense*, such as, is it natural for the user to switch and tap fingers to reach enhanced functions via the *FingerSense* interface? How fast can the user performance be? We feel that both theoretical analysis and usability study are necessary to answer these questions.

4.4.1 Modeling of Finger Switching Task

It is evident that hitting a *FingerSense* enabled button is an interaction task composed of a series of sub-tasks, including:

t_1 Withdraw the formerly used finger (prev-finger) from the button face.

t_2 Cognitively determine the intended finger (cur-finger) that maps to the anticipated function.

t_3 Extend the intended finger.

t_4 Visually search the target button from a list of button candidates.

t_5 Move the intended finger from its starting position to the target button and press it.

In the worst case, the total time used to hit one button is:

$$T = t_1 + t_2 + t_3 + t_4 + t_5 \tag{4.1}$$

Note that the time consumed in t_1 and t_3 are not constant - it will depend on the prev-finger and the cur-finger involved and some other existing conditions, which we will discuss in detail later.

t_2 can be improved by practice. The power law of practice [22] models such learning effect.

t_4 corresponds to the choice reaction time of a user. Reaction time can be modeled by Hicks' law [22]. Since the mapping from functions to fingers is fixed in our system, according to [119], the choice reaction time should only be considered for the performance of novice users, for expert users who are familiar with the keyboard layout, time consumed in t_4 can be ignored.

t_5 is also called target acquisition, which can be modeled by Fitts' law [32, 22], note that the index of performance (IP) in Fitts' law equation might not be deemed as a const in our system because the five fingers may have different performance on target acquisition. Essentially, sub-task t_5 is the same as other pointing tasks (e.g. [82]) modeled by Fitts' law.

Note that some sub-tasks are not executed in a uniform and sequential manner. There are at least two variations. First, if the prev-finger and the cur-finger are the same, t_1 and t_3 will take less time than usual since it is not necessary for the user to fully withdraw and spread the same finger in order to hit the target button. In this case we rename the sub-tasks as t_1' and t_3' correspondingly. Second, t_1 and t_3 can be performed in parallel if the prev-finger and the curfinger do not hit the same button. In addition, when t_3 is finished, the finger is ready to perform follow-on sub-tasks even if t1 is not completely finished. A user must perform t_1 and t_3 in a fully sequential manner only if

1. the prefinger and the cur-finger are not the same.

2. the prefinger and the cur-finger hit the same button.

To measure the performance of *FingerSense*, we need to understand the corresponding time consumed in each subtasks. As described above, time for t_2, t_4, t_5 can be modeled and measured with existing knowledge in human-computer interaction. Unfortunately there is no existing method to measure t_1 and t_3 in our system, so estimating t_1 and t_3 is the primary goal of our initial usability study. Using the conditions described above, we can decompose t_1 and t_3. For example, when an expert user is hitting different buttons with different fingers, and if no target acquisition is necessary (i.e. the button to push is right below the curfinger), the task conducted here can be represented as:

$$T_{e1} = t_2 + t_3 \tag{4.2}$$

Note that t_2 here is the cognitive generation time for expert users, an expert user spends less time than novice user due to practice effects. As an example, in random character entry tasks, the probability that we use two different fingers to hit two different buttons on a *FingerSense* enabled telephone keypad can be estimated 2 as $27*(27-3)*(2/3)/(27*27) = 0.59$. So this is the most common "actual" task for expert users. Similarly, the task for expert users to use different fingers to hit the same button can be represented as:

$$T_{e2} = t_1 + t_2 + t_3 \tag{4.3}$$

This event has a probability of $27 * 2/(27 * 27) = 0.07$ to occur on random character input. In addition, the task of expert users to use the same finger to hit the same button can be represented as:

$$T_{e3} = t'_1 + t_2 + t'_3 \tag{4.4}$$

This event has a probability of $27 * 1/(27 * 27) = 0.04$ to occur.

Lastly, the task of expert users to use the same finger to hit different buttons has a probability of $27 * (27 - 3) * (1/3)/(27 * 27) = 0.30$ to occur. Since this task was measured and analyzed by previous research [82], we did not measure the performance of this task in the following study. Based on the three conditions represented in equations 4.2 C 4.4, we designed an experiment to measure the user performance parameters of *FingerSense*.

Three subjects participated in this preliminary usability study - two males and one female with an average age of 26. Two of them are graduate students at UC Berkeley. All of them are right-handed and had former experiences with mobile devices such as cell phone and PDA. A within subject test was conducted. The three conditions are T_{e1}, T_{e2} and T_{e3}. Each subject was presented with all three conditions. The order of the conditions presented was counter balanced in a Latin Square pattern across the three subjects.

4.4.2 Experimental Task

In the experiment, each user was tested under all three conditions in the experiment. For conditions T_{e1} and T_{e2}, we measure the usages of all five fingers as the pre-fingers and the cur-fingers respectively so $5*4 = 20$ potential finger transitions are measured. For condition T_{e3} we measure the performance of the usage of all five fingers, i.e. 5 subconditions. We tested each condition at least 20 times.

4.4.3 Result

The results of conditions T_{e1} and T_{e3} are shown as a bi-tap transition matrix in Table 4.1 below (the unit is millisecond). The diagonal cells represents results related with T_{e3} and all the other cells are results for equations T_{e1}.

Similarly, the bi-tap transition matrix for conditions T_{e2} and T_{e3} are shown in table 4.2 below. Similar as table 4.1, the diagonal cells represents results related with equation 4.4 and all the other cells are results for equation 4.3.

The four findings in the usability study are -

1. The performance of T_1 and T_3 (Table 4.2) depend on the actual prev-finger and cur-finger pair and the performance is asymmetric among any two fingers.

2. In most of the testing cases (i.e. T_{e1} represented by equation 4.2), t_1 and t_3 can be paralleled and finger switching is faster than single finger tapping no matter which

	Thumb	**Index**	**Middle**	**Third**	**Little**
Thumb	172.37	159.22	115.13	113.56	107.50
Index	144.56	130.16	116.38	103.20	126.67
Middle	122.63	115.13	137.53	114.30	159.11
Third	105.00	100.11	146.78	141.79	143.67
Little	132.88	116.75	189.13	206.38	145.47

Table 4.1: Finger switching speed matrix (ms) for condition T_{e1} and T_{e3}. Diagonal cells represent $t'_1 + t_2 + t'_3$, other cells represent $t_2 + t_3$.

	Thumb	**Index**	**Middle**	**Third**	**Little**
Thumb	172.37	275.00	264.78	280.56	281.44
Index	278.30	130.16	234.67	270.22	271.56
Middle	245.40	248.40	137.53	257.11	273.67
Third	286.30	268.60	281.30	141.79	287.11
Little	277.50	309.40	268.40	301.40	145.47

Table 4.2: Finger switching speed matrix (ms) for condition T_{e2} and T_{e3}. Diagonal cells represent $t'_1 + t_2 + t'_3$ and are duplicated from table 1. All other cells represent $t_1 + t_2 + t_3$.

finger is involved. This finding yields the insight that that *FingerSense* systems should be designed to facilitate parallel typing in order to get better performance.

3. If t_1 and t_3 must be carried out in a sequential order, the time for finger switching will be significantly slower than single finger tapping. In this case, some combinations such as the third finger + the little finger, are especially inefficient. The worst finger combination is about 100% slower than single finger tapping.

4. The results of single finger, same button condition T_{e3} ranges from 130ms to 172ms, which accords with the performance 200ms per keystroke of skilled user quite well. (the difference should be considered as t_5 - horizontal movement for target acquisition)

4.5 Conclusion

In this chapter, we proposed a novel technology named *FingerSense* to enhance the expressiveness of physical pushing buttons by fingertip identification. We surveyed potential technologies which can be used by *FingerSense*. We created a computer-vision based prototype which uses color tags to facilitate finger identification as a proof-ofconcept. After a GOMS analysis of *FingerSense*, we derived the related bi-taping matrixes in a preliminary user study. A major port of the results reported in this chapter previously appeared in [134].

Chapter 5

End-User Generated Location-Aware Applications on Mobile Devices

"Space is the opportunity; place is the understood reality."

—Steve Harrison, 1996 [47]

5.1 Motivation

A user's current location plays an important role in her everyday activities. With the popularization of true-GPS enabled cell phones, location aware applications such as Mapping, Point-of-Interests (POI) finding (a.k.a. local search) [4], driving directions [4, 125], location-aware reminders [66, 80], and location-based gaming have become increasingly popular in recent years. However, when compared to the millions of daily tasks that involve participants' locations, researchers have only scratched the surface of what can be done. This situation is due, in part, to the high level of expertise and a significant amount of time required in building a typical location-aware mobile application. As a result, many of the users' diversified and ad-hoc needs, such as virtual garage sale, virtual bulletin board, virtual lost-found, ride-sharing and street parking finding, etc, are unlikely to be addressed in the near future.

At the same time, commercial products (e.g. [4, 125] are unlikely to be a feasible solution in the foreseeable future. Considering the costs of developing, testing and distributing products, companies tend to focus on uniform, well defined products with a large potential user base, and will avoid applications that are can only be used in small communities or applications are highly dynamic in nature. Un-fortunately, a major portion of the everyday location-aware applications fall into the "low commercial value" category in many commercial companies' eyes.

As a step toward addressing this problem, we present a system called *GLAZE* (**G**eneralized **L**ocation **A**ware model**Z** for **E**nd-users) which enables end-users to create location-aware applications on their cell phones (shown in figure 5.1). *GLAZE* also supports sharing of

Figure 5.1: The *GLAZE* client running on a Motorola v325 cell phone. A. main menu. B. top part of the "write-to" primitive definition template.

generated applications with friends and some community members.

5.2 The Design of *GLAZE*

5.2.1 The REWIND Model

Our hypothesis is that a large set of everyday location-aware applications can be developed by providing a set of three simple primitives encapsulated in what we call the REWIND model: READ, WRITE and FIND. The use of these three primitives is easy enough for end-users to use but expressive enough to implement a significant set of interesting location-aware applications.

In the REWIND model, location-aware applications work by exchanging and filtering messages (plain text, picture and special tags) coupled with multiple constraints on people, time and location. Messages can be exchanged with other people, places, or virtual tags with additional constraints.

The READ and WRITE primitives allow reading and writing messages from and to given sources and destinations (people, locations, virtual tags etc). A continuous version of the read primitive named "listen-to" and a continuous version of the write primitive named "write-to" provide sup-port for "push model" tasks such as tracking and monitoring.

The FIND primitive retrieves messages or virtual tags by providing constraints on people,

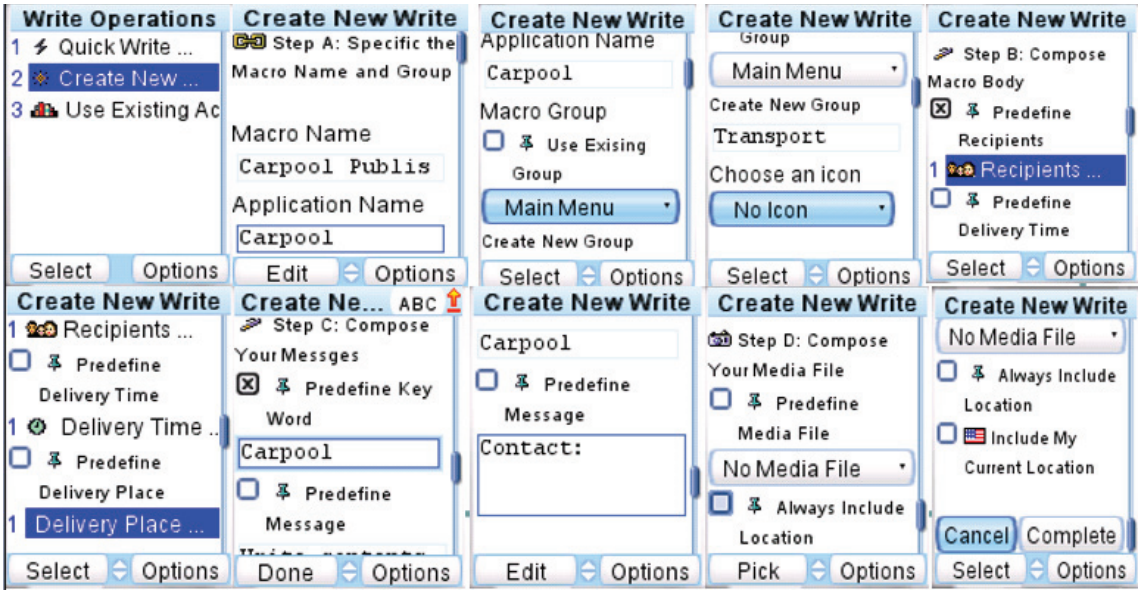

Figure 5.2: Brief steps for an end user to generate a "ride-sharing" application called "Carpool" in *GLAZE*.

time and location. Map-ping and driving directions functions are supported in *GLAZE* via the Find primitive.

In addition to using longitude and latitude information directly, *GLAZE* also supports using the notation "place" to refer to locations. *GLAZE* supports defining places by re-verse geo-coding and collecting place-annotation interactively from end-users.

As an example, A Location-aware reminder application [80] can be created by first defining a write operation to the desired place, then defining a continuous read operation ("listen-to") to read any text messages that are within a given radius from the receiver. Figure 5.2 shows another example for creating a ride-sharing application named - "Carpool" by using the REWIND model. Applications that the current REWIND model does not create fall into the following two categories -

- Leveraging information or resources (e.g. UPC bar-code, voices, network packets, tour guide video clips) that are currently not available in the implementation of *GLAZE*
- Using complex interaction logic (e.g. in-game AI) or using complex graphics/UI effects as a key function. The REWIND model is not intended to be a silver bullet to all the possible location-aware applications. Instead, it is designed to significantly lower the barrier to creating location-aware application , allowing average users to generate useful location aware applications.

Please note that the REWIND model is not intended to be a silver bullet to all the possible location-aware applications. It is designed to significantly lower the floor of loca-

tion aware application creation, allowing average users to generate useful location aware applications.

5.2.2 Community Exchange

In addition to creating a location-aware application by the user herself, she can also browse and download location-aware applications that are created and shared by other users via the *GLAZE* cell phone client. Considering that a major portion of everyday location-aware applications are group based, downloading and using the applications created by other people will let a user not only take advantage of the *GLAZE* system, but also contribute to the corresponding application community at the same time.

To encourage expertise sharing and group interactions in a small community, *GLAZE* provides two types of incentive mechanisms to encourage users to create and share location-aware applications that are might be useful to other people. According to the current incentive mechanism, a user receives "Activity Tokens" (AT) when she shares her con-tents/applications. A user will also receive "Status Tokens" (ST) when other users use the contributed con-tents/applications from this user. The amount of AT and ST that a user earn are shown in the user's public profile and on the ranking lists maintained in *GLAZE*. AT and ST earned can be used to unlock "bonus functions" (e.g. camera phone based motion sensing games) in *GLAZE*, but can not be transferred.

GLAZE is an end-user programming (EUP) system running on mobile devices for location based services. However, the design philosophy of *GLAZE* differs from traditional end-user programming systems such as EAGER [25] and ToonTalk [67] in two major aspects. First, *GLAZE* supports creating "group applications" while most of the applications generated by traditional EUP systems are essentially single user applications. Second, by taking advantage of the communication channels in cell phones, *GLAZE* supports and encourages the sharing of user generated artifacts with others. A typical *GLAZE* user does not have to create her own applications in order to take advantage of the *GLAZE* system and contribute to *GLAZE*. *GLAZE* is not a Programming By Demonstration (PBD) system [77], a popular solution to enable end-users to automate some repetitive tasks; in contrast, *GLAZE* leverages wizard/template style form-filling interfaces to generate mini LBS applications.

5.2.3 System Architecture

The *GLAZE* system is composed of two parts - the *GLAZE* server and the *GLAZE* client. Figure 5.3 is a high level diagram of the *GLAZE* system.

The first iteration of *GLAZE* server was written in C#, running on an ASP.net application server for Windows. In the recent iterations, it is written in Ruby on top of the Ruby on Rails [110] framework. All the user profiles/logs/generated contents are transimited in JSON format [65]. Most of the Map and POI information are from Microsoft MapPoint via a COM interface and via Google Map API. Additional Map and POI data come from other publicly available online data sources such as Tiger/Line. The *GLAZE* server communicates with the *GLAZE* client via the HTTP protocol.

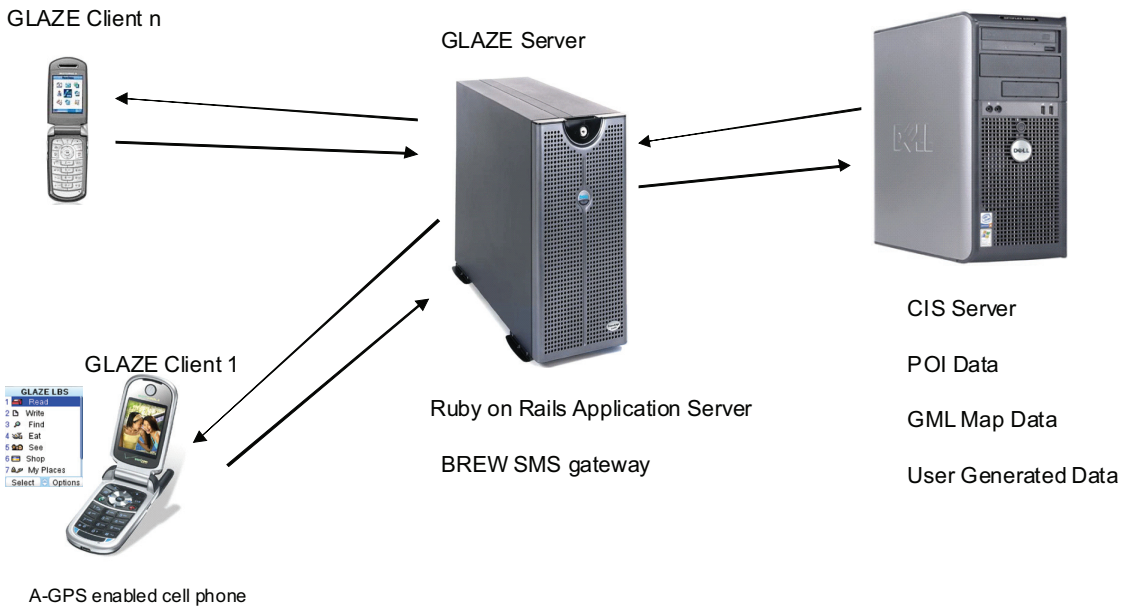

Figure 5.3: The Architecture Diagram of *GLAZE*.

The *GLAZE* client is written in Adobe Flash Lite 2.0 and C++ for BREW [13]. Some screen shots of the *GLAZE* client are shown in figure 5.4. The *GLAZE* client runs on Motorola v325 and v710 cell phones. These two models are entry level CDMA phones from Verizon Wireless. The built-in gpsOne Assisted-GPS [41] in these handsets support two critical location sensing features that are not available in regular GPS [80] or GSM cell tower based solutions [66, 74] -

1. No time-to-first-fix delay.

2. Works both indoors, outdoors and in "urban canyons".

These two features make the location-aware interaction scenarios highly practical and usable.

The *GLAZE* client address the common "battery drain" weakness in most GPS cell phone based tracking and notification solutions by leveraging a technique named BREW directed SMS (BDSMS) [6]. BDSMS is an official API supported by all BREW enabled CDMA phones for launching specific applications on a cell phones via an SMS message with special encoding. With BDSMS, the *GLAZE* client does not need to keep running in the background to be ready for receiving server initiated notifications or sending updates to the server. As a result, the client application no longer interferes with the power-management functions of the cell phone, the battery life of cell phones is no longer significantly reduced a running background application. The battery of Motorola v310 phones can last more than

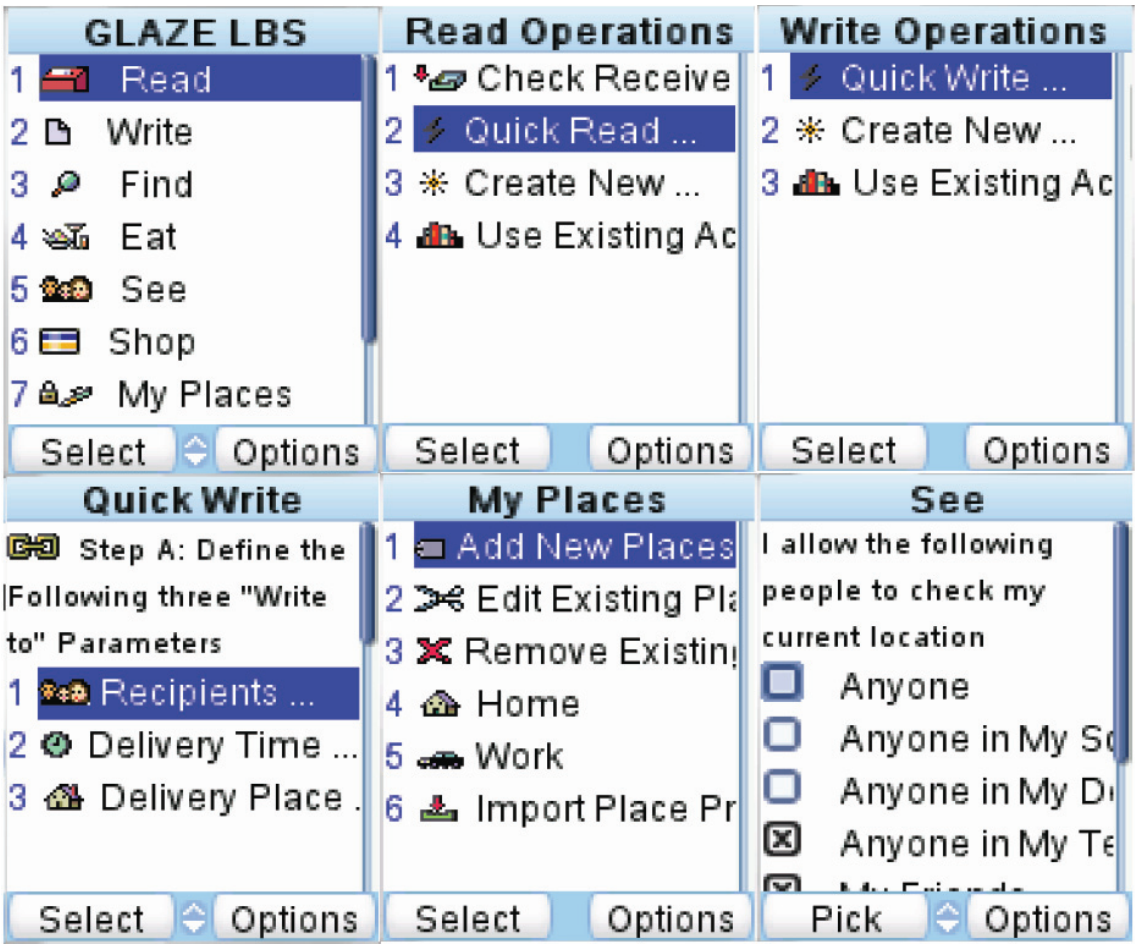

Figure 5.4: Screen shots of the current *GLAZE* client.

three days with normal use for BDSMS based *GLAZE* client implementation, in comparison, the same battery can only last nine hours when the *GLAZE* client is implemented as an "always alive" background application. Another advantage of the BDSMS based implementation is that inquiries and notifications to the cell phone are guaranteed to reach the cell phone later even if the user shut down her cell phone temporarily or the cell phone is not in the wireless carrier's service region at the time of notification.

5.2.4 Privacy Issues

Privacy issues are extremely important in almost any con-text-aware systems. Due to the scope of this research, in the current *GLAZE* system we implemented a role based access control list (RBACL) for controlling the disclosure of location related information. This meets or exceeds the privacy protection in place in current commercial systems. However, several researchers (e.g. [124]) have proposed more sophisticated means for preserving personal or group privacy in ubiquitous computing. There is no innate architectural limitation that prevents *GLAZE* from adopting other more advanced privacy preserving techniques in the future.

5.3 Discussions

Toolkits for location-based services such as Location Stack [52] and PlaceLab [74] provide LBS libraries and frameworks to programmers. It still requires a high level of programming expertise and considerable time to use them. Lash-Ups [11] provides light-weight APIs for grass-root programmers to retrieve existing information on the web or distribute web applets based on locations. Instead, *GLAZE* is a system for end-users with little or no programming knowledge. Topiary [75] is an informal prototyping tool for location enhanced applications. Topiary provides Wizard-of-Oz support for interaction designers to walk-though the interaction scenarios of their intended applications in the early stage of design. BrickRoad [79] is another informal prototyping tool aims at an even earlier design stage than Topiary, no only senor output and UI, but also the interaction logic can be simulated by using WOz technique by designers. iCAP [118] is another informal prototyping tool for designers to mock-up a context aware system via an actor and sensor centered approach. *GLAZE*, on the other hand, is used by end-users and *GLAZE* generated applications are fully functional and can be directly used in daily life.

GLAZE is a system to support the "long tail" of location-aware applications as GPS enabled cell phones are becoming pervasive in daily lives. Different from existing LBS applications in this area, *GLAZE* can be considered as a "meta" LBS application, enables average end-users to build and share the location-aware applications they need. Based on our experiences in the design and implementation of *GLAZE*, there is an interesting trade-off between ease of use and the flexibility of an EUP system. In *GLAZE*, we intentionally took a different direction from previous prototyping tools and designed a system that can be used directly by average users to directly build usable location-aware applications on their cell phones. To our knowledge, *GLAZE* is the first system that enables the creation

of location-aware applications directly from cell phones. Due to the same trade-off, *GLAZE* is not-capable of creating applications that require highly customizable "look-and-feel" or applications that requires complex interaction or data processing logic.

GLAZE can also be used as an unobtrusive data collection platform for ubicomp projects that need to collect data via the experience sampling technique. We believe it will be interesting to observe and analyze the social dynamics in a small community via system logs and user generated artifacts. Particularly, it would be interesting in knowing how the sharing of location-aware applications compares to the sharing of user created content on the web.

5.4 Case Study : The End-User Place Annotation Component

Due to the complexity of *GLAZE*, the systematic, longitudinal deployment and evaluation of *GLAZE* is an still an ongoing work and is beyond the scope of this chapter. In this section, we present one case study - i.e. the design, implementation and informal evaluation of the place annotation component in *GLAZE*. Instead of putting some edit boxes directly on the screens of cell phones, we show that more efficient, easier to use place annexation interfaces can be designed by leveraging the unique sensors on cell phones, such as the camera (again), the microphone and the GPS module.

In almost every location-aware application, there is one critical conceptual mismatch [50] that needs to be addressed regarding the access and reference of locations - i.e. while computers work with physical locations like latitude and longitude, people usually think and speak in terms of places, like "my home" or "Tom's office" which adds personal, environmental and social meaning to a location [39, 50]. Therefore, most location-aware applications will encounter the problem of translating raw location data (longitude, latitude, cell tower information, access point information, etc.) to more symbolic, personally meaningful places. Most of the existing LBS toolkits such as LocationStack [52] and PlaceLab [74] acknowledged such a conceptual mismatch between location and place and provided abstractions and models for mapping locations to places, however, none of them provided effective and scalable techniques for collecting the data to address this mapping problem. As such, most of the current LBS prototypes [66, 88] still require system developers to collect and load place names for users manually.

It's possible to discover personal meaningful places from users' history location traces automatically [3, 76] by using spatial and/or temporal data clustering algorithms. It's also possible to identify important place in everyday life semi-automatically by a set of heuristics such as GPS signal availability [88]. However, these methods still can not eliminate human intervention completely: First, automatic/semiautomatic methods will never become 100% accurate; Second, most of the automatic algorithms nowadays are focusing on discovering the positions of places; it is still a humans responsibility to annotate the names of places.

Due to the above reasons, we feel it is compelling to identify intuitive and non-intrusive place annotation methods to collect place annotations from end-users. In the rest of this

paper, we propose four prototypes that can be used to collect place annotations interactively from mobile devices, and describe the results of an experiment which aims to obtain preliminary understandings of the following two questions.

- How to effectively collect place labeling information from grassroots contributors?
- Whether and what users are willing to disclose regarding their personal place profiles?

5.4.1 Place Annotation - The End-User Approach

In this section, we focus on end-user manual annotation methods for two reasons: First, pure pre-loading approaches used in current prototypes do not scale well with more users. Second, even for semi-automatic or automatic methods, they still need end-users to enter feedback and corrections, input the label name and additional properties etc, so manual annotation is still required to complement automated methods.

We designed and implemented four techniques for end-users to make place annotations on mobile devices. We explored different modalities and input capabilities available on cell phones in these prototypes.

Direct Text Annotation The first end-user place annotation interface we created is "*Direct Text Annotation*" (a.k.a. *DText*, figure 5.5). *DText* is a relatively straightforward method C after pressing a hot key, the primary interface of *DText* will pop up, the user can give a name to her current location, specify a type of the location (There are 7 predefined location types in the current version, i.e. unlabeled, home, work, shopping, dining, leisure and others), define the valid range of the current annotation (current configuration includes: 20 feet, 50 feet, 100 feet and custom), and choose the access control property for the current annotation (existing options are self, family members, friends and anybody). The *DText* method is not just a standalone annotation technique. It was also used by the other three techniques to prompt the user for the name and other properties of the current place at appropriate moments.

Map Selection and Text Annotation The second place annotation technique is "*Map selection and Text annotation*" (a.k.a *Map*, figure 5.6). Unlike *DText*, this interface is designed to allow users to define places other than their current location. After pressing the hot key, a map corresponding to the user's current location will be downloaded to the cell phone. The users can pan, zoom-in and zoom-out the current map. He can also move the hot-spot cursor on the map by using the built-in cursor keys and context menu keys of the cell phone. After pressing the ok button, the *DText* interface described in technique 1 will pop up to collect the place information for the location under the hot-spot cursor. We propose this technique to give users the flexibility to annotate places other than their current location through selecting places from a graphical map display.

Photo Memo plus Offline Text Annotation The third technique is "*Photo Memo plus Offline Text Annotation*" (a.k.a. *Photo*, figure 5.7). After pressing a hot key, the user

Figure 5.5: Cell phone Interfaces for *DText*. (number order is left to right, top to bottom) figure 5.5.1 represents the upper part of the primary menu. 5.5.2 represents the middle part. 5.5.3 is the place type collection sub-menu. 5.5.4 is the place range collection sub-menu. 5.5.5 is access control sub-menu. 5.5.6 is the bottom part of the primary menu.

Figure 5.6: Cell phone interfaces for *Map*. After pressing the ok button, the *DText* interface will pop up for place annotation collection.

Figure 5.7: Cell phone interfaces for *Photo*. (number order - left to right) 5.7.1 is the initial interface after activation. 5.7.2 is the photo capture interface. 5.7.3 is the photo history. After pressing the ok button in 5.7.3, the *DText* interface will pop up.

Figure 5.8: Cell phone interfaces for *Voice*. 5.8.1 Initial interface after activation. 5.8.2 voice capture interface, 5.8.3 voice history. After pressing the ok button in 5.8.3, the *DText* interface will pop up.

can take a picture using her camera phone to create a reminder of such a place that she is visiting, after which, when it is convenient for the user, such as when she is waiting for a bus at the bus stop or waiting for food in a restaurant, she could activate the label picture function and annotate places she visited based on the photo memos she had taken. After annotation, only the properties appeared in the *DText* interface will be transmitted to the server, the pictures taken will only stay in the user's local cell phone as a reminder and will not be shared by other users. This design is motivated by our observation that when people are in personally significant places, it may not be the ideal time for them to spend significant amount of time to annotate the place immediately even though the place is meaningful to them.

Voice Memo plus Offline Text Annotation The fourth and the last technique we created is "*Voice Memo plus Offline Text Annotation*" (a.k.a. *Voice*, figure 5.8). The basic idea is similar to the photo memo method, except that instead of taking a picture as the reference to the interested place, the user records a short voice memo to describe the place she is interested in. When it is convenient for her subsequently, she can then retrieve her previous voice memos and label them the associated places after hearing the content of the voice memos. Similar to the photo memo method, voices recorded are only used as reminders. They are not shared with other people as part of the place annotations.

5.4.2 Usability Test

We conducted a trial usability study to investigate the performance of the four place annotation techniques described above.

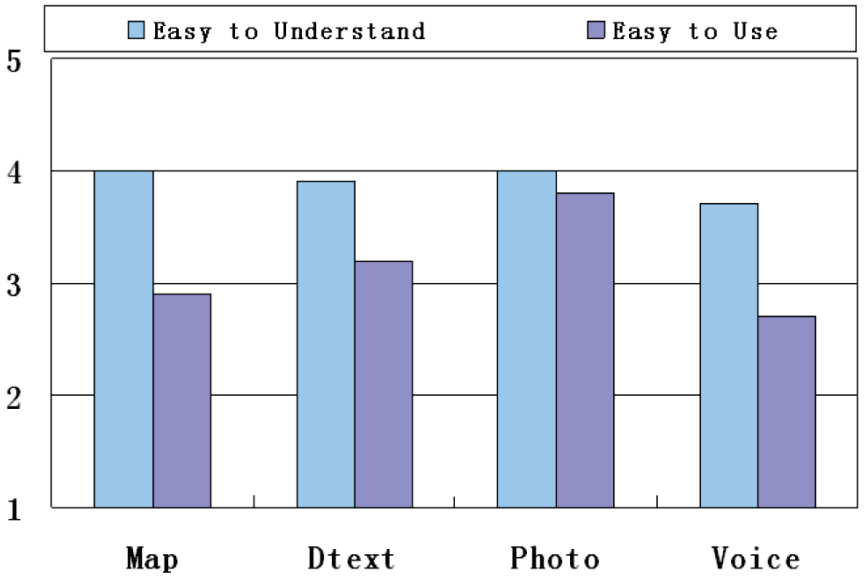

Figure 5.9: End user ratings on the four annotation techniques. Ratings are on a 5 point Likert scale (1 means worst, 5 means best and 3 means neutral).

Experimental Design

Six subjects participated in this preliminary usability study - three males and three females with an average age of 26.7. Five of them are undergraduate and graduate students at a local university. The user study consists of three parts. First, subjects complete an opening questionnaire, which include a background survey, a task asking the user to identify the important places he/she had visited in the past one week and assign a name to each of the place identified; and a form asking the user whether they are willing to share the location to place information that they provide during the study with their family members, their friends, their classmates/colleagues or just anybody. Second, the subject is asked to use each of the four prototype interfaces to annotate a place that she visited in the past two hours and the restaurant where she had lunch (or planned to) on the day of the study. The order of interfaces tested in this part was counterbalanced by the Latin-Square pattern.

Third, the subject is required to complete a closing questionnaire. Each subject gives each prototype interfaces tested a score on the whether the user interface is easy to understood and a score on the ease of use of the interface. The subject is also given blank spaces to leave her own usability feedbacks and suggestions in this section. Opening Questionnaire Results In the place identification task, all the six subjects identified 74 personally meaningful places totally (12.3 places on average. min = 6, max = 21). The average length of place names used by subjects is 1.8 words (min=1, max=3) or 7.1 English characters (min=3, max=15).

Top referred places from the opening questionnaire include - home, office, restaurant, shopping mall, bus stop, ATM etc. One interesting finding is that about 78% (58 out of

74) of the places are indoor places. This finding not only provide supports to the lost signal heuristic used in the ComMotion [88] project, but also suggested us that effective place annotation techniques shouldn't use the assumption that "live" GPS fix signals are available at the time of activation. Instead, the application should try to use either historical data or switch to different positioning techniques (e.g. AGPS [41], cell tower ID) when direct GPS signals are not available. All of the subjects indicate that they won't mind to share all of their place profiles with friends and family members. Four of them even indicated that they won't mind to share their location profiles (the customized mappings from locations to places) with anybody. These numbers are interesting because they appeared to differ from people's opinion on disclosing their current location [24]. There are two possible explanations to this phenomenon: First, people may think information related with their personal places, such as their home address are no longer a secret already. Indeed, considering that most people won't pay the telephone company to hide their contact methods from phone books and there are so many companies that collect and sell personal contact information, such reaction looks reasonable. Second, people may simply not define places which they don't want to disclose via the annotation application. Performance of Four Prototypes All the six subjects completed the assigned tasks successfully. Their rating of the four method are summarized in figure 5.9. From figure 5.9, we can see that users believe all the four designs are very easy to understand. (Average score =3.9, min=3.7, max = 4.0). The users also have mixed feeling on the four techniques according to the criteria of "ease of use". The *Map* selection method received a relative low score (2.9) according to this standard. We assume that the low score on the *Map* method is caused by the relative clumsy interaction methods to navigate on a map without proper pointing devices. It may also be caused by the fact that defining a place other than the user's current location might not be a very frequent task. Last, 78% of the significant places are indoors, which can not be located effectively from standard maps. The *Voice* annotation method received the lowest score (2.7) score on ease of use, among the four techniques. Most subjects dislike this method for two reasons C recording a voice memo for a place might look silly in many scenarios. What is more, selecting a desired voice memo later is often difficult since the users must listen to the voice memos one by one.

The *Photo* method received a significantly higher ease of user score among the four. Subjects thinks the idea of using photos as reminders and annotate the place name later is a neat idea, considering the relative slow text entry speed on cell phones.

Closing Questionnaire Results

Many usability issues have been identified from the usability study and the closing questionnaire. For example, some users feel that a confirmation screen is necessary when a place has been annotated successfully. More mode switching supports in the place label editing box are desired according to the users' feedback. E.g. one subject got stuck for more than 5 minutes in the usability test when he switched to the T9 input method by mistake and couldn't figure out how to switch back to the default MultiTap input method.

Although users gave the photo memo plus offline text annotation method the highest score in ease of use, they also indicated their desire to have more than one annotation

method in the final application. Thus they can choose the one that best fits their need according to the given context.

5.5 Conclusion

We have presented *GLAZE*, a system which enables average users to create and share everyday location-aware applications directly on their cell phones. The *GLAZE* system provides a set of location-aware primitives named REWIND (READ, WRITE and FIND) to help end-users model and generate their intended applications though the help of a set of form-style smart templates. The *GLAZE* system is designed to lower the threshold of location-aware application creation and encourage both expertise sharing and group interactions in a community.

As a case study, we presented the design and implementation of four interfaces for collecting end-user place annotations interactively on cell phones. In addition to the baseline interface that relays on entering texts direction in edit boxes, most other interfaces leveraged the unique sensors (such as the camera, the microphone and the GPS) and provided asynchrony functionalities to support offline annotation. A trial user study is suggest that while all the four methods got similar preference ratings in understandability, the "*photo memo plus offline editing*" method is the most favorite approach in ease of use. In addition, users indicate their desire of adopting more than one place annotation methods in location aware applications. Part of the results reported in this chapter previously appeared in [135].

Chapter 6

Event Maps : A Collaborative Calendaring System for Navigating Large-Scale Events

"minimizing computer admin debris; spatial distribution of information rather than temporal stacking."

—Edward Tufte, 2008 [129]

6.1 Motivation

Attending conferences and trade shows is important to our professional and social life [108]. Large conferences often have many concurrent sessions, and it may be challenging to get a good sense of all available activities and plan one's attendance. Having a guiding map can help tremendously.

Online schedules on traditional conference web sites [23] are built as a static collection of hyperlinked pages, where users may have to click multiple times and go back and forth to locate the information they need. Such a design makes it difficult to simultaneously explore both depth and breadth, and the user loses context rapidly. It is also hard to create tailored views of the schedule according to your preference.

Exchanging information with other conference participants, a key objective while attending conferences, is also not supported by this design. Although many conferences do provide complementary blog/wiki systems, they are generic to the conference and not tightly coupled to specific sessions. Why can't comments of senior researchers on an award winning paper be made available to new students?

Finally, according to our informal survey, creating the schedule on a conference web site and keeping it up-to-date is a challenging, tedious and error-prone task for conference organizers with present day systems.

Event Maps is a rich internet application we have built to address these shortcomings

in navigation, collaboration, personalization, and organization. *Event Maps* aims to make the experience of interacting with an online conference calendar engaging, productive and intuitive. It supports seamless navigation between broad and deep views of the schedule information, asynchronous interaction in the context of individual sessions, and the ability to tailor preferences and favorites. As such, it suggestively sets up the conference calendar as a focal point for interaction and data mining for participants, organizers, and any other interested parties, which has value before, during and after the conference is held.

The main research contributions of the work described in this chapter are in devising and exploring the use of:

Tabular Timeline A zoomable calendar interface for large event navigation to minimize page switching.

Active Corners Compact decoration widgets for awareness and activation of features.

We provide tangible evidence that, aided by these mechanisms, *Event Maps* indeed supports task efficiency while being enjoyable to use.

Event Maps also makes it easier for conference organizers and administrators to maintain an updated conference schedule. The system also provides analytic administrative tools that help get insights from visiting patterns and statistics.

We have been focusing our attention on the applicability of *Event Maps* to multi-track conferences. However, its design makes it more broadly suited for handling rich schedule-related data such as group calendars and personal calendars.

The rest of the chapter is organized as follows. After discussing related work, we describe the *Event Maps* system, its evolution and implementation. We then describe the user studies and deployments to date, followed by a discussion of lessons learned, future work, and conclusion.

6.2 Related Work

Research in group/collaborative calendaring systems has explored topics such as scheduling a group activity based on the availability of each team member, ambiguous time constraints [15] or special audiences such as family members [95]. However, less research has been devoted to addressing the needs of conference participants.

The idea of using a zoomable interface (ZUI) to view calendars is not new. In 1991, Furnas [38] proposed a textual Lisp-based calendar program with fisheye distortion. Bederson et al. [8] later extended this idea and built a scalable visual fisheye calendar named DateLens. *Event Maps* differs from DateLens in several ways: First, DateLens focuses on personal events while *Event Maps* is targeted at *multi-track conference events* with selective expansion of more than one track. Second, DateLens is a single user application while *Event Maps* supports asynchronous collaboration in the context of conference sessions. Finally, as DateLens is not a web-based application, it lacks facilities for user tracking and data analysis to help administrators and system developers.

Where	**When**
Google/Yahoo Maps	***Event Maps***
Who	**What**
Social Networks, Organizational Charts, Directory Services	Wiki, Web Widgets, Blog

Table 6.1: The Design Space of Web 2.0 Services

Event Maps deals with temporal data [64]. Visualization systems for temporal data have been available for decades, supporting activities such as abnormality detection, record management [1] and decision making. LifeLines [100] presents a timeline style interface to manage personal medical record and supports zooming, panning and text filtering. TimeSearcher [54] lets users select stock price patterns of interest via a direct manipulation interface named TimeBox that filters through thousands of temporal records of numerical data.

Several pieces of work address the scalability problems encountered when visualizing temporal data [1, 14, 127, 54, 143]. In addition to data level transformations [69], most of the solutions are domain specific and fall into the following two categories:

1. Using a zoomable interface to highlight the region of interests when necessary [14, 127].

2. Designing a query interface, either by setting up parameters via on-screen data widgets such as sliderbars, or by using techniques such as direct manipulation or query by example to refine targets of interests [1, 55, 95, 143].

Recently, we have seen the emergence of collaborative visualization systems like Many Eyes [132], sense.us [49], and Swivel [122], which support creating visualizations from statistical data, and also provide text comments, tagging and view-sharing through book-marking. "*By partitioning work across both time and space, collaborative visualization offers greater scalabilitygreater scalability for group-oriented analysis*" [48]. Our work extends some of these principles to the arena of multi-track conference schedules.

6.3 The *Event Maps* System

A collaborative calendaring system optimized for large, multi-track conferences. *Event Maps* focuses on organizing temporal information via a web based interface (Table 6.1). The major features of the *Event Maps* system include:

1. A zoomable interface that allows quick transition between views at different levels of detail, and simultaneous viewing of multiple levels of detail.

2. In-place support for asynchronous collaboration features such as highlighting, and commenting directly in the context of conference sessions.

3. Functions for conference organizers to create and import conference schedules, keep them updated, and obtain insights from visiting patterns and statistics.

Our greatest challenge in designing *Event Maps* has been to support the rich feature-set in a clean, intuitive and efficient interface. Quick event navigation and location is supported via a zoomable interface coupled with features such as progressive searching, brushing [86], and details on demand [20]. Contextual personalization and collaboration are supported via compact decoration widgets. *Event Maps* encourages spontaneous interface exploration by mapping frequent operations to mouse hovering clicking.

6.3.1 User Interface

The primary interface for *Event Maps* is shown in Figure 6.1. In the center region is a zoomable widget named *Tabular Timeline* to display all the activities in a conference. The top panel provides a button to set the *Tabular Timeline* to the original state, a search box (Figure 6.1.f), widgets for selecting the current view and time zone, and color-coded session-category labels. The bottom panel is an area for managing the user's "favorite" sessions; sessions with time conflicts will be marked in red in the "My Favorites" region. The rectangles inside the *Tabular Timeline* are conference sessions, color-coded by their associated categories. The top-left and top-right corners of each session are called *Tabular Timeline*, and are used to support personalization and asynchronous.

6.3.2 *Tabular Timeline*

Most *Event Maps* interactions happen inside the *Tabular Timeline*. The initial view of *Tabular Timeline* is an overview of the entire conference, with each of the tracks [1] collapsed. Our *Tabular Timeline* widget is a combination of traditional calendar visualization and a zoomable timeline. *Tabular Timeline* differs from existing zoomable timeline visualizations such as [14, 127, 101] in that it uses a tabular layout to maintain the "look-and-feel" of a calendar rather a scatter plot. At the same time, regardless of the zooming level and the state of the system, time information is always represented in the horizontal axis in a timeline fashion. This behavior differs from existing calendar interfaces such as MS Outlook, Google Calendar or DateLens [8], where the time representation usually jumps between axes depending on the viewing mode (day, week, month etc.). Both informal feedback and our user study show that maintaining a consistent timeline in a zoomable interface helps maintain a consistent user experience.

Along the vertical axis of *Tabular Timeline* are tracks which can be independently expanded and collapsed via mouse clicks (similar to the design of nodes in DOI trees [21]). The colored rectangles within tracks correspond to sessions. When a track is collapsed, concurrent sessions are collapsed into a single rectangle. Clicking on a session in an expanded track will bring up its details pop-up (Figure 6.2). Clicking on the "Related Information"

[1] The definition of track depends on the conference: it can either be a theme including sessions happening in multiple locations at the same time (e.g. Lotusphere 2009), or sessions in a single location (e.g. CHI 2009 [23]). *Event Maps* supports both definitions.

Figure 6.1: The *Event Maps* System. (a) Map area in global overview state; each colored block is a conference activity; hovering reveals details of the activity; clicking on it zooms in (both in X and in Y) to show more detail. (b) Region for saving a user's favorite sessions, highlighted scheduling conflicts. (c) The X-axis is a zoomable timeline; first click zooms to expand the corresponding day, second click zooms back out to the original view. (d) Color-coded session category labels; hovering will highlight (via embossing) all the sessions in the corresponding category. (e) Reset view button, for restoring the overview state. (f) The search box; search results, updated on each keystroke, show up on a panel on the left, and corresponding sessions are highlighted on the map. (g) Collaboration corner widget; hovering displays comments associated with the corresponding session, click pops up a commenting/annotating widget. (h) Switches for controlling the current view mode and displayed time zone.

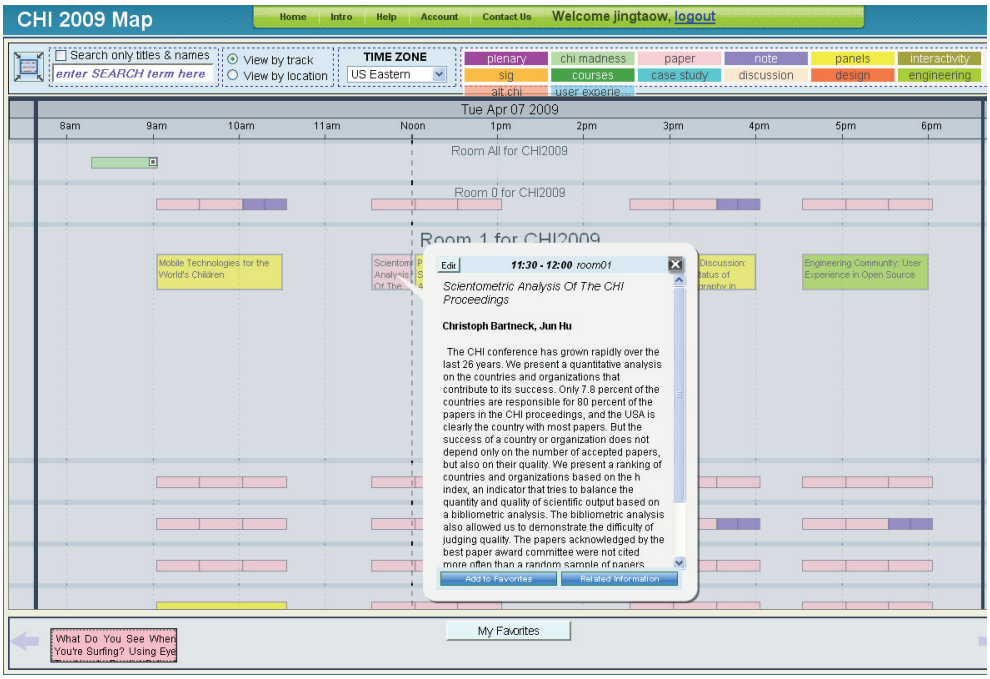

Figure 6.2: The *Tabular Timeline* zoomed into the day view, with one track expanded and the details pop-up open.

button on the details pop-up creates a large frame populated with an external web page pertinent to the session. This frame can be closed to return to the previous visual context, or detached to a separate browser tab.

In addition to discrete zooming (by clicking on different regions of the view), the *Tabular Timeline* supports continuous zoom and pan via mouse dragging. Hovering over a category label on the top right region highlights corresponding sessions on the *Tabular Timeline*. With this quick action the user can see how sessions of a certain type are distributed throughout the conference.

Regardless of the current zoom level, clicking the "reset view" button (Figure 6.1.e) on the top left will reset the *Tabular Timeline* to its original global overview state.

6.3.3 Embedded Search

When the user types into the search box, search results, updated on each keystroke, show up on a panel on the left, and corresponding sessions are highlighted on the map (Figure 6.3). For example, typing in "Washington" and restricting the search to titles and names will quickly show the sessions for papers authored by individuals from the University of Washington. Clicking on a search result shows/hides its session details popup in place on the map.

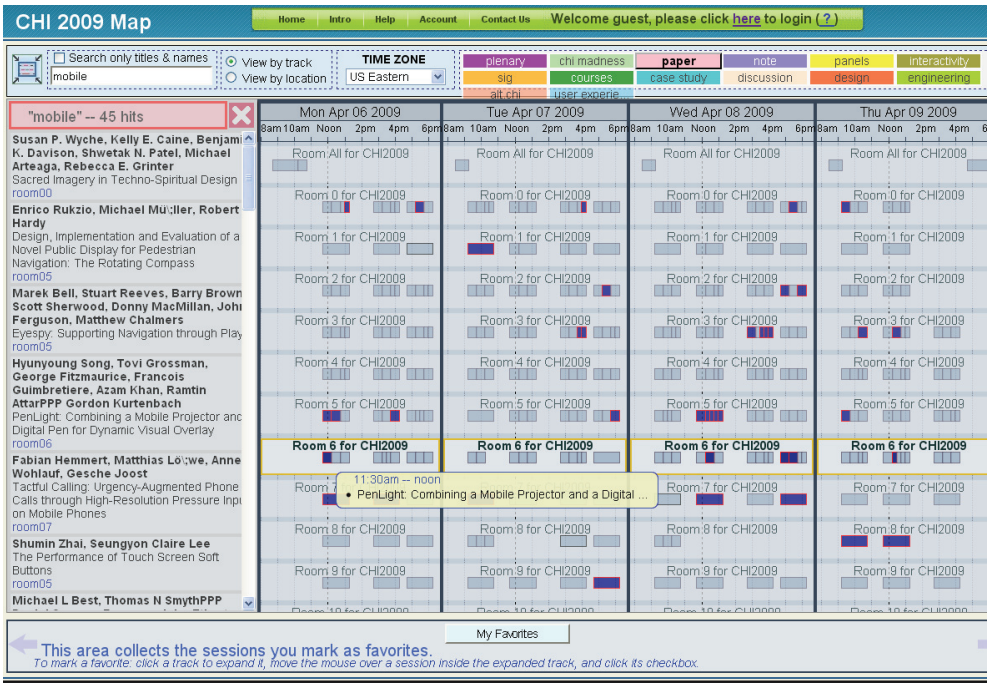

Figure 6.3: The search feature of *Event Maps*.

6.3.4 *Active Corners*

Rather than allocating designated regions to support personalization and asynchronous collaboration, we implanted a compact decorator widget named *Tabular Timeline*. *Tabular Timeline* serve both as an awareness indicator and a trigger for supported features. In the current implementation, the top-left corner of an event rectangle is used as the "personalization corner" and the top-right corner is used as the "*collaboration corner*". In the future, the two corners on the bottom of an event rectangle can also be mapped, e.g., for showing visiting statistics. Each *Active Corner* can have three states -

1. The default state is the awareness state, shown as a small icon at the corresponding corner to indicate Boolean information such as whether this session is marked as a favorite by the current user, or whether there are comments associated with the session.

2. If a session is expanded, hovering over an *Active Corner* can provide a quick preview of information associated with that corner.

3. If a session is expanded, clicking on the corresponding *Active Corner* can trigger an action, such as modifying the favorite state of the session or popping up a widget for browsing and adding comments on the session. We describe in detail the two *Tabular Timeline* currently in use in *Event Maps* below.

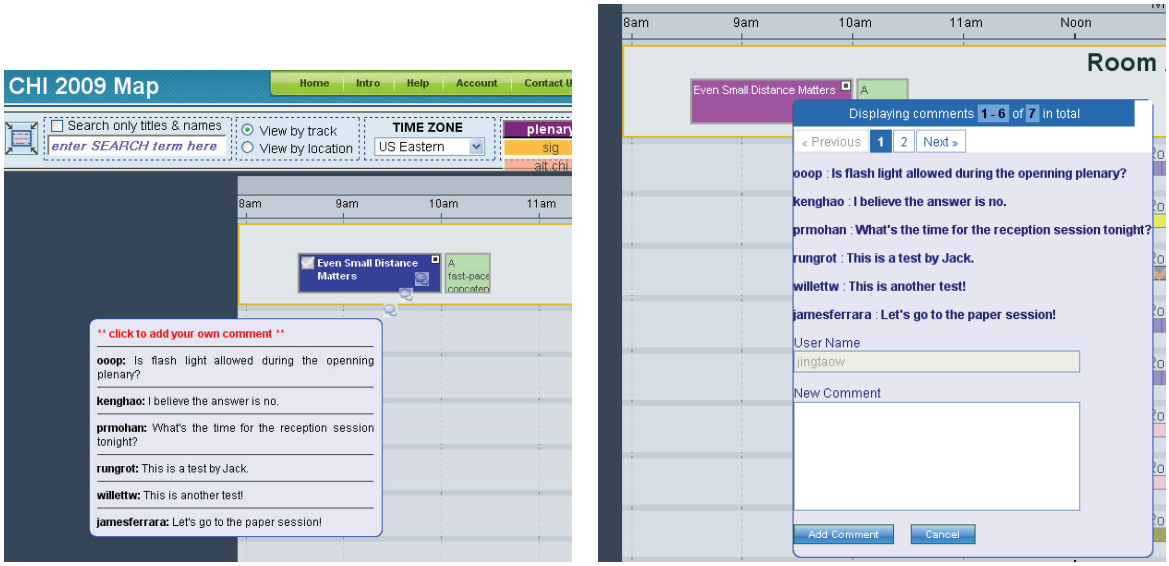

Figure 6.4: Collaboration features in *Event Maps*. (Left: a solid square on the top right corner indicates that there are associated comments; hovering over the indicator will display them. Right: clicking on the indicator brings up a comment-entering widget).

Personalization/Favorites Management

After logging in, a user can add an event to "My Favorites" by simply checking the check box at the "personalization corner" of a session or clicking the "Add to Favorites" button in its details pop-up. Proxies for favorite session are placed at the bottom of the screen. A user can remove a favorite by unchecking the check box the same button in the details pop-up, now labeled "remove from favorites". Any sessions that a user has added to his favorites are shown with a check mark on the "personalization corner" thus giving the user a quick indication of when she is free to schedule other activities.

Asynchronous Collaboration

In contrast to collaborative visualization web sites such as sense.us [49] and swivel [122], which were designed to support and promote collaborative sense-making, we believe that different *Event Maps* users may have different goals in mind. Some might only be interested in getting the schedule information they need quickly and exhaustively, while others might be interested in exchanging information or discovering interesting things from other participants. Thus, our major design principle for the asynchronous collaboration feature is that it can be ignored easily by people who do not need it, yet be discovered and accessed conveniently by interested parties.

Asynchronous collaboration features are accessed from the top-right *Active Corner* (a.k.a. "*collaboration corner*") of any session. Mouse hovering on it will display comments associated with the session (Figure 6.4, left). After logging in, mouse clicking on the *collaboration*

corner will pop up a commenting/annotating widget (Figure 6.4, right). A tiny black dot will show up on the "*collaboration corners*" of sessions with comments or user generated tags. Again, adhering to our design principle, the user does not have to go to a new destination to access the collaboration features.

Like in a wiki system, a user who has logged in, optionally with administrator privileges, can edit the event details in place by clicking an "edit" button on event details pop-up.

6.4 The Evolution of *Event Maps*

Event Maps draws its design from its earlier prototypes and a broad set of inspiring earlier works. While the individual features of *Event Maps* represent variations of existing approaches, our primary contribution is the integration of several techniques to create a novel application that is both usable and useful in an important domain. As of this writing, *Event Maps* has gone through three major rounds of iterative design, Here we report the insights and lessons learned during this process.

Our first design (P1) used a hierarchical, tree style zoomable layout as the primary interface as shown in Figure 6.5, Figure 6.6 and was drastically different from the traditional look and feel of a calendar. P1 was implemented as a Rich Internet Application in Adobe Flex 3.0. The primary view of P1 is a zoomable "event tree". Conference events are aggregated and placed on different levels based on properties, such as day, time, topic, speakers, location and keywords of the event by using an approach similar to Potter's Wheel [102]. We also implemented features such as semantic zooming [7], brushing, and drilling down [20] in P1. To support collaboration we implemented a double-linked commenting feature. Comments are shown in a foldable panel on the left, a separate comments view and as miniaturized, color encoded dots on conference events.

Our first prototype received mixed feedback from early testers - on the one hand, many people were excited by the rich interactions provided by our prototype. On the other hand, we also identified significant usability problems, especially from people who were less technically savvy.

For example, the tree-based interface requires extra learning time before people can take advantage of the rich interactions supported by the interface. Based on the authors' rough estimation, it took a novice user around ten minutes to get familiar. A frequent question we received was "*how are conference events placed at different levels*?" In retrospect, a tree-based hierarchy might not be the most intuitive conceptual model for conference events. Promoting a drastically new interface should take into account leveraging users' expertise in existing interfaces.

In the process of creating P1, we also discovered the effectiveness of certain features, which we carried out to future prototypes. Two important lessons were:

1. It is effective to use semantic zooming [7], i.e., revealing information differently based on the zooming level, to keep a good balance of legibility vs. visibility in a zoomable interface for temporal data.

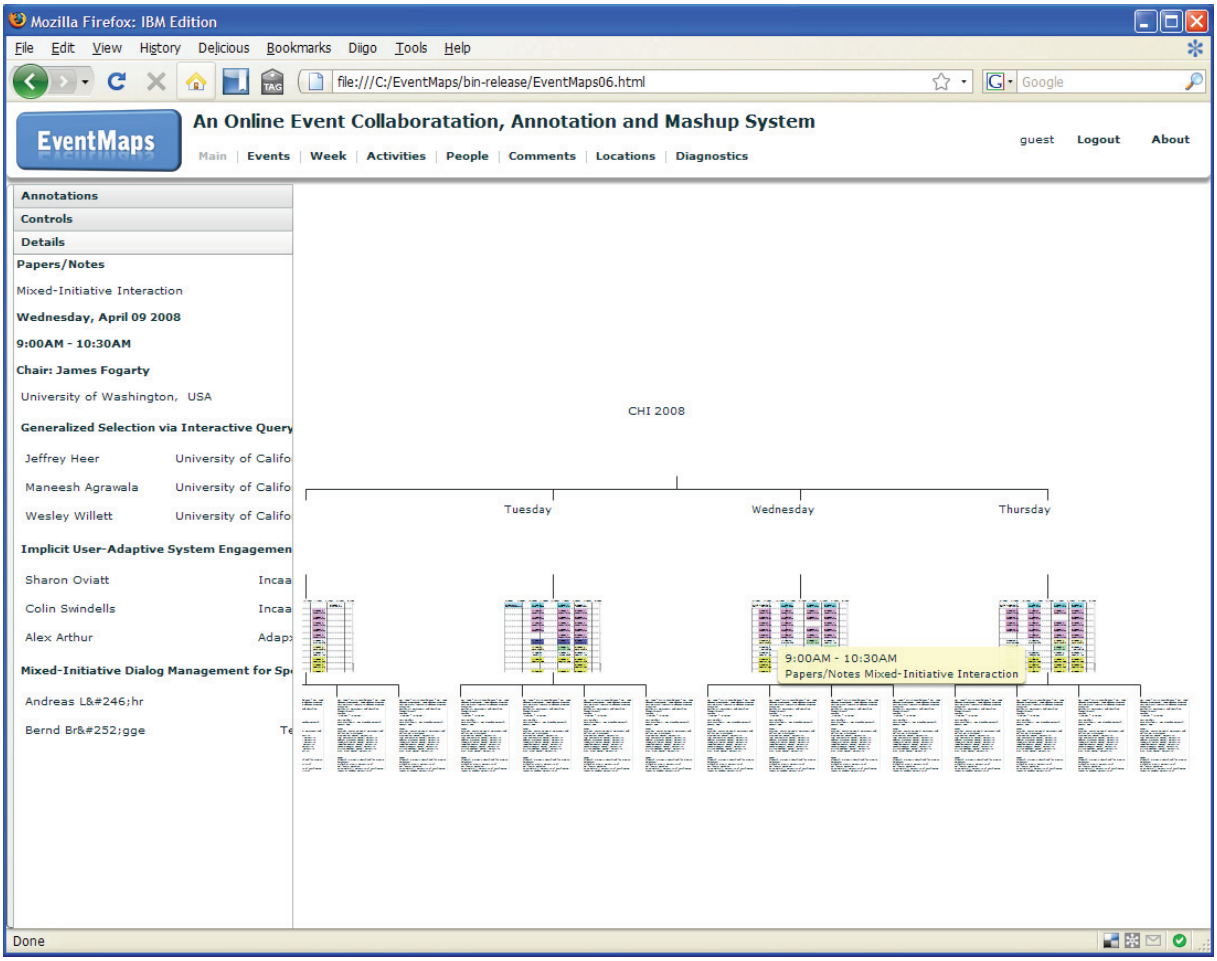

Figure 6.5: The first iteration of *Event Maps* implemented in Adobe Flex. (In overview mode, events are organized in a tree structure).

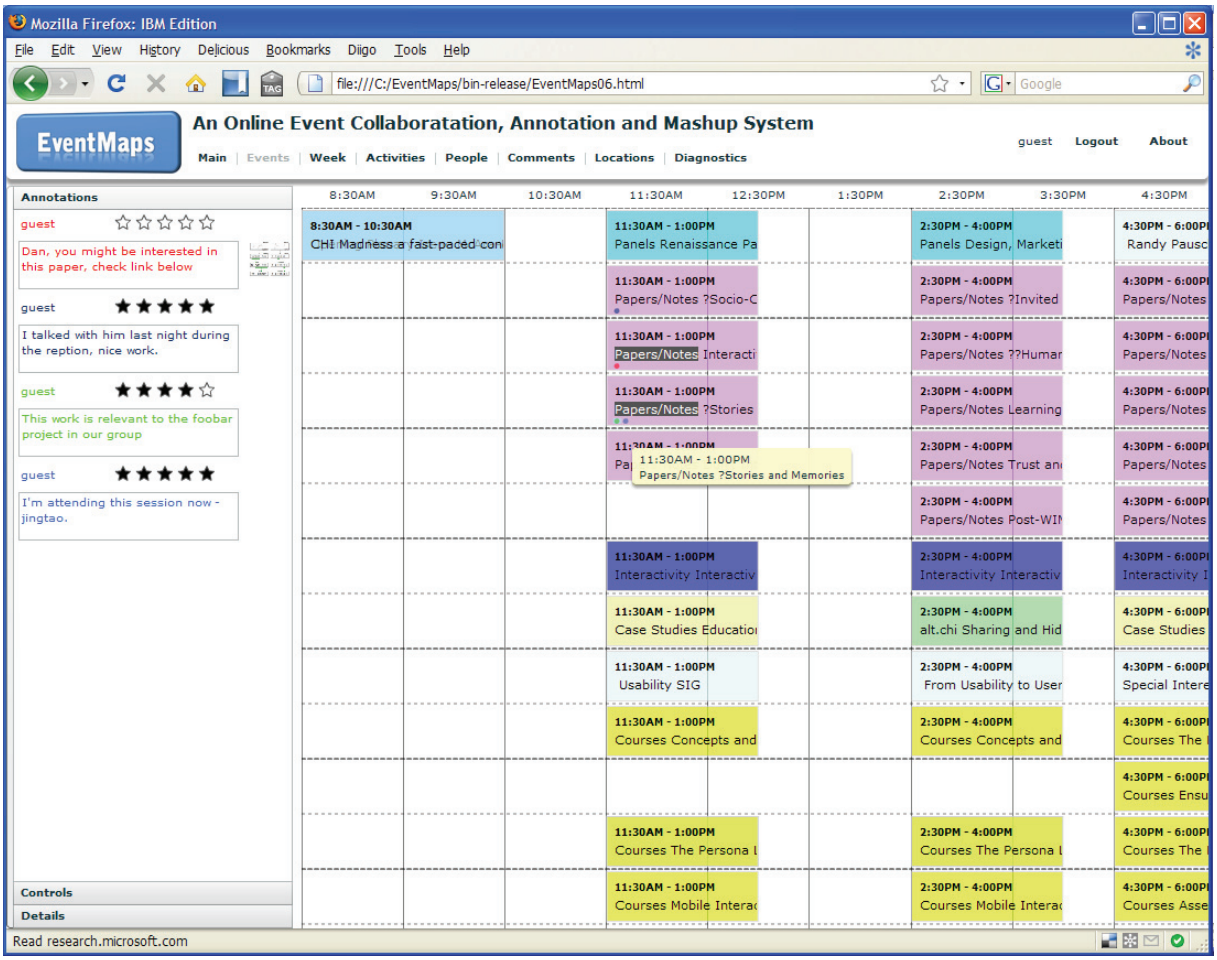

Figure 6.6: The first iteration of *Event Maps* implemented in Adobe Flex. (The zoomed in view of a day).

2. Although the entire schedule information might easily overwhelm the human eye, it can be stored quite compactly on a computer in standard format. For example, the entire CHI 2009 conference schedule, even with paper abstracts not available on the official website, is around 498KB in plain text. With compression this goes down to 150KB. We learned that we can provide a more responsive system by sending the whole schedule at the start rather downloading session details on demand via fine-grained web services.

We built the second system (P2) based on the lessons learned from the first prototype. We switched the browser side technology from Adobe Flex to JavaScript/AJAX to enable access from even mobile device browsers. The look and feel of the second prototype is similar to the current prototype (P3) shown in Figure 6.1. The biggest change we made in P2 is the introduction of the *Tabular Timeline* interface. P2 was deployed in an intra-company conference named SRS 2009 (described in detail in the next section). Based on both quantitative analysis of access logs on the server and qualitative interview results with 6 active users of *Event Maps* during the conference, we built P3, and ran a 12 participant user study to compare P3 directly with a state-of-the art conference schedule website (that of CHI 2009 [23]). P3 was hosted in a public conference - IEEE SCC 2009 (http://conferences.computer.org/scc/2009/), International Conference on Services Computing between September 21 and September 25, 2009. The major changes made between P2 and P3 were:

- Addressed usability problems identified in previous studies. Two examples, the search box and the reset view button, will be detailed in the next section.
- Designed and implemented the *Tabular Timeline* feature for personalization and asynchronous collaboration.
- Built the server side administrative interface and infrastructure for creating/importing new conferences and updating existing events efficiently.
- Improved the backend logging, analytic functions and interfaces for conference organizers.

With an improved server side administrative interface for managing conference events, it becomes convenient for administrators to import and maintain the schedule. We have tested P3 with event data from 19 conferences so far, some sample conferences we tested are listed in Table 6.2.

6.5 Implementation

The current implementation of *Event Maps* includes two parts: a client side and a server side. The client side is implemented in JavaScript/AJAX and can run on major web browsers such as FireFox, Safari, Internet Explorer, and Chrome. The server side is

Conference	Total Events	Categories of Events	Duration (days)	# of Rooms
CHI 2009	475	14	4	14
CHI 2008	467	12	4	14
Lotusphere 2009	351	7	5	19
SCC 2009	241	14	5	7
OSCON 2009	306	24	5	19
SRS 2009	126	8	4	9
RailsConf 2009	115	4	4	10
MySQL 2009	190	22	4	16

Table 6.2: Statistics of some sample conferences supported by *Event Maps*

implemented in Ruby on Rails 2.3 [110]. MySQL or IBM DB2 is used on the server side for data persistence. Excluding third party libraries and code for unit testing, the current *Event Maps* implementation includes a total of 11,032 lines of code in JavaScript that runs in the browser and a total of 12,980 lines of code in Ruby.

The JavaScript client is object-oriented, with objects corresponding to the major model, view and controller elements. It uses only standard JavaScript with no special libraries, and relies heavily on dynamic manipulation of the DOM and associated style sheets. The various animations are performed via timeout loops, and the DOM is only partially recomputed during an animation for improved performance. As browser support for JavaScript efficiency improves (the recent upgrades to Firefox 3.5 and IE 8 are good examples), so does the performance of *Event Maps*.

In our system, the client side and the server side use JSON [65] over HTTP to exchange user-profile data and user-generated data. The conference schedule data (including detailed times and descriptions) are sent to the client as a single text file (around 498KB for CHI 2009 [23] before compression) in iCalendar format [107]. By adopting open standards, when necessary, both the client side and the server side can be changed to a different technology in the future without impacting the interoperability.

As a result of our decision to use only standard JavaScript and AJAX, *Event Maps* can run on a wide range of operating systems and browsers. For example, Figure 6.7 shows two screen shots of *Event Maps* running on an iPhone. *Event Maps* maintains a similar user experience to that on desktop computers and portable, and the pinch operation on the iPhone maps rather nicely to the basic zoom operation. That said, the effectiveness of *Event Maps* on small form-factor devices needs to be evaluated. We plan to study this issue, as well as providing some user interface optimizations for portable devices e.g., leveraging multi-touch gestures as alternatives to mouse-hovering.

6.6 User Study

We now report both the insights and lessons learned during the iterative design process for building *Event Maps*.

Figure 6.7: Screen shots of *Event Maps* running on iPhone (top-left: CHI 2009 schedule in overview mode, top-right: zoomed in view with in-place event editor enabled, bottom-left: CHI 2009 schedule in zoomed in day view, bottom-right: showing the details popup windows of an event).

6.6.1 First User Study and Feedback

Initial user feedback on an early version of P2 was obtained via an informal study during an hour-long department meeting. After a short general introduction to *Event Maps* with no training, the 16 participants were given a link to a questionnaire hosted on SurveyMonkey.com that guided them through 3 tasks as if they were conference attendees (before, during, after the conference). Each task required some sort of result (e.g., a session name) and a free-form response on how well *Event Maps* supports the task scenario. The final page listed 18 *Event Maps* features and asked, for each one, to check "didn't use it", "don't like it", "it's OK", or "nice!". Some noteworthy lessons we learned from this study were:

Support unexpected interactions double-click caused a bug in a place in the code we had designed for single-click.

Don't be overly cute in the "related information" frame we used a semi-transparent red X in the middle. Some users did not see it, and were stuck, unable to close the frame!

Support freedom of use the following comment led us to make the "related information" URL explicit: "I found the recording of this talk, which was very valuable, but I wish the page would open in a new window, so that I can add it to my Firefox bookmarks for listening it later."

Nudge them (within reason) Paradoxically, it was a good idea to make the free-form questions mandatory - that's where we learned the most. It would have been good to also ask users who did not like a feature to explain why.

6.6.2 Live Deployment

We conducted a live deployment of *Event Maps* version P2 for an internal Software Research Strategy meeting (SRS 2009) held across eight IBM Research sites worldwide. It had 126 events spread over four days (Table 6.2). Participants could either attend local events or watch live online broadcasts of remote sessions. A link for *Event Maps* was posted on the primary conference website and an announcement was made during the opening plenary.

During the course of this conference, we collected a total of 3,297 significant logs from 298 unique IP addresses. Figure 6.8 reveals the visitation frequency from these unique addresses. Similar to most web-based activities; the visitation frequency conforms to the power law distribution.

76.1% of users were Windows users and 23.9% of users were Mac users. Firefox 3 was the most popular browser (52.9%) and Internet Explorer 7 accounted for 23.2% percent of the total access. 13.1% used Safari, and 6.4% used Chrome (figure 6.9). The effort we put to target and test on multiple browsers had paid dividends.

While nearly 69% percent of the traffic was from three sites in U.S., there were visitors who came from ten different time zones of the globe. This fact confirmed the importance of the "time zone" selection feature we implemented.

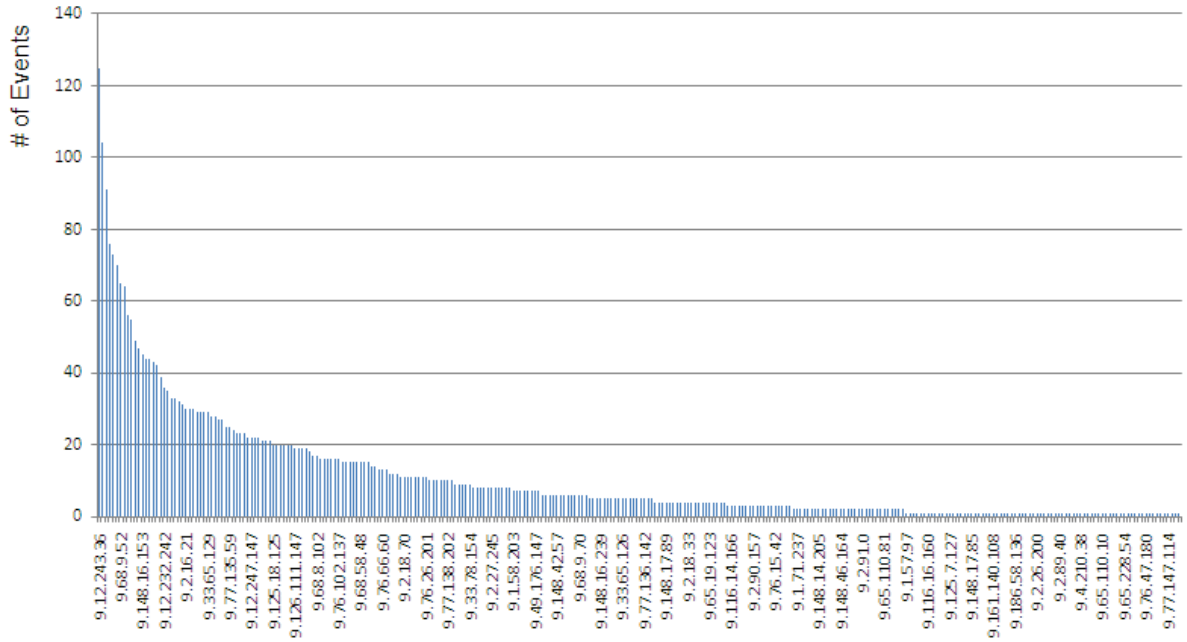

Figure 6.8: Visiting frequency by IP during SRS 2009. No IP includes events from two or more registered users.

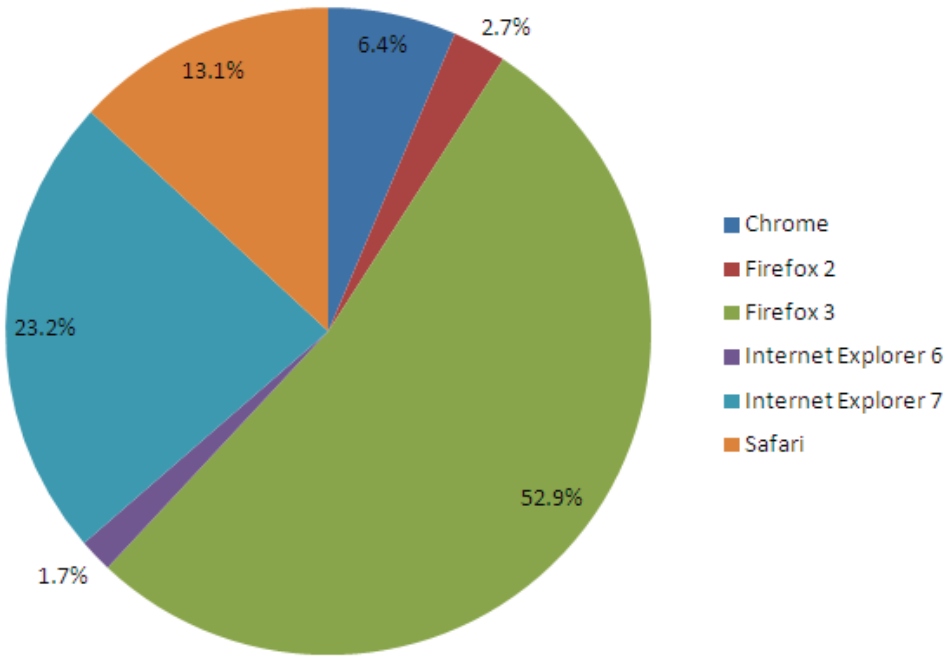

Figure 6.9: Distribution of browsers for the SRS 2009 deployment.

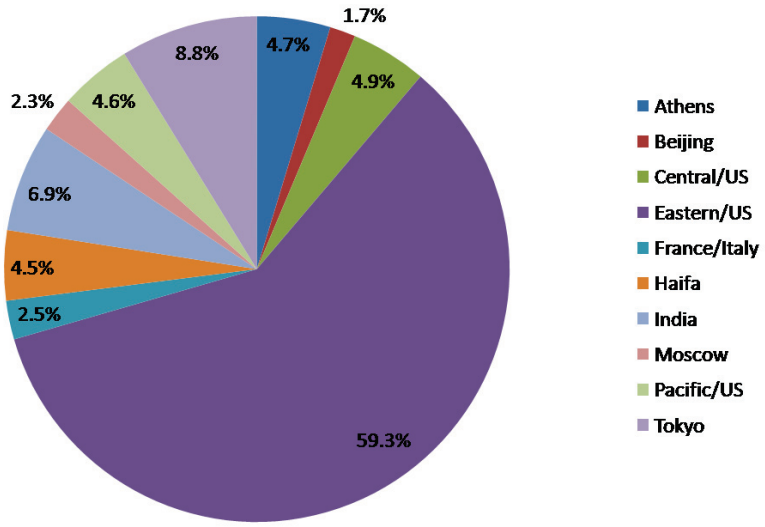

Figure 6.10: Distribution of users' location and time zone for the SRS 2009 deployment.

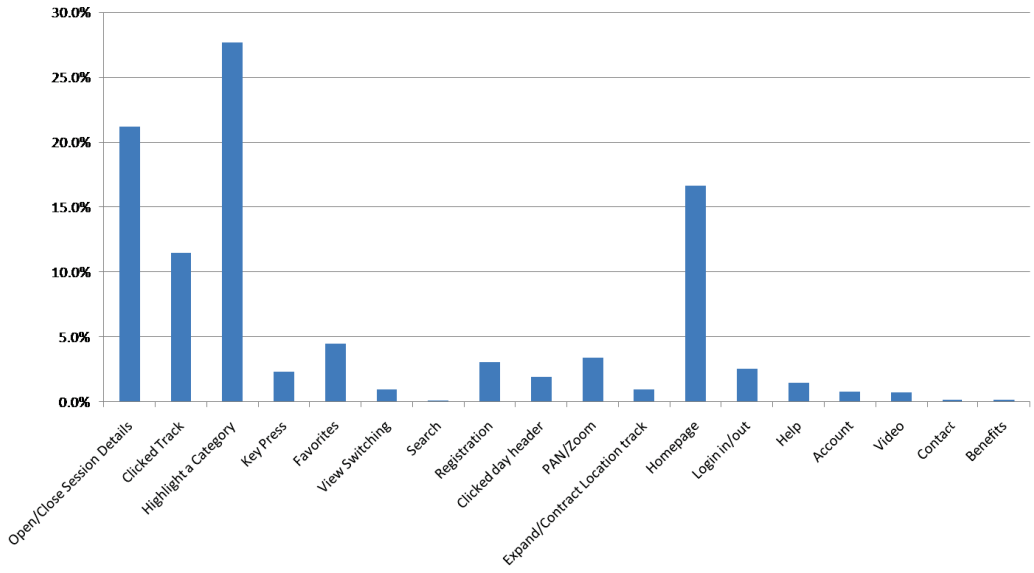

Figure 6.11: User activity by type of operations for the SRS 2009 deployment.

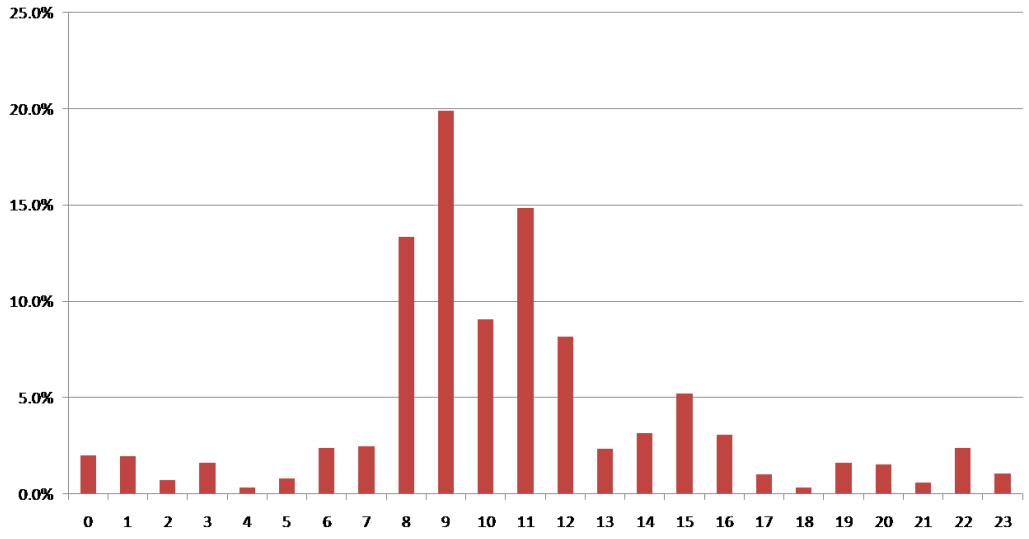

Figure 6.12: Site visiting frequency by hour of the day for the SRS 2009 deployment.

By analyzing the traffic log, we also identified several areas for improvement in *Event Maps*:

- Despite the relatively large number of visitors, only 19 visitors (around 6%) registered an account on *Event Maps*, most people used the web site anonymously. As a result, they didn't have a chance to try features such as persistent favorites or preferred time zone. Only 53 users re-visited *Event Maps* after more than one hour.
- To our surprise, the search function was rarely used during the deployment; most interactions were drill down operations for obtaining detailed information.
- Although we provided a simple tutorial in a highly visible tab on the website, less than 3% of the users actually visited it. Literally no one used the feedback form on the website during the study, though we did receive suggestions from several via e-mail.

6.6.3 Interview Results

After the conference, we identified frequent users that had registered, and sent them interview invitations. 6 users (2 females and 4 males) accepted the invitation and agreed to have an interview with us regarding their experiences with *Event Maps* (four interviews were conducted face-to-face; two were conducted over the phone). No incentives were offered to participate. The interviews were semi-structured including questions on how they heard of *Event Maps*, their experiences with specific features, such as favorites and search, and their opinions on prospective features to be implemented. We also went through open questions such as likes, dislikes, suggestions and comments. The Interviews lasted 30 to 40 minutes; five were audio-recorded with explicit consent of the interviewees.

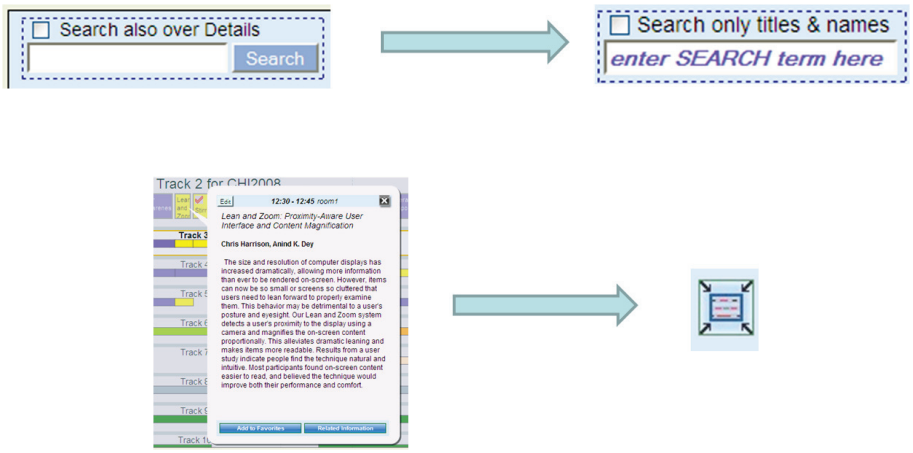

Figure 6.13: Sample UI change after the SRS 2009 deployment.

Highlights of the face to face interviews:

***Tabular Timeline* is Intuitive** most interviewees said that the zoomable interface of *Event Maps* was easy to master and use. The color-coded events and the hovering-to-highlight category labels were used by most interviewees. In general, they all liked this feature and indicated that it improved their awareness of ongoing conference events.

Not looking at any help, mouse click as the probe Rather than looking for a help page, almost all the users explored the interface directly. Clicking and hovering the mouse on various regions of the screen were the two most popular probing activities. At the same time, dragging, drag and drop, right button clicking and keyboard shortcuts (one interviewee "didn't even think of it!") were rarely used to explore the interface. As a lesson learned, when we add new features to *Event Maps*, we always make sure that they can be discovered by left mouse clicking or hovering.

Interests in other people's opinions In addition to all the "official" information (including slides and video), most interviewees indicated their keen interest in knowing other people's opinions and interests on conference events, especially people they know and trust.

As detailed in the design evolution section, most of the new features we added in P3 were driven by the lessons and insights from the SRS 2009 live deployment. In addition to new features, we also made various adjustments to the UI to address usability problems we had identified.

Figure 6.13 shows a sample UI revision we made - since the keyword search function is completely locally without internet connection, Removing the search button suggests that searching is responsive and encourages the users explore with different keywords. We also noticed via analyzing the access logs that the most common action after accessing the detailed information of a session was going back to the global view, so we added an explicit

the reset view button (figure 6.1.e) in addition to keyboard shortcut on the top left corner to support this action. The reset view button now becomes one of the most frequently access features in *Event Maps*.

6.6.4 Lab Study

After completing P3, we ran a formal lab study to evaluate the effectiveness of *Event Maps* in completing common tasks in a conference, especially its relative performance when compared with an existing state-of-the-art conference web site for accessing the same schedule information.

Participants

12 people (3 female, 9 male) between 22 - 36 years of age (mean = 26.1) participated in the user study. Subjects were collected via recruiting emails to multiple internal mailing lists in a university. All participants were right-handed. All participants have some previous experiences in attending a conference before the study. 10 subjects had an educational background in engineering, one in business administration and one in linguistics.

Apparatus

The experiment was conducted using a Lenovo Thinkpad T400 Laptop computer with an external Microsoft Laser Mouse 6000 as the pointing device. The computer had an Intel Core 2 Duo 2.26GHZ CPU, a 14 inch LCD monitor at a resolution of $1440 * 900$ pixels. The operating system was Windows XP SP3. Firefox 3.5 was used as the browser for all the tasks in our study. The *Event Maps* server used in this study was hosted on a Dell PowerEdge server running Ubuntu Linux 8.04 server edition, with a Pentium 4 2.8 GHZ CPU and 2GB Ram.

We pre-loaded 14 conferences to the *Event Maps* server, including those listed in Table 6.2. Subjects could freely browse these conferences on *Event Maps* during the exploration stage of the test. Subjects were asked to complete tasks for ACM CHI 2009. We also mirrored the ACM CHI 2009 conference schedule pages [23] on the same server as an alternative technology. The two major reasons for mirroring were to have both systems studied have the same network access speed, and to enable logging of users' visitation behavior on the web pages during the study. No modifications were made for to mirrored web pages.

Experimental Design

The experiment used a within-subject design, using two sets of comparable tasks. Each set included 12 tasks. The tasks were created by interviewing frequent conference participants about their high-frequency activities in a conference and by consulting domain experts in CSCW.

The nature of the tasks in each set is summarized below -

1. Locating the starting time of a talk;

2. Locating an author based on a keyword in the title of a talk;
3. Planning a short-term (2 hour) schedule for a given interest;
4. Determining the time and location of a talk by a given author on a given day;
5. Obtaining details of an event, given the time and event category;
6. Planning a long-term (half day) schedule for a given interest;
7. Counting events in a given category within a certain time span;
8. Obtaining details of an event given relatively vague time constraints;
9. Obtaining details of an event given accurate time and location information;
10. Finding a co-author of a given author;
11. Comparing the popularity of two technologies (in terms of keyword frequency);
12. Counting events in a given category within part of a day.

Brief tutorials (around 7 minutes) were provided prior to each set of tasks on each system. During the tutorial, the experimenter first demonstrated major features of the corresponding system. Then we encouraged users to spend some time to explore the system. We waited to start the actual tasks until the participant indicated explicitly to the experimenter that he/she was comfortable with the current interface. Most participants spent around 5 - 10 minutes in exploring each interface before starting the actual tasks. The participants performed the set of 12 tasks mentioned above using each calendar. The order of calendar use and task set for the calendar were both counterbalanced in order to minimize the effects of training, or the possibility of one task-set being slightly more difficult than the other. However, tasks within a set were not randomized.

Results

The average completion time of all the tasks are shown in Figure 6.14. Average completion times for *Event Maps* were shorter than on the official website in all tasks. Using *Event Maps*, the average completion time for each task was less than 46 seconds; while in the traditional web site, the average time for some tasks, e.g. task 6 (long planning) and task 11 (counting), could be more than 2 minutes. However, the differences are statistically significant only for tasks 1, 4, 6, 8, 10 and 12 (task 1: $p < 0.005$) , task 4 ($p < 0.001$), task 6 ($p < 0.005$), task 8 ($p < 0.005$), task 10 ($p < 0.02$), task 11 ($p < 0.001$), task 12 ($p < 0.001$).

The difference in average completion time for task 2 ($p = 0.12$), 3 ($p = 0.15$), 5 ($p = 3.76$), 7 ($p = 0.21$), 9 ($p = 0.12$) were not statistically significant.

An interesting observation was the performance of task 3 (planning a 2 hour schedule for a given interest) and task 6 (planning a similar half-day schedule) in both conditions.

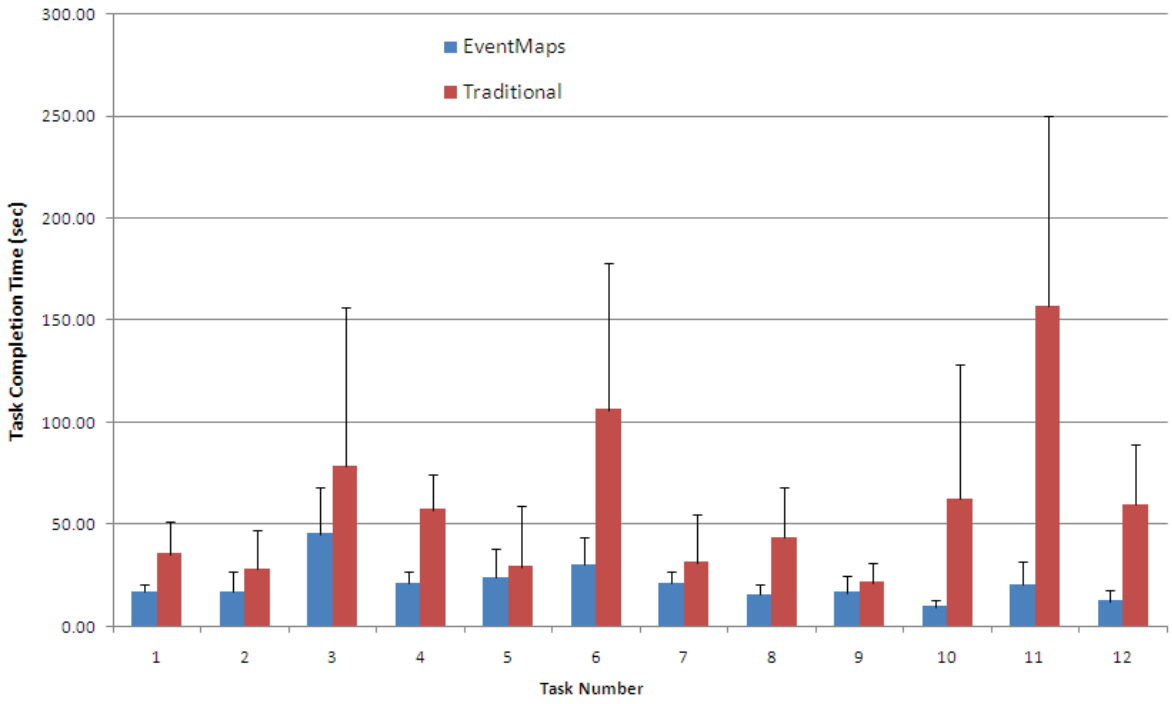

Figure 6.14: The average completion time (seconds) vs. task number. The error bar indicates the standard deviation of corresponding completion time.

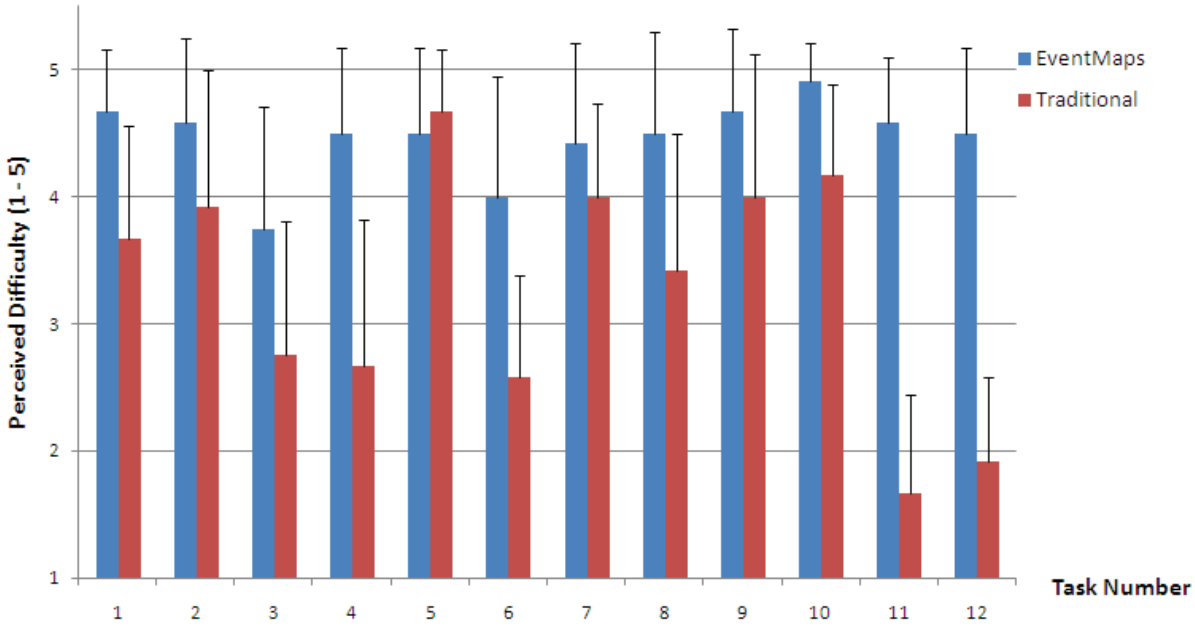

Figure 6.15: The average perceived difficulty for each task. (1 = extremely difficult, 3 = neutral, 5 = extremely easy). The error bar is the corresponding standard deviation.

Unsurprisingly, on the traditional web site, the time spent in completing the task increased as the required planning time-span increased, as more page switching was needed. However, although the difference is not statistically significant, on average it took less time to schedule a longer trip in the *Event Maps* condition! We attribute this observation to two reasons: First, these two tasks can be completed in the global view mode with either mouse hovering or keyword search, as there is no view switching involved despite the extended planning time-span. Second, there might be learning effects between task 3 and task 6, since the order of tasks in each test set is not randomized.

Figure 6.15 shows the subjective "perceived" difficulty of the tasks on a 5 point Likert scale [78] (5 means extremely easy, 3 means neutral and 1 means extremely difficult). In accordance with the actual performance, *Event Maps* was perceived easier to use than the traditional web site in most tasks. Interestingly, task 5 (Locating the name of a panel that meets the provided date and time constraints) was perceived to be easier to finish via the traditional web interface even though the average completion time of that task was slightly shorter using *Event Maps*.

In the closing questionnaire, many users indicated explicitly that they like the zoomable *Tabular Timeline* layout and the search feature in *Event Maps*. Sample quotes include

> *"[*Event Maps*] links time with location clearly."*

> *"Presentation is clear, Integrated search is very helpful and is more helpful even*

Satisfaction question regarding *Event Maps*	Mean	Stdev
Overall, I am satisfied with this system	4.58	0.67
It was easy to learn	4.25	0.62
Improved my awareness of ongoing events	4.50	0.67
Improved my ability to find info on a specific topic	4.75	0.45
Helped me to keep track of events I'm interested in	4.25	0.45
Saved my time in locating the information I need	4.83	0.39
I could effectively complete the tasks in my mind	4.42	0.67
I was able to complete the tasks and scenarios quickly	4.42	0.51
The "look-and-feel" of this system was pleasant	3.92	0.67
Organization of information in this system was clear	4.42	0.51

Table 6.3: Summary of the satisfaction rating in the closing questionnaire (on 5 point Likert scale, 1 means strongly disagree, 3 means neutral, 5 means strongly agree)

> *than the type ahead search in the [Firefox] browser which I use constantly and leveraged to complete the tasks on the old web site."*
>
> *"The search ability, and the representation of all events in categories in the timeline fashion made [it] easy for navigation."*

During the study, at least 4 users clicked on category labels while exploring the system. When asked why, the given answers were similar: those labels are button-shaped, when the mouse is over a label, that label gets embossed and looks exactly like a button, implying that they are click-able. Some users noted that when hovering on the "course" category label, they could not see anything because those courses are out of the current view port - they suggested that it might be better to pan the current view automatically to make most of the "course" events visible. One user suggested that when a category is clicked, events of all other categories become homogeneously dark, until the same category label or other category labels are clicked. In this way, the users could scroll the current view up and down to get some global feelings about a specific category. Interestingly, the iPhone already supports "sticky" category highlighting in lieu of hovering support!

Table 6.3 summarizes the average satisfaction rating after using *Event Maps*. Overall, the ratings were highly favorable. Participants strongly believed that using *Event Maps* could save their time in locating the information they need (mean = 4.83, stdev = 0.39). The "look-and-feel", especially the color theme of the current system was considered to still have significant room for improvements.

6.7 Discussions

Based on evidence from studies and deployments we have demonstrated that users feel *Event Maps* provides a good mental map and faster access to several typical pieces of information conference attendees need through its use of a zoomable interface and by exposing

information on demand conveniently via tool tips and mousing. Our practical experience in building *Event Maps* leads us to the following observations:

- *"It is faster to make a four-inch mirror, then a six-inch mirror than to make a six-inch mirror"* - The Thomson's Rule for First-Time Telescope Makers [9] also applies to building interactive systems.
- Build simple components as services with clean interfaces for adoption in broader contexts.
- Avoid premature optimization to get to the right design faster.
- Disk is cheap, so log and analyze more data than less.
- Scripting languages speed up prototyping [97].

Some areas for future work include the following:

- Customize *Event Maps* for access from popular mobile devices with particular attention to usability, i.e., interaction in mouse-less environments, legibility on small screens, and limitations in compute power.
- Use information provided by attendees, including browsing patterns within *Event Maps*, to improve the conference experience.
- Enable micro-tagging on a wide variety of conference material - including sections of papers, authors, institutions, recordings, etc.
- Make the conference site the hub for activity even after the end of the conference to benefit both attendees and those not fortunate enough to get the live experience.
- Incorporate indoor floor maps to enable several location-based services pertinent to conference attendees.

6.8 Summary

Event Maps is a highly interactive web-based collaborative calendaring system optimized to provide quick access to information at multiple levels of detail for large, multi-track conferences. Further, *Event Maps* can be used as a "collaboration space" for visualizing and collaborating on large, complex temporal data with a hierarchical nature. The results reported in this chapter previously appeared in [138, 139].

The *Event Maps* system has been refined twice based on quantitative and qualitative feedback from user studies, lab studies and deployments. The feedback we have received thus far on how *Event Maps* enhances the conference attendee's user experience has been very encouraging and we will thus be pursuing even larger deployment opportunities to gain insights into how conference collaboration substrates get used during various stages

of a conference and from different device form factors, build a prioritization of the feature set based on several factors, and use the findings to allow *Event Maps* to meet the user's needs in various times and situations. We will analyze and report in depth the insights from face-to-face interviews, lab based studies and real world deployments throughout the iterative design process. In addition to a holistic evaluation of the overall effectiveness of *Event Maps*, we are also interested in getting a deeper understanding of the impact of each specific feature in its corresponding design space.

Chapter 7

Conclusion

"All understanding begins with our not accepting the world as it appears."

—Alan Kay, 1998 [113]

7.1 Summary of Contributions

This thesis summarized research work on understanding how emerging sensors on mobile phones, such as built-in cameras, microphones, touch sensors and the GPS module can be leveraged to make everyday interactions easier and more efficient. We presented studies and models to quantify the capabilities of these sensing channels, and illustrated how effective interfaces for text entry, gaming, and CSCW can be built on mobile phones.

The main contributions presented in this dissertation are:

- We have developed a technology named *TinyMotion* for camera phones. *TinyMotion* detects the movements of a cell phone in real time by analyzing image sequences captured by its built-in camera, providing a usable analog pointing channel to existing cell phone users. In contrast to earlier technology, *TinyMotion* does not require additional sensors, special scenes, or backgrounds.

- We quantified *TinyMotion*'s human performance as a basic input control sensor. We found target acquisition tasks via *TinyMotion* follow Fitts' law , and Fitts' law parameters can be used for *TinyMotion*-based pointing performance measurements. We established a benchmark baseline for future improvements in both hardware and detection algorithms.

- We designed and implemented *Mobile Gesture*, an application that can convert the camera phone into a stylus capable of capturing and recognizing more than 8,000 characters and gestures written in English, simplified Chinese, traditional Chinese and Japanese characters at the speed of 35ms per character on a Motorola v710. This is a first-in-the-world application to demonstrate the feasibility of using the camera phone

as a handwriting capture device, and for performing large vocabulary, multilingual, real-time handwriting recognition.

- We designed and implemented an effective mobile text input method named *SHRIMP* (**S**mall **H**andheld **R**apid **I**nput with **M**otion and **P**rediction). *SHRIMP* runs on camera phones equipped with a standard 12-key keypad. *SHRIMP* maintains the speed advantage of Dictionary-Based Disambiguation (DBD) driven predictive text input while enabling the user to overcome collision and OOV problems seamlessly without explicit mode switching.

- We invented a novel interaction technique, *FingerSense* to enhance the expressiveness of physical buttons on space-constrained mobile devices. In a *FingerSense* enabled input device, pressing actions are differentiated according to the finger involved. We modeled the human performance of *FingerSense* interfaces via a GMOS analysis and derived related parameters from a preliminary usability study.

- We presented *GLAZE*, **G**eneralized **L**ocation **A**ware model**Z** for **E**nd-users, for creating location-aware mini-applications. The *GLAZE* system allows average users, who do not have programming experiences, to create many everyday location-aware applications through a set of form-filling and menu selection operations on mobile devices.

- We designed and implemented *Event Maps*, a web-based calendaring system targeted at improving the experience of attending and organizing large, multi-track conferences. Through its zoomable *Tabular Timeline*, users can navigate a conference schedule, seamlessly moving between global and local views. Through a compact decoration widget named *Active Corners*, *Event Maps* enables contextual asynchronous collaboration before, during, and after the conference. *Event Maps* has been successfully deployed in large, real world conferences including IEEE SCC 2009, Lotusphere 2010 and CHI 2010.

7.2 Experience Designing Effective Mobile Interfaces

This thesis highlighted about six years of research work in creating effective interfaces on mobile devices. It covered multiple projects on different aspects of mobile interfaces under the umbrella - "*perceptual and context-aware interfaces*"[19]. But what can be said in general? Here we summarize briefly some reflections and insights generated in the process of designing, building, and evaluating these technologies.

The importance of iterative design First and foremost, we can't emphasize enough the importance of rapid iteration in building research systems. Instead of citing some classic text books, we'd like to draw from a few concrete examples from previous chapters. (a), Just as the *Thomson's Rule for First-Time Telescope Makers* goes - "*It is faster to make a four-inch mirror, then a six-inch mirror than to make a six-inch mirror*"[9]. Indeed, even under tight time constraints, starting from simpler implementations and

building multiple prototypes can save the overall time spent. For example, in the *Event Maps* project (chapter 6), a total of three prototypes were built and the look-and-feel of P1 differs significantly from P3. Many failed ideas were thrown away quickly and effective solutions were kept in the iterative process. It's clear that we wouldn't be able to deploy the *Event Maps* system quickly and successfully in real world conferences without all these rapid iterations. (b), To create really compelling innovations, it's critical to build the prototype and have the creators use it everyday, a.k.a. "*eating one's own dog food*". This approach does help researchers to understand the technology, identify usability problems as well as fix them in follow-up iterations. Many designs described in this thesis, e.g. the tactile feedback technique in chapter 2, the *SHRIMP* predictive input in chapter 3 and the numerous revisions of *Event Maps* in chapter 6 are inspired this way. (c), User studies can indeed help to capture unexpected problems and bottlenecks of interaction. For example, the "moving the frame" vs. "moving the background" problem and the "clutch" problem in chapter 2, the limitation of tree based visualization of temporal events and the non-clickable category label problem in chapter 6 are just a small sample of the countless usability problems we identified through user studies. Such experiences can be explained by the Zone of Proximal Development (ZPD) theory [133] in learning.

Design is about making trade-offs One common theme, which gets reinforced in almost every research project, is - we researchers shouldn't expect "*silver bullets*" when designing mobile interfaces. Bill Buxton summarized this observation well - "*Everything is best for something and worst for something else. The trick is knowing what is what, for what, when, for whom, where, and most importantly, why*" [17]. As an example, according to our benchmark in chapter 2, the motion sensing channel enabled by *TinyMotion* is reliable, but has a very low input bandwidth. Directly mapping the *TinyMotion* channel to traditional interaction tasks may lead to inferior results, as we measured in the menu selection task in chapter 2. However, with careful design, it's possible to leverage the strength of *TinyMotion* (i.e. an input channel that is parallel to the existing channels) and hide its weakness (low bandwidth), and create efficient, easy-to-learn and enjoyable-to-use input methods such as Vision TiltText and *SHRIMP*. We made multiple similar trade-offs among 'familiarity to users', 'consumption of on screen resources' and efficiency in the *Event Maps* project and the *GLAZE* project.

Development environments and UI toolkits are important BREW [13], a C/C++ based mobile phone API created by Qualcomm Corp. for non-smart feature phones was adopted in multiple projects described in chapter 2, 3 and 5. In retrospect, there are both pros and cons about this decision. On one hand, the C/C++ based API in BREW allows us to get the maximal possible performance from non-smart feature phones and hack deep into hardware components that are usually inaccessible via more high level APIs and frameworks. These feature phones account for more than 90% of the mobile phone market and are highly constrained in memory and computing power. Running our systems successfully on feature phones meant that our research

results are applicable to nearly any mobile phone on the market, not just high-end smart phones. The number of citations and follow-up projects of *TinyMotion* indeed supports this judgment. On the other hand, choosing BREW as the primary development platform significantly hindered our productivity in building more compelling prototypes. Just to name a few challenges in using BREW - (a), Main stream versions of BREW (i.e. 2.x - 3.x) at the time did not provide a complete set of necessary APIs to build even regular UI elements. Multiple add-on toolkits were proposed by Qualcomm as a solution, but none of them hit the market in time, or met the expected standard. (b), The API and development environment are unfriendly to developers. For example, almost every system call, including file/network IO, are asynchronous, as a result, call backs and complex exception handlers have to be set even for trivial system calls; worse, the emulator on the PC could neither emulate the exact look-and-feel, nor the behavior of programs on the real devices. Meanwhile, the debugging support on real devices is close to non-existent and developers have to relay on logs to figure out what's happening when something suspicious is observed. (c), Most tools and services relevant to BREW are private (e.g. the service for testing signature generation, the access to the BREW application loader and extranet), expensive (e.g. the RealView ARM Compiler, VeriSign signature service) and tightly controlled by Qualcomm (e.g. enabling the debug model on BREW phones in early days). (d), The documentation and implementations of BREW are inconsistent, full of bugs and typos(please refer to [12] for some bugs/typos we found and reported). Behaviors of APIs could be quite different on different handsets. As a conservative estimation, at least two person years of efforts could have been saved if we had chosen a more mature mobile development platform at the beginning. Considering the benefits, is this kind of additional investment worthwhile? It is up to the readers to decide.

KISS - Keep It Simple and Straightforward As researchers in computer science, we have the innate impulse to solve a problem as general as possible, and once and for all, even in the planning stage. A commonly advocated technique is - when a choice decision is needed, add another level of abstraction to postpone the design decision hence providing more "flexibility" to prospective users. Another common trend is - taking the extensibility/scalability/performance problem of *every* component in the system into account even in the planning stage. Third, solving problems with more complex solutions even when a much simpler solution could achieve close or better performance. Instead, we believe it is important to resist the temptation of "*treating complexity as beauty*" for at least two reasons. First, it is hard for most people, including experienced researchers, to estimate the true bottlenecks when approaching a *new* problem at the very beginning. On one hand, the expected performance bottleneck may not exist even under high workload (e.g. the case of loading all the evens to the browser at the beginning rather than request a small portion of them on demand in *Event Maps*); on the other hand, certain features, which researchers might be very proud of and have spent a lot of time refining, may not be extremely attractive from the end-user's perspective. It's important to allocate limited resource on things that truly matter from the user's perspective. Second, complex solutions, although

they may look compelling from the intellectual perspective, usually contain additional or unnecessary assumptions that may not hold in real world scenarios. Complex solutions can also lead to higher costs in implementation, debugging and maintenance. The pixel level motion estimation algorithm in chapter 2, the simple, modeless solution of *SHRIMP* mobile input in chapter 3, and the brute-force iCalendar file generation algorithm in chapter 6 are three examples of our own experiences.

Bibliography

[1] Christopher Ahlberg and Ben Shneiderman. Visual information seeking using the filmfinder. In *CHI '94: Conference companion on Human factors in computing systems*, pages 433–434, New York, NY, USA, 1994. ACM.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[3] D. Ashbrook. Learning significant locations and predicting user movement with gps. In *ISWC '02: Proceedings of the 6th IEEE International Symposium on Wearable Computers*, page 101, Washington, DC, USA, 2002. IEEE Computer Society.

[4] AtalasBook (a.k.a. VZ Navigator), by Networks in Motion. `http://www.atalasbook.com`.

[5] Rafael Ballagas, Michael Rohs, and Jennifer G. Sheridan. Sweep and point and shoot: phonecam-based interactions for large public displays. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1200–1203, New York, NY, USA, 2005. ACM.

[6] BREW Directed SMS on the BREW platform. `http://www.openmarket.com/docs/OpenMarket_BREW_Directed.pdf`.

[7] Benjamin B. Bederson. Fisheye menus. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 217–225, New York, NY, USA, 2000. ACM.

[8] Benjamin B. Bederson, Aaron Clamage, Mary P. Czerwinski, and George G. Robertson. DateLens: A fisheye calendar interface for pdas. *ACM Trans. Comput.-Hum. Interact.*, 11(1):90–119, 2004.

[9] Jon Bentley. Programmimg pearls. *Communications of the ACM*, 28(9):896–901, 1985.

[10] Why Book Means Cool. `http://www.languagehat.com/archives/002415.php`.

[11] Joel Brandt and Scott R. Klemmer. Lash-Ups: A toolkit for location-aware mash-ups. In *UIST '06: UIST '06 extended abstracts on User Interface Software and Technology*, New York, NY, USA, 2006. ACM.

[12] BREW Camera API documentation bugs identified and reported by the author. http://brewforums.qualcomm.com/showthread.php?t=9060.

[13] BREW: (Binary Runtime Environment for Wireless) Homepage. http://brew.qualcomm.com.

[14] British History Interactive Timeline. http://www.bbc.co.uk/history/interactive/timelines/british/.

[15] Mike Brzozowski, Kendra Carattini, Scott R. Klemmer, Patrick Mihelich, Jiang Hu, and Andrew Y. Ng. groupTime: preference based group scheduling. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1047–1056, New York, NY, USA, 2006. ACM.

[16] Bill Buxton. Gartner: Mobile To Outpace Desktop Web By 2013. http://www.mediapost.com/publications/?fa=Articles.showArticle&art_aid=120590.

[17] Bill Buxton. Multi-Touch Systems that I Have Known and Loved. http://www.billbuxton.com/multitouchOverview.html.

[18] The Development of Camera Phone Module Industry, 2005-2006. http://www.okokok.com.cn/Abroad/Abroad_show.asp?ArticleID=1034.

[19] John Canny. The future of human-computer interaction. *Queue*, 4(6):24–32, 2006.

[20] Stuart K. Card. *Human Computer Interaction Handbook*, chapter Information Visualization, pages 509–543. Lawrence Erlbaum Associates, 2008.

[21] Stuart K. Card and David Nation. Degree-of-interest trees: a component of an attention-reactive user interface. In *AVI '02: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 231–245, New York, NY, USA, 2002. ACM.

[22] Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.

[23] CHI 2009 Conference Program. http://www.chi2009.org/Attending/AdvanceProgram/monday.html.

[24] Sunny Consolvo, Ian E. Smith, Tara Matthews, Anthony LaMarca, Jason Tabert, and Pauline Powledge. Location disclosure to social relations: why, when, & what people want to share. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 81–90, New York, NY, USA, 2005. ACM.

[25] Allen Cypher. EAGER: programming repetitive tasks by example. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–39, New York, NY, USA, 1991. ACM.

[26] Marc Davis, John Canny, Nancy Van House, Nathan Good, Simon King, Rahul Nair, Carrie Burgener, Bruce Rinehart, Rachel Strickland, Guy Campbell, Scott Fisher, and Nick Reid. MMM2: mobile media metadata for media sharing. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 267–268, New York, NY, USA, 2005. ACM.

[27] Steven P. Dow, Kate Heddleston, and Scott R. Klemmer. The efficacy of prototyping under time constraints. In *C&C '09: Proceeding of the seventh ACM conference on Creativity and cognition*, pages 165–174, New York, NY, USA, 2009. ACM.

[28] Stephan A. Drab and Nicole M. Artner. Motion detection as interaction technique for games & applications on mobile devices. In *in Proc. Pervasive Mobile Interaction Devices (PERMID) Workshop at PERVASIVE*, pages 48–51, 2005.

[29] Mark D. Dunlop and Andrew Crossan. Predictive text entry methods for mobile phones. *Personal Technologies*, 4(2), 2000.

[30] Mark D. Dunlop and Finbarr Taylor. Tactile feedback for predictive text entry. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 2257–2260, New York, NY, USA, 2009. ACM.

[31] EyeMobile Homepage. `http://www.eyemobile.com`.

[32] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. 1954. *J Exp Psychol Gen*, 121(3):262–269, September 1992.

[33] George W. Fitzmaurice, Shumin Zhai, and Mark H. Chignell. Virtual reality for palmtop computers. *ACM Trans. Inf. Syst.*, 11(3):197–218, 1993.

[34] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[35] William T. Freeman, David B. Anderson, Paul A. Beardsley, Chris N. Dodge, Michal Roth, Craig D. Weissman, William S. Yerazunis, Hiroshi Kage, Kazuo Kyuma, Yasunari Miyake, and Ken-ichi Tanaka. Computer vision for interactive computer graphics. *IEEE Comput. Graph. Appl.*, 18(3):42–53, 1998.

[36] Masaaki Fukumoto and Yoshinobu Tonomura. "body coupled fingerring": wireless wearable keyboard. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 147–154, New York, NY, USA, 1997. ACM.

[37] Borivoje Furht and Borko Furht. *Motion Estimation Algorithms for Video Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1996. Performed By-Greenblatt, Joshua.

[38] George Furnas. The fisheye calendar system. Technical Report TM-ARH-020558, Bellcore, Morristown, NJ, 1991.

[39] Randy L. Genereux, Lawrence M. Ward, and James A. Russell. The behavioral component in the meaning of places. *Journal of Environmental Psychology*, 3(1):43–55, 1983.

[40] Jun Gong and Peter Tarasewich. Alphabetically constrained keypad designs for text entry on mobile devices. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 211–220, New York, NY, USA, 2005. ACM.

[41] gpsOne Position-Location Technology. `http://www.cdmatech.com/download_library/pdf/gpsone.pdf`.

[42] Dale L. Grover, Martin T. King, and Clifford A. Kushler. Reduced keyboard disambiguating computer. In *Patent No. US5818437, Tegic Communications, Inc.*, October 1998.

[43] Jari Hannuksela, Pekka Sangi, and Janne Heikkila. A vision-based approach for controlling user interfaces of mobile devices. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 71, Washington, DC, USA, 2005. IEEE Computer Society.

[44] Thomas Riisgaard Hansen, Eva Eriksson, and Andreas Lykke-Olesen. Mixed Interaction Space: designing for camera based interaction with mobile devices. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1933–1936, New York, NY, USA, 2005. ACM.

[45] Thomas Riisgaard Hansen, Eva Eriksson, and Andreas Lykke-Olesen. Use your head: exploring face tracking for mobile interaction. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 845–850, New York, NY, USA, 2006. ACM.

[46] Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co.

[47] Steve Harrison and Paul Dourish. Re-place-ing space: the roles of place and space in collaborative systems. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 67–76, New York, NY, USA, 1996. ACM.

[48] Jeffrey Heer and Maneesh Agrawala. Design considerations for collaborative visual analytics. In *VAST '07: Proceedings of the 2007 IEEE Symposium on Visual Analytics Science and Technology*, pages 171–178, Washington, DC, USA, 2007. IEEE Computer Society.

[49] Jeffrey Heer, Fernanda B. Viégas, and Martin Wattenberg. Voyagers and voyeurs: supporting asynchronous collaborative information visualization. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1029–1038, New York, NY, USA, 2007. ACM.

[50] Jeffrey Hightower. From position to place. pages 10–12, 2003.

[51] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, 2001.

[52] Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello. The Location Stack: A layered model for location in ubiquitous computing. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 22, Washington, DC, USA, 2002. IEEE Computer Society.

[53] Ken Hinckley, Jeff Pierce, Mike Sinclair, and Eric Horvitz. Sensing techniques for mobile interaction. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 91–100, New York, NY, USA, 2000. ACM.

[54] Harry Hochheiser and Ben Shneiderman. A dynamic query interface for finding patterns in time series data. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 522–523, New York, NY, USA, 2002. ACM.

[55] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[56] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. Technical report, Cambridge, MA, USA, 1980.

[57] Yijue How and Min yen Kan. Optimizing predictive text entry for short message service on mobile phones. In *in Human Computer Interfaces International (HCII 05). 2005: Las Vegas*, 2005.

[58] Magnus Ingmarsson, David Dinka, and Shumin Zhai. TNT: a numeric keypad based text input method. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 639–646, New York, NY, USA, 2004. ACM.

[59] International Telecommunication Union, Arrangement of Digits, Letters and Symbols on Telephones and Other Devices that can be used for gaining access to a Telephone Network, ITU recommendation E.161, 1993.

[60] Anil K. Jain and David Maltoni. *Handbook of Fingerprint Recognition*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[61] Christina L. James and Kelly M. Reischel. Text input for mobile devices: comparing model prediction to actual performance. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–371, New York, NY, USA, 2001. ACM.

[62] Bernard J. Jansen, Mimi Zhang, Kate Sobel, and Abdur Chowdury. Twitter power: Tweets as electronic word of mouth. *Journal of the American Society for Information Science and Technology*, 60(11):2169–2188, 2009.

[63] Frederick Jelinek. *Statistical methods for speech recognition*. MIT Press, Cambridge, MA, USA, 1997.

[64] Christian S. Jensen and Richard T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11:36–44, 1999.

[65] JSON (JavaScript Object Notation). `http://www.json.org`.

[66] Younghee Jung, Per Persson, and Jan Blom. Dede: design and evaluation of a context-enhanced mobile messaging system. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 351–360, New York, NY, USA, 2005. ACM.

[67] Ken Kahn. ToonTalk–an animated programming environment for children. *Journal of Visual Languages & Computing*, 7(2):197 – 217, 1996.

[68] Matthew Kam, Jingtao Wang, Alastair Iles, Eric Tse, Jane Chiu, Daniel Glaser, Orna Tarshish, and John Canny. Livenotes: a system for cooperative and augmented note-taking in lectures. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 531–540, New York, NY, USA, 2005. ACM.

[69] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 151–162, New York, NY, USA, 2001. ACM.

[70] Per-Ola Kristensson and Shumin Zhai. $SHARK^2$: a large vocabulary shorthand writing system for pen-based computers. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 43–52, New York, NY, USA, 2004. ACM.

[71] Henry. Kucera, W. Nelson Francis, and John B. Carroll. Computational analysis of present-day american english. *Providence, Rhode Island: Brown University Press*, 1967.

[72] Peter M. Kuhn and Kuhn Peter M. *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[73] Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 258–264, New York, NY, USA, 1994. ACM.

[74] Anthony Lamarca, Yatin Chawathe, and el al. Place lab: Device positioning using radio beacons in the wild. In *In Proceedings of the Third International Conference on Pervasive Computing*, pages 116–133. Springer, 2005.

[75] Yang Li, Jason I. Hong, and James A. Landay. Topiary: a tool for prototyping location-enhanced applications. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 217–226, New York, NY, USA, 2004. ACM.

[76] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. *Artif. Intell.*, 171(5-6):311–331, 2007.

[77] Henry Lieberman. *Your Wish is My Command: Programming By Example*. Morgan Kaufmann, 2001.

[78] Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[79] Alan L. Liu and Yang Li. BrickRoad: a light-weight tool for spontaneous design of location-enhanced applications. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 295–298, New York, NY, USA, 2007. ACM.

[80] Pamela J. Ludford, Dan Frankowski, Ken Reily, Kurt Wilms, and Loren Terveen. Because i carry my cell phone anyway: functional location-based reminder applications. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 889–898, New York, NY, USA, 2006. ACM.

[81] I. Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. Letterwise: prefix-based disambiguation for mobile text input. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 111–120, New York, NY, USA, 2001. ACM.

[82] I. Scott MacKenzie and R. William Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17:147–198, 2002.

[83] I. Scott MacKenzie and R. William Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 754–755, New York, NY, USA, 2003. ACM.

[84] I. Scott MacKenzie and Shawn X. Zhang. The design and evaluation of a high-performance soft keyboard. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 25–31, New York, NY, USA, 1999. ACM.

[85] Scott MacKenzie. Mobile text entry using three keys. In *NordiCHI '02: Proceedings of the second Nordic conference on Human-computer interaction*, pages 27–34, New York, NY, USA, 2002. ACM.

[86] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[87] Marklogic Content Server. `http://www.marklogic.com`.

[88] Natalia Marmasse and Chris Schmandt. Location-aware information delivery with commotion. In *HUC '00: Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 157–171, London, UK, 2000. Springer-Verlag.

[89] M. S. Mayzner and M. E. Tresselt. Table of single-letter and digram frequency counts for various word-length and letterposition combinations. *Psychonomic Monograph Supplements*, 1(2):13–32, 1965.

[90] International Telecommunication Union: Worldwide mobile cellular subscribers to reach 4 billion mark late 2008. `http://www.itu.int/newsroom/press_releases/2008/29.html`.

[91] Number of Cell Phones Worldwide Hits 4.6b. `http://www.cbsnews.com/stories/2010/02/15/business/main6209772.shtml`.

[92] Mackenzie's text entry test phrase set. `http://www.yorku.ca/mack/phrases2.txt`.

[93] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Comput. Vis. Image Underst.*, 81(3):231–268, 2001.

[94] Mathias Mohring, Christian Lessig, and Oliver Bimber. Optical tracking and video see-through ar on consumer cell phones. In *In Proc. of Workshop on Virtual and Augmented Reality of the GIFachgruppe AR/VR*, pages 193–204, 2004.

[95] Carman Neustaedter, A. J. Bernheim Brush, and Saul Greenberg. The calendar is crucial: Coordination and awareness through the family calendar. *ACM Trans. Comput.-Hum. Interact.*, 16(1):1–48, 2009.

[96] Antti Oulasvirta, Sakari Tamminen, Virpi Roto, and Jaana Kuorelahti. Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile hci. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 919–928, New York, NY, USA, 2005. ACM.

[97] John K. Ousterhout. Scripting: Higher-level programming for the 21st century. *Computer*, 31:23–30, 1998.

[98] Kurt Partridge, Saurav Chatterjee, Vibha Sazawal, Gaetano Borriello, and Roy Want. Tilttype: accelerometer-supported text entry for very small devices. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 201–204, New York, NY, USA, 2002. ACM.

[99] Andriy Pavlovych and Wolfgang Stuerzlinger. Model for non-expert text entry speed on 12-button phone keypads. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 351–358, New York, NY, USA, 2004. ACM.

[100] Catherine Plaisant, Brett Milash, Anne Rose, Seth Widoff, and Ben Shneiderman. Lifelines: visualizing personal histories. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–227, New York, NY, USA, 1996. ACM.

[101] Plurk. `http://www.plurk.com`.

[102] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[103] Harald Rau and Stevens S. Skiena. Dialing for documents: an experiment in information theory. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 147–155, New York, NY, USA, 1994. ACM.

[104] RealView Compilation Tools for BREW. `http://www.arm.com/products/DevTools/RealViewForBREW.html`.

[105] Jun Rekimoto. Tilting operations for small screen interfaces. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 167–168, New York, NY, USA, 1996. ACM.

[106] Jun Rekimoto and Yuji Ayatsuka. CyberCode: designing augmented reality environments with visual tags. In *DARE '00: Proceedings of DARE 2000 on Designing augmented reality environments*, pages 1–10, New York, NY, USA, 2000. ACM.

[107] RFC 2445 Internet Calendaring and Scheduling Core Object Specification. `http://tools.ietf.org/html/rfc2445`.

[108] Tony Rogers. *Conferences and Conventions*. Butterworth-Heinemann; Second Edition: A global industry, 2008.

[109] Michael Rohs and Philipp Zweifel. A conceptual framework for camera phone-based interaction techniques. In *Pervasive Computing: Third International Conference, PERVASIVE 2005*, pages 171–189, 2005.

[110] Ruby on Rails - open source web development framework. `http://rubyonrails.org`.

[111] SemaCode homepage. `http://semacode.org`.

[112] MacKenzie's text entry test phrase set. `http://www.yorku.ca/mack/phrases2.txt`.

[113] Dennis Shasha and Cathy Lazere. *Out of their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Springer, 1998.

[114] ShotCode homepage. `http://www.shotcode.com`.

[115] SHRIMP Mobile Input Homepage. `http://bid.berkeley.edu/projects/shrimp`.

[116] Miika Silfverberg, I. Scott MacKenzie, and Panu Korhonen. Predicting text entry speed on mobile phones. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 9–16, New York, NY, USA, 2000. ACM.

[117] S. L. Smith and C. N. Goodwin. Alphabetic data entry via the touch-tone pad: A comment, the mitre corporation. *HUMAN FACTORS*, 1971.

[118] Timothy Sohn and Anind Dey. iCAP: an informal tool for interactive prototyping of context-aware applications. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 974–975, New York, NY, USA, 2003. ACM.

[119] R. William Soukoreff and I. Scott MacKenzie. Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard. *Behaviour and Information Technology*, 14:370–379, 1995.

[120] David J. Sturman and David Zeltzer. A survey of glove-based input. *IEEE Comput. Graph. Appl.*, 14(1):30–39, 1994.

[121] Atsushi Sugiura and Yoshiyuki Koseki. A user interface using fingerprint recognition: holding commands and data objects on fingers. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 71–79, New York, NY, USA, 1998. ACM.

[122] Swivel. `http://www.swivel.com`.

[123] T9 Predictive Input. `http://www.nuance.com/t9/`.

[124] Karen P. Tang, Pedram Keyani, James Fogarty, and Jason I. Hong. Putting people in their place: an anonymous and privacy-sensitive approach to collecting sensed data in location-based applications. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 93–102, New York, NY, USA, 2006. ACM.

[125] TeleNav Driving Directions Service. `http://www.telenav.net`.

[126] Feng Tian, Fei Lv, Jingtao Wang, Hongan Wang, Wencan Luo, Matthew Kam, Vidya Setlur, Guozhong Dai, and John Canny. Let's play chinese characters: mobile learning approaches via culturally inspired group games. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 1603–1612, New York, NY, USA, 2010. ACM.

[127] Timeline Web Widgets for Visualizing Temporal Data. http://www.simile-widgets.org/timeline/.

[128] TinyMotion Homepage. http://tinymotion.org.

[129] Edward Tufte. iPhone Interface Design. http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=00036T.

[130] Andries van Dam. Post-wimp user interfaces. *Commun. ACM*, 40(2):63–67, 1997.

[131] Andrew Vardy, John Robinson, and Li-Te Cheng. The wristcam as input device. In *ISWC '99: Proceedings of the 3rd IEEE International Symposium on Wearable Computers*, page 199, Washington, DC, USA, 1999. IEEE Computer Society.

[132] Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. ManyEyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

[133] L. S. Vygotsky.

[134] Jingtao Wang and John Canny. FingerSense: augmenting expressiveness to physical pushing button by fingertip identification. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1267–1270, New York, NY, USA, 2004. ACM.

[135] Jingtao Wang and John Canny. End-user place annotation on mobile devices: a comparative study. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1493–1498, New York, NY, USA, 2006. ACM.

[136] Jingtao Wang and John Canny. TinyMotion: camera phone based interaction methods. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 339–344, New York, NY, USA, 2006. ACM.

[137] Jingtao Wang and Jennifer Mankoff. Theoretical and architectural support for input device adaptation. In *CUU '03: Proceedings of the 2003 conference on Universal usability*, pages 85–92, New York, NY, USA, 2003. ACM.

[138] Jingtao Wang, Danny Soroker, and Chandra Narayanaswami. Event Maps: A collaborative calendaring system for navigating large-scale events. Technical Report RC24971, IBM T.J. Watson Research Center, NY, 2009.

[139] Jingtao Wang, Danny Soroker, and Chandra Narayanaswami. Event Maps: A collaborative calendaring system for navigating large-scale events. In *CHI EA '10: Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 3691–3696, New York, NY, USA, 2010. ACM.

[140] Jingtao Wang, Shumin Zhai, and John Canny. Camera phone based motion sensing: interaction techniques, applications and performance study. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 101–110, New York, NY, USA, 2006. ACM.

[141] Jingtao Wang, Shumin Zhai, and John Canny. SHRIMP: solving collision and out of vocabulary problems in mobile predictive input with motion gesture. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 15–24, New York, NY, USA, 2010. ACM.

[142] Jingtao Wang, Shumin Zhai, and Hui Su. Chinese input with keyboard and eye-tracking: an anatomical study. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 349–356, New York, NY, USA, 2001. ACM.

[143] Taowei David Wang, Catherine Plaisant, Alexander J. Quinn, Roman Stanchak, Shawn Murphy, and Ben Shneiderman. Aligning temporal data by sentinel events: discovering patterns in electronic health records. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 457–466, New York, NY, USA, 2008. ACM.

[144] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[145] Kevin R. Wheeler and Charles C. Jorgensen. Gestures as input: Neuroelectric joysticks and keyboards. *IEEE Pervasive Computing*, 2(2):56–61, 2003.

[146] Daniel Wigdor and Ravin Balakrishnan. TiltText: using tilt for text input to mobile phones. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 81–90, New York, NY, USA, 2003. ACM.

[147] Daniel Wigdor and Ravin Balakrishnan. A comparison of consecutive and concurrent input text entry techniques for mobile phones. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 81–88, New York, NY, USA, 2004. ACM.

[148] Jacob O. Wobbrock, Duen Horng Chau, and Brad A. Myers. An alternative to push, press, and tap-tap-tap: gesturing on an isometric joystick for mobile phone text entry. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 667–676, New York, NY, USA, 2007. ACM.

[149] Jacob O. Wobbrock and Brad A. Myers. Analyzing the input stream for character-level errors in unconstrained text entry evaluations. *ACM Transactions on Computer-Human Interaction*, 13(4):458–489, 2006.

[150] Jacob O. Wobbrock, Brad A. Myers, and Duen Horng Chau. In-stroke word completion. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 333–336, New York, NY, USA, 2006. ACM.

[151] Jacob O. Wobbrock, Brad A. Myers, and John A. Kembel. EdgeWrite: a stylus-based text entry method designed for high accuracy and stability of motion. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 61–70. ACM, 2003.

[152] Ka-Ping Yee. Peephole displays: pen interaction on spatially aware handheld computers. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–8, New York, NY, USA, 2003. ACM.

[153] Shumin Zhai, Michael Hunter, and Barton A. Smith. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 119–128, New York, NY, USA, 2000. ACM.

[154] Shumin Zhai, Michael Hunter, and Barton A. Smith. Performance optimization of virtual keyboards. *HUMAN-COMPUTER INTERACTION*, 17:89–129, 2002.

[155] Shumin Zhai and Per-Ola Kristensson. Shorthand writing on stylus keyboard. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 97–104, New York, NY, USA, 2003. ACM.

[156] Zhengyou Zhang, Ying Wu, Ying Shan, and Steven Shafer. Visual panel: virtual mouse, keyboard and 3d controller with an ordinary piece of paper. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces*, pages 1–8, New York, NY, USA, 2001. ACM.