Using Haptics to Address Mobile Interaction Design Challenges

Protoyping and User Evaluation with a Handheld Tactile Display

by

Joseph Kurachi Luk

B.S., Cognitive Science, University of California – San Diego, 1999

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia
July 2006

Abstract

Current user interfaces for mobile and handheld computing platforms principally offer user interaction through the visual and auditory modalities. However, mobile devices are often used in contexts where vision and hearing are impaired. At the same time, more and more functionality is being layered upon mobile devices, while the physical size of the display and keypad has remained small. This limits the rate of information that can be exchanged between the user and the system, and poses an interaction design challenge. Haptics offers a potential solution by providing an additional modality that is also especially well-suited to the demands of portable, personal devices that are in contact with the user's skin.

In this work we identify ways that interaction through the sense of touch can enhance mobile user interfaces. We describe the synergistic process of design of user interaction concepts and novel handheld tactile display hardware based on the principle of piezoelectric actuated lateral skin stretch. Following the realization of the prototype hardware, we performed perceptual characterization studies to determine the expressive capabilities of the new device in the hands of a human user. Informed by the results from the initial user studies, we built and tested a handheld browser application with tactile enhancement. The results of user testing with the browser

application suggest that the current implementation of directional tactile stimulation alone is not sufficient to enhance performance (task time) in spatial navigation; however, the user study also brought to light some encouraging qualitative feedback and ways to improve the interaction design and haptic feedback.

By conducting a full iteration of a user-centred design process in haptics, we have provided a case study to inform future development efforts, a flexible platform for prototyping, and an indication of promising future directions for using haptics to solve mobile interaction design challenges.

Joseph Kurachi Luk

Contents

List of Tables

List of Figures

Glossary

haptic icon Also known as a *hapticon* or *tacton* [10], a haptic icon is a brief, distinctive signal delivered to the user through the haptic apparatus, or its software representation.

High fidelity prototype As compared to *low-fidelity prototyping*, a methodology for expediting the involvement of users in an interaction design process by shifting the development effort from detailed implementation of features to early usability testing. In contrast, a high-fidelity prototype is one which is relatively close to the final product in terms of functionality and the level of interactivity which is supported.

piezo, piezo actuator Refers to an individual piezoelectric bending motor element.

position control, or pControl In this mode, the scrolling motion of the cursor or page follows the slider position directly. This type of control is also used in jog dial or mouse wheel controls.

slider In this document, refers to the mechanism that allows the tactile display to be moved up and down.

stretch image A graphical representation of the amount of gap between adjacent piezo actuators at a given point in time. Assuming the user's

finger has been in constant contact with the TD since prior to the application of signal, the levels depicted in each location of the stretch image correspond to the amount of relative skin displacement ("skin stretch") at that location on the TD. Described further in section 4.9.

subpixel rendering As used in this document, refers to the technique of rendering graphics into an offscreen buffer with higher resolution than the physical resolution of the display, then downsampling to the display resolution using anti-aliasing filters. For graphical displays, this increases the effectively usable resolution at the cost of some sharpness. In this document, this term does not refer to the technique of using an LCD's red-green-blue subpixels to increase resolution.

subtaxel rendering Equivalent to subpixel rendering for a tactile display.

tactile flow The perception of the movement of a tactile stimulus across the skin over time. See Section 4.8.

tactile window The portion of the page map that is currently being rendered to the TD. See Section 6.4.1.

taxel Just as a pixel is a logical element of a multi-element graphical display, a taxel is a logical element of a multi-element tactile display. In the case of the device prototype discussed in this thesis, a taxel represents the voltage delivered to a single piezo actuator. The TD described in this thesis therefore represents eight taxels.

TD Tactile display.

tweening In animation, refers to the creation of motion by adding intermediate frames that depict a gradual progression from one state to

another. For the tactile display described in this thesis, "tweening" refers to the successive displacement of a pattern of piezo activation (voltage levels) across adjacent piezo actuators, to create the sensation of *tactile flow.*

velocity control, or vControl In this mode, the slider position is mapped to the scrolling velocity. Therefore, as long as the user keeps the slider in one place, the cursor or page moves with constant velocity. This type of control is also used in joysticks and shuttle controls.

voltage image (volt image)[1] A graphical representation of the voltage applied across the array of piezo actuators at a given point in time, as described in section 4.9.

voltage image (volt image)[2] An object in the STReSS library API which contains data in the form of voltage levels.

XML Extensible Markup Language. A generic markup language, standardized by the W3C (World Wide Web Consortium). In the current project, extensible documents conforming to the XML syntax are used for various purposes including representing tactile stimuli.

XPCOM Cross Platform Component Object Model. Part of the open-source Mozilla application platform. Used in the tactile enhanced HTML browser (Chapter 6) for inter-process communications.

XUL XML User Interface Language. A markup language for specifying the visual user interface components in the Mozilla application platform. In the current project, XUL is used to customize the Mozilla browser for use as a handheld application.

Acknowledgements

I would like to acknowledge the guidance of my supervisor, Dr. Karon MacLean, who was always a champion for her students, working tirelessly to encourage us to do quality, meaningful research, and to overcome our personal challenges, perfectionism included.

Much of the research described in this thesis was the result of an exceptional collaboration between laboratories at the University of British Columbia and McGill University in Montréal. In particular, Jérôme Pasquero and I participated in an extended exchange which resulted in numerous learning experiences since we each brought a different, sometimes opposite, point of view to the research problems. I am grateful to Jérôme and other members of the McGill Haptics Lab, especially Professor Vincent Hayward, Vincent Lévesque, and Qi Wang, for their patience in engaging me in endless scholarly debates, their warm hospitality, and their dedication to the project. The contributions of Don Pavlasek and Jozsef Boka to the mechanical design and fabrication are also sincerely appreciated.

The members of my thesis committee, Dr. Rodger Lea, Dr. Ron Rensink, and especially Dr. Steven Wolfman, gave valuable feedback on this research. Finally, I would like to express my application to Mario Enriquez for help with the MDS testing methodology, and to Steve Yohanan for sharing his operating system expertise.

Chapter 1

Introduction

Mobile, portable, and handheld computing environments offer significant promise for the future of interactive computation. Compared to a typical desktop computer, a mobile device enables opportunistic uses for information that stretch the boundaries of the popular definition of data: instead of working with documents and files, mobile devices are called upon to manage an increasingly varied collection of data, from voice calls to photos and location based information snippets. The increasing popularity of mobile devices also represents a continued shift away from the physical instantiation of data and towards an increasingly miniaturized and networked medium. Information that was once tangible, in the form of paper, magnetic tapes or other media, is now invisibly contained within the device used to access it or simply delivered over ubiquitous, wireless networks.

The dissociation of information from its physical encapsulation is driven by pressures to improve portability by reducing device weight and volume. However, interactive applications can never be completely removed from physical constraints due to the necessity of producing a human sensory percept.[1] Furthermore, the history of computer user interfaces shows that

[1] Brain-computer interfaces enable the possibility of a completely non-physical user interface, but are too poorly understood at the present time to consider them a viable alternative in mobile device interaction.

significant progress in expanding the breadth of accessibility to computers can be achieved when interfaces, such as the graphical user interface (GUI), leverage the intuitive physics [38] that evolved in human beings as a means for dealing with the environment.

The design of effective mobile information devices is thus confronted with a classic dilemma: balancing the power of abstraction and freedom from physical constraints against the usability advantages of a system that interacts effectively with human beings on a natural physical scale. All too often, increased mobility comes at the cost of decreased usability.

1.1 Problems with Contemporary Mobile Interaction Design

The user interfaces of contemporary mobile devices suffer from *sensory bandwidth limitation* and *environmental competition for visual and auditory attentional resources.* Artificial haptics, incorporating synthetic tactile signalling, may be a useful tool for solving these problems.

1.1.1 Sensory Bandwidth Limitation

Throughout most of the 1990s, there was a steady decrease in the size (volume and weight) of personal mobile information devices such as mobile phones, personal digital assistants (PDAs), media players and laptop computers, spurred by steady technological progress in integration of electronic circuits, portable power, and digital wireless transmission. However, the trend towards miniaturization has since leveled off in many categories of devices, despite continued progress in hardware integration and anecdotal

evidence from users that the portability of ever smaller devices is highly desirable. The conventional wisdom is that input and output devices such as keypads and screens can not be made smaller without negatively impacting usability. Thus the factors limiting device portability have ceased to be technical in nature; instead, user interface considerations now dictate the practical limits to miniaturization of today's mobile devices.

For the increasingly complex tasks being performed on mobile devices, the information capacity of the small slice of the visual field covered by a small LCD display has become a limitation. Various head-mounted displays such as eyepieces and goggles have been developed to increase the effective display area while retaining portability, but thus far they have not proven widely practical because of their intrusiveness.

Similarly, most mobile information devices today include an audible transducer and microphone input, but aside from making phone calls and limited command-based voice recognition, information-rich auditory user interfaces have yet to be demonstrated on mobile devices.

The use of multiple simultaneous sensory modalities, known as *multimodal* interaction, offers a method to increase the available *user interface bandwidth*, or information volume per unit time that can be exchanged between user and device.[12] With visual and auditory modalities already receiving widespread research interest, and taste and smell being thus far relatively impractical as user interface modalities, the domain of touch appears to be a useful area for study as an additional modality to enrich the information carrying capacity of the user interface of a mobile device.

An abstract example of an interaction design problem created by limited sensory bandwidth in mobile devices is shown in Figure 1.1. For a given interface with a number of functions, "porting" the application to a mobile

device from a more conventional personal computer imposes limitations in the amount of information that can be practically presented at any one level of navigation hierarchy. Designers wishing to preserve functionality are therefore forced to implement "deeper" navigation hierarchies requiring more user interaction to access the desired function. The increased navigation overhead, in turn, decreases the practicality of the application in the hands of a busy, mobile user. While this example may seem simplistic, it is a relatively accurate description of the problems with the first generation of mobile phone applications based on the wireless application protocol (WAP).

1.1.2 Environmental Competition for Visual and Auditory Attentional Resources

Haptics may be especially useful for multimodal interaction in a mobile use context because it does not share the action-at-a-distance property of visual and auditory signalling, and is a more "parallel" sensory modality than vision and hearing.

When moving through the environment, a person typically uses visual and auditory modalities for navigation, avoiding obstacles, and learning about the environment. Any system that requires those modalities for user interaction must compete for the user's attention with information originating from the environment.

Competition for limited visual attention can present serious safety issues: for example, whereas use of a mobile device while walking on a crowded street might be marginally acceptable, use while driving is becoming an increasing public safety concern, and use while landing an aircraft might be

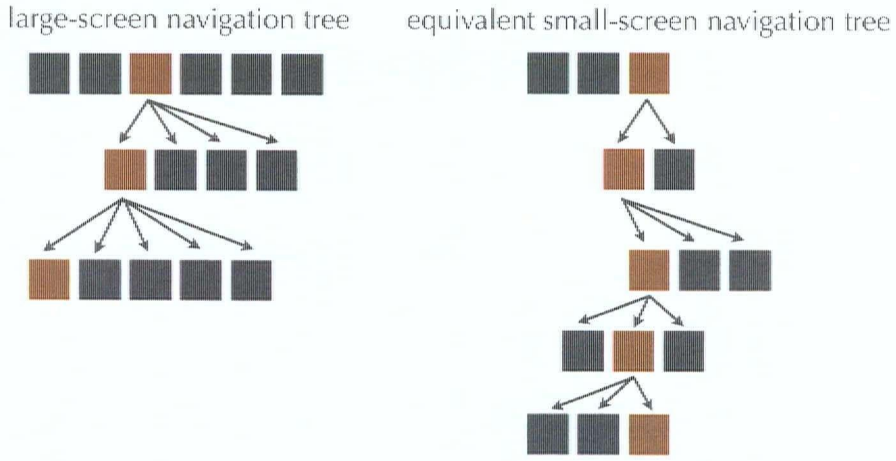


Figure 1.1: Examples of deep versus wide navigation trees. The limited sensory bandwidth available at any level of the hierarchy forces a "deep" configuration, which requires more effort to navigate.

considered completely unacceptable [49]. In practice, the user is only able to devote a portion of their visual attention to a mobile device, and repeatedly shifts their attention between device and environment [40]. It follows that a mobile user interface can not be guaranteed to deliver a visual cue to the user unless the information is retained onscreen long enough for the user to direct their attention to the display. This reduces the interface bandwidth relative to desktop computers even further than a simple comparison of screen size would suggest.

Auditory interfaces suffer from similar limitations. A very noisy environment such as a crowded street presents challenges in making the sounds produced by the device audible, as well as recognizing sounds produced by the user. Auditory interfaces therefore scale poorly to large numbers of densely packed users. On the other hand, in a very quiet environment, such as a meeting or library, use of an auditory interface might be completely unacceptable because it would disturb others. Headphones and microphones that isolate exterior noise improve user-device communication, but only at the expense of user-environment communication, causing similar concerns as visual impairment.

In comparison to vision or hearing, the sense of touch is relatively unobstructed in a mobile use context. The vestibular system is used for balance, and the feet or hands may be in contact with control surfaces, but other areas of skin are likely to be available for interaction. Unlike vision or hearing, which must be sensed through discrete organs, touch receptors are distributed throughout the body and active simultaneously — thus, the system can be characterized as more "parallel" in nature.

Tactile signals are only active at the point of direct contact, enabling discreet interaction. In addition, a portable device is also in contact with

the user's body more often than a desktop computer, making it a good platform for tactile interaction.

Indeed, a non-multimodal, *purely* haptic interface is conceivable and is currently used in limited contexts such as vibrotactile signalling of incoming calls. However, haptics can not be considered a generic replacement for visual or auditory interfaces, since each modality has its own profile of applications for which it is optimally suited. Work is ongoing to define novel application spaces for haptics, including affective, subconscious, and interpersonal interfaces. Within the scope of this thesis work, however, the potential benefits of haptic *augmentation* to existing mobile applications with demonstrated, proven utility will be explored. This will enable effort to be focused on implementing a high-fidelity prototype, including hardware development, application conceptualization, and usability testing, completing a full iteration of an interaction design cycle within the scope of a master's thesis.

1.2 Thesis Research Questions

The goal of this thesis research is to increase the understanding of the application of haptics to a mobile computing context. The following research questions are addressed:

1. *What are the problems with existing mobile user interfaces that may be addressed using haptics?*

2. *How can one implement haptics on a mobile device despite power, size, and weight restrictions?*

3. *What are the expressive capabilities of the hardware prototype in the hands of a human user?*

4. *What are the engineering challenges associated with building a high-fidelity hardware and software prototype with handheld use in mind?*

5. *Is* tactile flow *an effective aid for user navigation?*

6. *How can user-centred design methodology be applied to haptics, where hardware technology is still the primary determinant of user interface capabilities?*

These questions are revisited in the Conclusion (Chapter 8) which summarizes the research findings.

1.3 Thesis Approach and Overview

The approach that was used to address the research questions identified in the previous section was to create a handheld haptics system in a user-centred, iterative fashion by incorporating user input as early as possible in the design process. The stages of this process are shown in Figure 1.2. Because the aim of the thesis research was to use this process to better understand the potential for haptics in a mobile user interface, rather than to produce a fully engineered product, priority was given to creating a handheld haptic user experience over a fullly-functional device; if its usefulness could be demonstrated, then future work could address the engineering challenges related to true mobility, such as battery power and custom integrated circuits. Therefore, early in the design process the heuristic was adopted that the prototype should attempt to replicate as much of the handheld user experience as practical within the resources available for the research, while

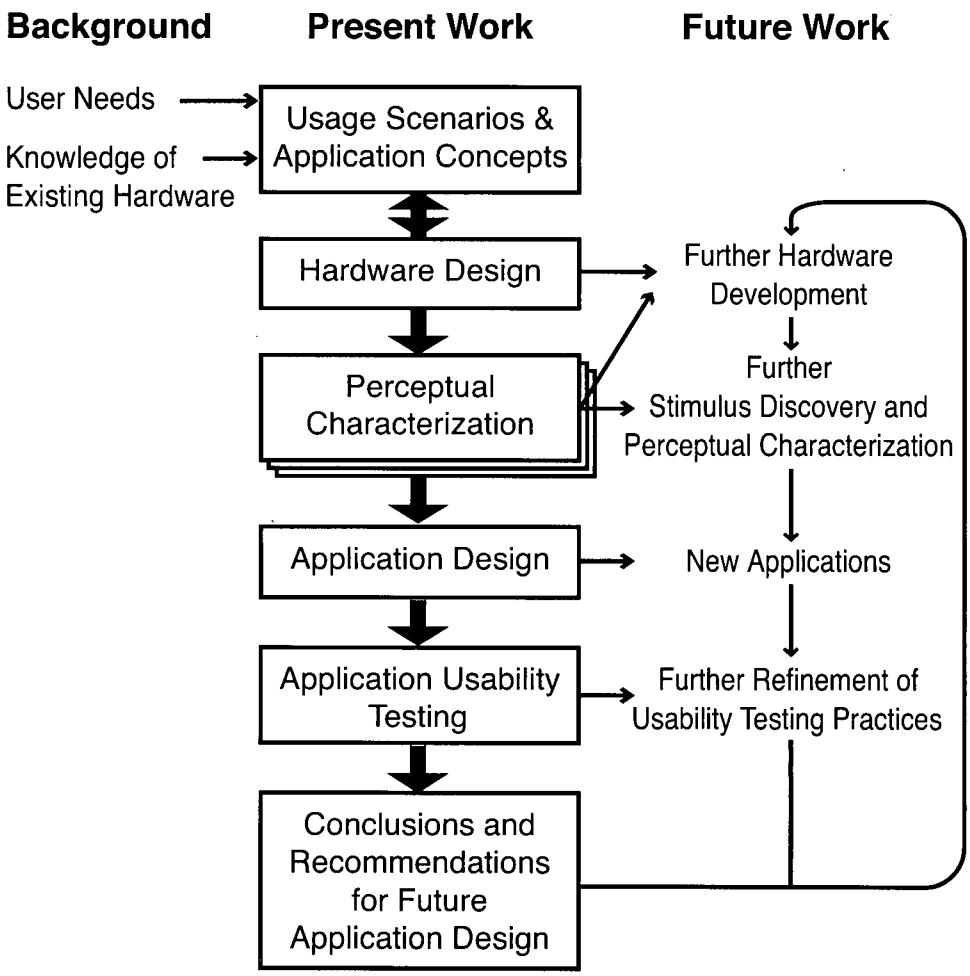


Figure 1.2: Overview of the design process used for this thesis research.

remaining tethered to a host PC and easily customizable; however, it should have a power, weight, and volume profile such that future integration into a wireless mobile device could be *conceivable* using known engineering techniques. Further details of this design decision are provided in Chapter 4, Hardware.

Rather than specialized haptic-only applications or, applications for users with visual or auditory impairment, we have chosen to focus on the general area of haptic augmentation of applications similar to those currently employed on mobile devices, because there is a clearer understanding of user needs and an opportunity to contribute to ongoing mobile development programmes. Therefore, for the purpose of the studies described in this thesis, the users are a general audience.

The hardware development and perceptual characterization stages were performed in collaboration with Professor Vincent Hayward, and graduate students Jerome Pasquero, Vincent Levesque, and Qi Wang of the McGill University Haptics Laboratory; and undergraduate research student Shannon Little of UBC. The contributions of each collaborator are noted in appropriate sections throughout the thesis.

This thesis is organized into the following sections:

Chapter 1 - Introduction describes the problem that is addressed in this thesis, and the motivation for the research.

Chapter 2 - Related Work discusses the existing research in mobile and haptic interfaces.

Chapter 3 - Design Concepts describes the initial conceptual design work that was performed to better understand the research domain

and to determine the appropriate application and hardware specifications for further development.

Chapter 4 - Hardware Prototype Development covers the engineering process that was performed to create a handheld haptics platform with an integrated piezoelectric tactile display. In addition to electromechanical design, this chapter also includes a consideration of the method for designing and representing tactile outputs, and a description of the control software architecture.

Chapter 5 - Perceptual Characterization describes user experiments conducted *before* detailed application development, to understand the capabilities and limitations of the device in producing a salient tactile percept.

Chapter 6 - Browser Prototype describes an in-depth, high-fidelity prototype of a mobile browsing application that was built using the handheld haptics platform. This chapter also includes the results of an initial exploratory user test, and hardware customizations for the browsing application.

Chapter 7 - Browser User Test presents a controlled user study designed to assess the performance characteristics of the haptically enhanced browsing application. New software created to manage the usability test, and a distraction task model, are also described.

Chapter 8 - Conclusion returns to the research questions identified in the previous section and considers how they were addressed through the activities conducted for this thesis research.

Chapter 2

Related Work

In this chapter we describe the previous research that informs the present study as it concerns mobile interface design challenges, haptic augmentation for multimodal enhancement, handheld haptic technologies, and perceptual evaluation of haptic devices. The decisions to focus on haptic augmentation for general mobile devices, to use piezoelectric lateral skin-stretch technology, and to use the perceptual characterization methodology are justified based on the existing knowledge in the aforementioned domains.

2.1 Mobile Interface Design Challenges

The hypothesis that limited sensory bandwidth due to small screens and keypads restricts mobile interface designs is based largely on widespread anecdotal evidence and the experience of the author in designing mobile interfaces professionally. As mobile devices are a field of high commercial interest and development, many usability studies are only published privately within companies [53]. At the same time, every research paper relating to mobile interface design begins with a description of the challenges posed by limited user interface bandwidth, resulting in the need for reduced and restructured content relative to desktop interfaces [6]. The need for more expressive interfaces that use physical, tactile affordances to support a con-

stellation of computing devices was articulated in the famous Tangible Bits paper by Ishii and Ullmer [24].

Limited visual attention in mobile use contexts has been described in [40]. By following mobile users in realistic usage environments, Oulasvirta et al. observed rapid and frequent visual attention switching between device and environment as users attempted to navigate while using a mobile browser. Based on broad field evidence, a related paper by Roto and Oulasvirta [47], suggests that non-visual, and particularly haptic, interfaces are needed in mobile applications to overcome the problems associated with loss of visual attention during system wait states.

2.2 Haptic Augmentation for Multimodal Enhancement

Given the attention-demanding nature of tasks performed on a handheld device in a mobile environment, studies of multimodal enhancement of interfaces under workload are relevant. In particular, Chan et al. [12] demonstrated that haptic signals could be learned and recognized despite distraction that placed a demand on the users' attentional resources. Tan et al. [50] demonstrated that performance in an attention-demanding vehicular navigation task could me improved with haptic feedback applied to a person's back.

In a navigation task, Dennerlein [17] demonstrated that haptic force-feedback could be used to improve targeting performance. Similarly, a patent by Novint Technologies [7] covers arbitrary force feedback produced when the user pushes "into" a screen boundary, for non-visual feedback of scrolling. *Directional stimulation* is a key component of these applications,

and is achieved in existing systems either through force feedback or spatially distributed vibrotactile stimulation [50]. Using multiple actuators to provide a directional signal provides significant increased utility over simple vibration ([27, 31] and Immersion Corp. patent [9]).

These examples support the feasibility of the application scenarios that are proposed later, including the use of directional and tactile feedback (but not through spatial patterns of vibration) to potentially enhance targeting and spatial navigation in an interface.

2.3 Handheld Haptics

While there is promise for the use of haptics on a mobile device, there are relatively few examples of functioning implementations. Some underlying difficulties are listed below.[1]

- Lack of mechanical grounding. Applying force-feedback to a user requires a fixed mechanical ground. In a mobile context, the forces must be created relative to the user, which imposes constraints on the physical design and force output capabilities. An alternative is tactile display, which generates no net force on the user, but consequently limits the scale of sensations transmitted.

- Stringent power, size, and weight constraints apply in mobile contexts. Use of a conventional motor for force-feedback introduces a significant impact on all three. [44]

- Since relatively few instances of integrated, rich haptic feedback exist today, it is difficult to justify inclusion in a mobile device until there

[1] *Portions of this section have been previously published in [34].*

is a better understanding of the added value it creates for the user.

The most common occurrence of haptic feedback in mobile devices today is the ubiquitous mobile phone or pager vibrator. Patterns of vibration are typically used to indicate various alerts, such as an alarm or incoming call. Recently there has also been commercial and research interest in putting vibration in more sophisticated applications [13, 14, 32]. Generally, vibrotactile stimuli are produced globally (across the entire device) and with only two levels (on or off), vibrotactile devices generally do not afford bidirectional interaction in the sense of the user actively exploring information through movement and the sense of touch [23, 48].

Devices that are capable of delivering grounded forces to the user have the potential for greater expressive capacity than vibration. Designs are restricted to minimal degrees of freedom (DoF) [35], yet must create enough added value to justify the power, size, and weight tradeoffs.

Piezoelectric actuation offers significant promise for mobile applications because it can achieve a smaller form factor without coils and magnets. Poupyrev et al. of Sony's Computer Science Laboratories used piezo elements to produce vibrotactile actuation of handheld devices or parts of them [46]. In the case of a touch screen [45], the user typically experiences the illusion of local actuation although the entire screen moves; this type of vibrotactile actuation of a flat surface is also mentioned in a patent by New Transducers Limited [8]. Creating true multiple loci of actuation on a small scale is significantly more complicated using vibrotactile signals [46].

Piezoelectric actuators may be configured in a way that also produces non-vibrotactile, low-frequency skin stimulation [23]. When the user places his/her finger on an array of actuators which collectively comprise a multi-

element tactile display, the relative motion of the individual piezo tips stretches the skin locally, activating skin mechanoreceptors. Applying specific patterns of distributed skin deformation can create the illusion of touching small-scale shapes and textures. A device based on this technology, called the Virtual Braille Display (VBD) [30], has been used to render legible Braille dots using only lateral stretching of the skin.

Similar sensations can be achieved using technologies that push *into* the skin [51], but the lateral skin-stretch configuration is mechanically simpler and makes the most efficient use of the range of motion of commercially available piezoelectric bending motors [15], resulting in favourable power, size, and weight profiles. Such a configuration also provides internal mechanical grounding, as forces are generated between adjacent piezo elements.

We thus eventually chose lateral skin-stretch as the most promising configuration for the hardware prototype. Our approach uses the same basic principle as the VBD, but miniaturized and embedded in a handheld form factor wherein the skin-stretch site is displayed to the users thumb, and mounted on a slider. The device is described in further detail later.

2.4 Perceptual Evaluation of Haptic Devices

When a novel haptic transducer is created, prior to the development of full-blown applications it is first helpful to understand the perceptual characteristics of its output, so that signals can be created that best use the device's expressive capabilities. The perceptual evaluation procedure described in Chapter 5 is based on work described by Enriquez and MacLean [18], using the multidimensional scaling (MDS) technique to categorize haptic icons, and the subgroup MDS technique has been further analyzed in [42]. The

procedure used in this thesis for designing the haptic signals also loosely follows the work by Brewster and Brown on tactons [10, 11].

Chapter 3

Design Concepts

This section describes application concept development that followed from the background research into the problem domain and existing haptic approaches.[1]

Design concept development proceeded in stages over the course of a year; the existence of the piezoelectrically actuated lateral skin stretch technology was not known until after initial hardware and application concept development had been done. Instead of creating applications for a particular hardware specification (*technology*-centred design), we sought to leave the hardware specification as open-ended as possible while focusing our efforts on understanding how different application designs could meet user needs. Of course, no application, especially haptic, can exist in a vacuum with *no* dependency on hardware. Therefore, during application concept development we also gave consideration to the *minimum* set of hardware features necessary to make the specified interaction possible. For the present project, we also had to keep in mind several practical considerations such as access to manufacturing facilities and setting an appropriate scope for a master's thesis project. In these ways, the interaction design and hardware design

[1]Sections 3.3.1 through 3.3.4 have been previously published in [34] and represent the collaborative work of authors on that paper. Figures 3.6, 3.8, 3.9, and Section 3.3.5 have not been previously published.

evolved synergistically, with the interaction needs being the primary driver.

In this chapter we describe the maturation of design concepts from an open-ended concept of simulating the haptic experience of reading a book, to a slightly more tightly specified concept of 1-D navigation with haptic feedback. Conceptual prototypes, including low-fidelity physical mockups and detailed usage scenarios, were used to drive the design process by elucidating details of the user interaction. Finally, we converged on a design based on 1-D interaction model, with piezoelectric lateral skin-stretch technology used to enable the application concepts.

3.1 Electronic Book Reader with Vibrotactile Feedback

An early design effort focused on haptic augmentation of the experience of using a mobile, handheld electronic book reader. The promise of a portable electronic device for reading information has been long recognized [26], but adoption of electronic readers has been hampered by poor display quality, weight and power, and slow navigation compared to traditional books. With the display quality, weight and power considerations being slowly mitigated due to technological advances, what remains is the user interface problem. Flipping through a paper book is a rich haptic experience, transmitting information such as the relative location of a page in the book, the speed of movement through pages, and even the usage history (books tend to naturally flip open to well worn sections).

We explored the concept of artificial haptic signalling to support book navigation by constructing the prototype shown in Figure 3.1. A vibrotactile transducer (a small speaker) was mounted on a rubber block that was shaped

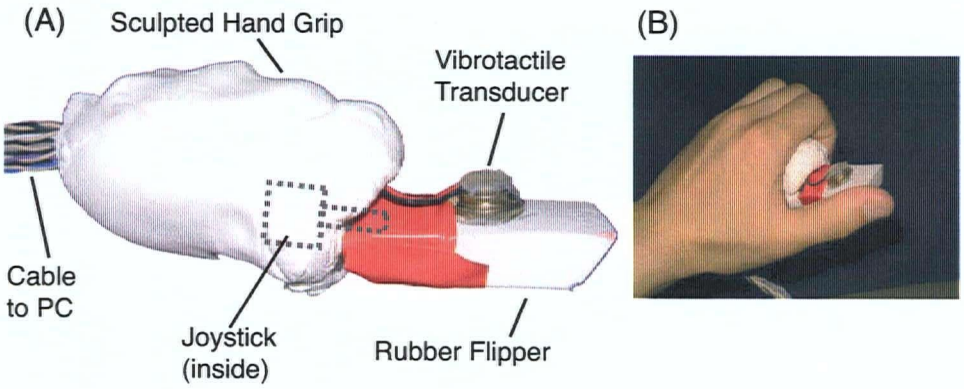


Figure 3.1: (A) Device for providing vibrotactile feedback to simulate the experience of page-flipping. The rubber flipper is mounted on a mini-joystick and can articulate in the up/down direction with spring return to centre. (B) The device being held in the hand. Note that the rubber edge meets the thumb at an angle similar to that of the edge of a book when pages are being flipped.

to simulate the angle of a book's edge when the device was held in the hand. The rubber "flipper" was mounted on a spring-loaded miniature joystick found in a typical PC game controller, and its motion restricted by a custom-machined plate so that the flipper could only move up and down. Finally, the apparatus was enclosed in a soft formed foam hand-grip (Crayola Model Magic).

The concept was that the user would use his/her thumb to tilt the rubber flipper up and down, producing a velocity signal which, relayed through the off-the-shelf USB game controller, would cause repeated page-up or page-down signals to be sent to electronic book reader software. Feedback would be provided for each page-turning event in the form of a perceptible click caused by the vibrotactile transducer, allowing the user to feel their movement through the pages of an electronic book. Finally, well worn pages would be simulated by a combination of stronger clicks accompanied by appropriate delays in the navigation model.

This concept was a first-iteration rapid prototype intended to study the ergonomic experience of a device with potential for integration into a mobile / handheld environment. The speaker itself did not generate sufficient force for a tactually perceptible click. Although it could have been upgraded with a voice coil or similar device with stronger force, attention shifted to a more versatile 1-D scrolling strip concept that could incorporate more expressive feedback.

3.2 Early 1-D Navigation Concepts

In a typical feature-packed mobile device, much of the chassis that faces the user (the $X - Y$ plane) is covered with input and output transducers

such as keypads, displays, and speakers. Furthermore, there is pressure to minimize the *thickness* of such devices in the Z direction, in order to maintain a form factor that fits unobtrusively against the body in a pocket or handbag. These factors make it difficult to incorporate a haptic device on the front or back surface of a mobile device. More accessible locations are the *side* surfaces, which are both relatively unused and also somewhat mitigates concerns about the physical depth of a haptic transducer, which may extend somewhat inside the case without significantly altering the typical aspect ratio of a handheld device.

In a typical box-shaped handheld device, this location is accessible to the user's thumb, which sweeps out an arc that can be used for 1-degree of freedom, linear input. Recently, several commercial products have been introduced that utilize capacitive touch sensors to digitize 1-D input (Figure 3.2). This provided the inspiration to use the side surface of a device as a bidirectional haptic transducer.

A conceptual schematic of a bidirectional touch input / tactile output system is shown in Figure 3.3. The tactile transducer component is assumed to be of the type that produces physical deformations perpendicular to the surface of the finger, and could be implemented using any of a variety of technologies. The following technologies were considered as candidates for the tactile transducer:

- **Electromagnetically actuated pins** produce local deflection and and form the basis for conventional Braille displays. With an appropriately chosen flexible covering acting as a low-pass filter, they can also be used to render smooth shapes. Investigation was made into using the mechanism from a dot-matrix printhead to create minia-

Figure 3.2: Examples of existing products utilizing linear capacitive touch sensors.

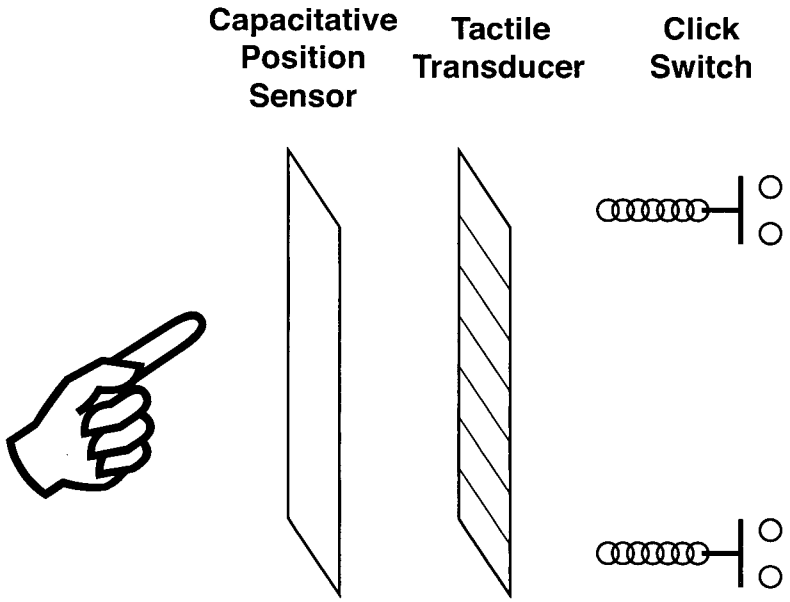


Figure 3.3: Schematic diagram of a concept bidirectional haptic device for 1-D linear input and tactile output. A flexible sensor for determining the position of the user's finger is layered on top of a deformation-producing tactile output transducer, and mounted on a movable apparatus so that the force of the user's finger pressing *into* the device can be detected for actions such as selection. (Components not drawn to scale)

ture, high-resolution deflections. Unfortunately, the drawbacks to electromechanical actuation are high power consumption and the weight of the coils.

- Just as piezoelectric speakers can be used as alternatives to electromagnetic speakers, a flat, segmented array of **piezo elements** could produce the tactile deflections. However, conventional piezo elements are made for producing audio, which has both higher frequency and lower amplitude than required for producing tactile sensations beyond vibration, which is poorly suited to dense spatial signalling.

- Small quantities of special **magneto- or electro-rheostatic fluids** may be configured as an array of separated cells. When a current or magnetic field is applied to a cell, the encapsulated liquid hardens, creating a tactile sensation when the user explores the array. Concerns with this approach include long-term reliability of the fluids and membrane, exotic micro-machining requirements, response time, crosstalk between cells (especially with the magnetorheostatic approach), and potentially limited amplitude.

- **Shape memory alloys (SMAs)** may be used in any number of configurations, such as bimorph benders, to produce local deflections. However, because the motion is temperature-related, crosstalk, heat-sinking, and especially reaction time remain significant concerns for a spatial array configuration.

Other considerations relate to the position sensing layer, which must be flexible if used on top of a deflection producing layer. Depending on the actuator technology, it must also be able to cope with magnetic and/or elec-

tric fields, which would cause problems for existing capacitive touch sensors. This could potentially be cancelled out in software, though at a cost of effective resolution. An alternative configuration consists of putting the sensing layer underneath (i.e., separated from the user's finger by) the actuation layer, which could eliminate the flexibility requirement but potentially exacerbate the interference problem. For the purposes of rapid prototyping for experimentation, preparations were made to mount a position sensing coil on the user's finger and to use a distally mounted drawing tablet (which works by radio frequency sensing at a distance of up to a few centimeters).

Leaving the specific choice of actuator and sensor technology open for the moment, interaction concepts were developed; an example is shown in Figure 3.4. Assuming a device that can produce localized orthogonal deflection while tracking the user's finger position with reasonable time accuracy, force-feedback may be simulated without resort to motors that physically push against the user's finger in the lateral direction. The concept is modeled on the sport of surfing, where a continuously moving ramp provides a lateral force on the rider due to the action of gravity and slippage along the surface. Similarly, a ramp that tracks the user's finger and provides continuous "uphill" resistance may be used to provide force feedback for a variety of interaction designs such as those described in [17] that improve steering and navigation.

3.3 Application Concepts

Under the working assumption of a linear, 1-degree of freedom bidirectional haptic device capable of delivering several distinguishable tactile stimuli combined with finger position sensing, several application concepts were

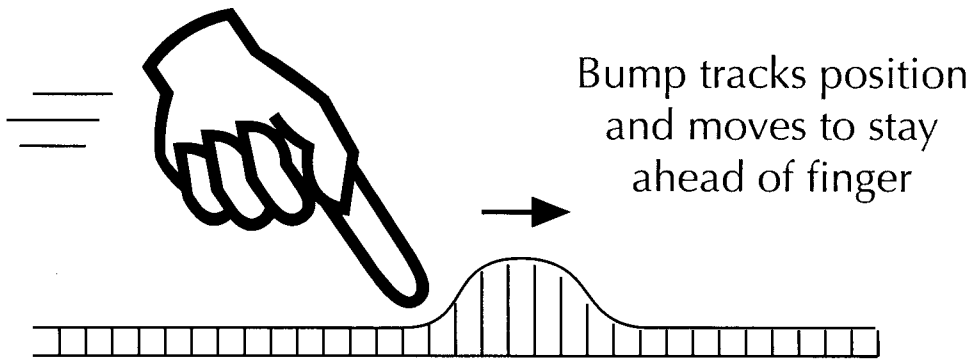


Figure 3.4: Simulated force feedback using a moving bump that tracks the user's finger, providing continuous "uphill" resistance.

developed to better understand how such a device might be useful in mobile interaction design. For ease of understanding each concept, a contextual scenario with a fictitious example user is used to motivate the application.

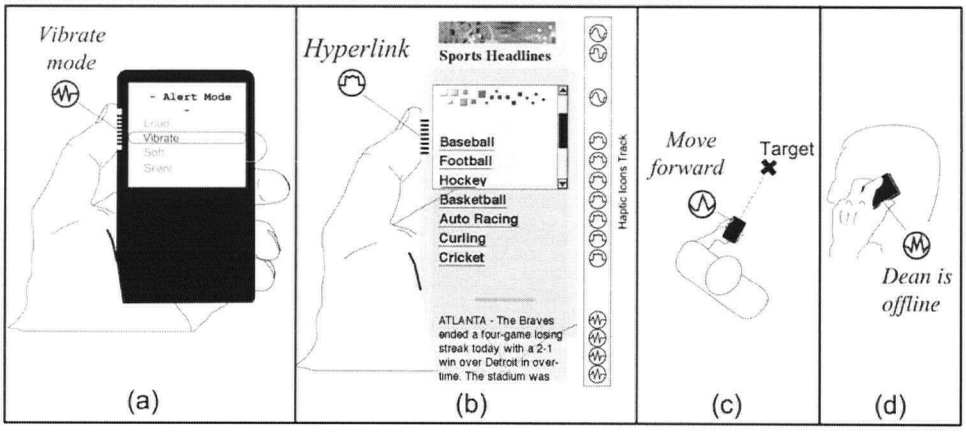


Figure 3.5: Overview of four application design scenarios. (a) List selection, (b) Scrolling, (c) Direction signalling, (d) Background status notification. The figures shown as callouts represent haptic icons. Figure by Jerome Pasquero and Joseph Luk.

3.3.1 List selection: Ringer mode application

(Figure 3.5, **a**, and Figure 3.6) *Linda is in a meeting and wants to set her phones ringer mode discreetly. Grasping her phone inside her purse, she explores the ringer mode menu by moving the selection highlight while receiving tactile feedback. Each menu item feels unique, like touching objects with different shape and texture, and she recognizes the sensation of the silent menu item because she has used this function before. She selects the silent mode and receives tactile feedback as confirmation.*

The scenario illustrates one way we can employ haptic icons [18], or

tactons — [10] brief, artificial tactile stimuli to provide assistance in making selections. A unique tactile stimulus is assigned to each item in a list menu; with repeated feedback, users quickly associate functional meanings to abstract haptic icons [11, 12].

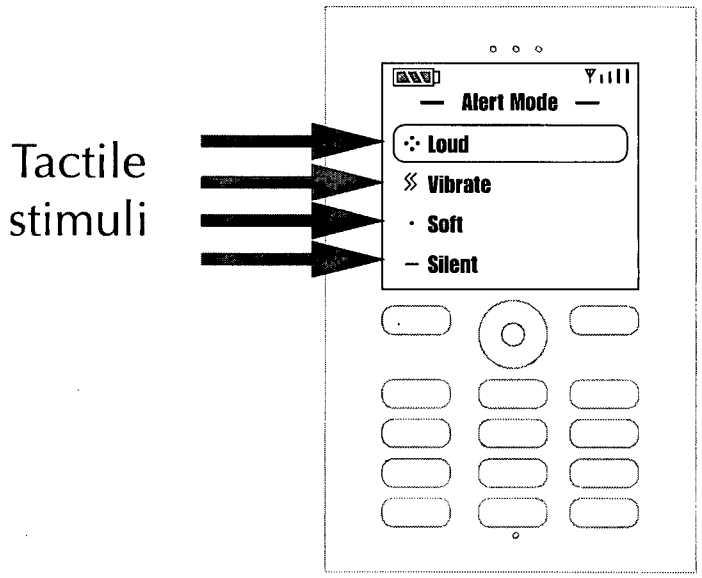


Figure 3.6: Example multimodal screen and haptic design for the list selection application concept. Distinct haptic icons are given an onscreen representation in this example as an associative aid.

The piezo tactile display technology described previously is capable of displaying small simulated surface features, such as bumps and gratings, with arbitrary motion relative to the users finger. It promises a rich vocab-

Figure 3.7: Low-fidelity foam mockup used for early user evaluation of the menu selection application concept. There is a textured plastic tactile strip mounted on the left side of the mockup.

ulary of haptic icons, which are later characterized in this document.

By mounting the tactile display on a slider that is also sensitive to thumb pressure, it becomes an input device. The user can select items in a vertical list menu by moving the display up and down. As the selection highlight is moved, the haptic icon associated with the selected list item is felt. Kinesthetic awareness of finger position allows the user to operate the device without looking, and to make a selection using the tactile display.

To gather information on user's impressions of this application concept, we constructed a low-fidelity foam mockup (Figure 3.7) and asked people to hold it in their hands and to imagine using the tactile sensations to help them make menu selections. The subjective feedback was generally positive and helped guide our prototype hardware development (Chapter 4).

3.3.2 Scrolling: Browser application

(Figure 3.5, **b**, and Figure 3.8) *Bob checks the sports news and scores many times each day. He didn't like using his old mobile phone's browser for this because he had to scroll around a lot to view content, which made him often lose his place.*

Bob accesses a sports website using his new haptically-enabled phone and scrolls down into a news story. He feels the sensation of his finger sliding over a textured surface while the text of the story moves up the screen. As he continues to scroll, he feels the headline of the next story (a distinct bump) and some links (each vibrates gently as it is highlighted). All the stimuli move smoothly past his finger in sync with the scrolling movement. Having scanned the page, Bob scrolls back up and quickly locates his area of interest (his home team's standings) aided by the memory of what that part of the

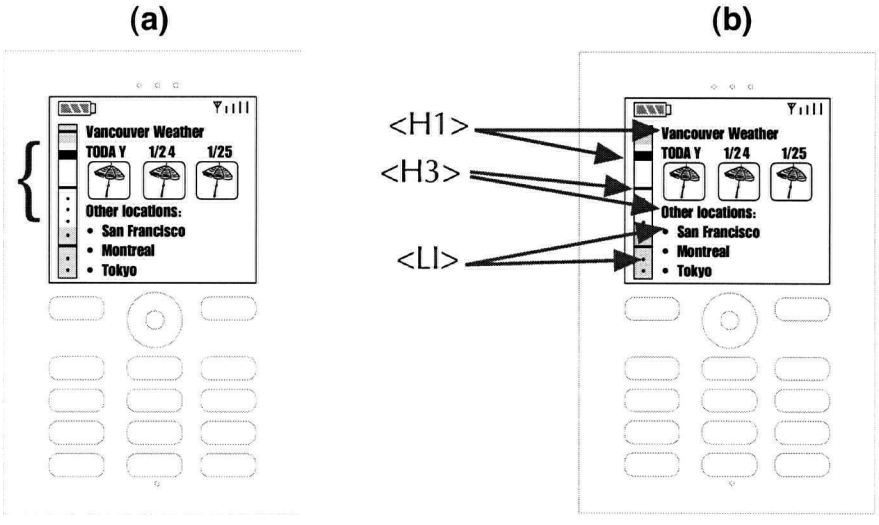


Figure 3.8: Example multimodal screen and haptic design for the browser application concept. **(a)**, The portion of the page currently visible is shown as a highlighted region of the scroll bar. **(b)**, Salient page elements are represented as haptic icons and shown in the scroll bar as an associative aid.

page feels like.

Small-screen mobile devices typically require more scrolling and/or selection actions to navigate a deep rather than wide information layout. Both place demands on the users visual attention. Haptic augmentation as vibrotactile feedback has been shown to improve performance in a handheld scrolling task [46]. However, a compact multiple-element tactile display offers additional capabilities such as smooth *tactile flow* rendering (a sensation moving across the skin).

Different page elements, such as headings, images, and links, can be rendered as haptic icons that are played when the user scrolls over them. Thus, each page has an associated haptic map that reflects its structure. Users learn to recognize familiar pages and can quickly scroll to desired sections or links. Improvements in scrolling efficiency would encourage user behaviours such as scanning to understand page structure and context, and increase the amount of information that can practically be presented on a page.

3.3.3 Direction signalling: Assisted navigation application

(Figure 3.5, **c**) *Mary is looking for a toy shop at a large, crowded shopping mall. Her location- and orientation-aware mobile device helps her find the shop with an active map and directions. The device also provides haptic feedback so she doesn't have to constantly look at the screen, allowing her to keep her eyes and ears on her surroundings.*

Mary holds the device discreetly at her side, with her thumb resting on the tactile display and pointing forward. The tactile display repeatedly strokes her thumb in the reverse direction (towards her back), indicating that the

device is pointed in the opposite direction from her destination. As she turns around, the sensation gradually weakens, then begins to build again in the opposite, forward direction; she is now correctly oriented. Mary starts walking while continuing to hold the device. The stroking becomes faster, indicating that she is approaching her destination.

Any application that assists the user in finding a spatial target could utilize an expressive tactile display to convey a direction cue. On a macro scale, this includes vehicle-based or walking navigation tasks, where the user must travel to a destination. On a small scale, the user could receive haptic assistance to orient a mobile device camera so image-recognition software can read a barcode or scene. Applications in between include finding wireless access points or other active distributed information devices, or people in a search-and-rescue scenario.

Vibrotactile stimulation at distributed points of the body has been considered for navigation [20], but in a non-intrusive handheld form factor, the display of tactile flow can be used to indicate 1-D direction. Other parameters (e.g. speed, amplitude, and wave shape) add information dimensions.

3.3.4 Display of background status information and alerts

(Figure 3.5, **c**) *Albert always feels in touch with his friends because they all share "presence" [39] and location information with each other via their mobiles, with status notifications as they become busy or free.*

Albert is composing a text message to a buddy. His fingers are busy entering text, but occasionally he places his thumb on the tactile display to move the cursor, and feels a subtle repeating haptic icon that indicates his friend Steve has come online. Albert can continue with his task, aware of

the status change.

Later, Albert is on the phone when his friend Dean goes offline. Albert feels a different haptic icon and is aware of Dean's status without having to interrupt his conversation or to remove the phone from his ear to look at the display.

Haptic alerts are commonly used on mobile devices, signalling events such as incoming or dropped calls [16]. Simple, high-amplitude signals such as vibration can be perceived through clothing and on various areas of the body, but more expressive tactile stimulation requires direct contact with sensitive parts of the skin, such as the fingertips. Therefore, it is best suited to situations where the device is being actively used and held in the hand, where the haptic feedback provides background status information. If the active application also makes use of haptics, the stimuli used for background notification must be distinct from the foreground applications haptic signals. Examples such as this underscore the importance of designing haptic icons in the larger context of their anticipated usage, and employing empirical data relating to their group perceptual characteristics.

3.3.5 Minimally Intrusive Interface for Rich Navigation of Music

(Figure 3.9) *Rich enjoys listening to music on his mobile device while on the go. He recently downloaded 30 songs onto his mobile music player, and wants to categorize them into songs he likes and songs he doesn't like. This is rather time consuming when he has to listen to the introduction of the songs, so he is in the habit of using the fast-forward/cue feature to skip to the "main", or chorus section of the song. Unfortunately, on his old music*

player this was a difficult task — since he never knew where the chorus section began, he often overshot it or was forced to listen to less relevant parts of the song before getting to the desired section. Moreover, since there were very few buttons on his miniature mobile music player, he had to press and hold the same button used to change tracks while visually monitoring the position counter, which was very difficult to do while walking or driving.

Using his new haptically enabled mobile music player, Rich can literally feel the structure of the songs in his collection. He runs his thumb up and down the slider and feels bumps and textures where chorus sections begin and end. Stronger musical themes feel like more significant tactile features, while subthemes feel like weaker features. He can easily cue to any of these features without looking, by moving his thumb to a position with a tactile feature and pressing to select.

With this method, he quickly sorts through his newly downloaded songs by repeatedly loading the next track, advancing to the chorus section, and listening to the main musical theme. He uses the device's rating feature to make note of which songs he likes, so he can enjoy their full length later.

This scenario is based on the work of Goto [21], who described a method to automatically parse and identify musical themes (chorus sections) in popular music. One application of the technique was in a retail music listening kiosk, where customers needed to sample multiple tracks on a CD and make a purchase decision within a limited amount of time. By providing cues as to the structure of the music — including main repeating themes as well as subthemes — the kiosk could help customers advance to the most salient, relevant, or memorable sections of music to make decisions about how much they like the selection. The goal of rapidly scanning tracks to identify and/or categorize them is shared by both users of listening kiosks

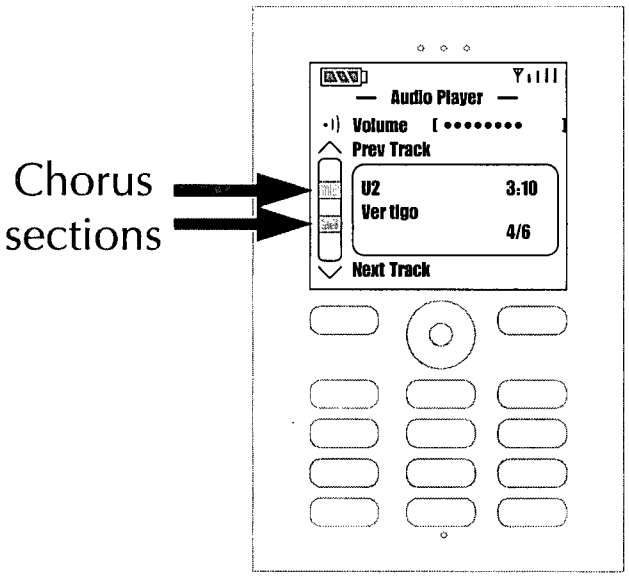


Figure 3.9: Example multimodal screen and haptic design for the rich music navigation concept. **(a)**, The portion of the page currently visible is shown as a highlighted region of the scroll bar. **(b)**, Salient page elements are represented as haptic icons and shown in the scroll bar as an associative aid.

and users of portable music players. Mobile music players are further challenged by limited space for physical buttons, making it less practical to add *more* controls than it is to add *richer* controls. Navigation of subthemes, main musical themes, entire tracks, and entire albums or playlists forms a continuum of salience which can be mapped to various levels of haptic interaction using a transducer with sufficiently wide bandwidth.

Chapter 4

Handheld Prototype Development

Working from the initial application concepts and hardware profile, we began an effort to create a high-definition, active hardware prototype that would enable significant investigation into the effectiveness of a portable haptic interface in the hands of a user.

The piezoelectric actuated lateral skin stretch technology introduced in Chapter 2 was selected for this project based on suitability for the problem domain and interaction designs, as well as practical considerations such as the maturity of the technology (allowing for a prototype to be realized within the scope of a master's thesis) and an existing relationship between the author's research group and the group developing the technology at McGill University.

Since much of the work in this chapter is the result of a highly collaborative effort between research groups at the University of British Columbia and McGill University, each section in this chapter includes a description of the present author's individual contribution to that aspect of the project. The hardware prototype, including the piezoelectric lateral skin-stretch concept, is described in detail in [43]. In this thesis we shall focus on the aspects that are relevant for the overall interaction design process of mobile haptics.

4.1 Design Philosophy

Because the options for specific implementations of the concept of handheld haptics are virtually unlimited, we followed a design philosophy that was informed by a combination of the conceptual design work discussed earlier (including user needs and application concepts), resources available for conducting the research, and general heuristics about practical implementation of the hardware concept.

4.1.1 Linear slide-mounted tactile display using piezoelectric actuated lateral skin stretch

As mentioned in previous chapters, the hardware concept consists of a linear (1-degree-of-freedom), slide-mounted tactile display based on the piezoelectric actuated lateral skin stretch technology, mounted in a handheld case and positioned under the user's thumb. We believe this option represents the best compromise for adding significant new haptic functionality without diverging too far from contemporary mobile device norms.

It would be conceivable to add multiple degree-of-freedom movement and additional features, but the additional costs in terms of size, weight, complexity, and power consumption must be justified against potential benefits. Since the benefits of rich mobile haptics have yet to be demonstrated conclusively, we felt it was best to adopt a conservative approach first; if significant benefits could be demonstrated, further iterations of the design could increase functionality in the areas where there would be an expected benefit.

The TD is placed on the left side of the device (accessible by the thumb on the left hand) because it is the most common location in existing products

for side-mounted controls such as buttons or jog wheels. The convention appears to be due to a preference, since most people are right-handed, for leaving the right hand available for jotting notes while talking on the phone, or writing on the screen of a PDA.

4.1.2 Use of off-the-shelf hardware components

All of the components of the hardware prototype should be readily available, mass produced parts. This design philosophy allows us to reduce costs and increase the speed of development, by focusing on the system *integration* issues rather than detailed engineering of components.

Our final hardware prototype may be replicated by any facility with basic mechanical and electronic prototyping equipment such as a hand-operated mill and soldering tools, at a final bill of materials cost of around $380 U.S. dollars, excluding the FPGA and PC used for control. The major component of this cost, retail piezo actuators ($160), declines significantly (by a factor of 4 or more) if ordered in quantity.

The final size, weight, and power profiles of the prototype tactile display mechanism make it not currently suitable for deployment as a mass produced consumer product. However, each of the critical components may be miniaturized with industrial micro-manufacturing facilities. In particular, the piezo elements, which currently require 50 V DC and extend into the device with a depth of 31.8 mm, may be specially manufactured as layered benders that could reduce the voltage requirement to 8-10V while simultaneously increasing rigidity and thus reducing the required depth [45]. In addition, the comparator and amplifier sections of the control electronics would be vastly simplified since reduced voltage would be well within the

range of mass produced microminiature DC-DC converters and integrated amplifiers.

4.1.3 Handheld operation while connected to a host PC

A significant part of the complexity of producing mobile devices derives from integrating batteries and wireless communications, and programming embedded computers — issues that are secondary to the haptic interface questions we are trying to investigate. Therefore, we have retained a tether to a host PC and power supply, while packaging the input and output transducers in a handheld form factor.

This allows us to investigate key aspects of the handheld user experience using a method similar to the "Wizard of Oz" paradigm: to the user, it appears that they are using a mobile device with a few subtle wires added. Despite the tether, the overall form factor generally *affords* mobility and users are able to conceptually distinguish the wires as additions possibly related to the testing status.

4.1.4 Author's Contribution

The part of the design philosophy related to the form factor, especially the thumb-mounted positioning, is primarily the work of the author. Other aspects (off-the-shelf components and tethered operation) were heuristics that were also shared in parallel by the collaborator, Jerome Pasquero, and the thesis supervisor, prior to the beginning of the project.

4.2 System Overview

The hardware components of the system, including labelled signal paths, are shown in Figure 4.1. The major components of this system are described in the sections below.

Since this thesis discusses a design process, it is instructive to examine how the hardware specification evolved over time. Within a month of beginning the hardware project, many of the components had been established (Figure 4.2). Along the way to the final design, the desk mount (which was expected to increase reliability and simplify cable management, but was not necessary because the cables were not as intrusive as expected) was eliminated, as was the provision for using an off-the-shelf game controller to digitize analog position and button signals, and additional pushbuttons on the device (though they could be added at any time if applications warrant).

4.3 Output Transducers

4.3.1 Tactile Output Device (Tactile Display)

The tactile display (TD) has an active area of 8.7 × 6.4 mm and contains eight stacked piezoelectric bending motors separated by thin (0.5 mm) brass rods. The bending "motors" (Piezo Systems model T215-H4-203Y) are thin (0.38 mm thick) rectangular sheets of relatively rigid ceramic-like material measuring 6.4 × 31.8 mm. Internally they consist of two bonded layers of alternately polarized piezoelectric material, such that when an electric field is applied to the motor, one layer contracts while the other expands, resulting in a bending deformation of the entire slab.

The piezo elements are mounted in a new dual-pinning configuration that

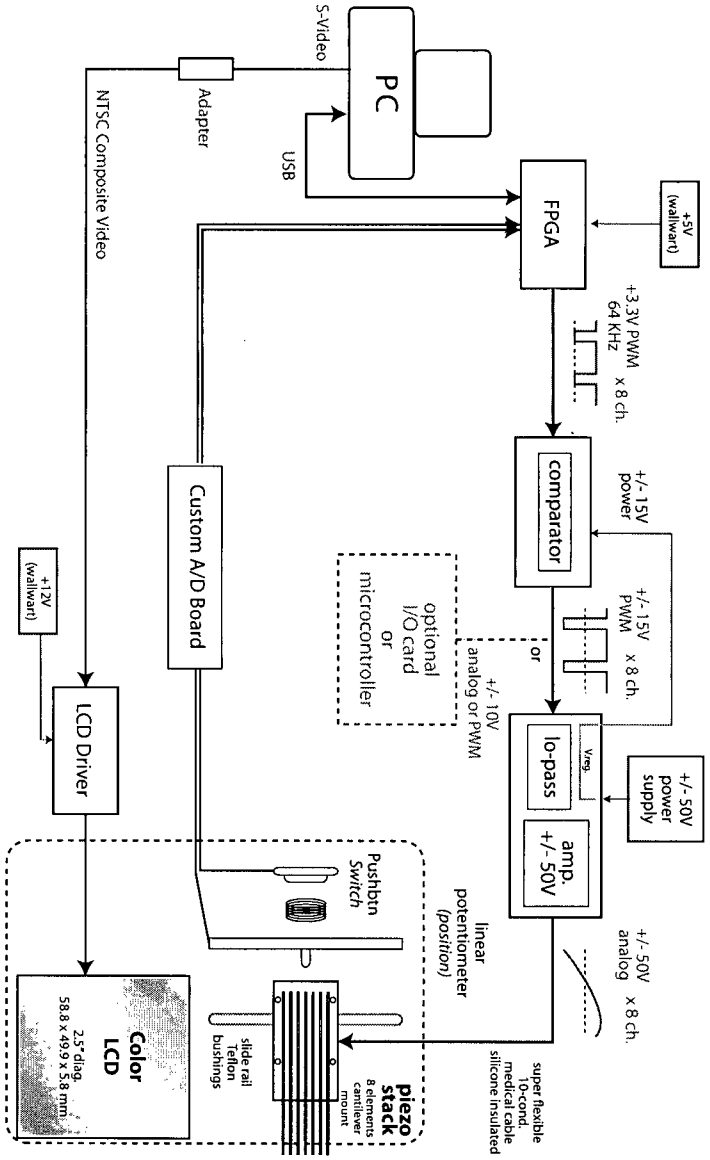


Figure 4.1: Overview of hardware components of the handheld prototype (final version used for user testing).

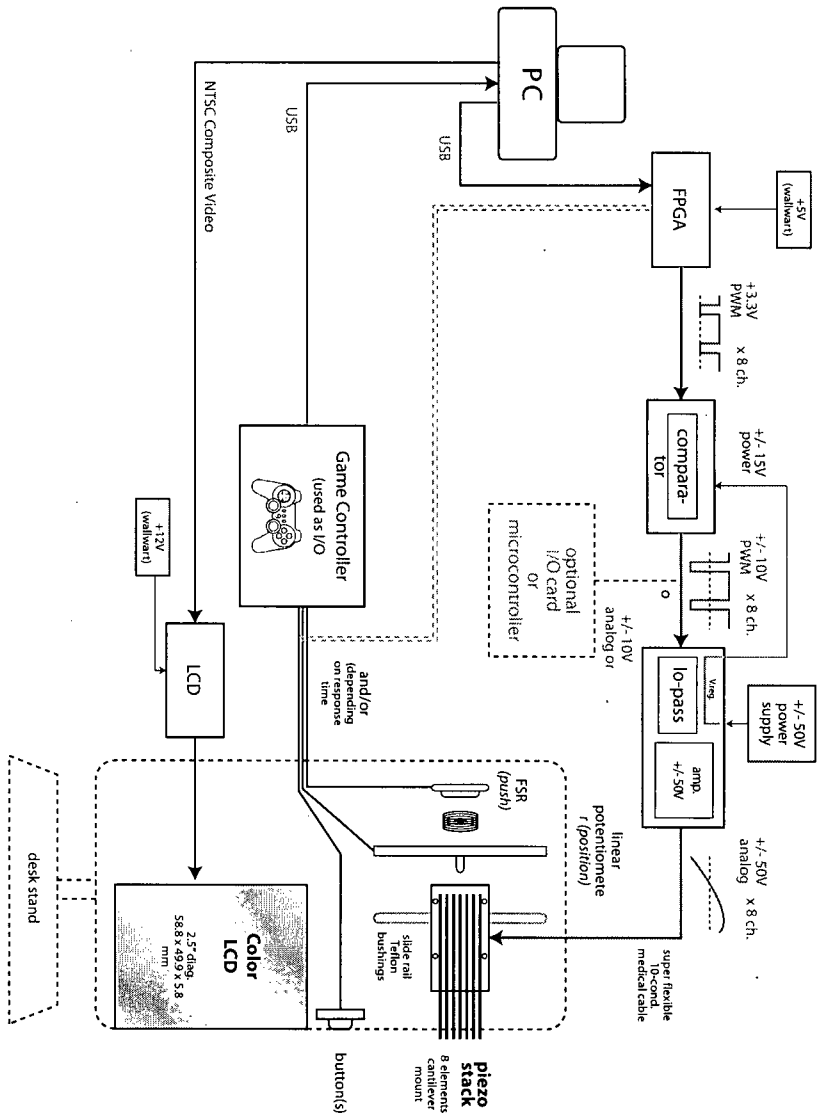


Figure 4.2: Hardware components as of March 2005

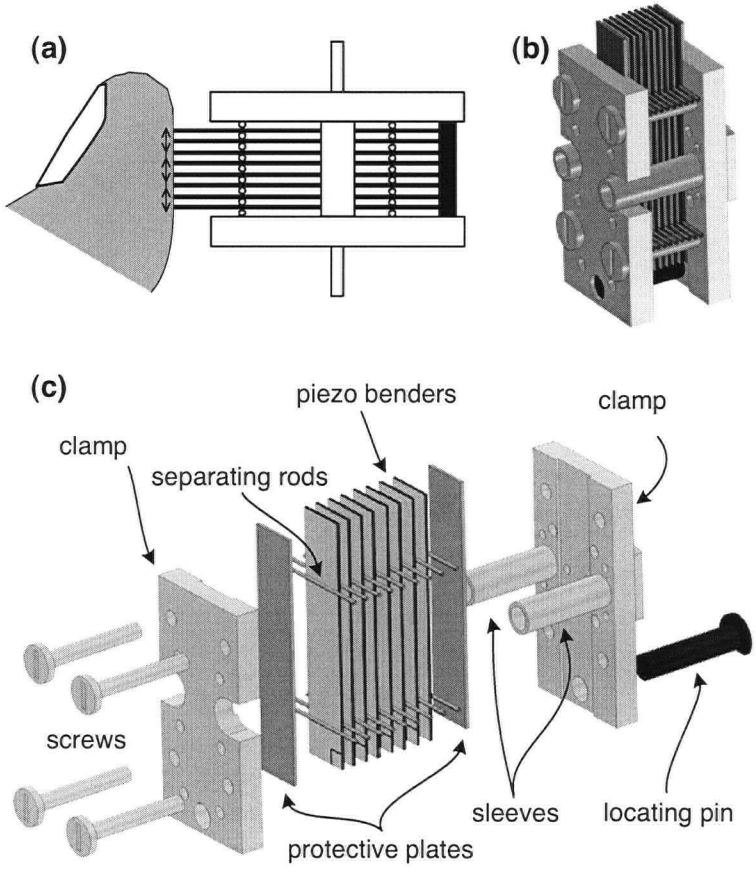


Figure 4.3: The tactile display. (a) Mechanism of action. (b) Assembled view. (c) Exploded view. Diagrams by Jerome Pasquero.

optimizes the force delivered to the user's finger and is more efficient than the cantilever mount used in the Virtual Braille Display [30, 43]. Because the distal ends of the piezo elements are fixed (i.e., mechanically grounded to the chassis that is being held in the user's hand), the proximal ends of the piezo elements that are in contact with the user's finger move when an electric field is applied. The portion of the user's skin covering the region between the ends of two adjacent piezo elements is stretched when the elements move away from one another, and it is compressed when they move towards one another. This activates the strain-sensitive mechanoreceptors in the user's skin, producing a tactile percept. Finally, it has been demonstrated that this produces the illusion of the sensation of something pushing *into* the skin, as occurs when the finger is placed on a surface feature such as a small bump [23].

Raw piezo elements are shipped as layered slabs with three electrodes: the two outer surfaces and an internal, central electrode. Preparation involves electrically connecting the two outer electrodes while creating an area to access the central electrode. Using a technique developed by the McGill collaborators, we abraded a small section of outer electrode and applied conductive tape, conductive paint, and insulating tape to create the required connections. Special solder flux provided by the piezo vendor was used to attach 30-gauge wire to each bending motor.

Note that although the piezos we used require relatively high voltage (+/- 50V), the current draw is on the order of a few milliamps and thus they are not high power devices. Because the piezo bender essentially consists of two oppositely charged electrodes separated by a gap, its electrical behaviour at sub-resonant frequencies is essentially that of a slightly lossy capacitor. As such, its steady state current draw is almost nil, and producing a deflection

by charging and discharging the plates also does not require high current.

Further details of the tactile display design can be found in [43].

4.3.2 Video Display

The video display is a conventional 2.5-inch colour TFT LCD monitor that includes driver circuitry to accept NTSC composite input signals. In the handheld prototype, the video display is mounted in the top half of the casing with the driver board positioned alongside, rather than under, the LCD panel. An 8-pin ribbon cable carrying the video signals emerges from the bottom of the casing and is routed around the top of the casing to join the power and signal wires. A second driver / interface board accepts the NTSC composite input via an RCA jack and 12V DC power (from a standard AC adapter) and sends the signals out on the ribbon cable.

After long term use the video display experienced hardware problems unrelated to its usage in the haptic prototype, and it was not possible to procure a timely replacement. Therefore, for the user tests related to the browser it was necessary to use a slightly larger 3.5-inch monitor affixed to the top of the case. While this did not produce an appreciably different screen image (both use NTSC composite input), it did cause a significant increase in the weight of the top half of the device.

4.3.3 Author's Contribution

The tactile display was designed mostly by Jerome Pasquero, while the author was responsible for preparing the piezoelectric components, and participating and providing feedback in design discussions. The video monitor is an off-the-shelf component; however, the author is responsible for its inclusion,

selection, and procurement.

4.4 Sensors

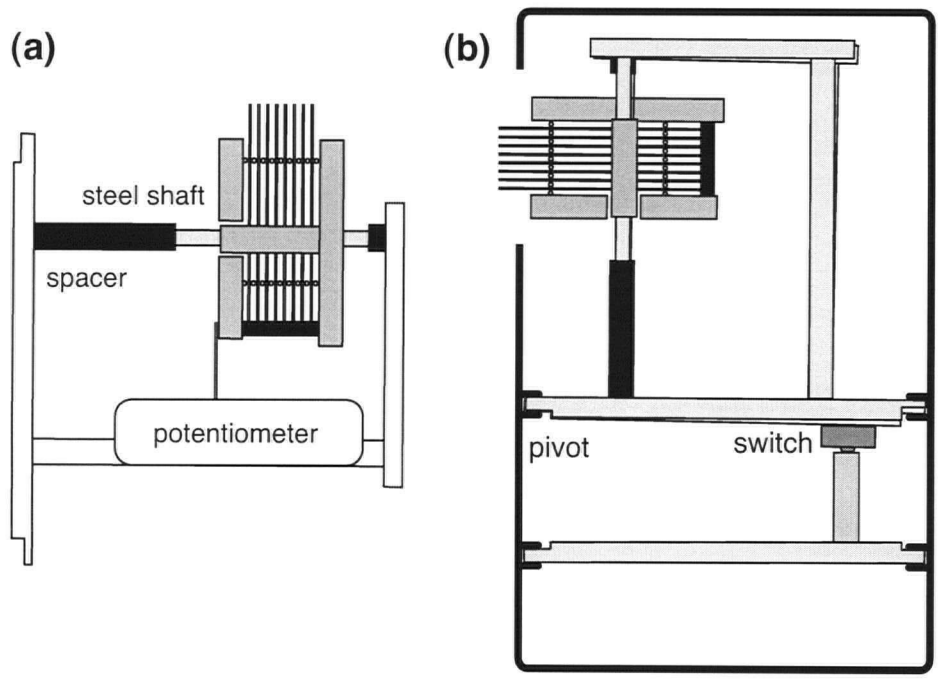


Figure 4.4: Location of input sensors. (a) Resistive position sensor. (b) Push-to-select sensor. Diagrams by Jerome Pasquero.

4.4.1 Slider Position Sensor

The entire TD is mounted on PTFE (Teflon) bushings that slide over 2.4-mm diameter steel rods, with a travel of 11 mm. Slider position is acquired using a standard analog 10K ohm potentiometer-type resistive position sensor. A custom A/D module (Section 4.5) digitizes the signal and sends it to the

control circuitry.

4.4.2 Push to Select Sensor

To facilitate a "click to select" behaviour, the entire assembly can articulate in an arc with a travel of approximately 1 mm when the user pushes *into* the TD (i.e., perpendicular to the active surface). The movement is sensed by a standard 3 mm pushbutton switch mounted on the distal end of the mechanism. The switch is connected via the A/D module to the control circuitry.

4.4.3 Author's Contribution

The author was responsible for the idea of the selection functionality, and component selection for the position sensor. A major part of the effort of integration of the sensor components was in achieving the right mechanical design for the slider and articulating mechanism; this was done by Jerome Pasquero in cooperation with McGill mechanical workshop collaborators Joe Boka and Don Pavlasek.

4.5 Interface Electronics

Control is achieved via a series of custom and off-the-shelf electronic modules. The tactile output signal begins at the PC running the application software, which sends out packets via a Universal Serial Bus (USB) connection to a field programmable gate array (FPGA, Constellation board 10K50E from Nova Engineering) running custom firmware. The function of the FPGA is to convert the PC output into an 8-channel 156.25-kHz pulse width modulated signal. The PWM signal is then fed through a comparator,

low-pass filter, and high voltage amplifier, resulting in an 8-channel analog signal that varies from +/- 50V. (see Figure 4.1).

Note that any equipment capable of generating a PWM signal may be used as the input to the comparator, filter, and amplifier stages. If desired, a standard I/O card, or embedded controller, may be connected in lieu of the PC and FPGA, eliminating the most costly and bulky components of the system and allowing for flexibility in prototyping.

On the input side, a standard 8-bit analog to digital (A/D) converter is used to digitize the resistive position signal.

4.5.1 Author's Contribution

Jerome Pasquero was primarily responsible for the design and specification of the interface electronics. The author was responsible for participating in design discussions (mostly to advocate the maximum level of flexibility for rapid interaction design prototyping), sourcing of some parts, and assembly of the custom electronics.

4.6 Power Supplies

The power supply for the tactile output consists of a pair of standard regulated 48V, 25W power supplies (Kaga Components Ltd. Part # SP30U-48S) normally used for telephone equipment. The fine-tuning voltage adjustment feature was used to tweak the output voltage to 50V. Additional power supplies include manufacturer-supplied standard AC adapters for the FPGA and LCD, and an internal +/- 15V power supply derived from the main +/- 50V source via voltage regulator.

4.6.1 Author's Contribution

The design and integration of the main power supply is the work of the author.

4.7 Control Software

The system is controlled by custom software modules on the FPGA and host PC.

The FPGA software consists of device specific firmware that handles the translation of USB packets into PWM output, and 8-bit A/D input to USB packets. It additionally offers buffer management features and can optionally support a non-hosted mode in which the FPGA operates in a closed-loop fashion, with piezo output dependent on a cached slider position function.

The PC software layer is based on the "STReSS Library",[1] [29] a cross-platform API written in C++, featuring buffered input and output functions and support functions. It is implemented using various open-source libraries, including the `ACE` component library for XML parsing, threading, and other support functions, and the `libusb` library for I/O. All applications described in this thesis were implemented on a 1GHz Pentium class PC running SuSe Linux.

Internally, the +/- 50V output signal is represented using 12 bits (4096 levels) on the FPGA, and the PC control software is able to handle both 12-bit and 8-bit signals. In practice, it was found that there was no readily

[1] The API is so named because it shares a large part of the code with an earlier project [41].

perceivable difference[2] between 12-bit and 8-bit resolution output, so the 8-bit (256 levels) signal was used for all the applications described in this thesis. This allowed for more efficient use of resources, since timing was a concern (see below).

4.7.1 Input and Output Timings

The basic software I/O loop runs at a rate of 333 Hz. Once every 3 ms, new information (consisting of voltage levels for each of the eight piezos) is written to the FPGA's output buffer, and the input buffer (consisting of slider position and click status) is read into the PC. All implementations described in this thesis were run on standard Linux without real-time extensions, so the actual I/O timings are subject to system load conditions.

Experiments and performance optimizations were performed to ensure stable real-world performance under load. Figure 4.5 shows the measured output at the amplifier (with the piezo load disconnected), while the system is running the full interactive browser prototype described in Chapter 6. In addition to generating piezo output, the system must read slider input, compute the output tactile image, and update the visual browser display.

Effect of output timing on qualitative experience

The top portion of Figure 4.5 shows the output of an early version of the software, which was able to produce signals at an irregular rate of approximately 12 Hz, while the bottom portion of Figure 4.5 shows the output

[2] Although this observation is based on the subjective experience of the author and other researchers, it is reasonable to assume that since the total piezo travel must be less than 1.0 mm, a 12-bit (0.2 μm maximum) resolution is well beyond the limit of human perception, which is roughly 1 μm for small surface features [25].

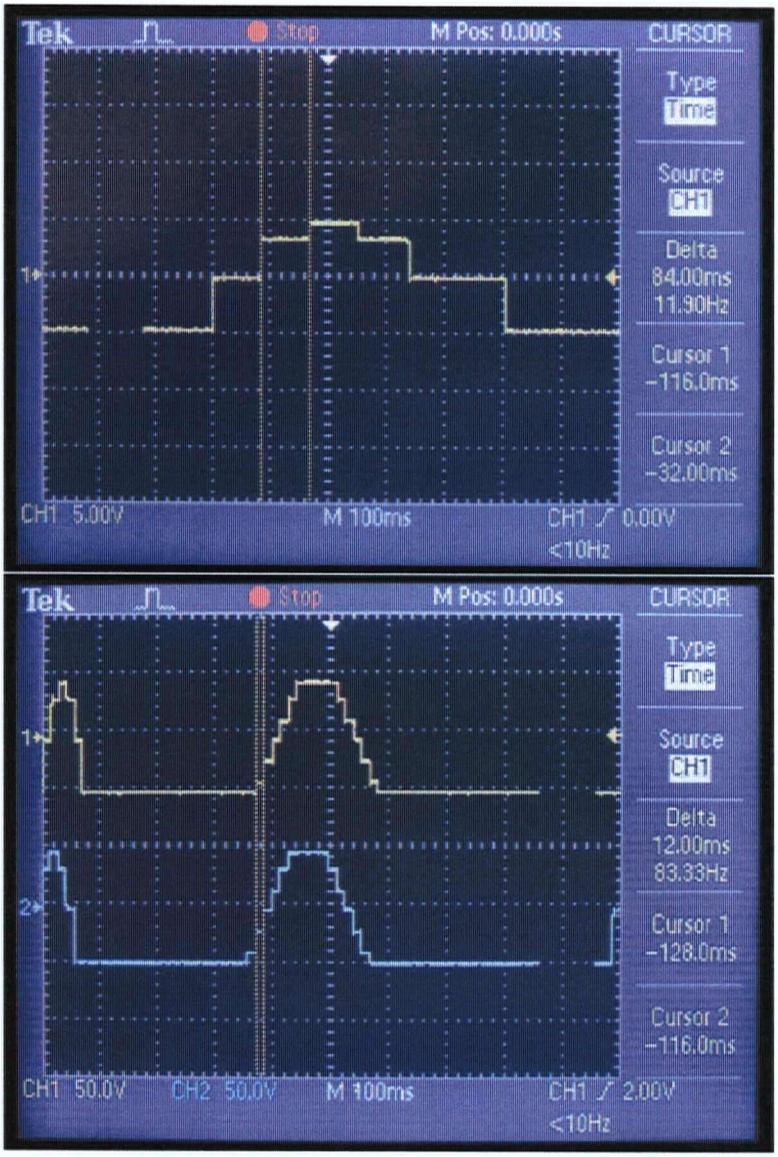


Figure 4.5: Measured outputs under conditions of load (running the browser application). *Above,* prior to optimization. *Below,* after optimization, showing two adjacent output channels at once.

after a series of software performance optimizations improved the rate to a relatively stable 83 Hz. At the low (12 Hz) sampling rate, the sensations produced by the TD were perceived by the author and other observers as rough, vibration-like stimuli without a clear sense of *tactile flow*, while the high (83 Hz) rate stimulus produced a much smoother sensation. Only the high sample rate stimulus could be compared to the sensation of touching a small bump. It is likely that the problems with low sample rate stimulus were the result of **aliasing** — undesirable high frequency transients are produced at the square edges of the wave, which become notably more pronounced at the lower sample rate.

Control software performance optimization

Software performance optimization to improve the output rate consisted of iterative development of both the FPGA firmware and PC software to optimize the buffering algorithm for rapid, interactive, closed-loop operation. This was especially important for the browser application (Chapter 6) which, unlike the earlier applications used for simple, triggered playback of haptic icons and tactile movies (Chapter 4), could not "anticipate" and cache significant amounts of output for smooth buffered delivery.

Further performance improvement was achieved by rewriting the applications to the number of threads, as interprocess communication overhead was found to be a significant source of irregular delays. The final implementation of the browser application uses only two "user" threads, plus one I/O thread in addition to the various threads internal to the Mozilla browser.

Finally, as described in more detail in Chapter 6, the browser application software is designed to take advantage of the fact that the visual sense

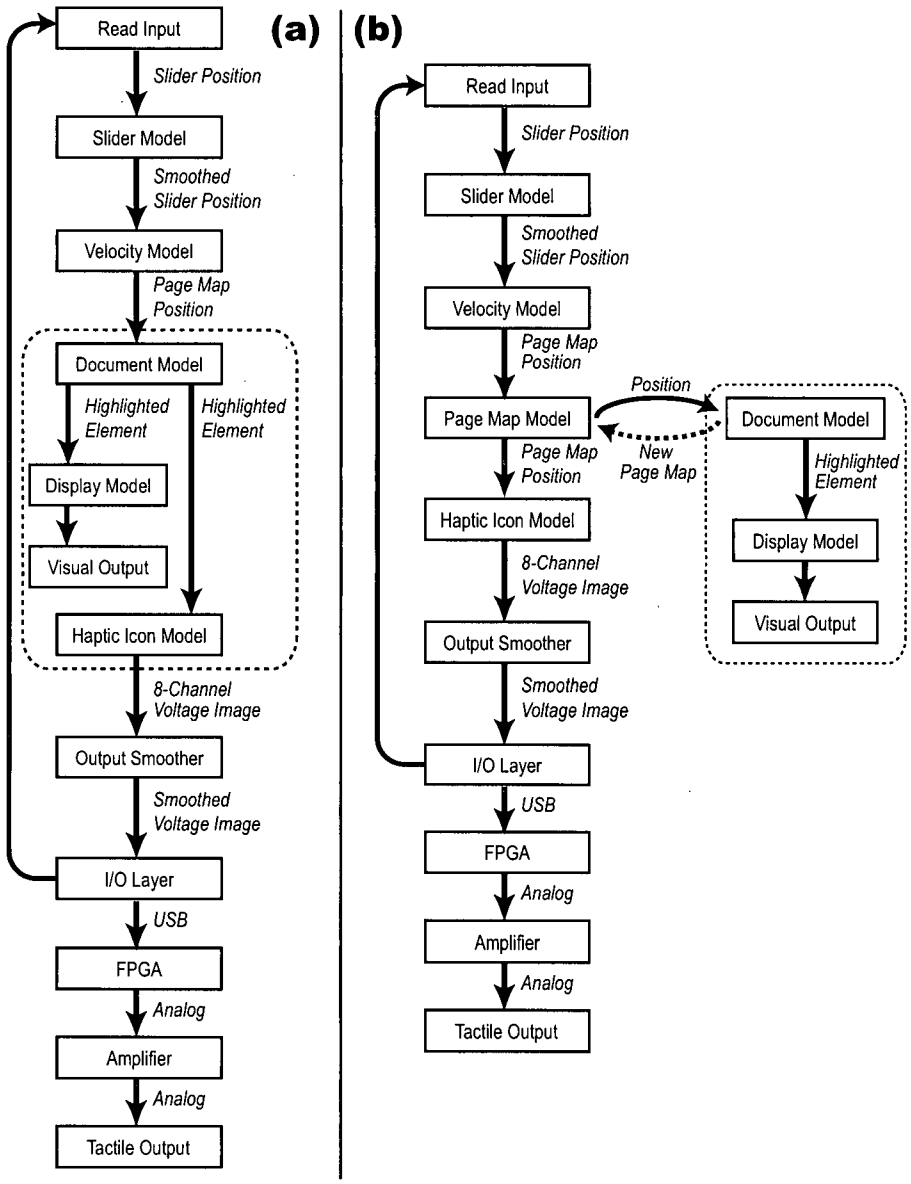


Figure 4.6: Implementation alternatives for the control software. **(a)**, large single loop, **(b)**, asynchronous design for improved tactile performance. The computation-intensive Mozilla portion is enclosed in a dotted line.

is less sensitive to timing irregularities than the tactile sense. Instead of one large I/O loop incorporating the Mozilla browser and its document model together with the tactile computation modules, the final software uses separate, asynchronous loops for tactile and visual components (Figure 4.6). The tactile loop, a slim 500K C++ program, runs at a higher priority than the visual loop (the approximately 20MB Mozilla suite), and includes its own internal, simplified representation of the web page so that it does not need to wait for the visual component.

The final resource usage of the full-blown control software stack, including browser, averages around 60% of the 1GHz CPU, as measured by the Linux "top" command.

4.7.2 Author's Contribution

The core control software architecture and implementation was done by Vincent Levesque. The author was involved in helping test and verify the system, including the measurements and optimizations (with the exception of the firmware update) described in Section 4.7.1.

4.8 Tactile Flow Rendering

The concept of *tactile flow* is described in various previous publications on related piezoelectric actuated lateral-skin stretch tactile displays [23, 28, 30, 41] and publications on the present prototype [34, 43]; however, since it forms the basis for many investigations throughout this thesis, we shall review it briefly here.

When a finger is swept over a small surface feature, patterns of skin

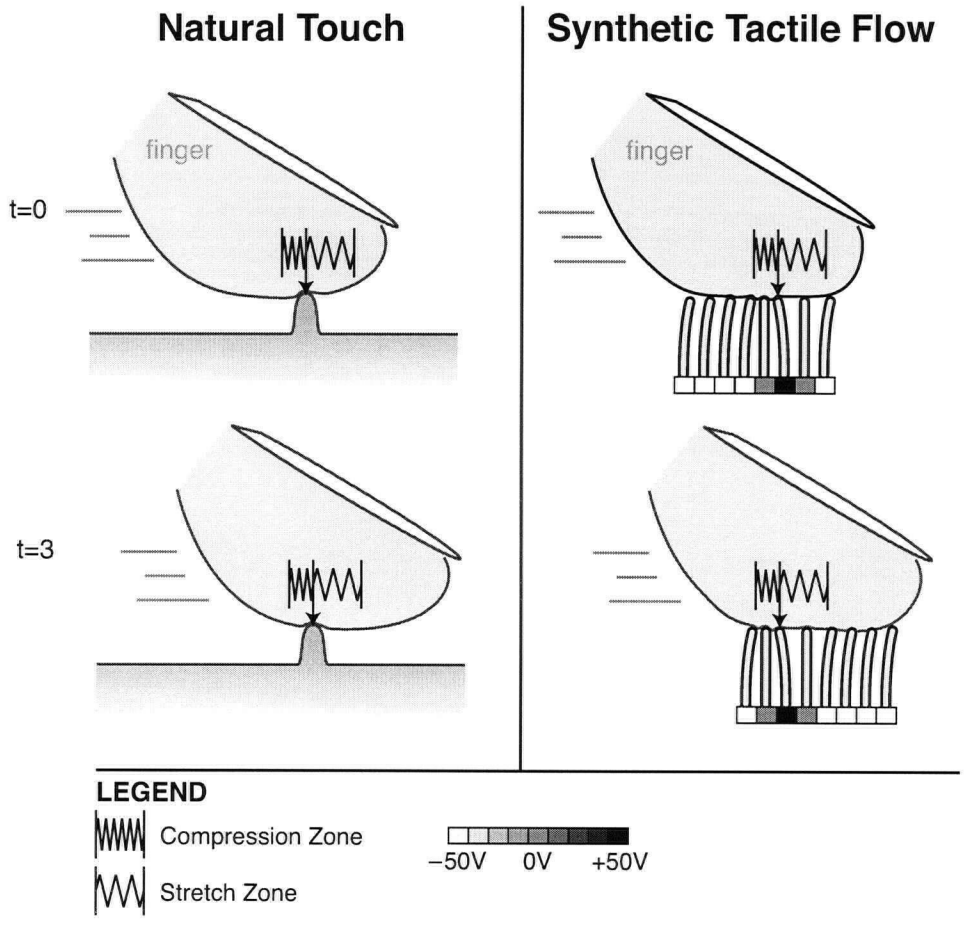


Figure 4.7: Moving patterns of skin stretch create a sensation of *tactile flow*. *Left,* the shear pattern associated with dragging a finger over a small surface feature. *Right,* synthetic re-creation of the pattern using the piezoelectric lateral skin-stretch TD. Parts are not drawn to scale.

stretch and compression are produced [28].[3] For example, the drag of the feature against the skin creates shear compression along the portion of skin behind the leading edge, and since the skin is a continuous elastic material, the area ahead of the leading edge undergoes a complementary shear expansion, or stretch (Figure 4.7, *left*). Furthermore, this pattern of skin stretch and compression *travels* across the finger as it is moved over the feature.

This moving pattern of skin stimulation is detected by mechanoreceptors and produces a sensory percept, which we have termed *tactile flow.*

It is possible to simulate an analogous moving pattern of skin stretch and compression using the tactile display in our handheld prototype. Figure 4.7, *right*, shows how the voltage-actuated bending of the piezo elements creates regions of shear stretch and compression. Note that at time $t = 3$, the pattern of voltage actuation is the same as at time $t = 0$, except that the signals are shifted to the left. In outward visual appearance, the sequential movement of adjacent piezo actuators during tactile flow display looks similar to biological *peristalsis* (sequential muscle contraction), or the pattern of movement of millipede legs (Figure 4.8).

Tactile flow is commonly associated with spatial movement, such as dragging one's finger over a surface to explore its texture, or losing one's grip on an object. It is a *directional* signal; moreover, it is more natural (in terms of being more commonly encountered in daily life, as one touches and explores objects) than spatially distributed patterns of vibration [9, 50].

Therefore, it is hypothesized that tactile flow may be useful as a spatial cue for use in navigating computer interfaces. This topic is covered in more detail in Chapter 7, *Browser User Study.*

[3] One can simulate this experience by sliding a finger over the tactile "keyboard hints" — small bumps present on most computer keyboards at the F and J key positions.

Figure 4.8: A millipede. Its legs move in a pattern that looks outwardly similar to the movement of the TD's piezoelectric actuators when displaying a *tactile flow* signal, except that the TD does not normally move relative to the user's finger.

4.9 Visualization of Tactile Stimuli

4.9.1 Problem

Depicting and communicating the tactile display output visually was a notoriously difficult problem. The output electrical signal is an eight-channel time-varying voltage ranging from +50V to -50V DC. Thus there is the problem of depicting *Voltage* × *Time* × *8 Channels* using two-dimensional media. The output may be represented by drawing the eight signal traces in a stacked fashion on the same time axis, like the display of a hypothetical eight-channel oscilloscope (Figure 4.5 and Figure 4.9, **a**). This "parallel wave graph" approach is used in previous publications [30, 41]. However, this representation has several problems, including:

- It only depicts the time-dependency aspect of the stimulus; it does not readily show the output as a function of slider position, which is the key feature that enables bidirectional interactivity and exploration of virtual texture that extends beyond the 8-taxel display window.
- The presentation format is somewhat difficult to understand for a lay audience. It is most easily understood by a technical audience that is accustomed to reading signal trace graphs.
- Because the vertical axis encodes two separate variables — piezo number and voltage — the spatial relationship between piezos is broken up by local axes (voltage). Thus the graph does not visually convey the propagation of a signal from one piezo to another that characterizes "peristaltic" motion or *tactile flow.*
- The "parallel wave graph" format requires a lot of space to depict

even a simple stimulus. A more compact representation would thus be desirable.

4.9.2 Novel Graphical Representations

To address these issues, a new way of thinking graphically about the piezo output was developed as part of the hardware development effort for the handheld tactile display. This includes zeroth-order shaded voltage map and first-order skin stretch image representations, and a haptic icon design process that leverages the graphical representations for rapid prototyping of tactile stimuli.

4.9.3 Shaded Graph for Voltage Signal

By replacing the height of a voltage trace with a greyscale shading value, it is possible to achieve a more compact representation that eliminates the overloading of the vertical axis with both voltage and spatial information. As illustrated in Figure 4.9, **b**, the shaded graph enables more rapid comprehension by visual inspection of the "shape profile" and spatial displacement pattern of the stimulus, as compared to the parallel wave graph.

Another key feature of the shaded graph is that instead of simply representing eight piezo outputs, the vertical axis may now be considered to encode *taxels*, or arbitrary spatial units along which tactile stimuli are organized. The TD can be considered a *display window*, spanning exactly eight taxels, into this larger tactile space. Therefore, the shaded representation serves a dual purpose: to visualize the output of the eight piezo channels over time, and also to visualize a tactile feature map, optionally one that evolves over time. The latter representation is used throughout the remainder of

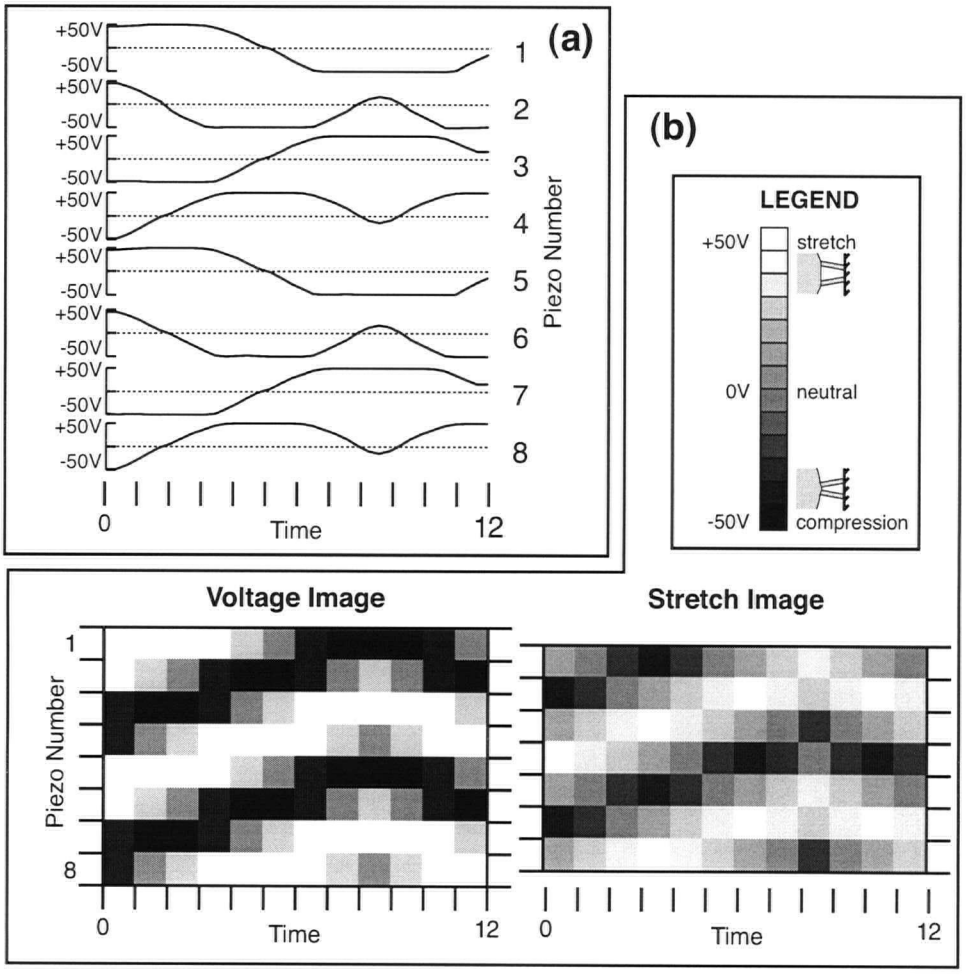


Figure 4.9: Graphical representations of a stimulus consisting of three small bumps that travel up, and then down, the TD. (a) Conventional parallel wave graph, used in previous publications. (b) Shaded graphs showing voltage and skin stretch (delta) images. The **time** axis is expressed in arbitrary units. Brass end-plates are not included in this stretch analysis.

this thesis.

4.9.4 Skin Stretch Image

The raw voltage level at a given time simply corresponds to the static displacement of the piezo element at that time. The tactile percept, on the other hand, is produced by transient patterns of skin stretch that activate mechanoreceptors in the user's finger. This may be caused by:

1. Skin stretch or compression produced by the movement of two piezos relative to each other. The area of skin in the gap between the piezos undergoes the stretch or compression.

2. Skin stretch or compression produced by the movement of a flanking piezo relative to the brass end-plates.

3. High frequency vibrotactile activation of a piece of skin, in which forces are produced against the inertia of the skin.

4. Active exploration of the static TD surface caused by the user moving or slipping their finger over the TD.

Of these various causes, (4) is unrelated to the synthetic haptics being discussed here, and (3) is subject to the limitations of vibrotactile stimuli that we described in Chapter 2, and as such we consider it secondary to the rich lower-frequency activations made especially possible by the current TD design.

By plotting the difference between voltages applied to adjacent piezo elements using a similar shading convention to the above, we can generate a graphical picture of the skin activation pattern due to causes (1) and (2). In the present work, we have chosen to use "bright" shading for "stretch"

(or expansion of the gap between two adjacent piezo-actuators), and "dark" shading for "compression" (narrowing of the gap). The resulting visual representation is known as a *stretch image*, and is contrasted with the raw *voltage image*, sometimes abbreviated *volt image*. The *stretch image* reflects more directly the user's tactile experience of the stimulus.

Stretch Image Caveats

The *stretch image* is an attempt to more accurately depict the tactile sensations produced by the device, and it is applicable to all of the stimuli described in this thesis, but it is still not a completely perfect picture. The following cases should be noted:

1. As mentioned in the previous section, the TD has sufficiently rapid response so as to be capable of producing vibrotactile stimulation that is not due to skin stretch created between adjacent piezo elements and therefore is not captured by the skin stretch representation. For example, if all eight piezo elements are vibrated in synchrony, the skin stretch image representation would be a flat grey, equivalent to no activation at all, since the piezos are not moving relative to one another. For this reason, stretch images are always presented alongside the source volt image in this work, as vibrotactile transients can be visually identified as areas of high horizontal contrast in the volt images even when they are not present in the stretch images.

2. Patterns of *static* stretch do not produce a tactile percept. For example, the piezos could be moved into a particular configuration and kept there for a long time (seconds), producing a momentary sensation when they are first moved, but one that dies out as the mechanorecep-

tors adapt to the pattern of stretch. This adaptation is not reflected in the *stretch image*.

The potential problem of too low-frequency stimulation should be kept in mind during tactile experience design. Further discussion of image-based compensation for mechanoreceptor adaptation can be found in the following section.

4.9.5 Automated Tools for Design

When tactile images are represented as an array of eight greyscale pixels, they can be created, manipulated, and analyzed using readily available tools for dealing with visual (bitmap or raster) images. Using a paint and image manipulation program such as Adobe Photoshop, tactile images can be easily painted using the mouse, and many convolution filters may be applied with qualitatively similar effects in both visual and tactile domains: *blur, sharpen, add noise, contrast*, and many others fall into this category. Since the image format provides an on-line visual preview of the tactile stimulus, the visual transformation produces a related tactile transformation. For example, blurring an image *visually*, smooths ("blurs") the *tactile* sensation as well.

Furthermore, *stretch images* can be automatically and rapidly generated in one click using Photoshop's *"Custom" Filter*. This is a generic convolution filter that accepts a user-specified kernel up to 5×5 in size, and may be used to generate a wide variety of visual effects. When applied using the settings shown in Figure 4.10, it takes the difference between two vertically adjacent pixels, normalizes it to the stretch image scale previously described, and outputs the image. Because the *stretch image* is one pixel smaller than

the *volt image*, the top pixel needs to be cropped off after using this filter. Alternatively, if desired, the interaction of the two end piezos (#1 and #8) with the brass end-plates can be investigated simply by adding two rows of neutral grey pixels to the voltage image prior to convolution. The resulting stretch image, after cropping, would then be 9 pixels high.

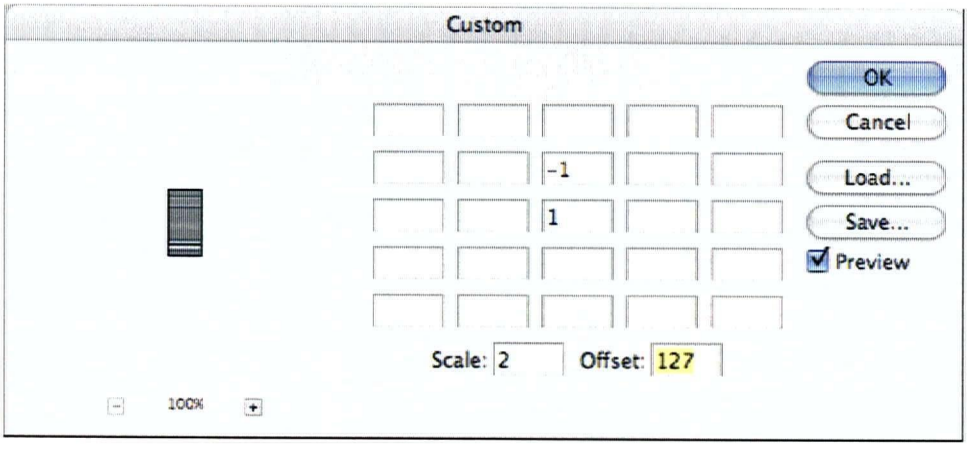


Figure 4.10: Photoshop Custom Filter settings to automate creation of *stretch images*. The depicted convolution kernel instructs the program to do the following: *Take the sum of the current pixel, multiplied by 1, and the pixel immediately above it, multiplied by -1. Scale the total by dividing by 2, then add 127 (half of the full-scale value of 255).*

More sophisticated algorithms could be employed to improve the accuracy of the stretch image by taking into account adaptation: local areas of skin stretch that are held for long periods of time could gradually "fade to grey" by detecting non-changing pixels (stretch zones) and pushing them towards grey on a timescale that is consistent with empirically measured adaptation phenomena.

From a practical standpoint, it was decided not to implement adaptation compensation in the image processing for several reasons. First, the adaptation rate is not constant and is dependent on a number of factors such as psychological state, skin properties and individual factors. Second, adding a real-time scale to the horizontal axis of the images introduces significant complication that undermines their purpose as a rapid prototyping and visualization tool. Finally, it is quite easy to visually detect and avoid long periods of unintentional lack of motion in haptic icon designs; it is far more likely that such artifacts would result from higher-level effects such as interaction between haptic icons and the speed of the user's navigation. For these reasons it was decided that the plain, uncompensated stretch image represented a good compromise between accuracy (correspondence of the visual representation with the felt tactile sensation) and utility as a prototyping and visualization tool.

4.9.6 `gif2hapticon` Tool

Prior to the introduction of the `gif2hapticon` tool, multichannel output for piezo tactile displays was usually constructed algorithmically, with tables produced in Matlab or similar tools through summation of sine functions, and used by applications in a simple spatially modulated lookup loop. Alternatively, using a Java-based application, the user could specify temporal output (a "tactile movie") by setting the value for each time-sample using onscreen slider widgets. The output from this application was an XML-like file[4] encoding the sample "frames", which was read by a tactile movie player

[4]XML was used primarily because of its extensible nature, which ensures that future functionality, such as more complex time/space relationships and support for different TDs, could be added without having to make major revisions to existing software. The

program and output in sequence to the hardware.[5]

Both existing methods required considerable time and effort to generate a new tactile stimulus. In practice, the stimuli that were created tended to be selected on the basis of convenience of the generating algorithm, rather than the quality of the tactile experience — for example, algorithmically "neat" simple sine waves or, when using the slider approach, square or linear transitions. Additionally, the algorithmic method restricts the design of haptic signals to a technical audience that is familiar and comfortable with mathematical functions. A method to allow more direct, rapid prototyping of haptic signals, leveraging the visualizations described previously, would enable more iterative design of stimuli and exploration of the stimulus space.

Leveraging the existing XML framework, a tool called `gif2hapticon` was developed by the author to rapidly transcode raster images in the GIF89a [2] format to the tactile movie XML files, or to the newer "haptic icon" file format (also XML based) described in Chapter 6. It is written in C++ and uses the free ImageMagick [4] library for reading GIF images.

With the `gif2hapticon` utility, any software that is normally used for creating animated GIF images for the Web may also be used to generate tactile movies. A sample workflow is shown in 4.11. Perhaps more importantly, the first steps of this workflow, where haptic icons are designed in a creative and exploratory process, are nearly identical to the creation of animated GIF icons for the Web. Therefore, anyone with experience in creating simple Web graphics can participate in haptic icon design, thus enabling a

actual implementation does not include a formal Document Type Definition (DTD) for reasons of expediency (although one could be added trivially in the future), therefore it is "XML-like" in its current state.

[5]These methods were developed and used by the group at McGill University.

much wider audience than those who are familiar with creating matrices in Matlab.

The code for the `gif2hapticon` tool is shown in Appendix C. It is a cross-platform ANSI C++ application, as is the required ImageMagick library. The general command line usage is:

```
gif2hapticon -t <infile.gif> <outfile.xml>
```

where `-t` specifies tactile movie mode and `infile.gif` and `outfile.xml` are the input and output filenames, respectively.

`gif2hapticon` is discussed further in Browser Prototype, Section 6, as it also features a mode for creating icons that are read by that application.

4.9.7 Author's Contribution

The above section is primarily the work of the author. Prior to the author's involvement with the project, the "parallel wave graphs" described in Section 4.9.1 were developed by the McGill collaborators, as well as a graphical representation, used in a Java tactile movie tool developed by Jerome Pasquero, that showed the relative angular orientations of individual piezo elements (not discussed in the body text above for reasons of brevity).

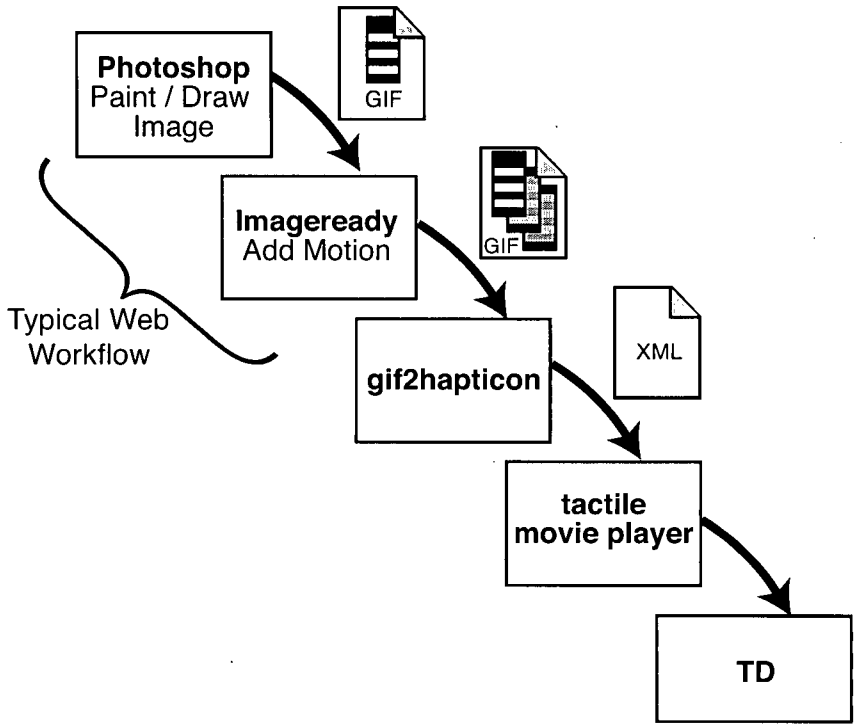


Figure 4.11: The process for creating a haptic icon using the image-based workflow is similar to the process of creating graphics for the Web. In this example, all creative steps are done in Adobe Photoshop and ImageReady, and the conversion to tactile signals is handled by automated tools, with the output of the Photoshop stage being a static GIF and the output from the ImageReady stage being an animated GIF. Tactile images of arbitrary size are supported. For the purposes of a simple tactile *movie*, the size would be 1×8 pixels, while a *haptic icon* (Chapter 6) or other spatial representation can have a greater height. All images represented here are *volt* images.

Chapter 5

Perceptual Characterization

5.1 Introduction

The development of the initial application concepts and handheld tactile display hardware was guided by an understanding of the general capabilities of the lateral skin-stretch technology, and ideas for how it could address user needs in mobile contexts. To proceed to the next stage of more detailed application design, we needed to quantify how users perceive the haptic signals generated by the new hardware. We then mapped some of the regions of the haptic "vocabulary" (range of stimuli that the device could generate), allowing us to assess suitability of the envisioned applications, and what stimuli would best match the roles specified in our concept designs.[1]

We used a similar approach to perceptual characterization as [12]. The core stimulus salience quantification method utilized multidimensional scaling (MDS), a tool for analyzing perception in complex stimulus spaces [21]. Given a range of stimuli, MDS analysis produces maps of how the perceptual space is organized.

[1] A version of this chapter has been published. Luk, J., Pasquero, J., Little, S., MacLean, K., Hayward, V., Levesque, V. (2006). Haptics as a Solution for Mobile Interaction Challenges: Initial Design Using a Handheld Tactile Display Prototype, in *Proceedings of ACM Conference on Human Factors in Computing Systems, CHI '06*, Montreal, Canada, April 2006.

Our new hardware can generate moving stimuli, but the range of detectable movement speeds was not known. We therefore performed a study to estimate this range. This enabled us to select speeds for icons for later MDS analysis.

5.2 Author's Contribution

This chapter was previously published in [34], and represents the collaborative effort of authors on that paper, including Jerome Pasquero, Shannon Little, Karon MacLean, Vincent Levesque, and Vincent Hayward. The author participated in every phase of this research, including software development, haptic icon design, running the experiments, data analysis, presenting and writing up the findings, decision making and project management. All user trials in the subgroup MDS experiment and the initial data analysis for the speed study were conducted by the author.

5.3 Study 1 - Range of Perceivable Stimulus Speed

The purpose of the speed study was to determine the available perceptual bandwidth in one possible dimension that could be used as a parameter for haptic icon design. The question we sought to answer was: "What is the upper limit on the movement speed of a virtual 'shape' that people are able to perceive?" To estimate the range of useable stimulus speed we hypothesized that the users' ability to perceive the movement direction would decrease as speed increased.

5.3.1 Speed Study - Experiment Design

We used a simple moving stimulus consisting of a square waveform that was "tweened" across the tactile display to achieve a sense of motion (Figure 5.1). Two waveforms were used, producing either a moving region of skin expansion ("stretching") followed by compression ("pinching"), or compression followed by expansion. The maximum stimulus speed was limited by the hardware sampling frequency to 3.40 m/s (taking 2.56 ms to cross the display). We conducted a simple pilot study among the authors to determine the approximate appropriate speed range for testing, setting the lower speed bound to a region where stimulus detection accuracy reached a plateau.

The independent variables were: speed (0.17 to 3.40 m/s); direction (up or down); and wave type (stretch-compress or compress-stretch). The dependent variables, measured with a forced-choice method, were: perceived direction (up or down), yielding an accuracy measure when compared to the actual direction, and confidence level (confident or guess).

5.3.2 Speed Study - Procedure

The trials were conducted on a Linux PC with the tactile device attached. On each trial, the computer selected random values for each independent variable. The user pressed a GUI button labeled "Play" to feel the stimulus, which was repeated three times with an intervening delay of 0.7 second. The user was then required to input the perceived direction and confidence level before proceeding to the next trial. There were five "training" trials where the user was informed of the actual direction via a modal dialog box just after entering their responses, followed by 40 "test" trials where the user received no notification.

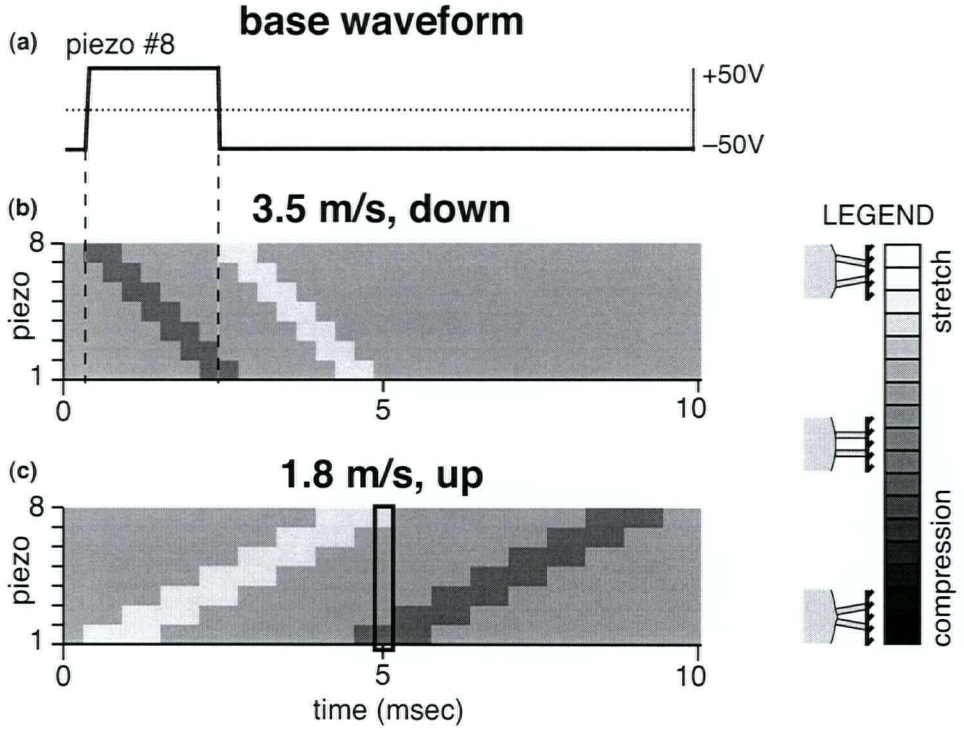


Figure 5.1: Examples of stimuli used for the speed study. (a) Voltage signal for one piezo element. (b) Pattern of lateral skin stretch produced with the 3.5 m/s stimulus. (c) Pattern of lateral skin stretch produced with the 1.8 m/s stimulus. The highlighted area represents one tactile frame in which there is the sensation of stretching and compression at opposite ends of the display.

5.3.3 Speed Study - Results

8 right-handed volunteers (5 male, 3 female, aged 20-40 years old) participated in the user study. Each user took approximately 5-10 minutes to run the study.

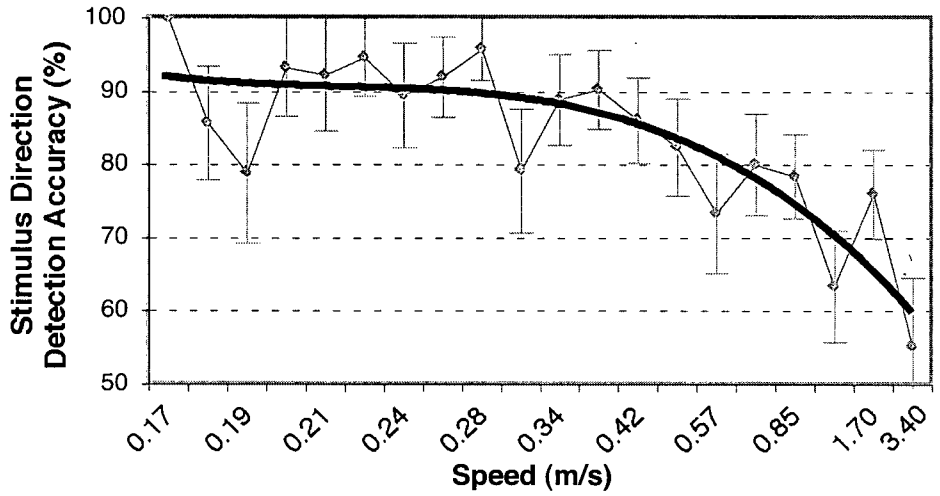


Figure 5.2: Results from the investigation of perceivable range of stimulus speed. The heavy line is the polynomial trend line; measured data points are in grey.

The overall accuracy results from the speed study are shown in Figure 5.2. The relationship of accuracy and speed was statistically significant with (χ^2 =43.00, p<0.01), supporting the experimental hypothesis. Accuracy fell to approximately chance levels at the maximum speed of 3.40 m/s, but approached 90% at 0.34 m/s using a polynomial regression. The measured accuracy at 0.19 m/s and 0.31 m/s appears to be lower than the surrounding data points. While likely due to random variation, this observation is

being further investigated. At the higher speeds, users reported that the stimulus felt like a "click" or small vibration and that direction was difficult to ascertain.

No significant effect was found for wave type ($\chi^2 = 1.87$, $p>0.01$). User-reported confidence level decreased as the speed was increased ($\chi^2 = 165.49$, $p<0.01$).

5.3.4 Speed Study - Discussion

The results from the speed study show that the device is capable of signalling the direction of stimulus movement over a large range of speeds. The sensation experienced is comparable to sliding one's finger across a surface with a small bump. It thus seems feasible to use a directional "tactile flow" signal in applications such as assisted navigation. In addition, the results suggest that speeds lower than approximately 0.34 m/s would be appropriate for designing abstract haptic icons that convey the sense of motion.

5.4 Study 2 - Haptic Icon Discrimination Experiment

The purpose of the haptic icon discrimination experiment was to assess the range and distribution of perceivable difference of some specific haptic icons rendered with this device. The multidimensional scaling (MDS) technique was used to map the organization of the stimulus space.

Factor	Levels
waveform	tri, roll, saw, bump, edge
amplitude	±50V ("full"), ±25V ("half")
speed	0.34, 0.23, 0.17 (m/s)
duration	(Calculated from *waveform* and *speed*) tri: {480, 720, 960} (milliseconds) roll: {221, 331, 442} saw: {86, 130, 173} bump: {74, 110, 147} edge: {74, 110, 147}

Table 5.1: Stimuli used in the MDS studies

5.4.1 MDS Study - Experimental Design

The stimuli were selected according to a 5 waveforms × 2 amplitudes × 3 speeds factorial combination, resulting in 30 haptic stimuli (Table 5.1 and Figure 5.3). These factors roughly correspond to stimulus components used in prior studies for tactile displays [11, 19]. The waveforms were chosen to represent qualitatively different tactile experiences based on first-pass experimentation with different signals, and included both repeating and non-repeating waveforms. For the speed parameter, we chose a range that produced an accuracy rate approaching 90% in the prior speed study.

A fourth "meta-parameter", duration, was calculated from the speed and waveform parameters, and represents the total amount of time the stimulus is present under the user's finger. We hypothesized that this parameter might be perceptually relevant and tracked it in later analyses.

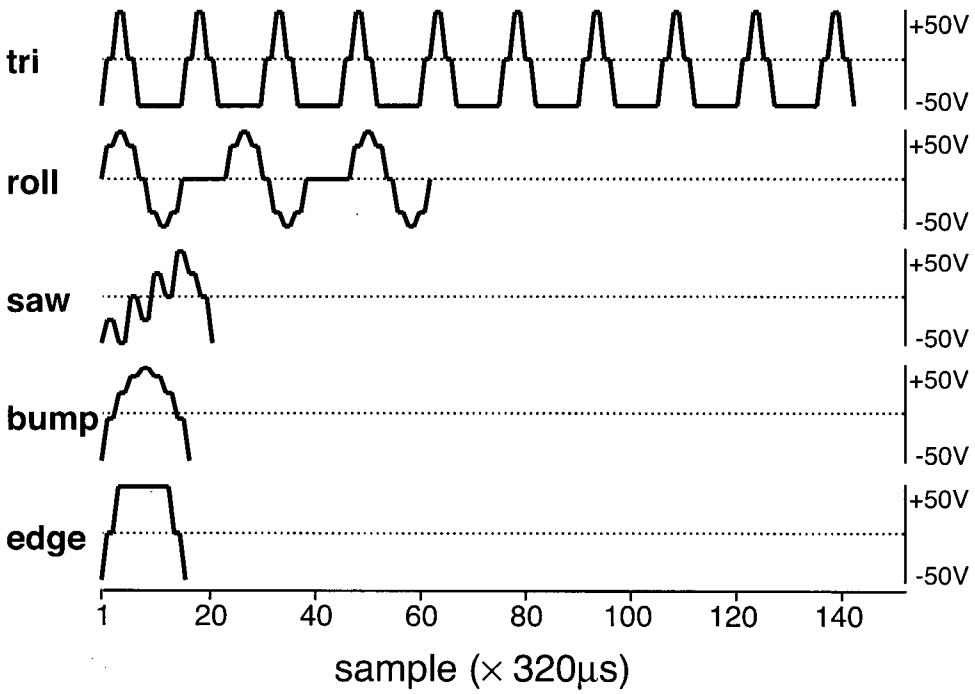


Figure 5.3: Waveforms used in the MDS studies.

5.4.2 MDS Study - Procedure

The participants completed five stimulus-sorting blocks in a method similar to that used in [36] and [52]. The sorting method is a way to efficiently measure perceptual similarity between pairs of stimuli. Participants were seated at a workstation and operated the mouse with the right hand while holding the device in their left hand[2] with the thumb resting on the tactile display. Slider position was ignored. Participants used a GUI that presented the 30 stimuli in a grid of approximately 1 cm^2 tiles. They could trigger

[2] Only left handed operation is supported on the handheld prototype. This is also the case with almost all commercial handheld products having side-mounted controls. See Section 4.1.

stimulus playback by clicking a tile with the left mouse button, and used the right mouse button to pick up, move, and drop the tiles into approximately 7 cm^2 regions on the screen, which represented clusters. On the first block, they could adjust the number of clusters using onscreen +/- buttons. In subsequent blocks, they were required to produce 3, 6, 9, 12, or 15 clusters, presented in random order; the number of clusters closest to the user-selected for the first block was excluded.

We also collected qualitative feedback from users in a post-task interview, seeded with the following questions:

- "How would you describe the tactile sensations you experienced to someone who had not experienced them?"
- "What aspects of the device felt comfortable or natural to use, and what aspects did not?"
- "Can you suggest any applications of the tactile sensations for a mobile device?"

5.4.3 MDS Study - Results

Ten right-handed individuals (7 male, 3 female, aged 19-31 years old) participated in the study, and were compensated $10 Canadian dollars. All subjects completed the tasks within one hour.

We performed an MDS analysis on the data obtained from the sorting task. Stimuli that are sorted together into a cluster were assigned pairwise similarity scores proportional to the total number of clusters in a given sort, because it is reasoned that when a user has more clusters from which to choose, the significance of placing two stimuli together in a cluster is

increased.

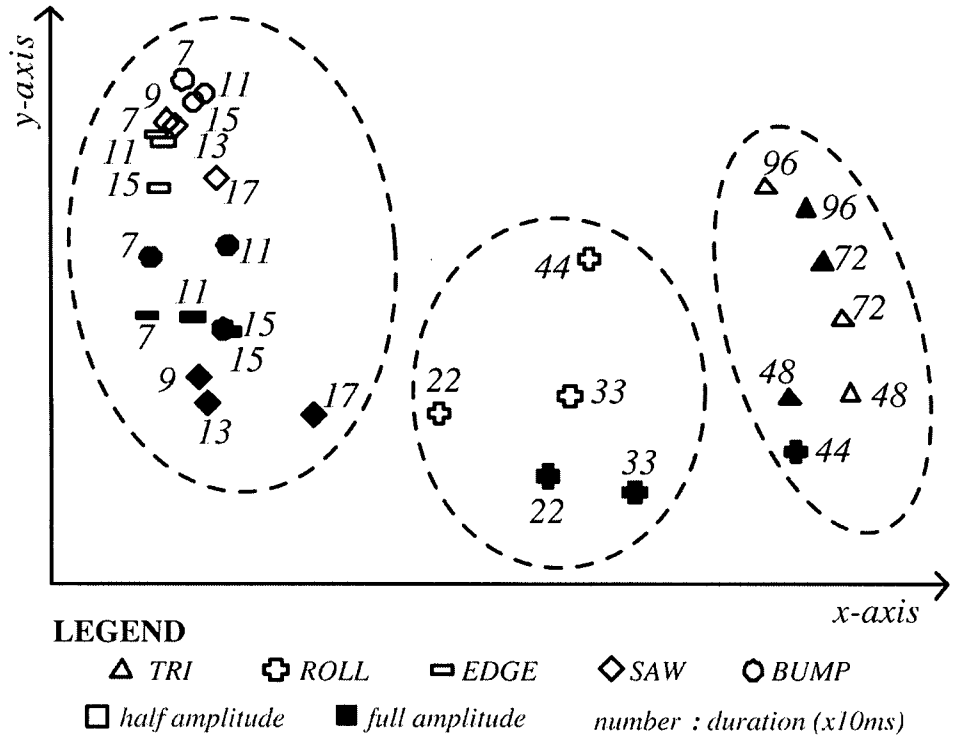


Figure 5.4: Results from the MDS analysis of haptic icons. Each point represents a stimulus, and dotted lines illustrate stimulus groupings. The axes may be rotated arbitrarily.

The results from a two-dimensional MDS performed with ordinal, untied data are shown in Figure 5.4. Analyses in 3-D and higher dimensions did not yield any additional structural information about the data.

The graph clearly indicates that users tend to structure the stimulus space in terms of waveform, with the tri stimuli clearly distinguished, and roll stimuli also being separated from the non-repeating waveforms bump,

edge, and saw.

The stimuli formed by the three non-repeating waveforms — bump, edge, and saw — were less clearly distinguished on the graph, indicating that users did not consistently sort them separately from one another. This suggests that the differences between these waveforms are not perceptually salient, possibly due to limitations of the hardware or skin sensitivity. Additionally, because the experimental paradigm uses relative perceptual data, the dominance of the repeating / non-repeating waveform difference may obscure subtle differences among the non-repeating waveforms [7].

A closer examination of the graph suggests that duration and amplitude may also be salient perceptual dimensions, but their organization in the overall MDS graph is not consistent. However, when subsets of the data were analyzed one waveform at a time, most of the graphs exhibited clear duration and amplitude structure along the $x-$ and $y-$axes. Because the data was collected in a task where users were required to sort all stimulus factors at once, we hypothesized that because the less salient dimensions are perceived qualitatively differently depending on waveform, a global MDS solution was unable to represent them all consistently. We therefore performed an additional experiment to determine the validity of the subset analysis.

5.5 Study 3 - Subgroup MDS Experiment

The purpose of the subgroup MDS experiment was to determine whether more subtle stimulus factors could be detected when the waveform was not varied.[3]

[3]A detailed analysis of the subgroup methodology is published in [42].

5.5.1 Subgroup MDS - Experimental Design

The subgroup MDS experiment consisted of four trials: a control trial similar to the first MDS experiment, and three subgroup trials where users performed sorting tasks using individual waveform subgroups. We chose the tri, roll, and edge waveforms for further analysis because the earlier MDS analysis and qualitative reports indicated that they were judged to be the least similar.

5.5.2 Subgroup MDS - Procedure

To avoid fatigue resulting from the increased number of trials, we reduced the number of sorting blocks per trial. For the control trial with 30 stimuli, subjects performed three sorts with a user-selectable number of clusters in the first sort, and 5, 10, or 15 clusters in subsequent sorts, presented in random order with the number of clusters closest to the user-selected number of clusters in the first sort excluded. For the waveform subgroup trials using 6 stimuli, after the first trial the clusters were 2, 3, or 4 clusters using the same presentation and exclusion criteria. The control trial was presented first, followed by the three waveform subgroup trials in random presentation order. All other data collection methods were the same as in the first MDS experiment.

5.5.3 Subgroup MDS - Results

Five right-handed people (3 male, 2 female, aged 19 to 35) participated in the subgroup experiment. None had participated in a previous experiment with the device. Participants were paid CAD $20 for a 90-minute session.

The subgroup MDS results confirmed the findings from the earlier sub-

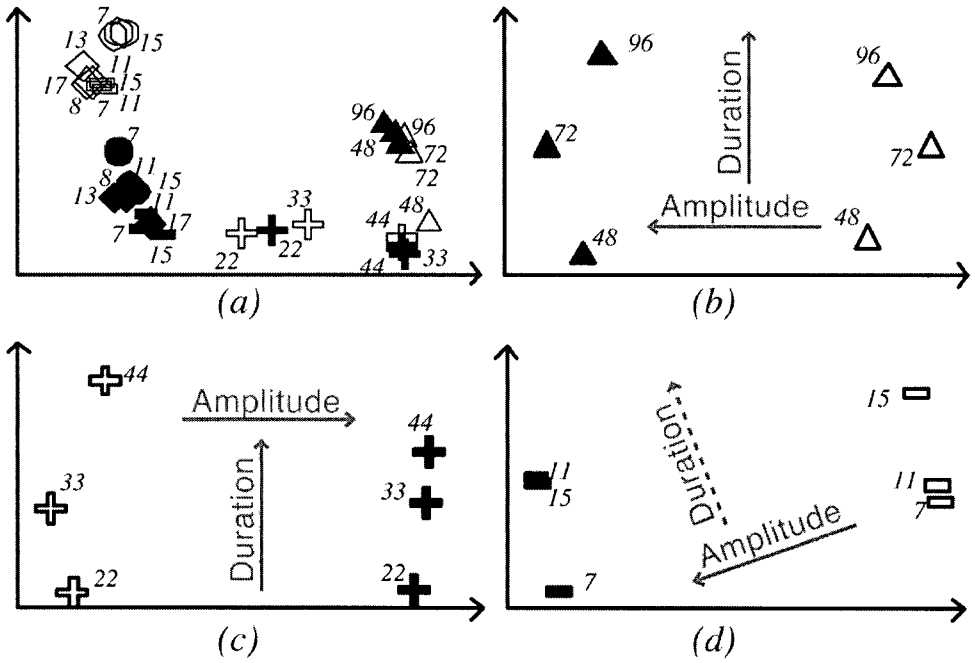


Figure 5.5: . Results from the subgroup MDS study. (a) Control trial with all 30 stimuli. (b) tri stimuli (c) roll stimuli (d) edge stimuli. The results exhibit organization along the dimensions of duration and amplitude.

set analysis, with duration and amplitude being clearly employed by users to organize the stimulus space. Figure 5.5 indicates no clearly discernible duration/amplitude organization in the control trial graph with all 30 stimuli, but when individual waveforms were tested separately, the organization became apparent. In the subgroup graphs, duration is aligned vertically and amplitude horizontally.

Additionally, the data from the control trial exhibited the same overall structure as the data from the first MDS study, providing further confirmation of the original results and the robustness of the technique despite differences in the number of clusters used in the sorting task. Taken together, the results indicate that duration and amplitude, while secondary to some differences between waveforms, are nevertheless discernible and useful as salient parameters for haptic icon design in this environment.

5.6 Summary of Perceptual Characterization Findings

The results from the three perceptual characterization studies suggest that users are capable of distinguishing a wide variety of stimuli produced by the hardware prototype. Direction, certain waveforms, duration, and amplitude are salient parameters that may be used in designing haptic icons for use in applications. The three-way grouping we observed among waveforms was especially interesting, because it empirically suggests how our first-pass parameterization model of haptic icons could be improved; for example, instead of treating waveform as a single parameter, in subsequent designs one could consider non-periodic versus periodic waveforms, and further subdivide the periodic group into different wave shapes (e.g., tri versus roll in the

present experiment).

5.6.1 Qualitative Findings

During user evaluation we were also able to learn how people perceive the device qualitatively. This information is especially useful for determining how users would perceive the value of the proposed applications. The key findings are summarized as follows:

- Universally (N=15/15), participants did not find the stimuli annoying or disruptive. Many participants reported that they preferred them to their mobile phone's vibration mode. A variety of reasons were given, including quiet operation and moderate stimulus amplitudes.

- Many (N=8/15) participants volunteered that they would find this type of tactile stimulus useful for alerts and notifications, such as identification of who is calling, information about a waiting message, or an alarm.

- Some (N=3/15) participants experienced mild tactile fatigue, usually expressed as numbness, which was overcome by repositioning the finger to use a different part of skin, or taking a brief (approx. one minute) break.

- In general, participants said they found the device comfortable to hold and ergonomically suitable for the tasks. Since the sliding function was not used in the perceptual characterization studies, it is not known whether this report would be affected by using the slider for input.

5.7 Perceptual characterization findings for application design

With some quantitative and qualitative data on low-level user perception of the prototype device, we can now consider whether the applications originally envisioned for the device are indeed appropriate, and to proceed with the next steps of application design.

5.7.1 List selection

Judging from the results of the perceptual characterization, haptic icons designed along the dimensions of waveform (periodic or non-periodic), duration, and direction are candidates for distinguishing items in a list. Because the most salient parameters are the direction and speed of the stimulus, it is important to decouple this rendered motion from illusions of relative stimulus motion generated as a result of the voluntary thumb movements to produce control input to the system. One way of avoiding this confound is to signify a discrete command such as scrolling an item up or down with a larger but mechanically grounded gesture that incorporates pressing the slider against an end-stop.

5.7.2 Scrolling

As originally envisioned, the browsing application uses rendered speed and direction parameters to provide haptic feedback to the user about the movement of the point of focus within the page. Haptic shape (waveform) is the only parameter available to provide information about the selected item (link, image, heading, etc.) However, the two MDS studies suggest that the

user's ability to discriminate haptic shape with this device may be somewhat limited when using non-periodic signals. It is possible to build and test the browser application using the currently identified set of haptic icons, but its usefulness may be limited by the relatively narrow choices of icons. Alternative next steps include (a) reiterating the haptic icon design and perceptual characterization stage to discover more choices for haptic shape; (b) re-examining the rendering method and electronic I/O characteristics to minimize electronic and mechanical filtering that may be reducing the resolution and bandwidth of the haptic signal output; and (c) reconsidering the mechanical construction of the tactile display itself with the aim of further amplifying signal strength and thus, presumably, the potential distinctiveness of different waveforms.

5.7.3 Direction signalling

The location-finding application concept relies on the tactile display's ability to convey direction information to the user. The user studies confirmed that direction of tactile flow is clearly distinguishable across a useful range of speeds. Intensity, waveform and rhythm of repeating stimuli may be used to provide additional information about the distance to the target, status, or movement of the target. Our results thus encourage prototyping and usability testing for this application according to the original design concept.

5.7.4 Alerts and background status indicators

User feedback obtained during interviews following the perceptual characterization sessions indicated strong potential for using the device for alerts,

based on the judgment that it would be pleasant and non-intrusive compared to currently available vibrotactile displays.

Data from the perceptual characterization suggests a hierarchy of salience that could be mapped to the relative importance or urgency of an alert. For example, a periodic signal would be useful for important alerts due to its high saliency. Less important changes in background status, such as the movement of passively monitored "buddies", could be conveyed with non-repeating signals.

Finally, if background status indicators are to be multiplexed with other haptic signals generated by the foreground (currently in-use) application, one of the dimensions identified in the user studies could be allocated for this display. For example, if the speed dimension was allocated to background status indicators, slow moving stimuli could be used for the foreground application, while fast-moving stimuli could indicate background alerts. However, because of the limited set of currently known salient haptic stimuli, it would be advisable to perform another iteration of haptic icon discovery before allocating a large chunk of the "vocabulary" to background indicators.

Chapter 6

Browser Prototype

Having constructed a hardware prototype and characterized its expressive capabilities in the hands of users, we were ready to begin prototyping and testing one of the application concepts described in Chapter 3. We chose the **browser** concept to investigate first, because it is the most generalizable in terms of an abstract scrolling / navigation model, unlike more specialized interfaces such as the music player. A browser with tactile enhancement is also a significant departure from existing approaches, unlike the notification scenario, which could be considered an evolutionary improvement of existing vibrotactile notifications. Finally, the concept of simulating the tactile sensation of spatial movement using *tactile flow* rendering, as an aid to cursor navigation, has not been previously studied.

Two levels of browser prototypes are described in this chapter. In Section 6.2 we describe a simple *image browser*,[1] allowing viewing and scrolling of an image with an associated "tactile track", and an informal user evaluation conducted to roughly determine the expected user experience with a higher fidelity prototype. The remainder of the chapter is devoted to the design and implementation of the high-fidelity *tactile web browser*, which is

[1]The image browser was implemented by Vincent Levesque and Jerome Pasquero based on a collaborative design, and the user study was conducted by the author with contributions to the experimental design from the McGill collaborators and thesis supervisor. The image browser is also described in [42].

capable of reading web pages, automatically generating haptic maps based on web content, and supporting interactive navigation with both visual and haptic feedback. A formal user evaluation of this browser is described in the following chapter.

6.1 Design Goals

The purpose of building the browser prototype is (1) to explore one of the aforementioned application areas in depth, (2) to discover what challenges exist for implementing effective haptics in this application area using the piezo tactile display device, and (3) to assess its effectiveness with usability testing.

As mentioned in Chapter 4, the aim was to develop, test, and validate an application in the handheld, tethered environment before going to the much more difficult and expensive mobile, untethered context. As such, the browser software does not need to be concerned with wireless issues and variable content delivery relative to signal quality. Web standards such as HTML, CSS, and JavaScript are used instead of wireless standards such as WML, MIDP, and Brew. While this decision potentially limits the untethered capabilities using the current browser architecture, it also greatly simplifies the design and eliminates costly technology licensing, allowing use of a fully open source software architecture.

Using Web standards means that the browser is capable of interacting with live content from remote servers. However, for the purposes of usability testing, locally cached web pages are used. The browser uses standard web formatting and display constructs, but does not attempt to automatically reformat pages originally designed for PC screens. Just as with mobile

devices built around existing Web standards, content creators will use the same techniques for authoring but must optimize their content for specific devices. Design for tactile browsing can be considered an extension of this process.

6.2 Low-Fidelity Prototype: Image-Based Browser

A prototype browser was implemented that takes a graphical image as input and allows interactive scrolling of the image, with synchronized tactile output. The software was tested informally with users.

The image browser is described in detail in [43]. Therefore, it is only briefly described here.

6.2.1 Design

The design concept, described previously in Chapter 2, is the work of the author. A key component of this design is the provision of a *tactile track*, represented graphically alongside the rendered page, with icons spatially arranged at locations corresponding to the vertical positions of elements such as headings, links, images, text, etc. (Figure 6.1). Furthermore, the height of the icons represented in the tactile track corresponds to the height of the page elements. When the user scrolls the page, the tactile track moves in synchrony with the page. These concepts were used to build the image-based browser.

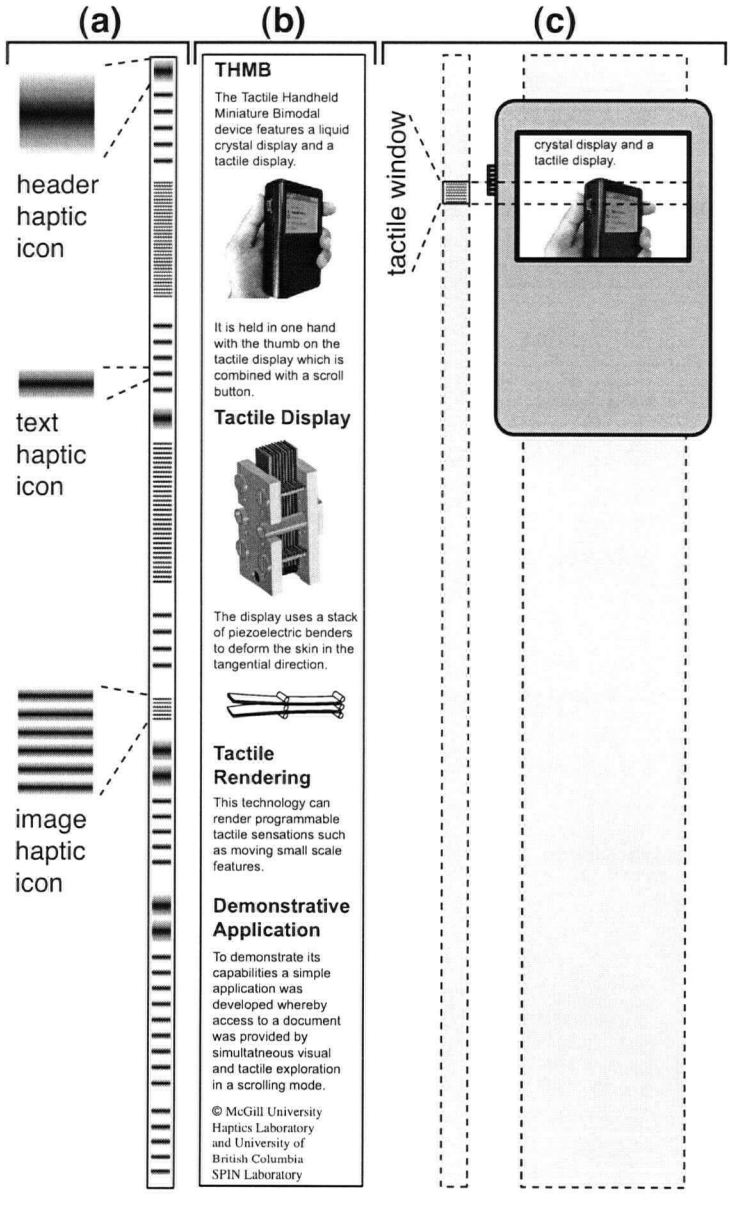


Figure 6.1: Overview of main features of the image browser. (a) "Tactile track" with callouts for icons. (b) Source image. (c) The viewable area is determined by the screen size and height of the tactile display ("tactile window"). Diagram by Jerome Pasquero.

6.2.2 Implementation

Implementation was done by the team at McGill, principally Vincent Levesque. The image browser application is implemented in C++ and uses the GTK library [3] for graphics display, and the STReSS library (Section 4.7) for TD input and output.

The browser software is capable of generating its own haptic icons parametrically. It supports three types of icons, whose positions and types are manually encoded in the input image:

1. Headers
2. Body Text
3. Embedded Images

In the image browser prototype, embedded images are represented by a high-frequency "grating", body text by sparse "dots", and headers by either two closely spaced "dots", or a single dot with a superimposed high frequency grating with adjustable magnitude. Based on informal user evaluation of the stimuli, the two-dot configuration for headers was chosen as the best candidate for being distinguishable from the other stimuli.

Scrolling is accomplished by pushing the slider up or down, which causes the page to continue scrolling as long as the slider was held in that position, in proportion to the distance from the centre of the slider travel (*velocity control*). For this test, the slider was equipped with a spring-return mechanism that centred the slider in the middle of its travel. There was no action associated with the TD's "click" mechanism.

6.2.3 Image Browser User Test

An informal user evaluation was carried out by the author to collect impressions from people who were unfamiliar with the device. The full results are reported in [43]. Evaluation with seven participants produced the following findings:

1. While the high-frequency "image" icon and the low-frequency "body text" icons were clearly distinguishable, 6 out of 7 participants had difficulty distinguishing the "header" icon, even when it was displayed graphically (in the tactile track) in a way that was visually distinguishable from the other icons, and they were asked explicitly to attend to the number of stimuli they could distinguish.

2. All (7/7) participants said they wanted to be able to access the ends of the page which were inaccessible due to the tactile window being fixed in the middle of the visual scrolling window.

3. 4 out of 7 participants reported that they could easily perceive the direction of movement of the tactile stimuli, and that it was in synchronization with the movement of the page.

4. 3 out of 6 participants reported frustration with the use of the slider mechanism for scrolling and targeting. One participant said that instead of velocity control, it would be nice if the device worked like a jog dial, supporting repetitive pushing motions in position-control mode

5. Participants' free-response suggestions included being able to make selections by pushing down on the TD ("click" action), page flipping,

eyes-free browsing, and notification.

6.3 Haptic Display of Web Pages

The previous section described a browser which operated on graphical *mockup images* of web pages, with the accompanying tactile stimuli being generated algorithmically from one of three templates, with placement and type manually encoded in the image. Link selection, reading of web source files (HTML, etc.), and other typical features of a web browser were not supported. The goal of the image-based browser was to serve as a first-pass low-fidelity prototype to obtain early user feedback as part of the ongoing process of developing the browser application.

We will now discuss an automated method for creating a haptic representation of a web page from its HTML source files. The key features of this approach are:

1. Simultaneous visual and haptic rendering of HTML elements.
2. An object-oriented, extensible framework for the haptic page map.
3. Support for an arbitrary variety of haptic icons associated with page elements.
4. Support for animated (i.e., time-varying) haptic icons.

6.3.1 The Haptic Page Map

The output of the rendering algorithm is a *haptic page map*, which is a *model* (in the object-oriented programming sense) of the page in the haptic domain. (Figure 6.2) In the haptic page map, icons are laid out spatially

along a one-dimensional *page map coordinate*, which corresponds to the vertical dimension in the visual representation of the web page. [2]. During navigation, the user moves a cursor along the page map coordinate, as described in section 6.4.

6.3.2 Mapping Haptic Icons to Page Elements

Page elements, such as links, images, and headings, are associated with haptic icons according to the following algorithm:

1. If the element contains the HTML `name` attribute, that property is used as the haptic icon filename, followed by an `.xml` extension.

2. Otherwise, the default icons `link.xml`, `image.xml`, `h1.xml`, `h2.xml`, etc. are used.

This approach allows page authors to specify haptic icons easily within the HTML markup. For example, the XHTML tag:

```
<img src='coupon.jpg' name='hapticonA' \>
```

places an image in the page with filename `coupon.jpg` and associates it with the haptic icon `hapticonA.xml`.

Combined with the animated GIF based workflow described previously in Section 4.9.6, this HTML-based markup scheme means that no special tools are required to add haptics to an existing web page, and page authors can leverage their existing skills in graphics creation and HTML markup.

[2] A coordinate transformation may apply in converting between graphical vertical coordinates and page map coordinates. See Section 6.4.10.

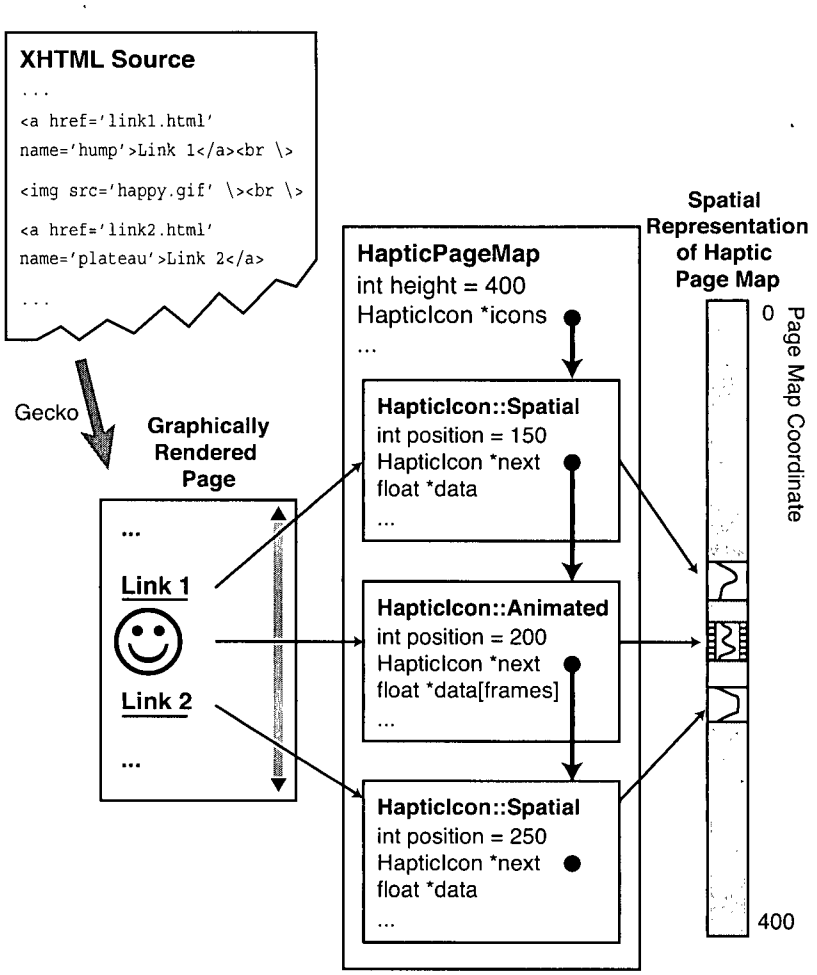


Figure 6.2: The (X)HTML source is rendered graphically by the Gecko engine and haptically using the `HapticPageMap` model, shown here in a simplified schema. The resulting spatial arrangement of icons is also shown. The `HapticPageMap` object includes a list of `HapticIcons`, which may be of type `SpatialIcon` or `AnimatedIcon`. Together, the objects form a complete representation of the web page in the haptic domain.

6.3.3 Spatial Layout

A simple spatial layout algorithm is used to assign positions to the haptic icons. The haptic icon's top coordinate corresponds to the top pixel-coordinate of the rendered element in the graphical representation of the web page.

Various approaches to deal with the case when multiple page elements are laid out horizontally, and thus occupy the same vertical space on the graphic display, were tested. There is functionality built into the layout engine for dynamically reconfiguring the haptic icons to avoid overlap Initial experimentation with this approach revealed that the mismatch between graphical and haptic representations created unsettling navigation inconsistencies. At the same time, it was found that horizontally laid out elements are generally rare on small screen displays, especially mobile phones with a "portrait" (as opposed to "landscape") oriented display. Taking both these factors into account, it was decided not to take advantage of the horizontal layout functionality in the final prototype.

Currently, the layout algorithm does not make use of the rendered pixel-height of the element; haptic icons are not dynamically resized to match the visual element height. This was considered a low priority for links and headings, since their heights are usually specified globally in a style sheet and do not often vary from element to element.

Both the variable-height haptic icon and horizontal layout features would be rather trivial to implement in the current framework, and represent opportunities for future improvement (Section 8.3.4).

Further details of the haptic page map generation algorithm are provided in Section 6.5.

6.4 Navigation Model

This section describes the user interaction model for the web page and its associated haptic page map.

6.4.1 Cursor Position

The navigation model maintains a cursor position in both the graphic and haptic domains. The coordinates may be the same, or they may be related by a linear transformation, as described in Section 6.4.10.

The haptic cursor position is logically an integer number in the haptic page map coordinate space. Signals are rendered to the TD from a portion of the page map that is centred on the cursor position. This is known as the *tactile window*. The height of the tactile window is normally eight units high but may be increased with subtaxel rendering, described in Section 6.4.10. Because there are an even number of piezos, by convention the cursor position is rendered slightly above centre, or piezo #3 if no transformation is applied.

The graphic cursor is logically a single row of pixels comprising the vertical point of focus. It is displayed as a semi-transparent yellow bar, achieved with the alpha channel features of the PNG image file format, centred on the logical cursor position, with height corresponding to the height of the tactile window in the current coordinate transformation scheme; it is eight pixels high when there is no transformation. Therefore, the graphic cursor provides the user with a visual display of the area of the page which is currently "visible" (or, more accurately, *tactable*) through the tactile display.

The graphic and haptic cursor positions are kept synchronized in software. The term *cursor position* may be used to refer to both.

6.4.2 Rendering Haptic Icons

As mentioned in the previous section, the *tactile window*, or portion of the haptic page map that is "viewable" by the tactile display at a given position, is rendered to the TD continuously. The following algorithm is used:

1. Every I/O loop cycle, the program traverses the list of haptic icons that is linked from the `HapticPageMap` object.

2. If a haptic icon is found whose extent lies within the tactile window (i.e., is *within focus*), the object's rendering function is called, which renders the icon (places voltage values) to a buffer.

3. The loop repeats until all icons have been checked.[3]

4. The portion of the buffer that is within the tactile window is output to the device.

Therefore, the case of overlapping icons (normally an error condition), icons that are lower in the list of icons may overwrite their predecessors.

Animated icons are handled implicitly by the rendering function of the `HapticIcon::AnimatedIcon` object, which maintains a real-time counter and renders different "frames" when it is called at different times. Under the current implementation, the counter starts from the first animation frame when the icon is first brought into focus (its extent is overlapped by the tactile window), but is not reset subsequently; the icon's animation may be

[3]Since the number of haptic icons is small (typically, less than 20) and is limited by the amount that the user is willing to scroll, the overhead associated with a simple iteration through the list of icons is negligible on the current system. Further improvement could be made by using a sorted data structure, such as a tree, to reduce search cycles.

considered to continue playing in the background during the time it is out of focus.

6.4.3 Page Element Focusing

The Mozilla browser keeps track of a focused, or highlighted, page element. If the user presses the selection button (usually a mouse button, but handled abstractly in the current implementation), the focused element is selected. For example, in the case of a link, the browser loads the new link.

The page element that intersects with the graphical cursor position receives the current focus. In simpler terms, the user can highlight an item by positioning the cursor over it. Because the cursor occupies a full row, the current design does not support selection of multiple horizontally distributed elements; it simply defaults to selecting the rightmost element.

Because the cursor is logically a single row, but is displayed as the height of the tactile window which occupies eight or more rows, it is possible for the cursor to be partially intersecting an element both in the visual and tactile space without the element being focused.

6.4.4 Graphical Display Scrolling

Because pages are often taller than the graphical display, cursor movement must also control scrolling of the display. A push-to-scroll model is used; when the cursor hits a boundary of the display, the display begins scrolling if necessary. On a PC, this type of model is commonly used in word processors and text-entry boxes, where vertically moving the cursor beyond the display boundary causes the display to scroll one line at a time.

Because the cursor is also being used for focusing, it is desirable for the

user to have some way of looking ahead before hitting an element. Therefore, scroll margins are implemented, as shown in Figure 6.3. From its initial position at the top of the display, the cursor moves down until it hits the scroll margin. If the user continues scrolling down, the cursor's displayed position remains fixed while the page scrolls under it. If the user reverses direction and moves the cursor up, the page does not scroll until the cursor hits the top scroll margin (hysteretic motion). When the display can scroll no more, the cursor moves past the scroll margin and to the top or bottom of the page. In this way, the whole page is accessible to the cursor, unlike the previously described image based browser.

6.4.5 Control of Cursor Movement

In the simplest case, the "throw" of the slider may be mapped to the overall height of the page map, such that if the user positions the slider at the top of its range, the cursor is positioned at the top of the page, and if the user positions the slider at the bottom of its range, the cursor is positioned at the bottom of the page. This is a direct *position control mode*; the cursor position corresponds directly to the slider position. Position control is the model used in jog dials, including mouse scroll wheels.[4]

Early experimentation with the device revealed that there was insufficient spatial resolution to display more than two or three distinguishable haptic icons using direct position control across the short slider range (11 mm). Furthermore, precise positioning would require extremely high dexterity, and may even be beyond the precision of the analog position sensor;

[4] Depending on the application, the mouse scroll wheel may also operate in a discrete pseudo-position control mode, simulating the effect of a user repeatedly pressing a scroll key.

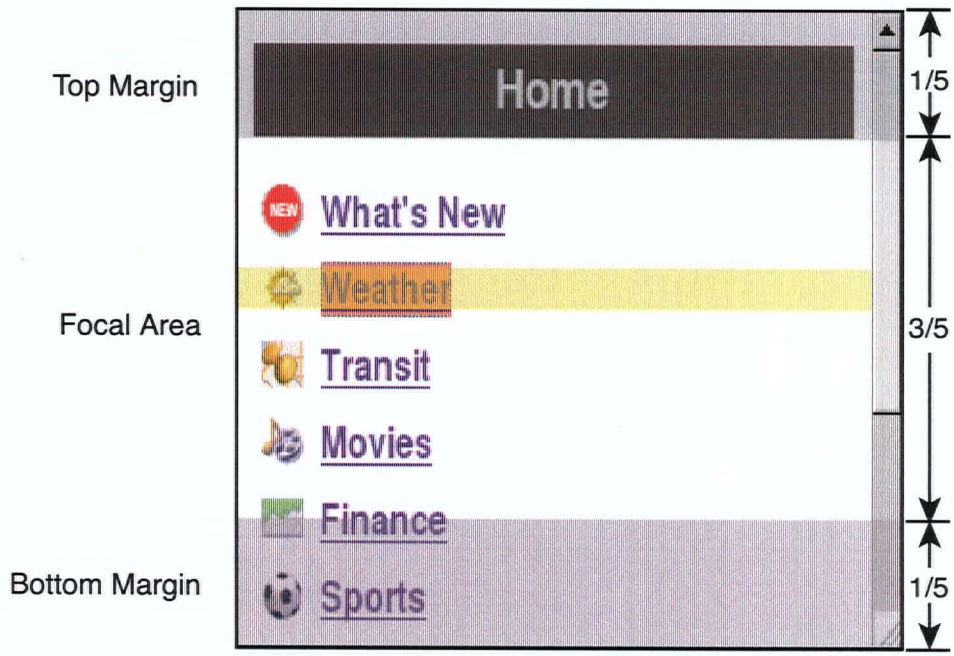


Figure 6.3: Scrolling margins for the graphical display, located at 20% from the top and bottom of the display. The display begins scrolling when the cursor reaches a margin.

for a typical web page 600 pixels high, the necessary resolution would be approximately 1390 dots-per-inch, far in excess of typical mouse hardware resolution. [5]

Because of the limited range of slider motion, it was necessary to use *velocity control*, which is the model used in joysticks and shuttle controls. In the simplest case (pure velocity control), the cursor movement velocity corresponds to the slider position, with the centre of the slider treated as zero velocity. A small "dead zone" may be used in the centre to reduce unwanted cursor motion. This method was used in the image-based browser (Section 6.2). A more sophisticated model was used for the web browser (Section 6.4.7), and a summary of the three cursor control models is shown in Figure 6.4.

6.4.6 Spring return to centre

Typical velocity control input devices have a spring-loaded return to centre mechanism which provides a resistance force when the user pushes on the device to cause motion. Informal user testing of the device with and without spring feedback revealed that users strongly preferred spring feedback when the device was used in simple velocity control mode.

A number of approaches were tried to implement spring feedback, including elastic foam, elastic bands, and compression springs. Due to the mechanical design of the slider, use of compression springs in a coil-over-shaft configuration was the most parsimonious and reliable approach. (Figure 6.5) This required finding springs with a minimum inner diameter of 2.4 mm and appropriate length and spring constant. Springs with appropriate

[5] The Microsoft Intellimouse Optical, for instance, has a resolution of 400 dots-per-inch.[5]

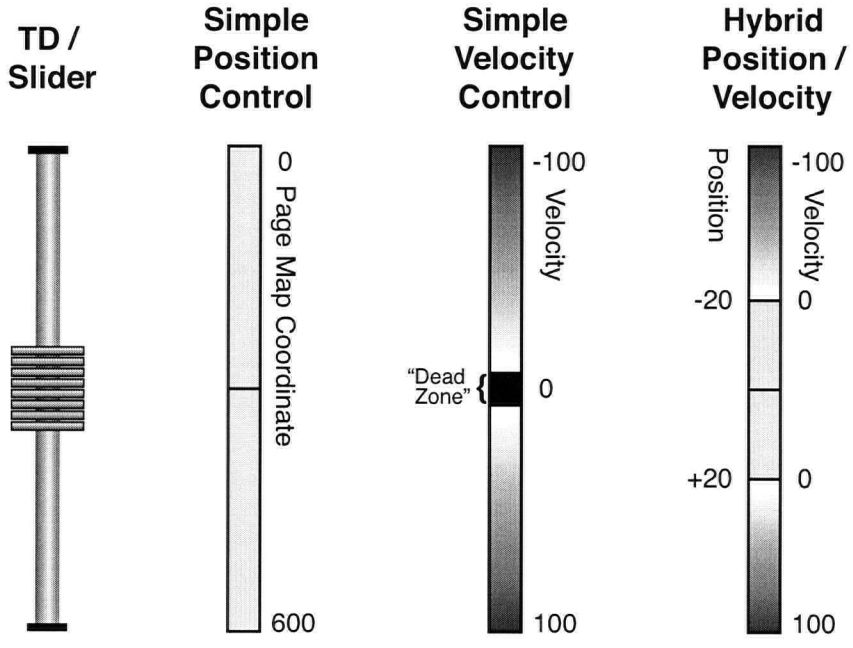


Figure 6.4: Slider control modes. In simple position control, the slider position is mapped directly to the cursor position, e.g., for a page with height 600. In simple velocity control, the slider position relative to centre is mapped to the velocity of cursor movement, with an optional central "dead zone" to make centering easier. In hybrid position/velocity control, a region of the slider travel functions in direct position control mode; if the user pushes the sider beyond that region, velocity control is active.

physical dimensions could be found in common retractable ball-point pens, and 16 different pens were acquired and disassembled to find the one with the most appropriate spring constant. However, due to the fact that typical use of a retractable ball-point pen involves compressing the spring by pushing directly on its end, versus lateral force applied in the case of the TD, most ball-point pen springs were too stiff for use with the device. Consequences of excessively stiff springs included user fatigue and diminished tactile stimulus intensity, due to the piezos being bent laterally against one another by the force against the spring-loaded slider.

Softer springs were found in multi-colour ball-point pens that also have lateral actuation. The softest spring was found in a Mitsubishi four-colour combination pen from Japan, and this spring was cut to size and used in the device. The spring was actually too soft to produce a reliable and accurate return to centre against the friction of the slider and the inertia of the TD; therefore, two slightly overlength springs in a mutually preloaded configuration were used to improve the centering properties. Unfortunately, this also increased the force necessary to push the slider all the way to its top or bottom stop.

This configuration was used in the image-based browser tests (Section 6.2) and early pilot testing of the HTML-based browser. While it was clearly necessary for velocity control (users reported strong preferences for the presence of the springs), extended use caused fatigue due to the stiffness of the springs, and premature failure of the insulating varnish coat due to the strong shear forces between the thumb and the TD. An extensive survey of spring vendors revealed that no softer springs could be found in the 2.4 mm inner diameter configuration, possibly due to metallurgical limitations in the minimum thickness of wire that can be used while remaining in its

elastic zone. Softer springs are available in a torsion configuration, but major re-engineering of the slider mechanism would have been required to accommodate them, and it is unlikely that such low spring rates would be sufficient to overcome the friction associated with the mass of the TD. Due to these limitations, it was decided to remove the springs prior to further testing of the browser prototype, and to defer broad re-engineering of the slider mechanism until the cursor control model could be studied in more detail.

6.4.7 Hybrid Velocity / Position Control Model

Through informal user evaluation, it was found that velocity control alone did not afford the bidirectional tactile exploration that was effective in creating the illusion of exploring small shapes with the Virtual Braille Display [30]. Furthermore, it was desirable to make effective use of the centre of the slider travel for precise positioning instead of simply having a "dead zone".

A hybrid velocity / position control model was adopted to meet these goals, as shown in Figure 6.4. When the slider is in the position control zone, velocity is zero and the user can move the cursor using position control within a local offset range. Outside of this zone, velocity control is active. The actual values of the parameters used are listed in Section 6.4.7.

To the user, the experience of navigation using the hybrid control model is of a small active area in the centre of the slider range within which they can interactively "browse" a haptic icon as if they were touching a small shape, moving the finger up and down to explore its edges. Pushing the TD further causes scrolling motion. If the TD is moved very fast (for example, rapidly from its bottom-stop to its top-stop), the cursor may rapidly move across its

position control range. If the maximum velocity setting is lower than this velocity, there may be a somewhat unexpected bimodal cursor behaviour in which it rapidly jumps and then moves more slowly under velocity control. However, in real-world usage, the user is unlikely to move the slider rapidly from end-stop to end-stop; the more common usage pattern involves moving the slider away from its centre detent and holding it there until the target is achieved, followed by small fine-tuning adjustments in the case of overshoot, undershoot, or local exploration of adjacent elements.

Hybrid Control Mode Feedback and Spring Return

While the hybrid control model allows fine-grained exploration within the central position control zone, users still reported discomfort using the device in its velocity control zone without spring feedback. In addition, in the absence of any haptic cues such as spring force feedback, it was difficult to determine where within the slider travel the mode switch occurred, which also resulted in negative user observations.

To deal with the latter concern, synthetic tactile notification of mode switching was implemented. Whenever the user crossed a mode switch boundary, the current tactile signal was overlaid with a rapid vibrotactile signal consisting of a half-height (50V peak-to-peak) cycle, repeated twice over 4 samples. The overlaid samples were clipped to +50V/-50V if necessary. This provided a "clicking" or "bumping" sensation as the slider was moved across the mode boundaries. The sensation was judged effective in early pilot testing for notifying mode switches, but only if the slider was moved very slowly across its travel. With rapid movement, the user could move the slider across its 11 mm travel within a fraction of a second, causing

the two mode switch notifications to blend into a generic vibratory signal with minimal spatial localization relevance. Furthermore, it did not address the need for proportional force feedback in velocity control mode.

The eventual solution was to adopt hardware springs in a hybrid configuration. Figure 6.5 shows the design after several testing rounds with various materials and thicknesses. Small pieces of latex foam were affixed to the slider stops using industrial double-sided tape. The extent of the foam pieces was approximately one-quarter of the total slider travel each (3 mm). Because of the compliance of the foam, the point at which the slider hits the foam is essentially imperceptible; however, after the foam is compressed more than halfway, it starts to provide perceptible resistance. There is approximately 1 mm of travel within this range, during which the force is perceived as essentially isometric. To the user, the experience is of moving the slider within a free-movement zone causing the cursor to track the slider position, followed by pushing the slider against its stops with variable force resulting in velocity control. Specific parameters were determined after informal optimization with three pilot users, and are listed in the section below.

Hybrid Control Delays

Further testing of the device under the hybrid control system revealed additional areas needing improvement for user comfort. First, when users wanted to stop cursor movement by moving the slider out of the velocity control zone, they typically released the slider, allowing the springs to move it back into the position control zone, or they actively slid the slider back towards centre. In either of these cases, the slider was unlikely to end up

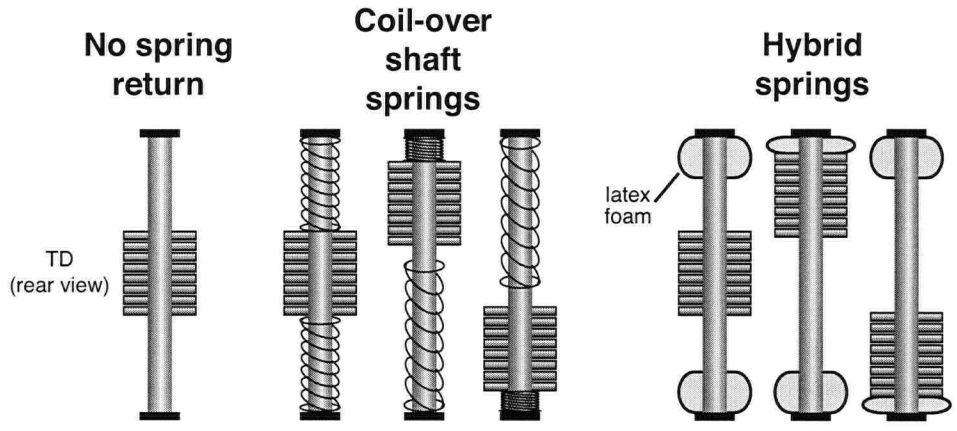


Figure 6.5: Force feedback using springs. The standard coil-over-shaft spring configuration and the hybrid spring configuration is shown in various slider positions.

exactly at the position control boundary; it was more likely to be somewhere within the position control zone. The effect of this on cursor movement was an unsettling "snap back" effect that made targeting difficult because the user would acquire a target using position control, release the device, and the cursor would move backwards a distance equivalent to the distance between the position control zone boundary and the final slider position, thus bringing the intended target out of focus.

The second problem was unintended cursor movement due to accidentally moving the slider into the velocity control zone. This situation occurred when the user was targeting a point just outside of the position control range. The user would approach the target by continuing to move the slider in the direction of the target, cross the zone boundary unintentionally, and cause unexpected continuous cursor movement.

Finally, due to the short velocity control range, and the low amount of force that the user could apply to the device laterally without causing fatigue or piezo deflection, there was a limited dynamic range available for use with velocity control. Users still wanted to move the cursor rapidly to traverse a web page quickly; however, if the maximum speed parameter was increased excessively, there was insufficient precision at low speeds for precise targeting.

These three problems were addressed by adding time controlled elements into the control software. The resulting control system's state diagram is shown in Figure 6.6. To address the first cause of unexpected cursor movement due to "snap back", a delay was implemented in the transition from velocity control to position control. Since the system remains in velocity control during this delay, the rapid return of the slider to its centre detent position simply causes the velocity to drop to zero, "absorbing" the large

negative position displacement. When the user replaces his/her finger on the display and starts moving it again after the delay has expired, the system is in the position control state, enabling interactive local exploration of tactile features.

The second concern of accidentally moving the slider into the the velocity control zone was addressed using a delay in the transition from position control to velocity control. Local exploratory behaviour in position-control mode consists of relatively rapid, back-and-forth small scale sweeps of the virtual surface. To make the transition to velocity control using the delay model the user must "push and hold", a more intentional action, before the system started scrolling under velocity control.

Finally, a linear acceleration model was added to the velocity control to enable increased scrolling speeds if the user held the slider in the velocity control zone for an extended time.

Hybrid Control Settings

After performing the above mentioned adjustment process, the final parameters used in the browser prototype are listed in table 6.1.

6.4.8 Reduction of Slider Jitter

Although the slider uses a high-precision resistive position sensor, there is considerable analog sensor noise. If unfiltered, this noise produces continuous high-frequency piezo motion that completely destroys the ability to render shapes using low-frequency piezo deflections. A historical averaging model was therefore implemented in the software to reduce jitter.

Raw slider input is first passed to a `SliderSmoother` object in soft-

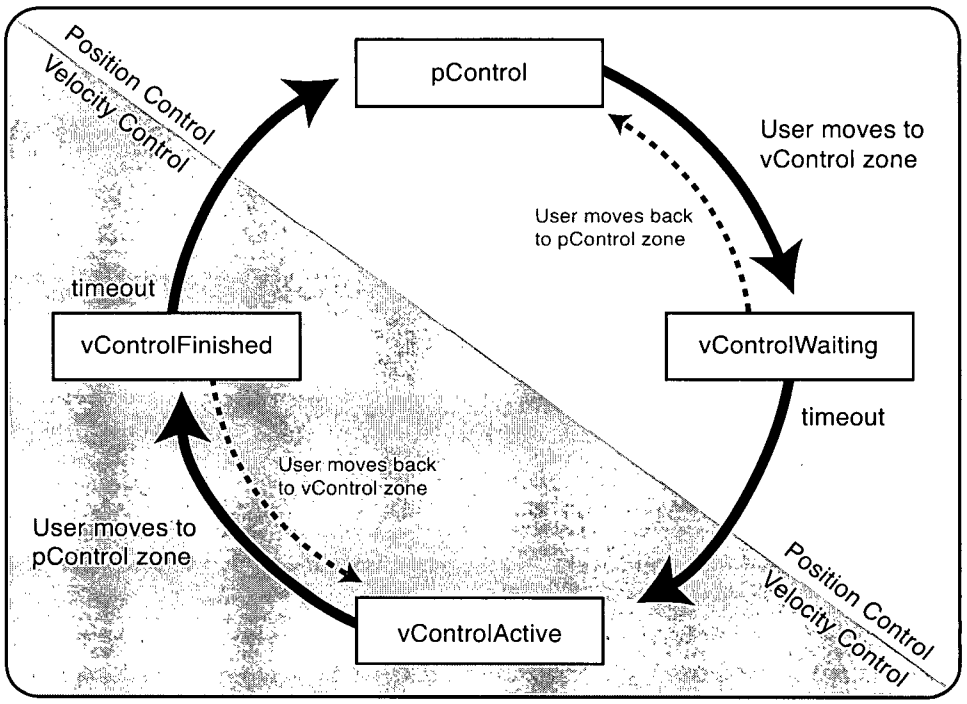


Figure 6.6: State transition diagram for the four velocity / position control states.

ware, which outputs the "smoothed slider position". It takes two parameters: `cache_size` and `threshold`, whose final values are shown in Table 6.2. When the slider is stationary or being moved slowly, an average of the last `cache_size` number of samples is taken and used as the final slider position, thus cancelling out any small, high-frequency noise. For high values of `cache_size` and large slider displacements, this can take a noticeable time to converge on the slider position, which is perceivable as a "rubber band" effect on the cursor. Therefore, an escape parameter, `threshold`, is implemented. If the user moves the slider rapidly, historical averaging is abandoned and the current slider position is used directly. Furthermore,

the cache of values is reset so that averaging re-starts at the current slider position when its velocity drops below the threshold value.

6.4.9 Reduction of High-Amplitude, High-Frequency Outputs

When a user is scrolling very rapidly through the page map, it is possible for two sequential samples to be taken from very distant areas of the page map. When there is a large voltage spread between adjacent samples, it creates a large, sudden displacement of the piezo that is perceived by the user as a rough "clicking" sensation. The high-frequency transients caused by rapid discrete movement of the piezo are especially bothersome, since they tend to result in user reports of undesirable "vibration", "zapping" or "electric shocks". It is also potentially damaging to the piezo motor itself.

To combat this undesirable effect, two measures are used. First, the slider input smoothing algorithm can spread displacements over time using its historical averaging algorithm, as described in the previous section. However, to reduce unwanted "rubber band" cursor behaviour, large slider displacements cause a buffer abort. Furthermore, when velocity control is being used with high maximum velocity settings, high-frequency, high-amplitude actuator transients can be produced even when the slider is held stationary in a high-velocity position. Therefore, an additional *output* smoothing algorithm was implemented to reduce sudden large piezo movements.

The algorithm first checks whether a given piezo actuator will have a large voltage change, as measured between the "current" sample and the immediately previous sample. If the threshold (currently set at 50V in either direction) is exceeded, then the actual output voltage delivered to the piezo

is "smoothed" out by averaging the current sample and the immediately previous sample. (We nicknamed this method "*lookback*" smoothing).

Because this tends to have the effect of smoothing out peaks and therefore limiting the dynamic range of the TD during high speed movement, there is also a *feedforward* component designed to "recover" part of the peak voltage in future samples. When averaging is used to produce the output of a given sample, the sample's original (pre-averaging) value is stored. That value is used to compute the next sample by averaging the cached and current sample values on the next timestep. The threshold test is then applied and, if necessary, the previous *actual* sample voltage is averaged back in using the "lookback" algorithm.

The net effect of these smoothing functions is to create the sensation of rapidly scanning across tactile features, while minimizing aliasing caused by sampling delay.

6.4.10 Speed Limitation

In the perceptual characterization phase (Chapter 5), a maximum speed of 0.34 m/s was found to elicit a reliably discernible sense of directional motion among users. We may use this figure as a reasonable estimate of an upper bound for the speed at which a user can traverse the haptic page map while maintaining a continuous tactile flow percept.[6] At maximum speed, given

[6]The speed study measured non-interactive tactile flow signals, and it is possible that the actual upper speed bound for an *interactive* (i.e., user-modulated) tactile flow signal may be slightly higher. An experienced user may also be able to detect higher speed signals. However, for the purposes of our initial calculation, it is clear that even with a 30% margin of error, requiring the user to spend 8 to 33 seconds to traverse a typical web page is still unacceptably long and requires countermeasures.

the physical dimensions of the TD it takes 30 ms for a haptic icon to move from one piezo actuator to the next.

If spatial layout of the haptic map is performed using a one-to-one mapping between pixels and taxels, it would require 18 seconds at maximum speed to go from the top to the bottom of a typical 600 pixel-high web page. Pages in the testing corpus for the browser ranged from 424 to 1698 pixels high, so it would require considerable time (12 to 50 seconds) to traverse the web page at speeds necessary to stay within the range of perceivable direction.

Two techniques were used to accomplish the decoupling of the scrolling motion on the screen and the TD:

1. Page map shrinking, and
2. Subtaxel rendering

These methods effectively comprise a cascading transformation scheme for the page map coordinates that can reduce the logical height of the haptic page map up to eight times relative to its graphical equivalent. If desired, it may be possible to further increase maximum speed using other techniques in combination or separately. These possibilities are discussed in Section 8.3.4.

Page Map Shrinking

The browser client supports a mode that transforms all page coordinates and requested icon heights to create a page map that is half the pixel-height of the web page. Using this technique, it is possible to double the effective speed limit of scrolling traversal, but because icons must be half-height, they must be simpler, thus limiting the differentiability of icons.

User testing with the page map shrinking feature suggested that the loss of icon fidelity negatively impacted the perception of tactile flow because it required designing more sharp-edged icons. Users perceived rapid changes in the voltage applied to actuators as "vibration" rather than tactile flow. This undesirable perception of high-frequency aberrations is analogous to the problem of harsh edges and other anomalies that occurs in computer graphics when images are downsampled excessively using simple nearest-neighbour scaling algorithms.

Further page map shrinkage beyond a factor of two was deemed impractical because the haptic icon height for a link icon, normally 21 page map units for the corpus of pages used in the user test (Chapter 7), would drop below 10, thus severely limiting the expressive capabilities of the icons and increasing the risk of delivering unintended high-frequency vibrotactile signals to the user.

Subtaxel Rendering

To reduce the movement speed of the tactile stimulus while still retaining good fidelity, a subtaxel rendering method was implemented in the browser server component. Using this method, the internal representation of the page map retains the same height as in the one-to-one case. However, when the signal is rendered to the TD, two or three (depending on a configurable mode setting) taxels from the internal map are averaged to form one output taxel. In this way, the effective "viewport size" of the tactile display (the *tactile window*) increases two- or three-fold, to 16 or 24 taxels, respectively. Because the location of the TD's window into the overall page map is maintained in the internal high-resolution coordinates, smooth movement results.

Parameter	Description	Value	Units
`maxP`	The page map range accessible in position control mode.	50	page map units
`maxV`	The maximum velocity at full acceleration.	200	page map units per second
`posControlZone`	The portion of the slider travel assigned to position control (the remainder is used for velocity control). Determined by the point at which the latex foam begins to provide noticeable feedback.	0.75	(ratio)
`timeBeforeVControl`	The timeout before entering velocity control mode. Used to prevent accidental activation of velocity controlled movement during exploratory behaviour.	0.05	seconds
`vControlAcceleration`	The time between entering velocity control mode and reaching the maximum velocity for a given slider position.	1.0	seconds
`timeAfterVControl`	The timeout before entering position control mode. Prevents "snap-back" of the cursor when the user releases the TD.	0.05	seconds

Table 6.1: Values of hybrid velocity / position control model parameters.

Parameter	Description	Value
`cache_size`	The number of slider values (collected once every 3 ms) to average to find the current slider position.	10
`threshold`	If, within the 3 ms sample interval, the slider is moved a distance greater than this value (in raw 12-bit slider position units), historical averaging is abandoned and the current slider position is used directly.	5

Table 6.2: Settings for the slider smoothing function using historical averaging.

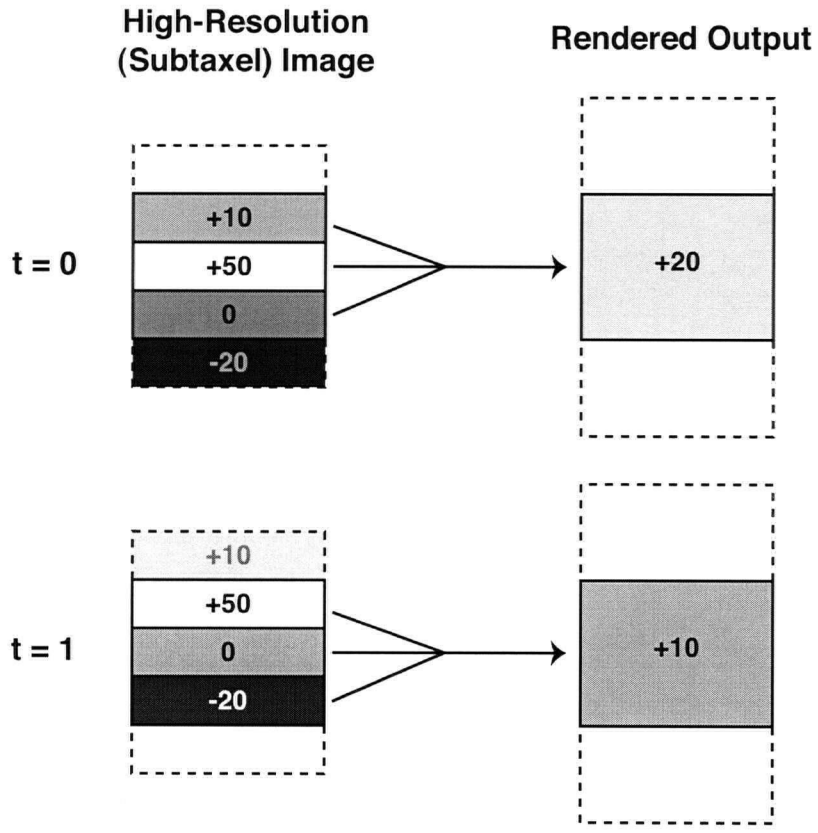


Figure 6.7: The subtaxel rendering technique. In this example, three taxels from the internal high-resolution representation are averaged to form one output taxel. The stimulus is moving up across the display, and two time samples are shown.

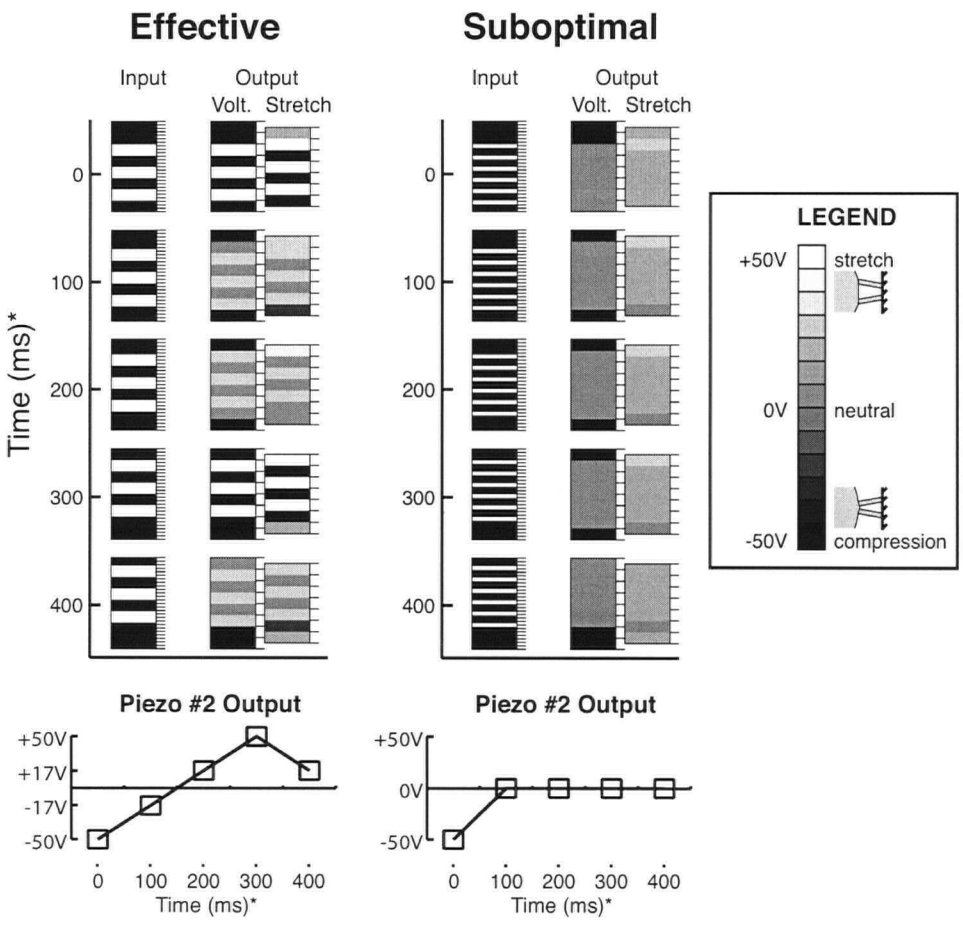


Figure 6.8: Effective icon design for subtaxel rendering. High-resolution page map representations are shown in the left column of the group, and the output of the subtaxel rendering algorithm is shown on the right column in both raw voltage signal and skin-stretch representations. Five time samples are shown as the stimulus is tweened upwards across the display at a constant speed, illustrating how the subtaxel algorithm provides fine spatial information in the form of temporal signal changes. Note that exceeding the resolution limit of the TD when designing high-resolution icons tends to create blurry results. Below the pictures, the output from one piezo (#2 from the top) is graphed over time. In the case of a well-designed icon, the piezo reaches full-scale values.

The method is, in effect, using temporal data to encode spatial data — using the high temporal resolution of the sense of touch [19] to compensate for the low spatial resolution of the tactile display device. The technique is shown in Figure 6.7.

Because data is lost during the downsampling, haptic icons must be designed carefully to ensure that they create the intended sensations. Figure 6.8 illustrates some examples of effective and suboptimal haptic icon design with respect to downsampling. By analogy, when visual images are heavily downsampled to low pixel counts using averaging, they tend to lose contrast and become blurry, and the same danger applies to the haptic icon images. Especially since the sensations produced by the TD tend to be quite subtle, it is important to retain as much dynamic range as possible when designing icons.

Fortunately, the effects of downsampling may be quickly previewed during icon design in Adobe Photoshop by scaling the image by 33% and using the "bilinear" setting, which is equivalent to the averaging method used by the software.

6.5 Browser Software Architecture

The browser software architecture is shown in overview form in Figure 6.9. A logical flow diagram is also presented in Figure 4.6. Briefly, the principal components of this architecture are as follows:

1. A **client component**, built on the open-source Mozilla API, that handles parsing of the HTML and CSS source documents, generation of the page map XML file, keyboard and mouse input, graphical display, loading of new pages, and logging data for the usability evaluation.

Described further in Section 6.5.1.

2. A **server component**, implemented in high speed C++ code and running a continuous I/O loop, generating the tactile signals from its internal representation of the page map, and translating analog slider input to page map coordinates and forwarding the input to the client component. Described further in Section 6.5.2.

3. **Interactive haptic icon design tools**, previously described in Section 4.9.6.

4. (Not shown on the diagram) **Web page generation tools** for creation of HTML and CSS source content. Haptic markup is optionally added to the HTML source as described in Section 6.3.2.

5. A **file-based interprocess communication system** using "position" and "click" files to transmit information from the server component to the client component, and a "flag" file for the client component to signal to the server component to load a new `HapticPageMap`. Described further in Section 6.5.3.

Author's Contribution

Initial work on specifying the architecture of the browser client was performed by undergraduate research student Shannon Little with support from the author. The current XPCOM and XUL components share approximately 50% of the code from that effort [33], which is in turn significantly derived from online XPCOM and XUL tutorials; the current JavaScript component, where the majority of functionality is implemented, shares less than 10% of code.

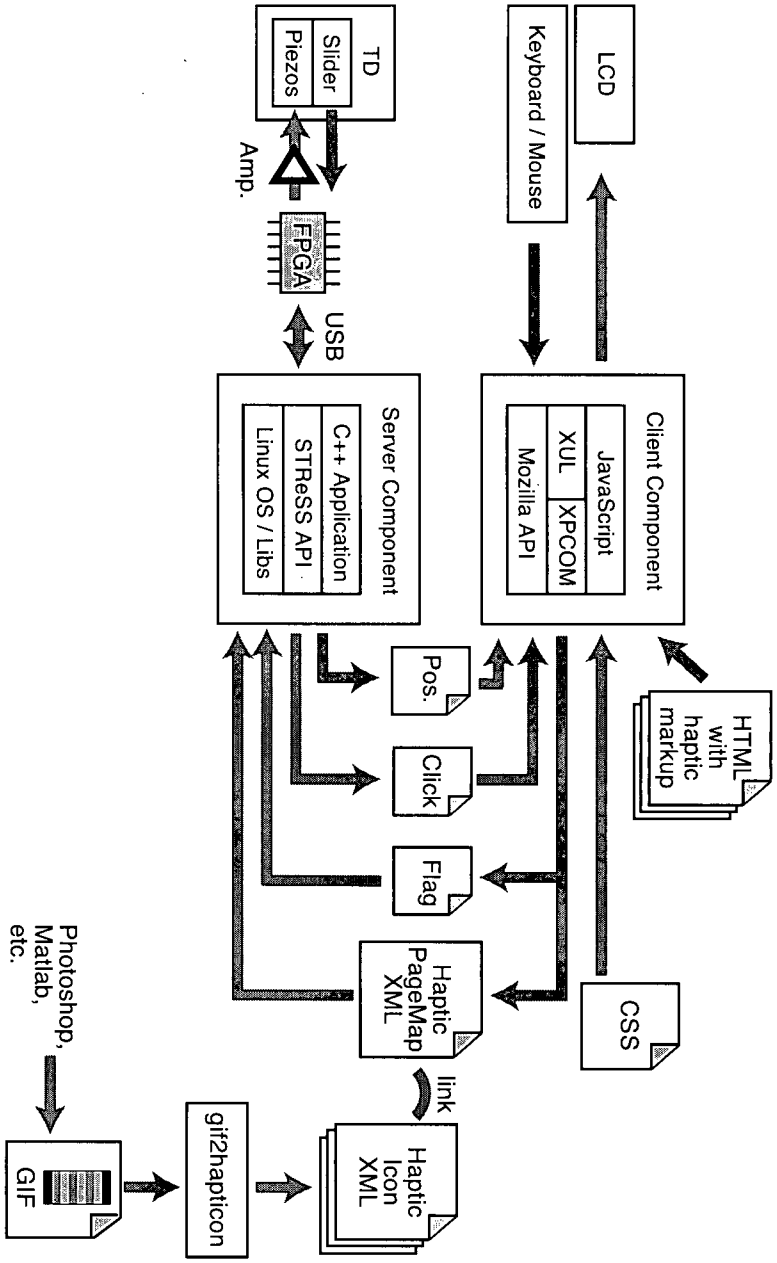


Figure 6.9: The browser software architecture, including software and hardware components and the data that is exchanged between them.

The I/O portion of the browser server is based on example applications provided by Vincent Levesque, a graduate student at McGill university, as documentation for the STReSS Library [29] API. Earlier examples of the "THMB Library" co-written by Vincent Levesque and Shannon Little based on this code were not used; the code was rewritten from scratch for more precise timing control.

Where open-source software was used, it is mentioned in the following sections.

All other portions, including the aforementioned haptic page map and navigation functionality, cursor display, usability testing support, page content, and style sheets are the work of the author.

6.5.1 Browser Client

The browser client component is built on the open-source Mozilla API and its Gecko rendering engine. Since Mozilla is the most popular extensible, open-source platform for web page browsing as of this writing, it ensures good compatibility with Web standards and good portability of the software. The browser client component consists of:

1. A JavaScript application,
2. An XPCOM (Cross Platform Component Object Model) component,
3. A XUL (XML User Interface Language) "chrome" component, and
4. The Mozilla API, its Document Object Model and security modules, and the Gecko parsing and rendering engine.

Most of the browser client component is implemented in the JavaScript application, because it has access to the Mozilla browser's Document Ob-

ject Model (DOM), allowing read access for the spatial layout of web page elements, HTML tags including specialized haptic markup, and mouse and keyboard events. The JavaScript application also writes to the DOM when it loads a new page, displays the cursor, highlights elements that are in-focus, or provides usability testing related displays to the user. Data-logging is accomplished using the standard terminal `dump()` command from JavaScript. The main I/O loop consists of reading the slider position and click count (via the XPCOM layer, described below), updating the cursor position based on the slider position, and comparing the click count to a previous value — if an increase is detected, the highlighted page element is selected.

The XUL layer consists of small wrapper code to run the JavaScript application within the Mozilla framework while suppressing standard PC browser features such as toolbars.

The XPCOM component handles file input and output, which is normally unavailable to JavaScript because of security restrictions. It consists of various functions to read the position and click files, and write the `HapticPageMap` file and its associated update-flag file. The file input and output functions are implemented in C++, and a compiled IDL (Interface Description Language) wrapper is used to create the abstract function interface that allows JavaScript to call the C++ functions.

Finally, the Mozilla API itself is based on Mozilla 1.01 browser, and is the work of various open-source authors.

6.5.2 Browser Server

All tactile device input and output routines are encapsulated in the browser server component, which is a separate executable running at the highest

priority available in the Linux kernel. This architecture ensures rapid, continuous updating of the tactile device, reducing the risk of irregular tactile signals that are perceived as undesirable vibrations or roughness.

The browser server begins by loading an XML-like[7] `HapticPageMap` file generated by the client component after reading an HTML source file. The `HapticPageMap` file contains links to the various XML-like haptic icon files that are associated with page elements. The browser server parses all of these files using the open source ACE[8] Library's XML parsing engine. The result is an internal representation of the haptic structure of the page in the form of an object model built around the `HapticPageMap` object (Figure 6.2). Since the object model exists within the compiled C++ component, it can be read rapidly during the I/O loop.

Device input and output is handled through Vincent Levesque's STReSS Library's API [29], which in turn makes use of `libusb` and firmware onboard the FPGA for hardware input and output.

The browser server's I/O loop[9] performs the following actions every cycle:

1. Check the page map flag, and reload the `HapticPageMap` if necessary.
2. Read the slider position and click count from hardware and perform the translation into page map coordinates.
3. Compute the eight-taxel tactile image based on the cursor position and the `HapticPageMap`.

[7] See the footnote to section 4.9.6.

[8] ADAPTIVE Communication Environment [1].

[9] The browser is not currently implemented on a real-time operating system, so its update interval can not be guaranteed to be 3 ms. See Section 4.7.1 for more information on measured timings.

4. Output the tactile image to the hardware.

5. Output the cursor position to a `position` file.

6. Output the click count to a `click` file.

The browser server implements a dual-threaded architecture as a further performance consideration. Filesystem operations are handled outside the main I/O thread, and mutex locks (via the ACE library) are used for data synchronization.

Browser Server Haptic Page Map Model

A key part of the design of the browser server component is its object orientation. Just as graphical web browsers have developed a rich Document Object Model around the HTML source, the browser server's haptic model is intended to be flexible and extensible using object-oriented tools. The base object (equivalent to the `document` object in the DOM) is the `HapticPageMap` object, which contains a list of `HapticIcons` (equivalent to the `document.elements[]` array in the DOM). Currently, two types of haptic icons, `SpatialIcon` and `AnimatedIcon`, are supported, each with their own content data stored in a class inherited from the `HapticIconContent` abstract base class (Figure 6.10).

This architecture facilitates adding additional types of haptic icons. For example, one could imagine a haptic icon that varies its position based on time and/or slider position. It could be rapidly implemented as another subclass of `HapticIcon`, or `SpatialIcon`. As long as it supports its own polymorphic `RenderIcon()` method, no changes to the core browser server code are necessary. In fact, the `AnimatedIcon` class was added after the browser server architecture was complete using the method just described.

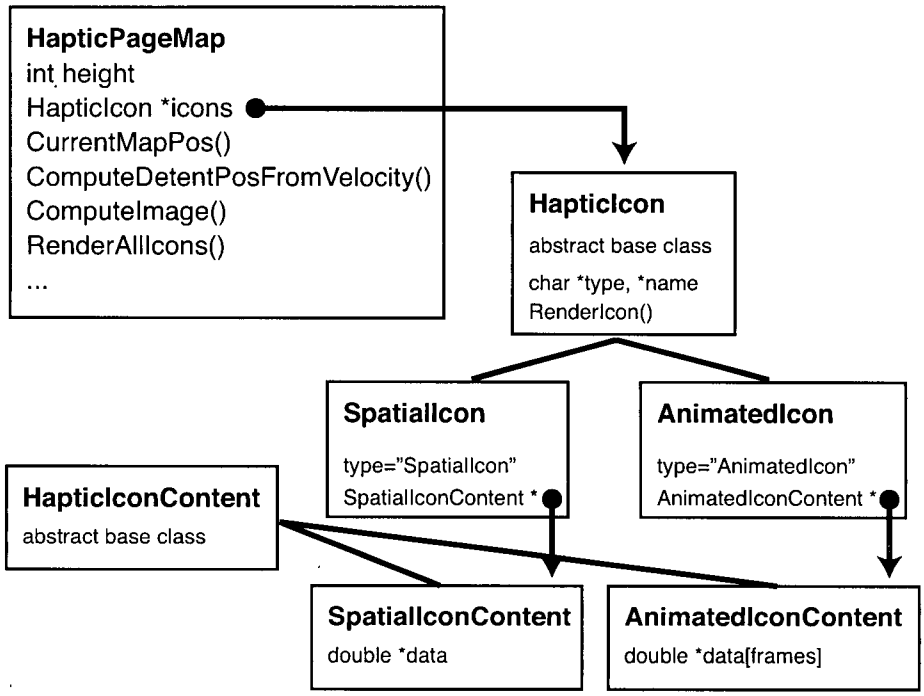


Figure 6.10: Inheritance and encapsulation diagram for the `HapticPageMap` object, `HapticIcon` abstract base class and its inherited classes, and `HapticIconContent` abstract base class and its inherited classes.

Further functionality could be achieved by scripting the `HapticPageMap` model (using JavaScript), in a similar fashion as is currently done with the DOM. This would enable a range of advanced haptic functionality and interactivity leveraging the existing core input/output code.

6.5.3 Interprocess Communication and Timing

Communication between the browser server and browser client components is achieved through the use of small files. This has the advantages of simplicity and leveraging the filesystem's implicit locking mechanisms to prevent simultaneous access and data corruption. It has the disadvantage of potentially low performance due to disk overhead. Timing studies were therefore required to validate the filesystem approach.

As described in Section 4.7.1, various web pages were run with an oscilloscope connected to the amplifier outputs, and a series of software optimizations achieved an empirically stable output rate. No specific timing studies were done on the GUI (Mozilla) loop, but it was observed that `position` and `click` files were not accumulating in the filesystem, indicating that they were being read in a timely manner by the client. No delays were observable between slider input and GUI output.

As mentioned before, various optimizations were done for high performance, including multithreading and careful task prioritization. Placing the file-exchange directory on a RAM disk did not have any performance effect, probably because the files were already being exchanged via the filesystem's memory cache rather than being written to and read from disk.

6.5.4 Browser Haptic Icons

As discussed above in Section 4.9.6, a graphical image-based workflow allows for interactive icon design and rapid analysis using the *stretch image* technique. The `gif2hapticon` tool not only supports the tactile movie output mode, but also a haptic icon output mode with the `-i` command line option (the default). The XML-like format is similar to the tactile movie syntax but is more optimized for haptic icons for use with web pages. Both static (`SpatialIcon`) and time-varying (`AnimatedIcon`) formats are supported, and the type is automatically determined by checking whether there are multiple frames stored in the GIF input. Icons are stored in a icon library directory accessible by the browser server and loaded at runtime when the `HapticPageMap` is parsed.

6.6 Known Software Issues and Caveats

6.6.1 Support for Element Height

Currently, haptic icons are XML-like files that are named under the convention `[name]-[height].xml`. The page rendering engine selects the height value based on the onscreen height of the associated element. Therefore, a separate haptic icon file (or at least a symbolic link to a valid file) must be present for every possible rendered height of the elements. This limits the generalizability somewhat, especially for images, which may have various heights. A possible solution would be to generate icons algorithmically (including repeated tiling), to use the closest-available height icon, or some combination of the two approaches. The XML format should then be reconsidered somewhat, moving the height parameter out of the filename and

into a separate tag, such as `<height>`, much as the `<at>` tag is currently used. Multiple height representations could also be combined into one file.

6.6.2 Opportunities for further software optimization

The browser prototype was implemented with the goal of eliciting user feedback on the application concept as expeditiously as possible. There are opportunities for improving the software architecture if the goal is to build a "production" version with more headroom for future expansion of features:

- A formal implementation of the XML standard, including a Document Type Definition (DTD) would support future extensibility while retaining compatibility with the existing "XML-like" implementation.
- The inter-process communication method, which currently uses high-overhead but simple to implement filesystem access, could be revised using more lightweight, optimized communication methods for greater performance.
- The application could be ported to a real-time operating system for more precise timing control.
- The modularity of the JavaScript code base could be optimized for better maintainability.

As mentioned earlier, while the software is far from being exhaustively optimized, none of the aforementioned optimizations would affect the operation of the browser in its current form. The current version of the browser has been tested to support the application concept and to deliver stable tactile performance. As such, it is ready for user evaluation, which is covered in the following chapter.

Chapter 7

Browser User Evaluation

This chapter describes a formal evaluation that was performed on the browser prototype, aimed at measuring its performance in the hands of a human user. We measured the time required to browse web pages to find a piece of information, such as a weather forecast, and compared the performance in three conditions: (1) the handheld prototype **with** tactile feedback, (2) the prototype **without** tactile feedback, and (3) a mouse scroll wheel used as a discrete navigation control. To simulate a perceptually demanding mobile environment, study participants were asked to simultaneously perform a *distraction task* (not identified to the participants as such) using pedals to respond to video cues. The data from the study, after normalization to compensate for differences in difficulty between the various tasks, did not demonstrate a significant performance difference between the two handheld prototype conditions, although the mouse scroll wheel was faster as expected. Qualitative reaction to the tactile feedback was generally positive, with a small proportion of users reporting minimal conscious awareness of the stimulation.

In this chapter, the volunteers that participated in the study are referred to as *users*, and the terms *user evaluation* and *user test* are equivalent.

7.1 Aims

The purpose of the user test was to evaluate the effects of tactile feedback on performance and qualitative user experience in a handheld context using the previously described browser prototype. The browser prototype implements a number of novel concepts, including the use of artificial tactile flow rendering to provide spatial cues. The results from the user test could serve to validate these concepts, and the utility of tactile feedback on a handheld or mobile device in general, by providing a working demonstration and performance statistics.

The study addresses the following questions:

1. Does the presence of tactile feedback, in the form of a synthetic *tactile flow* signal, affect the time required to complete a web navigation task?

2. How does the experience of browsing and navigation on the handheld prototype compare to a more common paradigm (the mouse scroll wheel)?

3. What are users' subjective reactions to the use of tactile feedback in this application?

4. What are users' subjective reactions to the handheld prototype hardware and browser software?

5. Are there any correlations between user demographics, such as age, gender, and previous mobile experience, and their performance in the browsing task with or without tactile feedback?

7.2 Study Design

Formally, the user study is designed as a *task time performance measure* with one primary *within-subjects* factor having three levels. The selected alpha level was 0.05. The full list of study variables is outlined below.

7.2.1 Study Variables

Primary Dependent Variable

TASK TIME: The amount of time (in milliseconds) required for the user to complete the task. The task **begins** when the user signals that they are ready to begin the task (and the browser screen is presented on the display). The task **ends** when the user signals that they have found the requested information.

Primary Independent Variable (condition)

3 levels: SLIDER+ (slider **with** tactile feedback), SLIDER- (slider **without** tactile feedback), MOUSE (mouse scroll wheel used for navigation).

Additional Dependent Variables

The following were also measured:

- Performance on the distraction task (described below in Section 7.6).
- Individual link access times, which yielded the amount of time spent on each page.
- A qualitative assessment battery (described below).

Additional Independent Variables

The following factors were randomized:

- The TASK TYPE (3 levels: WEATHER, TRANSIT, and MOVIES)
- The specific TASK:
  - WEATHER group: 18 levels
  - TRANSIT group: 12 levels
  - MOVIES group: 10 levels
  - (total 40 different tasks)
- Various parameters related to the distraction task (see Section 7.6 below).
- User demographics.

In addition, a variable, PRESENTATION ORDER, is defined as the presentation sequence number after flattening for CONDITION. More information on PRESENTATION ORDER is given in Section 7.3.4.

7.2.2 Normalization for Task Difficulty

Each of the 40 TASKS is similar in the sense that they all involve searching for information in a web page hierarchy using identical navigation methods. However, because the information that the user is requested to find is slightly different in each TASK, each TASK will have an intrinsic task completion time, referred to here as the TASK DIFFICULTY. The task difficulty is influenced by a number of factors, including the depth of the hierarchy at which the information is found, the position of links on the web page (i.e., the distance

that the user must scroll to access the link), and the complexity of the cues (e.g., link text) employed by the user during navigation.

Since TASK DIFFICULTY is a confounding variable in our analysis of the effect of CONDITION on task time, it is desirable to decouple the measured task times from the relative differences among tasks, using a normalization procedure. The first step in this procedure requires the computation of a statistical estimate of relative TASK DIFFICULTY. Tasks that are relatively more "difficult" (i.e., have a higher intrinsic task completion time) would have their measured task completion times depressed by the TASK DIFFICULTY score, and vice-versa for relatively "easier" tasks.

Theoretical models such as Fitts' Law and GOMS could provide predictive estimates of task difficulty as a normalization score, but there is no single existing published model that takes into account all the factors influencing task difficulty in the specific context of the current experiment. Furthermore, it is not within the scope of the present work to derive and validate an in-depth user model of browsing using the hybrid position/velocity control system on a handheld device using the specific content and layout that was developed for the user study. Therefore, to maximize accuracy, an *empirical* TASK DIFFICULTY score was used, calculated from the measured data according to the following procedure.

A *normalization vector* was computed, having a normalization score for each of the 40 TASKS. For each TASK, an average score (across all subjects and conditions) was computed and compared to the average for all TASKS. The ratio of current task time to average overall time score was used as the "difficulty" score, and the normalization factor was computed by taking its inverse.

By deriving the normalization score from a pooled statistic across all

users, the task difficulty normalization does not take into account individual user differences. Sources for these differences might include variations in familiarity with certain tasks versus others, or differences in the relative effects of various factors that influence the task time (navigation distance, verbal complexity, etc.). The validity of the normalization procedure in light of individual subject differences will be explored in Section 7.8.5.

7.3 Methodology

A total of 16 volunteers were recruited for the study (three pilot subjects and thirteen main study subjects), which was performed under the auspices of UBC Ethics Approval #B01-0470 (see Appendix A). Each study session took approximately 50 minutes and consisted of:

1. Briefing and collection of demographic data.
2. A pre-task attitudes survey.
3. Verbal delivery of task instructions.
4. A **training**, or practice session, minus the distractor task.
5. A second **training** session, including the distractor task.
6. The main study task (described further below).
7. A post-task attitudes survey.
8. A post-task interview.
9. Compensation ($10 per subject).

Gender	38% female
Age	median: 28.5 std. dev: 4.6
Right handed	81%
Previous experience with a musical instrument	77%
Among subjects with previous musical instrument experience, number of years played	median: 7 std. dev: 4.0
Regular user of a mobile phone	77%
Regular user of a PDA	15%
Regular user of a game controller with vibrotactile feedback	15%
Regular user of a mouse with scroll wheel	94%

Table 7.1: Demographic characteristics of the subject pool (16 subjects) that was used for the browser user evaluation, including both pilot subjects and main study subjects.

Due to variability in the experimental procedure during the three pilot runs (mostly related to fixing bugs in the computer setup), only data from the 13 main study subjects were used in the quantitative and qualitative analyses of the experiment.

7.3.1 Recruitment of Study Participants

Participants were recruited through the HCI@UBC subjects database, and through friends and associates of the author who met the study criteria. The conditions for participating in the study were a good command of English and no prior experience with the browser prototype or the present research

in general. The final corpus had demographic characteristics as described in Table 7.1. The subject pool appears to be a typical random selection of potential users of general mobile data services, slightly skewed towards younger participants due to the university setting. Since no distinction was made between pilot subjects and main study subjects other than the order in which they booked their appointments, both groups of subjects can be considered to be randomly drawn from the distribution described in Table 7.1.

7.3.2 User Test Environment

All user tests were conducted in a dedicated, soundproof usability testing lab at the University of British Columbia during normal business hours or in the early evening. The room was arranged as shown in Figure 7.1. The following devices were arranged in front of the study participant:

1. The tactile display prototype, including an embedded LCD display. Because the originally specified 2.5-inch LCD panel malfunctioned prior to the start of the experiment and could not be promptly replaced, it was necessary to use a 3.5-inch NTSC TFT colour LCD monitor, attached to the top case with double-sided tape. This resulted in an increased weight, especially in the top part of the device.

2. A mouse (Logitech Scroll Mouse) with scroll wheel.

3. A standard 101-key PC keyboard. The two largest keys, ENTER and SPACE, were used for signalling readiness to begin the task and completion of the task, respectively, and were labelled "START" and "STOP", using 7.5-cm wide yellow sticky notes.

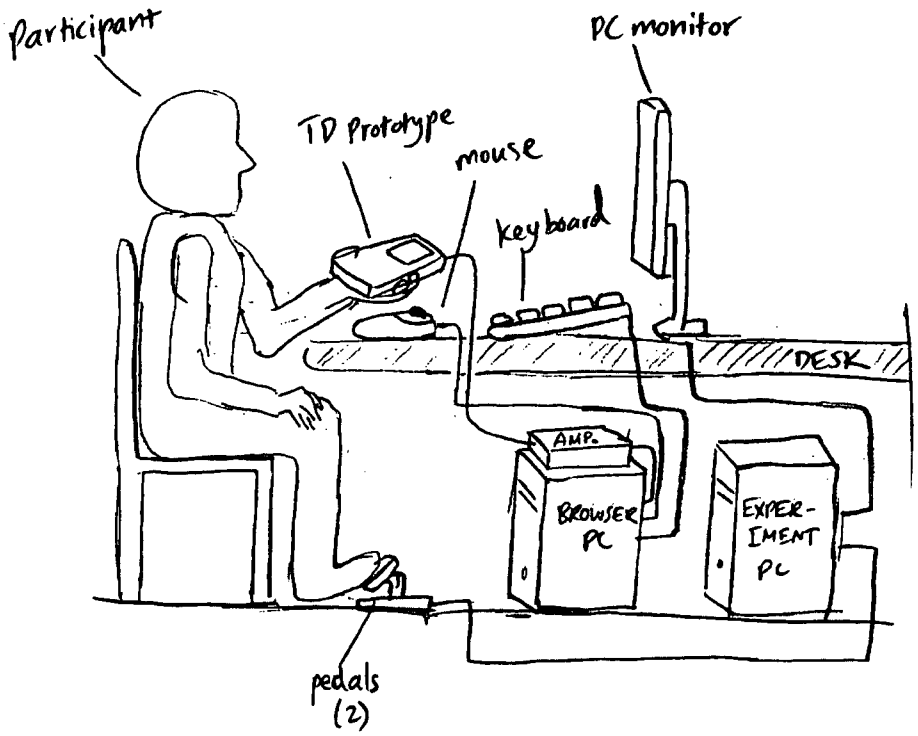


Figure 7.1: Browser user test environment.

4. A second PC keyboard (not shown in the diagram), connected to the experiment PC and used by the facilitator for resetting the distractor task.

5. A 17-inch colour TFT LCD PC monitor running at 1280x1024 resolution. Prompts to the user were displayed in 120-pixel high type, centred in the display on a white background.

6. A set of pedals, of the type used commonly in home game systems for driving games (ACT LABS, Performance Pedals). Participants were instructed to place one foot over each pedal. The pedals themselves were modified to function as left and right mouse buttons, and were connected to the PC via a USB mouse.

7. A *browser PC*, running the Linux-based browser prototype software and connected to the tactile display, LCD, mouse with scroll wheel, keyboard, and local-area network.

8. An *experiment PC*, running Windows 2000 and the browser experiment software (described in Section 7.7), and connected to the PC monitor, pedals, and local-area network.

9. The driver electronics, connected between the browser PC and tactile display prototype.

During the experiment, the facilitator remained in the room with the participant, delivering instructions read from the verbal protocol (Appendix A) and verbally collecting the information the participant was asked to retrieve. The experimenter also remained logged in to the browser PC via a remote terminal connection from a separate laptop, performing such functions as

starting and stopping the browser prototype software, changing configuration settings (for example, SLIDER+ to MOUSE), and monitoring the data collection process. The experimenter also used a keyboard connected to the experiment PC to signal task completion and to present the next task to the participant when they verbally indicated that they were ready to proceed.

In the SLIDER- condition, all software settings were the same as the SLIDER+ condition, except the final amplifier electronics were powered off. This ensured that there were no confounding variables, such as timing differences, introduced as a result of changing conditions between SLIDER+ and SLIDER-.

7.3.3 Briefing and Collection of Demographic Data

Study participants were first asked to read and sign the standard UBC Ethics Consent form (included in Appendix A), and each participant was given a copy for their records. All participants were verbally reminded that they could abort their participation in the study at any time without penalty (i.e., they would still receive their compensation). Participants were also reminded that their frank and honest opinions were requested, and that the purpose of the study was to obtain a performance assessment of the device, not the study participant.

All study participants then completed the demographic survey and pre-test attitudes survey; the results are shown in Table 7.1 and Figure 7.7.

As noted in the verbal protocol (Appendix A), all participants were instructed to hold the device in their left hand[1] with their thumb resting

[1] Only left handed operation is supported on the handheld prototype. This is also the case with almost all commercial handheld products having side-mounted controls. See Section 4.1.

on the tactile display. Participants were told not to abrade or pick at the surface of the TD, and were reminded when necessary to maintain a hand position such that the thumb was in full contact with all 8 elements of the TD. When the mouse scroll wheel was used as an input device, the user was also asked to use their left hand to operate the mouse. In that case, the device (still required for visual display) was placed on the table and propped up at an an angle for better visibility.

7.3.4 Task Blocks

The study is organized into BLOCKS of 6 TASKS each. Each BLOCK is performed under one CONDITION (SLIDER+, SLIDER-, or MOUSE). Within a group of three BLOCKS, each condition is presented; and each subject receives a sequence that is randomly drawn from the 6 possible permutations of condition presentation order. Between BLOCKS, study participants get a one-minute break to relax and stretch.

Within each BLOCK, the 6 TASKS are also presented in a randomized order. TASKS may be categorized into TASK TYPES, consisting of WEATHER, TRANSIT, and MOVIES, each of which is represented twice per block. There are 6 possible permutations of the presentation order of 3 TASK TYPES. For each BLOCK, a presentation order is drawn at random from this set, and the sequence is repeated twice to form a 6-TASK BLOCK. Finally, for each TASK TYPE, the presentation order of specific TASKS is randomized.

The presentation sequence of an example run through the main part of the study can be visualized as follows (six blocks shown):

(SLIDER+)A16.C5.B8.A11.C9.B1 $\longrightarrow$ (break) $\longrightarrow$
(MOUSE)C3.B4.A3.C2.B5.A7 $\longrightarrow$ (break) $\longrightarrow$

(SLIDER-)B3.A15.C1.B2.A17.C4 $\longrightarrow$ (break) $\longrightarrow$

(SLIDER+)A2.B1.C8.A5.B11.C6 $\longrightarrow$ (break) $\longrightarrow$

(MOUSE)C1.A18.B9.C7.A1.B9 $\longrightarrow$ (break) $\longrightarrow$

(SLIDER-)B10.C5.A4.B6.C9.A10 $\longrightarrow$ (break) $\longrightarrow$

...

where each letter (A, B, or C) represents a TASK TYPE, the number following the letter represents a specific TASK, and each line is a BLOCK. Note that the sequence of TASKS does not repeat until all tasks of a particular TASK TYPE have been exhausted; the sequence for that TASK TYPE then repeats in the same order as before. When, for example, all 12 tasks of TASK TYPE **A** have been exhausted, the next TASK drawn from TASK TYPE **A** is the same as the first task that was drawn during the first pass through the sequence.

The meta-variable PRESENTATION ORDER is the ordinal TASK $\times$ BLOCK presentation sequence number, with CONDITIONS "flattened" out. Formally:

$$PO_{task} = t_{task} + T(ceil(b_{task}/3) - 1) \tag{7.1}$$

Where:

- PO_{task} is the PRESENTATION ORDER number for the given TASK,
- t_{task} is the sequential presentation number for the TASK within its BLOCK (ranging from 1 to 6)
- b_{task} is the sequential presentation number of the BLOCK to which the TASK belongs (ranging from 1 to 9)
- T is the number of TASKS per BLOCK (6)

In the example above, TASKS **A16, C3, and B3** each have a PRESENTATION ORDER of 1, and TASKS **C5, B4, and A15** have a PRESENTATION ORDER of 2. The sequence continues through TASKS **A2, C1, and B10**, which have a PRESENTATION ORDER of 7. By grouping TASKS that are presented at approximately the same ordinal position in different CONDITIONS under a single PRESENTATION ORDER score, a *within* PRESENTATION ORDER analysis of variance of task times is enabled. The results from this analysis are presented in Section 7.8.1. — The full task inventory is included in Appendix A; however, some examples are:

A1 What is the weather in London today?

A2 What will the weather be like in London tomorrow?

A3 What will the weather be like in London the day after tomorrow?

A4 What is the weather in Paris today?

B1 If you take the 99 B-line from UBC at 1pm, when will you arrive at Broadway station?

C1 When is the movie "L'Enfant" playing at the Ridge Theatre?

The various randomizations and permutations are handled automatically by the JavaScript component of the Test Software (described in Section 7.7).

Error cases, defined as the user clicking on the wrong link, becoming distracted (in ways other than typically induced by the distraction task), or experiencing a problem, were recorded but excluded from the analysis of task times. When a participant made an error, he/she was asked to inform the facilitator of the error but complete the task anyway. A "back

to previous page" link was provided in a standardized location on all pages of the sample corpus for this purpose.

7.3.5 Training Sessions

Participants first completed two training sessions to become familiar with the tasks. In the first training session, subjects completed 3 shortened BLOCKS of 2 TASKS each, in each of the three conditions (SLIDER+, SLIDER-, or MOUSE). In the second training session, subjects also completed 3 shortened BLOCKS of 2 TASKS each, but with the addition of the distraction task (described below in Section 7.6).

7.3.6 Main Data Collection Session

In the main part of the study, subjects completed 9 full BLOCKS of 6 TASKS each, or 54 TASKS in total.

7.3.7 Post-Task Assessment

Following the main data collection task, subjects were given a one-minute break and then asked to fill out a post-task attitudes survey. The survey is included in Appendix A, and the results are discussed in Section 7.9.

Following the survey, the facilitator conducted an interview using the protocol documented in Appendix A. The participants were then given their compensation and were free to go after signing a receipt.

7.4 Pilot Study

A pilot study was first conducted with three participants to assess, debug, and optimize the experiment protocol and browser prototype design. Pa-

rameters in the browser prototype software were continuously optimized during the pilot phase using a participatory design model (the final values are documented in Table 6.1). In addition, the decision to implement the hybrid control spring design (Figure 6.5) was made based on user feedback from the pilot study.

7.5 Stimuli Used in the Study

The final selection of haptic icons used for the study are shown in Figure 7.2. Two types of haptic icons were used. Large headings were marked with the high-frequency *grating* icon, and links were marked with the modified single-bump *crater* icon. To arrive at these haptic icon selections, more than 20 variants of haptic icon representations were created using the image-based rapid prototyping method (Section 4.9.6) and participatory design feedback was obtained during the pilot phase and used immediately to tweak and optimize the icons in an on-line fashion. Icons were selected for high salience (perceived amplitude), distinguishability from one another, and clarity of the directional *tactile flow* signal.

To achieve these design goals, several points were taken into consideration. First, the results from both the MDS haptic icon discriminability study (Section 5.4.1) and the Image Browser User Test (Section 6.2.3) indicated that the most salient dimension employed by users for distinguishing haptic icons was whether the icons were high frequency and periodic or low frequency and relatively non-periodic, therefore, the two selections were made on opposite ends of this spectrum. The decision to use only two icons was based on the following justification: the browser user study was intended primarily to address the effectiveness of *tactile flow* as a design concept; pre-

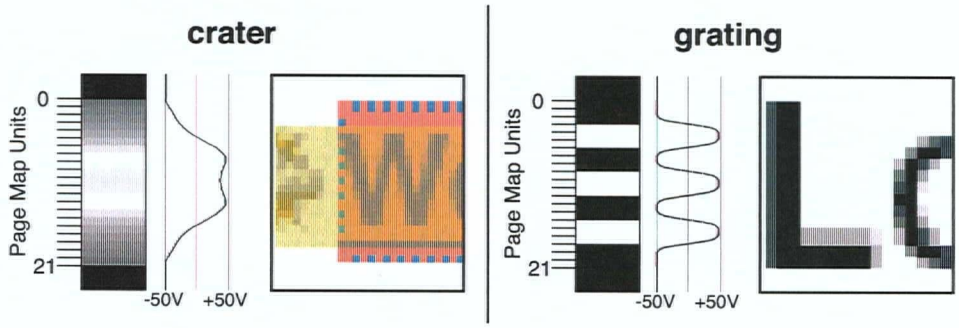


Figure 7.2: Haptic icons used in the browser user study. For each of the two icons, the volt image representation, voltage trace, and crop of an associated page element are shown. The height of the *crater* representation (21 page map units) is the same as the height of a link (in pixels) when rendered in the browser window. The rendered height of headings may differ slightly from the height of the associated *grating* icon; in that case, the haptic icon is top-justified.

vious studies had already addressed the distinguishability of a wide variety of haptic icons; and the use of a large corpus of haptic icons might introduce undesired variability in the salience of the tactile flow signal which the present study was not powered to evaluate. On the other hand, if the effectiveness of tactile flow as a spatial cue was proven, a subsequent study could examine the detailed design of an optimally sized corpus of haptic icons for use with the browser.

Animated icons containing time-varying moving components were explored, but since the goal of the experiment was to demonstrate the usefulness of the *tactile flow* signal as a spatial cue, it was felt that the icons at this stage of development should not interfere with the *tactile flow* signal by superimposing additional motion. Therefore, static icons were selected.

The decision to use the *grating* icon for headings and the *crater* icon for links, as opposed to vice-versa, was informed by the observation that the directional sensation of *tactile flow* was much stronger for the non-periodic signals, and links were more frequent and irregularly distributed than headings in the test web page corpus, thus the importance of effective *tactile flow* display for links was higher.

Finally, the icons were designed such that, using the 3-to-1 subtaxel rendering setting (Section 6.4.10), individual piezos would still reach maximum (or near-maximum) amplitude displacement. To avoid icon overlap and because pilot testing did not reveal any significant advantages in terms of tactile flow perception, the page map shrinking feature of the browser prototype software was not used.

7.6 Distraction Task

Divided attention between the information appliance and the user's environment is a key feature of mobile device usage in the real world [40]. To simulate this experience while retaining the benefits of a controlled laboratory testing environment and without incurring the costs of engineering the prototype for true mobility, a visual distraction task was implemented as part of the experiment protocol.

Every seven seconds (beginning seven seconds after the start of the browsing task), the user was prompted to perform an independent task with his/her feet, so as to not affect the hand-based interaction with the mobile device. The prompt was displayed on a PC monitor on the desk, requiring the user to shift his/her gaze from the hands (below eye level) to horizontal eye level. The prompt was displayed in large (120-pixel high) type, such that a change could be observed using peripheral vision while looking at the device held in the hand; however, the prompt was displayed in ALL CAPITAL letters in a serif font to minimize preattentive processing of the message. Since the various prompts were morphologically similar and complex, the user was forced to read the message each time.

There were six distraction task variants, each of which involved responding to one of the following commands:

1. Press the left pedal once.
2. Press the left pedal twice.
3. Press the right pedal once.
4. Press the right pedal twice.

5. Press the left, then the right pedal.

6. Press the right, then the left pedal.

To balance initial difficulty against increasing user familiarity as they acquired experience with the task, only the first four distraction tasks were used during the first half of the experiment (3 out of 6 blocks in training phase 2, and 4 out of 9 blocks in the main experiment); after that, the full selection of 6 tasks were used.

Feedback was given at seven-second intervals together with a refresh of the task prompt. A cumulative positive reinforcement model was used, similar to a typical video game design. Successive correct performance resulted in the onscreen presentation of a reward image, consisting of a briefly (1 second) animated character such as a smiley face, with increasing valence. For example, the smileys became larger, more numerous, more animated and more embellished to reinforce a string of correct performance. Six levels of reinforcement were supported. If the user provided an incorrect input during a seven-second interval, the correct-response counter was reset to zero and the background of the prompt area was changed from white to red. A correct response during the next seven-second interval resulted in the background reverting to white, the counter being incremented by one, and the appropriate reward animation being presented.

The effectiveness of the distraction task is assessed from a quantitative perspective in Section 7.8.8 and from a qualitative perspective in Section 7.9.2.

7.7 Browser Experiment Software

As described in Section 7.3.2, User Test Environment, two PCs were used during the experiment, one running the browser prototype software, and the other running the experiment software. The experiment software provides the following functionality:

1. Randomizing the selection of tasks in blocks, using the procedure described in Section 7.3.4.
2. Displaying task prompts to the user (e.g., "What is the weather like in Paris today?").
3. Displaying the timed distractor task prompts to the user (e.g., "Press the left pedal twice."). Described further in Section 7.6.
4. Providing feedback to the user on their performance in the distractor task.
5. Recording the distractor task events from the pedals, and tagging the data with timing information.
6. Sending the collected data to the facilitator's email account via the network.
7. Reading the keyboard events from the facilitator.
8. Prompting the user to take a one-minute break.

In addition, for the most accurate timing, the browser *prototype* software also includes several functions related to the experiment:

1. Receiving events from the keyboard when the user signals the start and end of the task.

2. Displaying a modal dialog box blocking interaction with the browser until the user signals the start of the task.

3. Supporting the mouse scroll wheel as a configurable alternate input device.

4. Resetting the browser upon user-signalled completion of the task.

5. Dumping timing data to the terminal (recorded on the experiment facilitator's remote laptop).

The browser *experiment* software is designed for any standards-based web browser (it was run on Mozilla Firefox 1.0 under Windows 2000 for the experiment) and consists of the following components (included in Appendix E):

1. A set of XHTML files with embedded style sheet (CSS 1.0) information, comprising the presentation layer of the software.

2. A set of JavaScript files, comprising the logic layer.

3. A CGI (Common Gateway Interface 1.1) script, based on the free "FormMail" script, running on an off-site server that accepts form data and forwards it to the facilitator's email account.

The general architecture is that of a web application using the modern AJAX (Asynchronous JavaScript and XML) methodology, running mostly on the client side. It uses the XMLHttpRequest object supported in all current (as of 2006) browsers to send data to a server-based CGI script in an online fashion, without reloading the page or causing a visual interruption to the user. This allows for a highly interactive, portable experiment

management application while providing convenient email-based collection of data with redundant copies distributed across networked computers for maximum safety. To the author's knowledge, the present software is the first known application of this technique for usability experiment management.

The logical flow of the browser experiment software is as follows:

1. The appropriate HTML file (training phase 1, training phase 2, or main experiment) is loaded into the browser by the facilitator.

2. The HTML file instructs the browser to load and executes the associated JavaScript code.

3. A start screen is presented, prompting the facilitator to enter a participant number, and in the main experiment version, an optional number of blocks (default is 9). The participant number is provided simply for convenience and is appended to all data output by the program.

4. When the participant indicates to the facilitator that he/she is ready to begin the first task, the facilitator presses a form submit button.

5. The JavaScript component creates an array for each of the three TASK TYPES. In the main experiment version, the arrays are randomly permuted, and the tasks are sequentially assigned to BLOCKS that are stored as associative arrays.

6. A display refresh is performed without reloading the page, accomplished by manipulating layer visibility parameters in the style sheet.

7. The first task prompt is displayed to the user. The software waits for a keyboard input from the facilitator, indicating that the participant

has read the prompt and has pressed the START key, signalling the browser *prototype* component to start the task.

8. The software enters a loop, presenting a distractor task prompt to the user at seven-second intervals. The software checks to ensure that no two sequentially presented distractor tasks are alike.

9. Pedal input from the user is read and stored. When the seven-second interval expires, the input is checked against the task requirement and feedback (correct or incorrect) is provided to the user.

10. When the facilitator presses a key on the keyboard, the task is finished, the distractor task prompt is removed, and the next task prompt is presented on the screen.

11. When a block has been completed, the software sends an interim copy of the data log to the server-based CGI script (and thus to the facilitator's email account). The user is prompted to take a one-minute break via a modal dialog.

12. When all blocks have been completed, the software sends the final copy of the data log to the server and presents a modal dialog thanking the user for his/her participation. Since the page is never refreshed, the data is retained in a text input field that is hidden below the scroll boundary of the window. Should the email transmission fail, the data can still be recovered from the page using copy and paste on the client PC.

7.8 Quantitative Results

A total of 645 valid, error-free observations of TASK TIME were gathered in each of the 3 CONDITIONS across the 40 TASKS and 13 participants, and used in the quantitative analysis of performance.

7.8.1 Effect of Condition on Task Time

The results of a repeated-measures ANOVA on the task time data across all 13 non-pilot subjects is shown in Table 7.2. For this analysis, TASK TIME was compared within PARTICIPANT and PRESENTATION ORDER variables. If an error occurred in any of the three CONDITIONS for a given level of PRESENTATION ORDER for a given subject, the data for that level of level of PRESENTATION ORDER was not considered in this analysis; only those combinations of task × subject with full repeated measures across all conditions were used, resulting in a slightly lower number of observations (N) than the overall data set.

The results show a statistically significant difference in TASK TIME across the three different CONDITIONS. However, when the CONDITIONS are examined in a pairwise fashion, there is a statistically significant difference between the MOUSE condition and the two SLIDER conditions, but not among the two SLIDER conditions (i.e., with and without active tactile feedback).

7.8.2 Individual Subject Differences in Performance

Could the lack of a statistically significant difference between the two SLIDER conditions be due to individual differences in the way users respond to the presence or absence of tactile feedback? The data were analyzed separately for each subject (with appropriate Bonferroni correction for the groupwise

Condition	Mean Task Time	Std. Error	N
MOUSE	12166.34	4415.81	181
SLIDER-	18200.52	5453.77	181
SLIDER+	18445.01	4682.06	181

Overall ANOVA

Conditions	F	p
Overall	132.750	0.000

Pairwise Comparison (Bonferroni Corrected)

		95% Confidence Interval		
Conditions	ΔMean	Low	High	p
MOUSE vs SLIDER-	-6034.18	-7182.19	-4886.19	0.000
MOUSE vs SLIDER+	-6278.67	-7292.42	-5264.93	0.000
SLIDER- vs SLIDER+	-244.49	-1373.88	884.91	1.000

Table 7.2: Results from a within-subjects, within-PRESENTATION ORDER analysis of **measured** task time. All measurement units are in milliseconds.

error rate), and the results are shown in Table 7.3.

As shown in Table 7.3, the magnitude of the difference between SLIDER+ and SLIDER- conditions was small for each participant, and none of the differences was statistically significant. The evidence therefore does not support the hypothesis that opposing effects of CONDITION in individual subjects was responsible for the overall lack of statistical significance.

7.8.3 Effect of Task on Task Time

The relationship between TASK and measured TASK TIME is shown in Figure 7.3. The graph clearly shows variation in the task times that suggest differences in task difficulty. The task requiring the most time to complete was:

C10 What rating did the movie "The Promise" receive in its review in the Washington Post?

The task requiring the shortest time to complete was:

A15 What will the weather be like in San Francisco the day after tomorrow?

Examining the web pages in the test corpus, we see that task **C10** requires a minimum traversal distance of 156 + 852 + 563 + 497 = 2068 pixels or page map units: the longest in the corpus. By comparison, task **A15** requires traversal of 99 + 484 + 254 + 176 = 1013 pixels or page map units: the shortest in the corpus. The observations are therefore generally in accordance with GOMS and Fitts' Law principles about the effects of distance and keystroke input on task completion time.

7.8.4 Effect of Task × Condition on Task Time

Could the lack of a statistically significant difference between the two SLIDER conditions be due to opposing effects in different TASKS? Figure 7.4 shows the TASK TIME data sorted by TASK across each of the conditions. Although the data is somewhat noisier due to a lower number of samples in each condition versus the overall task time, each of the three conditions follows the same general trend with regard to variation in task difficulty; we do not observe any large interaction effects between CONDITION and TASK TIME

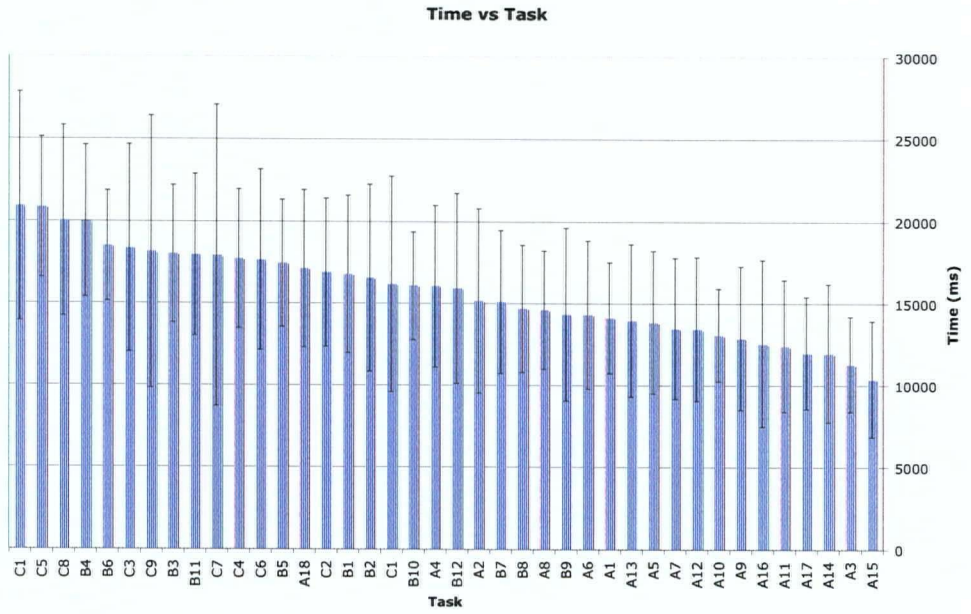


Figure 7.3: The 40 TASKS are sorted according to measured TASK TIME and displayed here with standard error bars.

Average Task Time vs Condition

Subj.	N	allcond	MOUSE	SLIDER-	SLIDER+	p
4	49	18878	16282	20527	19986	1.000
5	54	16294	13487	17358	18036	1.000
6	53	15748	10841	18227	18314	1.000
7	49	16892	10384	19566	19960	1.000
8	47	17775	13170	20226	19558	1.000
9	53	13339	9155	15875	15084	1.000
10	51	13203	10654	13763	15376	0.968
11	46	15513	11392	17517	17963	1.000
12	49	17509	13987	19053	19529	1.000
13	48	13650	9055	15613	16283	0.932
14	46	18171	12587	21419	20913	1.000
15	50	21640	17069	23146	24026	1.000
16	50	13187	9248	15413	15031	1.000
ALL	645	15287	12056	18245	18499	1.000

Table 7.3: **Measured** task time for each of the 13 main test participants. "Allcond" stands for "all conditions". The p-values are the result of Bonferroni corrected t-tests between the two SLIDER conditions. All measurement units are in milliseconds. Further statistics are provided in Appendix B.

across the TASKS. Furthermore, although the performance in the MOUSE condition in the most difficult TASK (**C10**) is worse than the performance in the SLIDER- condition in the *least* difficult TASK (**A15**), when compared in a TASK-matched fashion, the performance in the MOUSE condition is *always* better than the performance in either of the SLIDER conditions. This does not support the hypothesis that the lack of an observed difference in the two SLIDER conditions was due to an interaction between task difficulty and CONDITION.

7.8.5 Validation of Task Difficulty Normalization

Before proceeding with the task difficulty normalization as described in Section 7.2.2, we should verify that the observed variation in performance on a task-by-task basis was indeed due to intrinsic TASK-related sources, and not due to the effect of the study independent variables. As discussed in Section 7.8.3, both the empirical data and theoretical models support the hypothesis that variation in task difficulty is associated with navigational distance. Furthermore, as Figure 7.4 indicates, the general trend of increasing difficulty with certain tasks is observed in *all three* CONDITIONS in a slightly noisy, but mostly monotonic fashion. Combined, these observations lend validity to the concept of task difficulty determined by intrinsic factors, which may be "normalized out" using an empirical model of task difficulty.

Consideration was given as to whether to use a single, pooled normalization score from across the three conditions for each TASK (i.e., a normalization *vector*), or to use separate normalization scores for each TASK $\times$ CONDITION (i.e., a 3×40 normalization *matrix*). There was no reason to reject the hypothesis of similar effects of task difficulty across all three

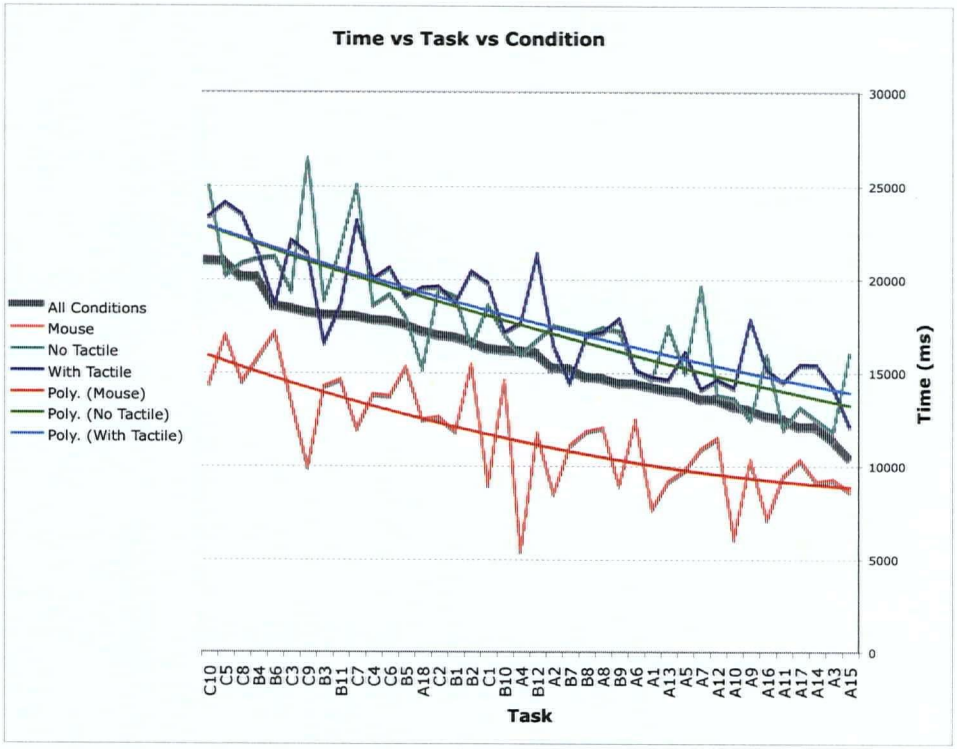


Figure 7.4: The coloured lines show measured TASK TIME vs. TASK in each of the three conditions, while the heavy black line shows the overall (averaged) time across all conditions. The TASKS are sorted in order of overall measured TASK TIME. Polynomial curve fit lines are also shown for each of the conditions. The data are noisy due to a low and variable number of observations at each point due to the randomization of task assignments. The following data points had zero observations and are reflected in the graph as smooth lines drawn between adjacent observations: Tasks A13 and B11 in the SLIDER- condition, and task A17 in the MOUSE condition.

conditions based on the data in Figure 7.4. Furthermore, due to the randomization of tasks, some cells in the 3 × 40 matrix would have had zero observations (the maximum number of observations in a cell was 13); using the overall data the lowest number of observations was 9, and the highest was 24 (see Appendix B). For these reasons, it was decided to determine the normalization score for a given task by considering the data across all three conditions.

The normalization scores were applied to the *measured* task times to determine the *normalized* task time, and ranged from 0.733 for the task requiring the most time to complete (C10), to 1.474 for the fastest task (A15).

7.8.6 Analysis Using Normalized Task Time

Overall Effect of Condition on Normalized Task Time

A revised version of Table 7.2, using times normalized for task difficulty, is shown in Table 7.4. Data variation due to differences in task difficulty is reduced, as evidenced by the lower standard error and increased F statistic; however, despite increasing the detection power through normalization, the overall results between the two SLIDER conditions are still not statistically significant.

Individual Subject Differences in Normalized Task Time × Condition

Normalizing the task scores did not affect the significance of the observations on an individual user basis. Full statistics are provided in Appendix B.

Condition	Mean Task Time	Std. Error	N
MOUSE	11474.34	3957.91	181
SLIDER-	17011.57	4455.32	181
SLIDER+	17179.28	3653.15	181

Overall ANOVA

Conditions	F	p
Overall	163.275	0.000

Pairwise Comparison (Bonferroni Corrected)

		95% Confidence Interval		
Conditions	ΔMean	Low	High	p
MOUSE vs SLIDER-	-5537.25	-6456.06	-4618.41	0.000
MOUSE vs SLIDER+	-5704.94	-6499.99	-4909.89	0.000
SLIDER- vs SLIDER+	-167.707	-945.67	610.26	1.000

Table 7.4: Results from a within-subjects, within-PRESENTATION ORDER analysis of **normalized** task time. Units are in milliseconds.

7.8.7 Learning / Practice Effects

Even if the difference between the two SLIDER conditions was not statistically significant across the entire experiment, could the presence of tactile feedback have a positive or adverse effect on learning the task and operation of the device? To address this question, we split the data into bins for each of the three three-BLOCK sets comprising a full run through each of the three CONDITIONS. The data are presented in Figure 7.5. A typical

asymptotic performance improvement is observed in each of the three CONDITIONS; however, it is unclear whether there are significant interactions in the two SLIDER conditions. The shape of the curves suggests that more familiarity with the device or practice with the experimental paradigm would not have produced a significant difference between the two SLIDER conditions, although the present data do not suggest an expected outcome for a long-term, longitudinal study.

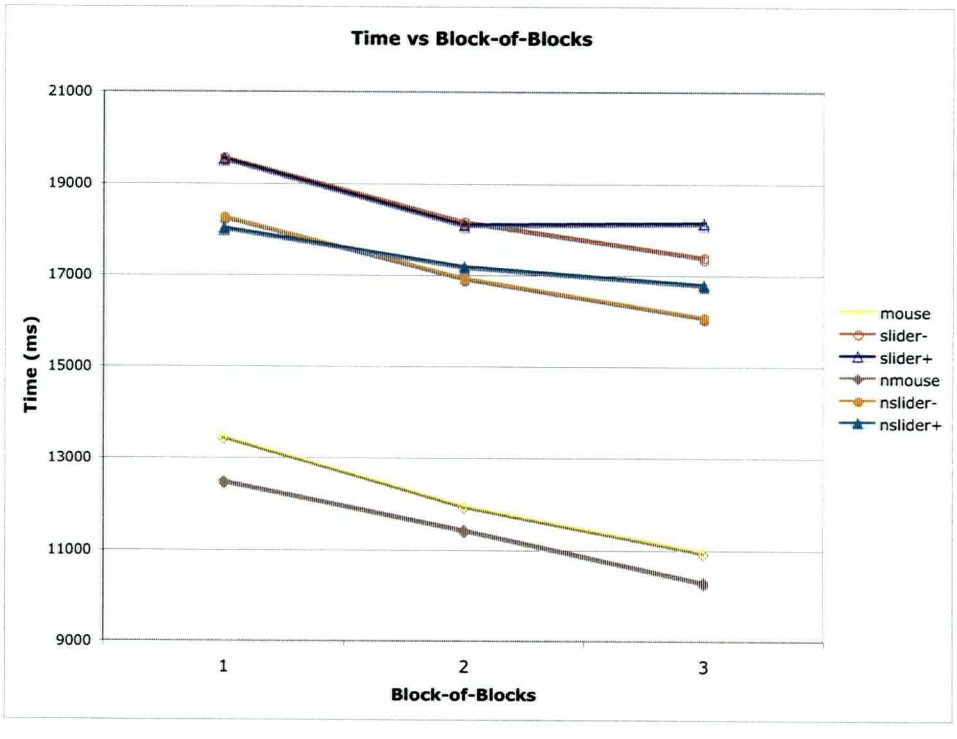


Figure 7.5: The measured (mouse, slider-, slider+) and normalized (nmouse, nslider-, nslider+) task times are plotted versus the presentation order sequence of a set of 3 BLOCKS ("block-of-blocks") comprising a full run through each of the three CONDITIONS.

For increased resolution, the data were analyzed by the previously described PRESENTATION ORDER meta-variable. The results with both measured and normalized task times are shown in Figure 7.6. Again, typical learning curves are observed, with a slight increase in task time at the end likely attributable to fatigue. The data do not indicate any effect of tactile feedback on learning; however, the relative flatness and consistency of the learning curves lend confidence to the validity of the observations.

7.8.8 Quantitative Validation of Distraction Task

The data gathered by the experiment PC were analyzed to ensure that participants were performing the distraction task per the experiment protocol, and the results are summarized in Table 7.5. While some participants showed higher accuracy than others on the distraction task, the overall accuracy rate was 81%, showing that participants were attending to and performing the distraction task. The qualitative findings related to the distraction task are discussed in Section 7.9.2.

7.9 Qualitative Results

7.9.1 Pre-Task Attitudes Survey

The results from the pre-task attitudes survey are shown in Figure 7.7. All questions were measured on a Likert scale. Questions 1, 5, 2, and 3 were intended to assess participants' attitudes about performing multiple tasks at once on a handheld device. Questions 4 and 7 were intended to gather some insight into participants' self-assessment of tactile aptitude and preference, which people have rarely considered before and therefore can

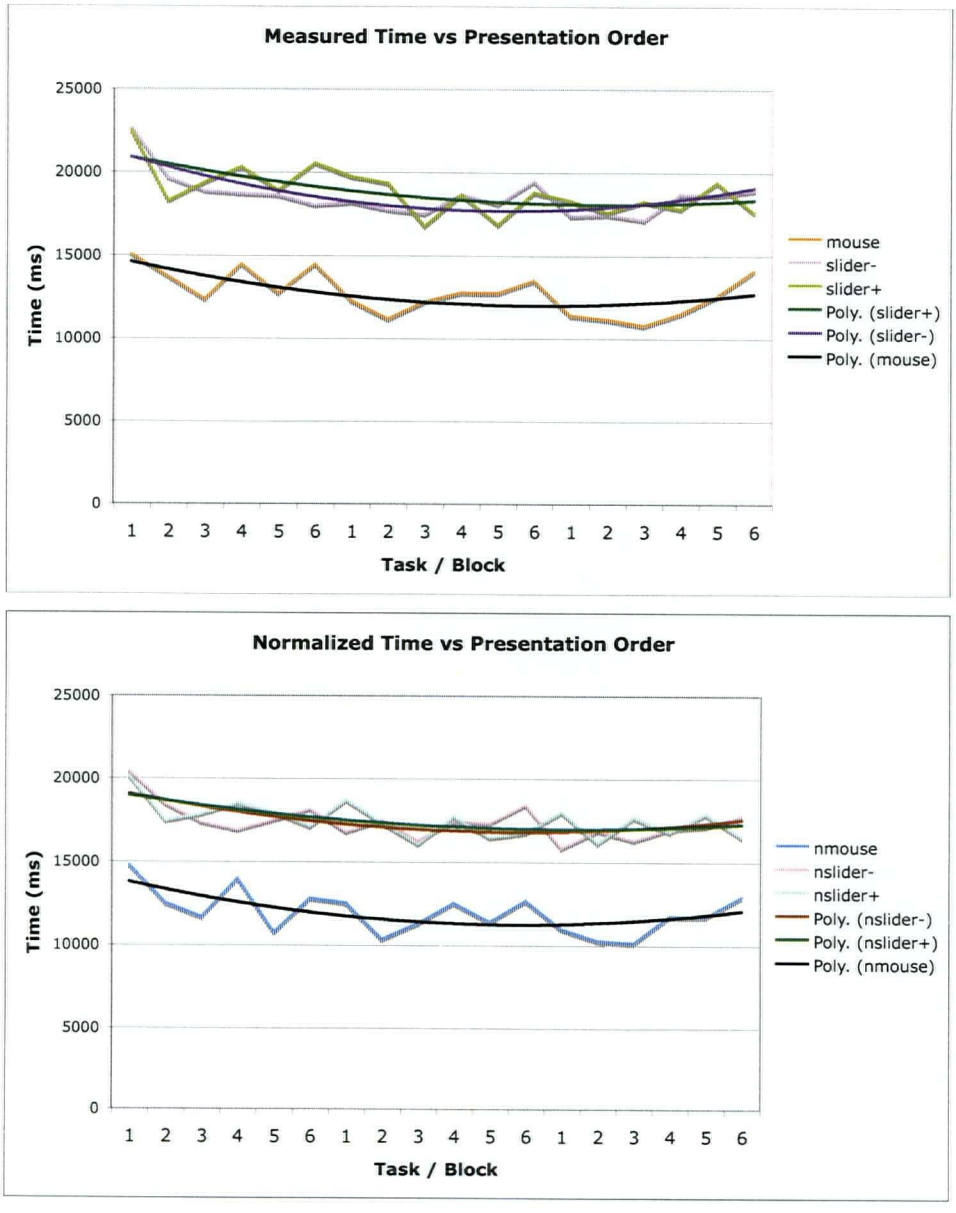


Figure 7.6: Task time and PRESENTATION ORDER, with polynomial curve fit lines added.

Participant	PedalPrompts	PedalCorrect	%Correct
4	70	67	96%
5	67	59	88%
6	58	57	98%
7	69	58	84%
8	72	31	43%
9	54	33	61%
10	46	42	91%
11	53	37	70%
12	71	62	87%
13	47	37	79%
14	73	50	68%
15	97	95	98%
16	62	55	89%
ALL	839	683	81%

Table 7.5: Performance on the distraction task by individual study participants. The table shows the total number of task prompts that were presented to the user, compared with the number of periods between prompts where the user made a correct response according to the prompt.

not be measured using direct questioning. Finally, question 6 was added based on qualitative feedback obtained during earlier studies in which users expressed a strong preference for the piezoelectric TD versus a typical mobile phone vibrator due to lower noise and intrusiveness; for this experiment, it was hypothesized that pre-existing attitudes about mobile phone vibration might be correlated to performance with the TD.

Unfortunately, because no significant effect was observed for the presence or absence of tactile feedback, it is impossible to correlate the measured attitudes with individual subjects' performance on the task. However, the presence of this data anchors the demographic picture of the subject population and reveals a significant diversity in attitudes about multitasking while using a handheld device, which lends ecological validity to the study.

7.9.2 Qualitative Evaluation of the Distraction Task

The goal of the distraction task was to cause the participant to split his/her visual attention resources in a manner similar to that of a person using a mobile device in a real-world usage context. Data from the experiment indicates that the task was mostly successful in this regard. The facilitator observed that all participants frequently shifted their visual attention between the handheld browser's LCD and the PC monitor (where the distraction task prompts were displayed). Video analysis of the author and several research colleagues performing the distraction task together with the main experimental task further confirmed that the interval between attentional shifts was on the order of a few (2 - 10) seconds and often coincided with times when the web page was being reloaded, which matches what was observed in previous field studies of mobile users [40].

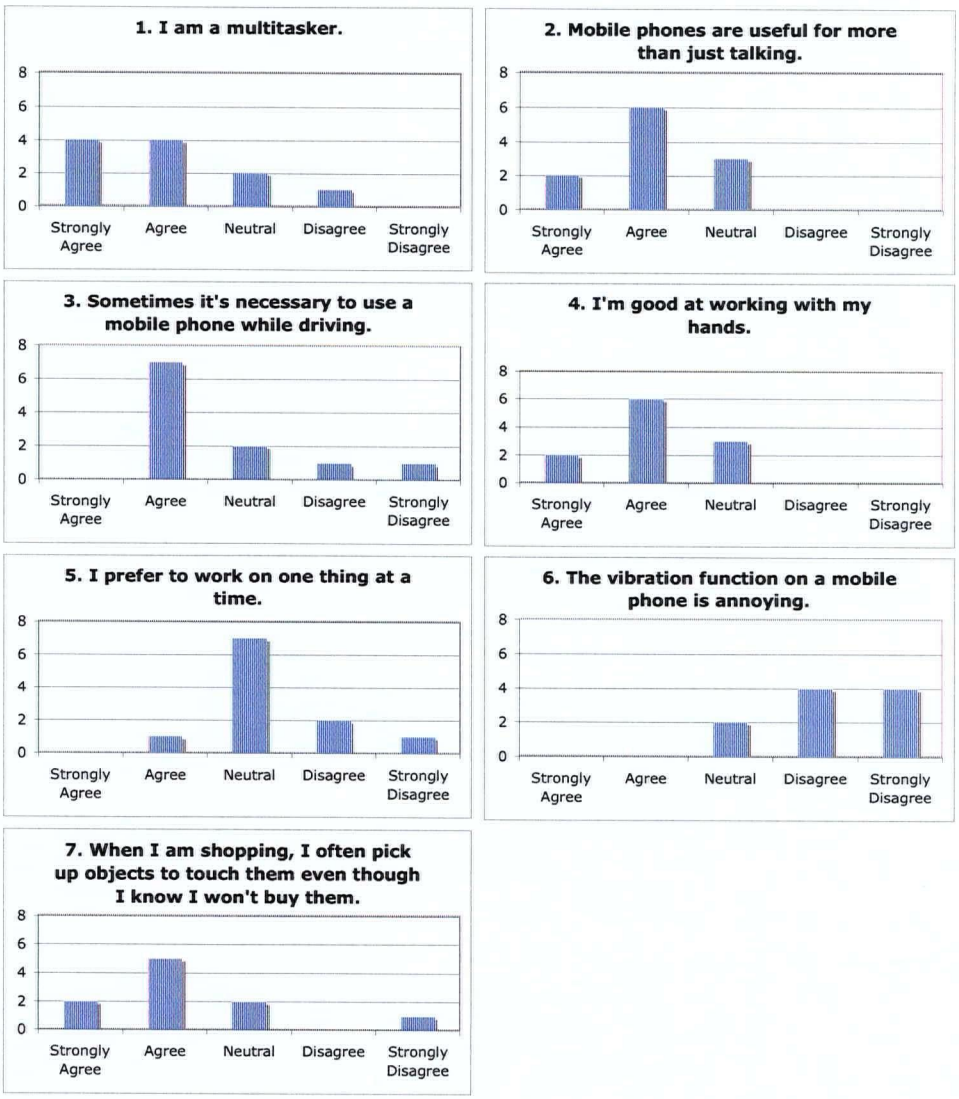


Figure 7.7: Results from the pre-task attitudes survey for the 13 main study participants.

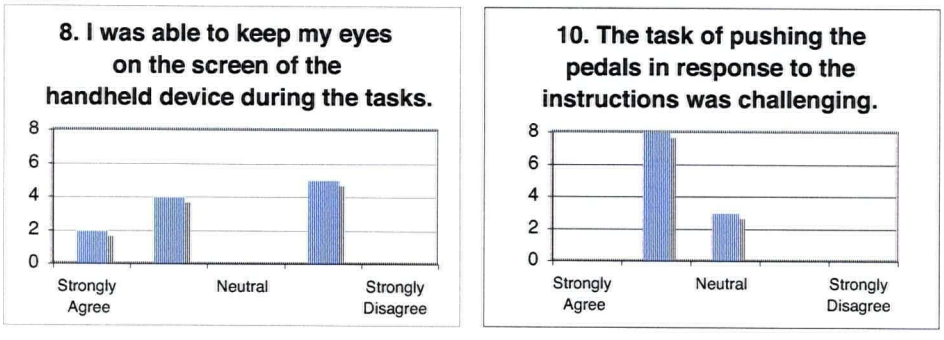


Figure 7.8: Results of the post-task attitudes assessment questions pertaining to the distraction task.

The qualitative data gathered from the participants after they had finished the tasks (Figure 7.8) mostly supports the hypothesis that the distraction task was successful in achieving its goals. There were mixed responses to Question 8 ("I was able to keep my eyes on the screen of the handheld device during the tasks..."), possibly due to a lack of awareness of where visual attention was directed; indeed, the participants were not asked to pay attention to where they were looking until after all the tasks had been completed. However, most subjects agreed with the prompt in Question 10 ("The task of pushing the pedals in response to the instructions was challenging..."), indicating that the distraction task was at least placing some attentional demands on the subject.

In post-task interviews, several participants expressed frustration with the level of system feedback on the distraction task. Feedback was only given at 7-second intervals when the prompt was changed, not immediately following the user input, so users were sometimes confused as to whether they had succeeded in entering the requested input. It was especially difficult for participants due to the use of two simultaneous tasks in an attention-starved paradigm (the facilitator observed that this caused subjects to forget which inputs they had made), and the poor mechanical feedback of the pedal hardware, which was originally designed for continuous, analogue input. However, the distraction task was intended to require users to divide their own attention within seven-second blocks rather than to interrupt or preempt the user at machine-specified times; immediate, asynchronous feedback would have interfered with this model somewhat.

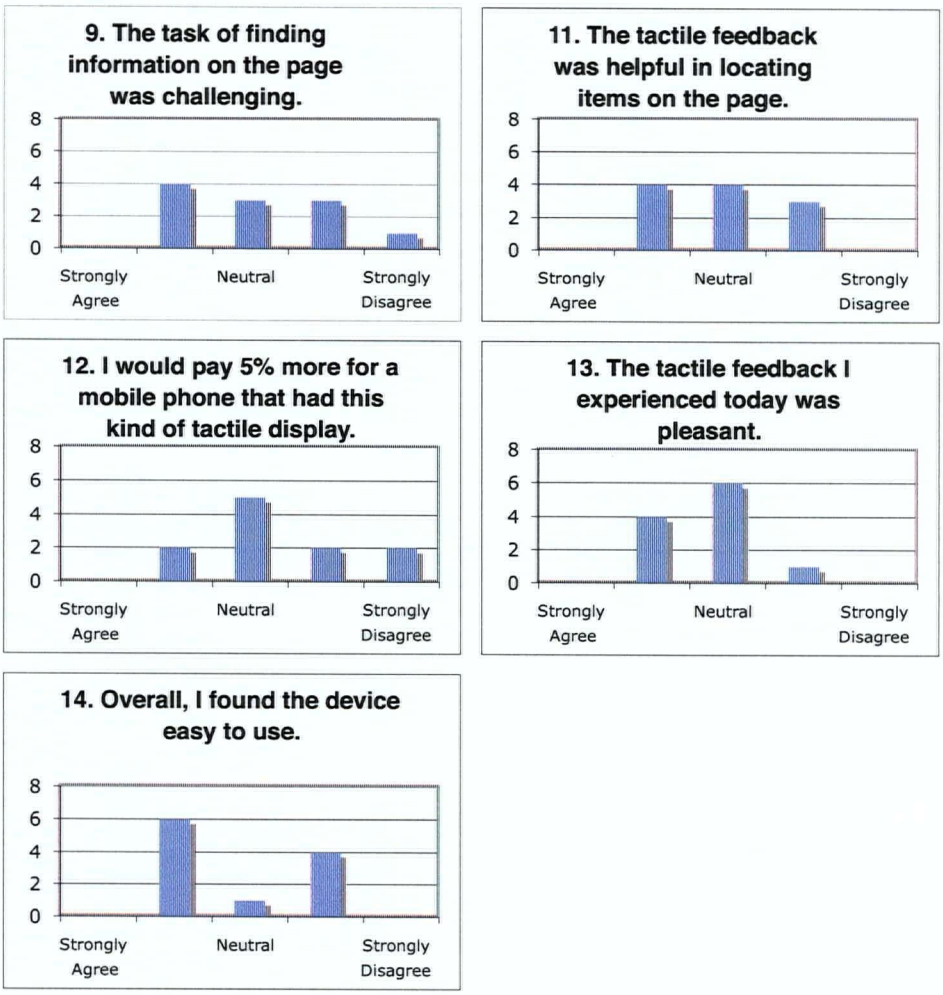


Figure 7.9: Results of the post-task attitudes assessment questions pertaining to the navigation task.

7.9.3 Qualitative Evaluation of the Navigation Task

The questionnaire data gathered from the participants on the main (i.e., navigation) task are shown in Figure 7.9. Additionally, the following feedback was obtained during the post-task interviews:

- Most (12/13) participants felt that their performance was best in the MOUSE condition. Hardware-related reasons included more familiarity with the mouse scroll wheel model (5/12) and better ergonomic placement of the hands (6/12).

- Five out of 13 participants said they thought their performance was better with active tactile feedback. Reasons cited included a qualitative preference for being able to feel the page elements.

- Almost all of the users criticized the velocity-control scrolling model compared to the mouse scroll wheel model. The most commonly cited reason was that continuous scrolling was less efficient than discrete (i.e., link-to-link) scrolling, although 3/13 users also expressed discomfort with the irregular jumping motion of the mouse scrolling in areas where links were sparse.

- Surprisingly, 4/13 users reported no awareness of the tactile feedback until they encountered the post-task assessment questionnaire. One user said they did not notice the tactile feedback until about halfway through the study. This was despite all users having been informed that they may experience tactile feedback during the study. The perceptual load due to the distractor task may have had an effect on this lack of awareness.

- Miscellaneous comments and suggestions included improving the hand position for better ergonomics, reducing the inertia of the slider for lower fatigue, and improving the contrast and clarity of the embedded LCD screen.

7.10 Discussion

Returning to the questions posed at the start of the study:

1. *Does the presence of tactile feedback, in the form of a synthetic* tactile flow *signal, affect the time required to complete a web navigation task?*

   It does not appear that tactile flow feedback aids targeting and navigation performance in the browser model used. By carefully inspecting the data along various dimensions (subject, PRESENTATION ORDER, etc.), and increasing the power of the experiment through the task difficulty normalization, we can more confidently state that there does not appear to be an effect of tactile feedback in the measurements taken. While this is clearly disappointing in light of the hypotheses formulated at the start of the experiment, the findings enable us to focus on areas for improvement of the device rather than on issues with the experimental design.

2. *How does the experience of browsing and navigation on the hand-held prototype compare to a more common paradigm (the mouse scroll wheel)?*

   The familiarity and ergonomic comfort, as well as the discrete scrolling model, of the mouse scroll wheel were preferred.

3. *What are users' subjective reactions to the use of tactile feedback in this application?*

   Participants were mostly either ambivalent about the tactile feedback or reacted positively.

4. *What are users' subjective reactions to the handheld prototype hardware and browser software?*

   The prototype was mildly criticized for its ergonomic profile and weight.

5. *Are there any correlations between user demographics, such as age, gender, and previous mobile experience, and their performance in the browsing task with or without tactile feedback?*

   Due to a lack of overall statistical significance between the two SLIDER conditions, it was not possible to obtain correlative data on these demographic characteristics.

Why was performance seemingly unaffected by the *tactile flow* rendering setting? Prior to the start of the experiment, we hypothesized that the following factors, if satisfied, may influence the measured performance in the tactile feedback condition:

1. The tactile rendering could be perceived as a sense of tactile flow by the user.

2. The tactile flow signal would simulate moving spatial cues that occur in the real world when the user is touching objects.

3. The simultaneous, multimodal input of both visual and haptic cues would allow the user to more quickly gain a sense of spatial location in a virtual space (i.e., web page).

4. Given a more rapid, more confident acquisition of spatial location, targeting performance would be increased.

5. Any performance changes would be measured by the experimental paradigm.

The first factor should be true, based on the fact that the parameters used were within the limits established by the Perceptual Characterization studies, and the haptic icon design effort to maximize the amplitude of the percept created by the TD. However, the fact that several users did not notice the presence or absence of tactile feedback under conditions of high perceptual load suggests that the amplitude may be on the low (i.e., weak) side. Since the signals were already mostly optimized, the remaining limitations are primarily in the hardware and the physical constraints of the piezoelectric elements used in the display. Weak stimulation could be responsible for the lack of a statistically significant performance difference, but it seems unlikely that *no* performance difference would be observed due to low amplitude alone, since users are universally able to detect the tactile feedback when their attention is directed towards it.

Regarding the second and third factors, the synthetic tactile flow rendering differs from typical real-world touch in a significant way: in natural touch, a tactile flow percept is generated when the fingertip is moved (i.e., slid) across a textured surface, usually accompanied by the proprioceptive sensation of the directed movement of the body part. In the browser model, the tactile signal is decoupled from its typical proprioceptive accompaniment and controlled with a novel force- and position-sensitive virtual scrolling model. Moreover, in the world of natural touch, *externally* modulated sliding (tactile flow) signals are usually limited to accidentally losing one's grip

on an object, or an insect crawling across the fingertip, where the semantic interpretation of the tactile signal is not one of spatial position, but of an alert. A synthetic, decoupled tactile flow signal, therefore, has a significant challenge to overcome in being a novel way of providing a spatial cue.

The fourth factor is likely to be true on the basis of general psychological principles of semantic activation and directed movement. In addition, a thorough analysis of the experiment, combined with the fact that performance differences were observed in the MOUSE condition, lends confidence to the fifth condition. Therefore, it seems that the most likely explanation for the lack of performance improvement was that the concept of decoupled tactile flow signal as a spatial cue was not validated by the current experiment.

How can the browser be improved? In addition to the performance data, a significant amount of qualitative data about the browser in the hands of a human user was collected. The study suggests that user comfort can be increased by improving the ergonomic characteristics of the device, including thumb position, size, and weight. Instead of continuous velocity-based scrolling, the navigation model could be discretized, either partially (as in the case of the mouse scroll wheel) or wholly (as in the case of scroll keys), allowing users to more easily target links. A discretized navigation model could use different haptic signals, including icons that use tactile flow for identification rather than spatial cueing, to widen the vocabulary of tactile signals and perhaps become useful not as spatial cues, but as eyes-free semantic cues as to the content of the web page. These opportunities for further development are considered in further detail in the conclusion in the following chapter.

In summary, the key contributions of the Browser User Experiment are that, having thoroughly tested the application of *tactile flow* in a browser model, we can rule out further optimization of haptic icons, scroll model parameters, browsing content, and experimental design, and instead direct our efforts towards application designs that do not rely on decoupled tactile flow rendering as a spatial cue. Qualitative feedback obtained during the study gives clues about how to significantly improve the hardware and navigation models. We have also established an experimental paradigm, including a distraction task and novel, web-based experiment management software, which can be used to evaluate further progress in handheld, multimodal interfaces.

Chapter 8

Conclusion and Future Work

In this chapter we summarize the key contributions of this work, revisit the research questions posed earlier, and outline promising future directions enabled by this work.

8.1 Summary of Key Contributions

This thesis makes the following contributions to the body of knowledge about interaction design for mobile haptics.

8.1.1 Identification of a novel multimodal approach to addressing limitations in mobile user interfaces

Conventional approaches to mobile interaction design using existing hardware suffer from limited interface bandwidth and poor suitability for mobile conditions of visual and/or auditory impairment. Contributions of this work include the articulation of this need, a survey of existing haptic technologies and the selection of the piezoelectric actuated lateral skin-stretch technology, mounted in a simplified 1-D slider configuration, as a tractable method for addressing the requirements of mobile haptics. Compared to existing vibrotactile approaches, our concept of mobile haptics enables richer tactile experiences while managing cost, power, size, and weight tradeoffs. The

validity of this selection has been demonstrated by the realization of a handheld prototype supporting interaction in *both* haptic and visual modalities.

8.1.2 Development of a new handheld haptics hardware platform

Having selected the problem domain and the tactile display technology, we[1] proceeded with implementation of a hardware and software platform for prototyping and testing handheld haptic applications. This inexpensive, reproducible prototype, including the various technical developments incorporated within, represents another contribution of this research project.

Even after the major engineering effort on the prototype (carried out at McGill University) was completed, the hardware and software specification evolved through insights obtained by using the platform for application development and conducting user tests. For example, the need for a spring-based return-to-centre feature for velocity control was identified during application development. The specification of the prototype platform at the time of writing represents the result of this iterative development. The existence of the prototype enables further research in handheld haptic applications without the need for development of entirely new hardware.

[1] The hardware prototype development was a collaborative effort with members of the McGill University Haptics Lab. The present author was a contributor at all stages of the prototype development, but the electronic and mechanical design was lead by Jerome Pasquero, and the control software design was led by Vincent Levesque. For more information, please see Chapter 4.

8.1.3 Evolution of application design concepts based on user studies and hardware development

This thesis describes a suite of novel application designs that are matched to the capabilities and limitations of the handheld prototype. In addition to the detailed initial specification based upon a design philosophy and background research, this work presents evolutionary development of the application concepts based on insights learned during the prototyping process. For example, the size of the "vocabulary" of haptic icons available for use with the browser application — and thus the range of haptic icon functionality — was discovered through perceptual characterization experiments (Chapter 5).

8.1.4 Method for rapid prototyping and graphical representation of tactile stimuli

As described in Section 4.9, novel approaches for visualizing voltage outputs, spatial *volt images*, and skin *stretch images* were developed to address the previously cumbersome process of creating, managing, and analyzing tactile stimuli. In addition, an accessible prototyping procedure using standard web-based graphical tools, and support utilities for automated parsing of the visual representations, are introduced. These methods can be generalized to any tactile display that produces spatially distributed patterns of skin stimulation.

8.1.5 Perceptual characterization of a novel miniature piezoelectric tactile display

Rather than immediately developing applications based on the developers' own vague and subjective impressions of the capabilities of the new system, we first conducted a detailed user evaluation to determine its "vocabulary" of signals that human users are capable of perceiving. Both the results of these user studies, and the validation of this method as a useful component of user-centred design for novel output hardware, represent contributions of this work.

We determined that directional *tactile flow* can be perceived reliably at speeds up to 0.34 metres per second, and that haptic icons may be designed along dimensions of — in order of decreasing saliency level — *repeating* versus *nonrepeating* waveform, *direction*, *speed*, and *amplitude*. These findings may be put to use by anyone who is interested in developing applications for a tactile display with similar physical parameters to the one we used.

8.1.6 Handheld browser application with tactile enhancement

As an example of one high-fidelity application prototype, we constructed a multimodal browser capable of reading web pages and displaying them simultaneously with visual and haptic representations. The method for translating web markup to a haptic page map, and the navigation model for a 1-D slider are examples of novel contributions of this effort.

Usability testing of the handheld browser did not demonstrate a performance (task time) advantage for navigation with directional tactile cues (*tactile flow*), although the qualitative feedback obtained from the users sug-

gests that there may be a subjective preference for the tactile confirmation of target acquisition. The results may be used to guide further development of applications (Section 8.3, below), and to highlight areas where hardware improvements are likely to have the most significant effect on usability (Section 8.4).

8.1.7 Method for usability testing of mobile applications

A contribution of this thesis is the method used for testing a handheld device in the laboratory while simulating some of the cognitive and perceptual load aspects of mobile interaction in the real world. The system incorporates a pedal-based distraction task that requires the user to engage the feet and visual system without interfering with the hands, similar to the scenario of a person walking while using a device.

An experiment manager software application, built using modern web technologies, provides cumulative reinforcement feedback to the user and collects data in a reliable, distributed fashion. This software is available for re-use by any researcher who wishes to conduct a user experiment using simulated mobile conditions.

8.1.8 Case study of a user interaction design process for haptics

Finally, the activities described in this thesis can be considered, from a case study perspective, as one full iteration of a user-centred design process (Figure 1.2) that begins with an analysis of user needs and proceeds in a stepwise cascade of discovery and increasing understanding of the interaction between the system and the user. The results from each stage of this process

provide input for further iterative improvement.

This work presents a specialized case of the classic user-centred design archetype, one that is specifically tailored to the needs of contemporary research in haptics. By demonstrating that it is possible to follow the user-centred approach even with nascent technology and heavy dependency on hardware configuration, this work lends validity to the concept of incorporating users early in a design process in order to focus on achieving good usability and to minimize system- or technology-centric design.

8.2 Research Questions

The research questions posed at the beginning of this thesis were addressed in the following ways.

1. *What are the problems with existing mobile user interfaces that may be addressed using haptics?*

   In Chapter 1 we identified two fundamental problems of existing mobile interfaces: *sensory bandwidth limitation* and *environmental (contextual) competition for visual and auditory attentional resources.* Haptics is well suited to address these challenges through parallel feedback in a sensory modality that is still mostly unused in mobile interaction.

   In Chapter 3 we described the initial stages of an iterative application design process that has identified

2. *How can one implement haptics on a mobile device despite power, size, and weight restrictions?*

   In Chapter 2 we discussed existing approaches for haptic transducers and identified the piezoelectric lateral skin-stretch technology as a

promising candidate for further investigation. The resulting piezo tactile display represents a compromise: it is smaller, lighter, consumes less power, and avoids the mechanical grounding challenges of motors used for force feedback, but it is also more complicated than a typical mobile phone vibrator. However, the technology we used also offers a significantly richer haptic experience than mere vibration.

In addition to the selection of the tactile display itself, our investigation into various alternatives for mobile haptics hardware resulted in the choice of a linear slide-mounted tactile display positioned on the side of the device as a working compromise between complexity (and thus, size and weight) and functionality. After application prototyping and user testing, we now know that features such as a spring return-to-centre and increased slider travel (perhaps enabled by a different mounting configuration) should be considered for the next iteration of design, in order to improve the usability of the hardware when used for cursor navigation.

Since our choice of technology was partially influenced by practical considerations such as access to manufacturing facilities, it is conceivable that if the process was replicated by someone else, a different hardware focus might have been chosen. For example, someone with access to MEMS (Micro-Electro-Mechanical Systems) facilities, or specialized intellectual property, might have been able to create a hardware prototype just as rapidly as we were, without using off-the-shelf parts. However, the interaction designs we proposed prior to the selection of the hardware technology are generalizable in the sense that they do not assume large grounded forces (which would be impractical on

a mobile device regardless of haptic display technology), and provide a "bridge" between existing mobile user needs and a rough hardware specification that may be implemented using a variety of technologies.

3. *What are the expressive capabilities of the hardware prototype in the hands of a human user?*

   In Chapter 5 we described the results of three perceptual characterization experiments that revealed the usable stimulus speed range, distinguishable haptic icon design parameters, and hierarchy of salience of various parameters such as speed, direction, waveform and amplitude.

4. *What are the engineering challenges associated with building a high-fidelity hardware and software prototype with handheld use in mind?*

   Through the process of designing and building the prototype platform and its application software, we discovered a variety of new information related to the realization of a handheld haptic device. For example, on the hardware side we were the first to validate the use of a stiffer double-pinning configuration for the piezo benders as a key factor enabling miniaturization (Section 4.3.1).[2] Through experiments with the control software algorithms,[3] we discovered that a 12-Hz refresh rate was insufficient, while an 83-Hz refresh rate was sufficient for producing smooth tactile sensations on this TD. Finally, we encountered the problem of visualizing and prototyping haptic icons, and addressed it using the methods described in Section 4.9.

[2]The double-pinning configuration was invented by project collaborators Jerome Pasquero and Qi Wang.

[3]The control software development and optimization was carried out in collaboration with Vincent Levesque.

By building a representative software application (the browser) we learned that, under the present system configuration, an asynchronous design with priority given to the tactile process over the visual process was necessary for good haptic performance. The need for a sophisticated cursor navigation model was also recognized, with iterative development eventually resulting in a hybrid position/velocity control model that promotes active tactile exploration within the central region of the slider travel while allowing traversal of larger areas than would be possible with position control alone.

5. *Is* tactile flow *an effective aid for user navigation?*

   The user studies conducted with the browser prototype did not demonstrate a significant performance (task time) advantage for *tactile flow* as an aid to navigation or target acquisition in the browsing context. However, qualitative feedback from the same studies was often positive, with many users indicating a subjective preference for the extra tactile feedback. Further user studies could perhaps better elucidate the ways this multimodal interaction could be used to better support user goals other than more rapid navigation.

6. *How can user-centred design methodology be applied to haptics, where hardware technology is still the primary determinant of user interface capabilities?*

   It is a common misconception that emerging fields like haptics require designers to do a lot of up-front effort on the display technology before involving users, due to the complexity and lack of standardization of the interface hardware. User testing, when done, is typically utilized

for fine-tuning the software applications, or simply validating the usefulness of a particular approach. At the same time, it is well known that incorporating user studies early in the design process is essential for achieving maximum usability benefit per development effort [22].

Some of the methods we used to incorporate user testing *early* in the design process, such as perceptual characterization, have also been used in previous studies (e.g., [12]). However, in the present study, we have conducted a full cycle of development, in which usability considerations were the primary driver at each stage — from concept work that began with an inquiry about user needs and delayed the detailed specification of hardware until *after* usage scenarios had been determined, to the further development of the application concepts informed by perceptual characterization studies, and finally, to the two-stage development of the browser application consisting of a low-fidelity prototype followed by user testing, and a high-fidelity prototype which was then formally evaluated.

Therefore, we have demonstrated that, even when the user interface is highly dependent on technological limitations, it is possible to follow a user-centred design process and to benefit from the streamlined path towards good usability that this approach allows.

8.3 Future Work: Application Designs for Further Investigation

Informed by the results of the hardware prototyping, perceptual characterization, and high-fidelity software prototype and user test with one selected

application, we may now return to the application design stage to consider the feasibility of further applications based on the same handheld haptics concept.

8.3.1 Applications Involving Shape Rendering

Applications involving shape rendering are those that take advantage of the piezoelectric tactile display's ability to mimic patterns of skin stretch analogous to those created by touching small objects and surface features.

Handheld Braille Reader

The ability of a piezoelectric lateral skin-stretch tactile display to render legible Braille dots via active tactile exploration has been previously demonstrated in [30]. A mobile version has various applications for blind users including discreet text messaging or reading printed characters with optical character recognition (OCR). The only hardware change that would be necessary for practical implementation would be the use of two rows of piezo arrays for each of the two vertical Braille dot positions. The existing hybrid position / velocity scroll model could be used to overcome slider travel limitations, but further testing will have to be done to determine the optimal parameters for legibility of Braille.

Medical Application: Displaying Small Shapes

Although not strictly a mobile application, the ability to digitally transmit and reproduce small shapes could enable a remotely located doctor to tactually palpate an area on a patient which is sensed by a high resolution tactile sensor such as the one described in [37]. An example might be detailed ex-

amination of the results of a tine test, which produces distinctive patterns of swelling depending on antibody status, which are often much more difficult to distinguish visually than tactually.

8.3.2 General Haptic Icon Applications

In the following group of applications, the TD is used as a fixed output transducer, with minimal direct coupling between user input and tactile output. This type of interaction was tested and verified in the MDS and speed study experiments, and does not assume untested concepts such as spatial acquisition by tactile flow rendering. Therefore, it is expected to have a high probability of successful deployment. However, factors such as environmental distraction and matching between the information content of the output and the TD's capabilities must still be considered.

Instant Messaging / Notification

As described in Section 3.3.4, tactile notification of presence status changes could enrich the experience of using instant messaging or other social applications. It is expected that the effects of haptic enhancement would be most pronounced under conditions of workload [12] or environmental distraction.

8.3.3 Spatial Signalling

This group of applications makes use of the TD's capacity to display a directional signal. As the MDS analysis demonstrated, users can distinguish at least two directions, three speeds, and two amplitudes in a optimal controlled laboratory setting. The semantic usefulness of those parameters in a real-world context of multiple distractions and variable tactile engagement

still needs to be determined.

Navigation

As described in Section 3.3.3, a directional signal may be utilized for navigation feedback. Based on the results of the browser user study, it is not clear that the directional signalling will produce an intuitive directional response in the user. Rather, it is perhaps more likely that the user would come to associate a specific haptic icon (in this case a directional signal) with a semantic meaning, and produce responses appropriately. However, this hypothesis requires further testing.

8.3.4 Browser Improvements

While the concept of tactile flow rendering to improve spatial navigation was not validated in the present study, there is room for numerous improvements in the browser prototype. These include:

- General software optimization to improve I/O performance, perhaps moving to a real-time operating system and/or embedded controller for the tactile loop.
- Dynamic haptic icon rendering based on the pixel height of page elements.
- An algorithm to properly handle horizontally laid out elements that overlap in the vertical domain. This would require a more complex 2D (visual page map) to 1D (haptic page map) transformation engine than is currently implemented.

- Maximum usable scrolling speed can perhaps be further increased by adding a "high speed mode"[4]: when the user is scrolling at high speed, the normal haptic icons can be dynamically substituted with alternate representations optimized for maximum amplitude, and perhaps "stretched" so that they remain under the finger longer, or by displaying only the most semantically important elements. This favours speed over "vocabulary". The range of distinguishable haptic icons would be reduced while the system is in high speed mode, since the *amplitude* parameter and some *waveforms* would no longer be available. Such a modification, therefore, must be considered carefully with respect to the desired haptic tagging scheme.

8.4 Future Work: Hardware Improvements

Of course, numerous opportunities exist for further improvement in the prototype hardware specification. Many of the challenges associated with building working applications using the prototype are related to the slider, which despite being auxiliary to the tactile output device has a major effect on the user experience. Increasing the slider travel (perhaps by reconsidering the mounting location), and finding appropriately stiff springs for the hardware return-to-centre feature could improve the scrolling model.

The first 1-D piezo tactile display, the Virtual Braille Display, had a slider travel that was an order of magnitude longer than the present miniature tactile display. This enabled a simple position-control mode, but importantly also allowed the user to use proprioception to sense the slider position without having to mentally integrate velocity and time. Neither of these usability

[4]This idea was contributed by thesis reader Dr. Steve Wolfman.

advantages are present in the current handheld version.

Finally, presently the push to select, or "click", feature of the device is implemented as a switch. By replacing the switch with an analog force sensor such as a force sensitive resistor (FSR), and perhaps reconsidering the mounting appropriately to deliver the force linearly to the sensor, additional interaction models may be considered that modulate the stimulus amplitude based on how strongly the user is pressing down on the TD, enabling richer tactile exploration.

8.5 Conclusion

In this thesis we have described one full iteration of an interaction design and hands-on inquiry process (Figure 1.2). We began by identifying a problem with broad relevance: the usability limitations of contemporary mobile devices. Under the working hypothesis that haptics might be a good candidate for addressing this problem, we conducted conceptual design work informed by an inquiry into existing usage patterns and user needs. User requirements and system requirements emerged concurrently, eventually allowing us to proceed with the development of a high-fidelity, "deep" hardware prototype using a number of novel technologies, in partnership with an expert collaborator. Once some of the device's expressive capabilities were known, we proceeded with the development of a high-fidelity browser application prototype. Finally, it was then possible to use more conventional usability techniques to assess the performance characteristics of the browser.

Through the exercise of designing, building, and testing the mobile haptics concept, we learned that user-centred interaction design for haptic interfaces requires the use of slightly different methodologies than conventional

user-centred interaction design for primarily visual interfaces. First, because richly expressive haptics — especially in the mobile domain — is still in its infancy, the hardware used for rendering computer signals into human sensory percepts is far more varied and weakly standardized than for the visual domain. This implies that it is not possible to do "pure" user-centred interaction design that is entirely system (and hardware) agnostic. Instead, application and hardware design must go hand-in-hand; however, in our approach we have striven to follow the spirit of good user-centred design techniques by beginning the inquiry into user needs and application scenarios *as early as possible*, using the minimum set of assumptions about the specific hardware implementation. As design work proceeded, both the target hardware profile and the application designs evolved synergistically, informed by increasing knowledge about user needs and candidate hardware designs.

Another special characteristic of designing for mobile haptics is the utility of formal perceptual characterization based on user studies. While the sensory gamut of a visual display might be readily characterizable using a set of standardized measures and a simple visual inspection, haptic signals are often far more subtle, complex, and difficult to characterize. Therefore, when a new haptic output device is developed, it is helpful to understand its sensory gamut — or measure of *expressiveness* — using the techniques described in this thesis, to inform the design of applications that best take advantage of the tactile display.

The identification and validation of practical interaction design techniques for mobile haptics, concept designs for applications based on a minimally assumptive hardware profile, high-fidelity hardware prototype, high-fidelity browser application prototype, and user study findings that shed

light on the characteristics of a miniature lateral skin-stretch haptic system represent major contributions of this work. For future work, it is reasonable to expect that another iteration of the *same* process, informed by the findings from each step of the previous iteration, will produce further improvements in hardware and application usability. Such predictability enables researchers and engineers to engage in mobile haptics development in a *planned* fashion, and it is hoped that this method will increase the accessibility, usefulness, and usability, putting this exceedingly promising technology in the hands of users.

Bibliography

[1] The adaptive communication environment (ace). http://www.cs.wustl.edu/ schmidt/ACE.html.

[2] Gif89a specification. http://www.w3.org/Graphics/GIF/spec-gif89a.txt.

[3] The gimp toolkit (gtk) library. http://www.gtk.org/.

[4] Imagemagick library home page. http://www.imagemagick.org/.

[5] Microsoft intellimouse optical specification. http://www.microsoft.com/hardware.

[6] Corin R. Anderson, Pedro Domingos, and Daniel S. Weld. Personalizing web sites for mobile users. *Proceedings of the 10th Conference on the World Wide Web (WWW10)*, 2001.

[7] T.G. Anderson. Human-computer interface including haptically controlled interactions. United States Patent 6,954,899, October 11 2005.

[8] R. Bown. Multi-functional vibro-acoustic device. United States Patent 6,911,901, June 28 2005.

[9] A.C. Braun, L.B. Rosenberg, D.F. Moore, K.M. Martin, and A.S. Goldenberg. Directional tactile feedback for haptic feedback interface devices. United States Patent 6,864,877, March 8 2005.

[10] Stephen Brewster and Lorna M. Brown. Tactons: structured tactile messages for non-visual information display. In *CRPIT '04: Proceedings of the fifth conference on Australasian user interface*, pages 15–23, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[11] L. M. Brown, S. A. Brewster, and H. C. Purchase. A first investigation into the effectiveness of tactons. pages 167–176, 2005.

[12] A. Chan, K. MacLean, and J. McGrenere. Learning and identifying haptic icons under workload. pages 432–439, 2005.

[13] Angela Chang, Sile O'Modhrain, Rob Jacob, Eric Gunther, and Hiroshi Ishii. Comtouch: design of a vibrotactile communication device. In *DIS '02: Proceedings of the conference on Designing interactive systems*, pages 312–320, New York, NY, USA, 2002. ACM Press.

[14] Angela Chang and Conor O'Sullivan. Audio-haptic feedback in mobile phones. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1264–1267, New York, NY, USA, 2005. ACM Press.

[15] R. Cholewiak and C. Sherrick. A computer-controlled matrix system for presentation to skin of complex spatiotemporal pattern. *Behavior Research Methods and Instrumentation*, 13(5):667–673, 1981.

[16] Immersion Corporation. Vibetonz system. Product Literature, March 2005.

[17] Jack Tigh Dennerlein, David B. Martin, and Christopher Hasser. Force-feedback improves performance for steering and combined steering-

targeting tasks. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 423–429, New York, NY, USA, 2000. ACM Press.

[18] M. Enriquez and K. E. MacLean. Common onset masking of vibrotactile stimuli - poster. In *Proc. World Haptics*. IEEE, 2005.

[19] F. Geldard. Some neglected possibilities of communication. *Science*, 131(3413):1583–1588, 1960.

[20] F. Gemperle, N. Ota, and D. Siewiorek. Design of a wearable tactile display. pages 5–12, 2001.

[21] Masataka Goto. Smartmusickiosk: Music listening station with chorus-search function. *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST 2003)*, pages 31–40, 2003.

[22] J.D. Gould. How to design usable systems. In *Readings in Human-Computer Interaction: Toward the Year 2000*, pages 93–121. 1995.

[23] V. Hayward and M. Cruz-Hernandez. Tactile display device using distributed lateral skin stretch. In *Proc. of the Haptic Interfaces for Virtual Environment and Teleoperator Systems Symposium (IMECE2000)*, volume DSC-69-2, pages 1309–1314. ASME, 2000.

[24] H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In ACM, editor, *Proc. of CHI'97*, pages 234–241, 1997.

[25] R.S. Johansson and R.H. LaMotte. Tactile detection thresholds for a single asperity on an otherwise smooth surface. *Somatosensory Research*, 1:21–31, 1983.

[26] Alan Kay. Personal dynamic media. *IEEE Computer*, 10(3):31–41, 1977.

[27] R. L. Klatzky and S. J. Lederman. How well can we encode spatial layout from sparse kinesthetic contact? In *Proceedings of the 11th Symposium on Haptic interfaces For Virtual Environment and Teleoperator Systems (Haptics'03)*, page 179, Washington, DC, March 2003. IEEE Computer Society.

[28] Vincent Lévesque and Vincent Hayward. Experimental evidence of lateral skin strain during tactile exploration. *Proc. Eurohaptics 2003*, pages 261–275, 2003.

[29] Vincent Lévesque and Jérôme Pasquero. Stress library. http://www.cim.mcgill.ca/ haptic/.

[30] Vincent Lévesque, Jérôme Pasquero, Vincent Hayward, and Maryse Legault. Display of virtual braille dots by lateral skin deformation: feasibility study. *ACM Trans. Appl. Percept.*, 2(2):132–149, 2005.

[31] Robert W. Lindeman, John L. Sibert, Erick Mendez-Mendez, Sachin Patil, and Daniel Phifer. Effectiveness of directional vibrotactile cuing on a building-clearing task. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–280, New York, NY, USA, 2005. ACM Press.

[32] Jukka Linjama and Topi Kaaresoja. Novel, minimalist haptic gesture interaction for mobile devices. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 457–458, New York, NY, USA, 2004. ACM Press.

[33] Shannon Little. Theory, software, and psychophysical studies for the tactile handheld miniature bimodal device. Technical Report TR-2005-21, University of British Columbia, Department of Computer Science, 2005.

[34] Joseph Luk, Jérôme Pasquero, Shannon Little, Karon E. MacLean, Vincent Lévesque, and Vincent Hayward. A role for haptics in mobile interaction: Initial design using a handheld tactile display prototype. In *CHI '06: Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 2006. ACM Press.

[35] K. E. MacLean, M. J. Shaver, and D. K. Pai. Handheld haptics: a usb media controller with force sensing. pages 311–318, 2002.

[36] K.E. MacLean and M. Enriquez. Perceptual design of haptic icons. In *Proc. EuroHaptics 2003.* IEEE, 2003.

[37] Vivek Maheshwari and Ravi F. Saraf. High-resolution thin-film device to sense texture by touch. *Science*, 312(5779):1501–1504, June 9 2006.

[38] Michael McCloskey. Intuitive physics. *Scientific American*, 248:122–130, April 1983.

[39] B.A. Nardi, S. Whittaker, and E. Bradner. Interaction and outeraction: instant messaging in action. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, pages 79–88, 2000.

[40] 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 Press.

[41] J. Pasquero and V. Hayward. STRESS: A practical tactile display system with one millimeter spatial resolution and 700 Hz refresh rate. In *Proc. Eurohaptics 2003*, pages 94–110, 2003.

[42] J. Pasquero, J. Luk, S. Little, and K.E MacLean. Perceptual analysis of haptic icons: an investigation into the validity of cluster sorted mds. In *Proc. Of Symp. On Haptic Interfaces for Virtual Environment and Teleoperator Systems (IEEE-VR)*. IEEE, 2006.

[43] Jérôme Pasquero, Joseph Luk, Vincent Lévesque, Qi Wang, Vincent Hayward, and Karon MacLean. Distributed tactile display for rich haptic interaction with a handheld infomation display. *To appear in IEEE Multimedia*, 2006.

[44] C.A. Perez, A.J. Santibañez, C.A. Holzmann, P.A. Estévez, and C.M. Held. Power requirements for vibrotactile piezo-electric and electromechanical transducers. *Medical and Biological Engineering and Computing*, 41:718–726, November 2003.

[45] Ivan Poupyrev and Shigeaki Maruyama. Tactile interfaces for small touch screens. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 217–220, New York, NY, USA, 2003. ACM Press.

[46] Ivan Poupyrev, Shigeaki Maruyama, and Jun Rekimoto. Ambient touch: designing tactile interfaces for handheld devices. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 51–60, New York, NY, USA, 2002. ACM Press.

[47] Virpi Roto and Antti Oulasvirta. Need for non-visual feedback with long response times in mobile hci. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 775–781, New York, NY, USA, 2005. ACM Press.

[48] K. B. Shimoga. Finger force and touch feedback issues in dexterous telemanipulation. pages 159–178, 1992.

[49] David L. Strayer, Frank A. Drews, and Dennis J. Crouch. A comparison of the cell phone driver and the drunk driver. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 48(2):381–391, Summer 2006.

[50] Hong Z. Tan, Rob Gray, J. Jay Young, and Ryan Traylor. A haptic back display for attentional and directional cueing. *Haptics-e: The Electronic Journal of Haptics Research*, 3(1):20, June 11 2003.

[51] C.R. Wagner, S.J. Lederman, and R.D. Howe. Design and performance of a tactile display using rc servomotors. *Haptics-e Electronic Journal of Haptics Research (www.haptics-e.org)*, 3(4), September 1999.

[52] L. Ward. Multidimensional scaling of the molar physical environment. *Multivariate Behavioral Research*, 12:23–42, 1977.

[53] Scott Weiss. An alternative business model for addressing usability: Subscription research for the telecom industry. *ACM interactions*, pages 62–63, July-August 2005.

Appendix A

Browser User Evaluation Documents

The following documents related to the browser user evaluation are included in this Appendix:

1. Participant Demographic Data Collection Form and Qualitative Assessment Battery
2. Verbal Protocol (Experiment Instructions)
3. Post-Study Interview Data Collection Form
4. Task Inventory
5. Examples of Test Web Pages

Physical and multimodal user interfaces – usability and psychophysics

Study Questionnaire Form *UBC Behavioural Research Ethics Approval #B01-0470*

Instructions

Please respond to all of the items listed below. If you have any questions, please ask the facilitator.

Part 1: Background

1. Age: _____ years 2. Sex: **female** or **male**
2. First language(s): _____

Part 2: Survey

1. a) Did you play a musical instrument as a child? **yes** or **no** (please circle one)

   b) If 'yes', for how many years did you play? _____ years

   c) If 'yes', which instrument(s) did you play? _____

2. Do you **regularly use** any of the following: (check all that apply)
   - ❑ Mobile phone
   - ❑ PDA (personal digital assistant), including BlackBerry
   - ❑ Game controller with vibration or force feedback (PlayStation Dual Shock, etc.)
   - ❑ Mouse with a scroll wheel

3. Please respond to the following statements by placing an X in the box according to the following scale:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1. I am a multitasker.					
2. Mobile phones are useful for more than just talking.					
3. Sometimes it's necessary to use a mobile phone while driving.					
4. I'm good at working with my hands.					
5. I prefer to work on one thing at a time.					
6. The vibration function on a mobile phone is annoying.					
7. When I am shopping, I often pick up objects to touch them even though I know I won't buy them.					

Thank you! Please inform the facilitator when you are done.

Physical and multimodal user interfaces – usability and psychophysics

Study Questionnaire

UBC Behavioural Research Ethics Approval #B01-0470

Instructions

Please respond to all of the items listed below. If you have any questions, please ask the facilitator.

Part 3 – Post-Study Survey

4. Please respond to the following statements by placing an X in the box according to the following scale:

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree

8. I was able to keep my eyes on the screen of the handheld device during the tasks.

9. The task of finding information on the page was challenging.

10. The task of pushing the pedals in response to the instructions was challenging.

11. The tactile feedback was helpful in locating items on the page.

12. I would pay 5% more for a mobile phone that had this kind of tactile display.

13. The tactile feedback I experienced today was pleasant.

14. Overall, I found the device easy to use.

5. The facilitator will now ask you some brief questions.

6. Additional comments:

_____

_____

_____

Thank you for participating in this study!

Experiment instructions

Thank you for participating in this experiment today. Please review these instructions and if you have any questions, ask the facilitator. If you feel uncomfortable, you may quit the experiment at any time without penalty.

Holding the device

Hold the device in your left hand with your thumb resting on the side-mounted tactile display (TD). You can slide the TD up and down, and you will feel some spring resistance. Do not rub, pick, or otherwise abrade the surface of the TD. For best results, try to keep the TD centred under your thumb as you operate the device.

Tasks

First, you will have a chance to "practice" to get familiar with the controls and tasks. Then you will do a series of tasks that involve finding some information on a web page. Every sixth task, you may be asked to change devices – for example, using the mouse instead of the TD. If you are using the TD, you might feel tactile feedback, or you might not. There will be a one-minute break after each block of six tasks.

In addition to browsing web pages, you may be asked to do another task **simultaneously**. Every 7 seconds, an instruction will appear on the computer screen. You must press the pedals according to the instructions within 7 seconds, before the next instruction appears.

Operating the browser

Depending on the task, you can move the cursor using either the TD or the mouse scroll wheel. Highlight the link you wish to select, and select it using the ENTER key.

Timing

Timing information is being collected, so it is important that you do all the tasks **as fast as you can**. The procedure is as follows:

1. The facilitator will explain the next task.
2. When you are ready to begin the task, press the ENTER key. Timing starts from this point.
3. When you find the information requested, press the SPACE bar as soon as possible. This stops the timer.
4. Inform the facilitator of the answer.

Again, it is important that you press the SPACE bar **as soon as you see the answer**. Do this **before** you tell the answer to the facilitator.

If you make a mistake

On the bottom of each web page, there is a link labeled "Back" which will take you to the previous page. If you make a mistake, please continue trying to complete the task as quickly as possible. When you are finished, please inform the facilitator that a mistake was made during the task.

Physical and multimodal user interfaces – usability and psychophysics

Stock Inquiry Questions

About this document
The experimenter will converse with the participant using the following 'stock' questions to guide the inquiry.

1. During the experiment, you were asked to find information using either the mouse, or the handheld device with or without tactile feedback. **Did you feel that your performance was better in any of the three conditions**? If so, why?

2. Aside from the task of pushing the pedals and the general experimental setup, do you have any **suggestions for improvement**, for example in the way the handheld device is used, or the way tactile feedback is given?

A.1 Task Inventory

A1 What is the weather in London today?

A2 What will the weather be like in London tomorrow?

A3 What will the weather be like in London the day after tomorrow?

A4 What is the weather in Paris today?

A5 What will the weather be like in Paris tomorrow?

A6 What will the weather be like in Paris the day after tomorrow?

A7 What is the weather in Tokyo today?

A8 What will the weather be like in Tokyo tomorrow?

A9 What will the weather be like in Tokyo the day after tomorrow?

A10 What is the weather in Hong Kong today?

A11 What will the weather be like in Hong Kong tomorrow?

A12 What will the weather be like in Hong Kong the day after tomorrow?

A13 What is the weather in San Francisco today?

A14 What will the weather be like in San Francisco tomorrow?

A15 What will the weather be like in San Francisco the day after tomorrow?

A16 What is the weather in Toronto today?

A17 What will the weather be like in Toronto tomorrow?

A18 What will the weather be like in Toronto the day after tomorrow?

B1 If you take the 99 B-line from UBC at 1pm, when will you arrive at Broadway station?

B2 If you take the 99 B-line from UBC at 1pm, when will you arrive at Granville?

B3 If you take the 99 B-line from UBC at 1pm, when will you arrive at Main?

B4 If you take the 99 B-line from UBC at 9pm, when will you arrive at Broadway station?

B5 If you take the 99 B-line from UBC at 9pm, when will you arrive at Granville?

B6 If you take the 99 B-line from UBC at 9pm, when will you arrive at Main?

B7 If you take the #44 bus from UBC at 10am, when will you arrive at Macdonald?

B8 If you take the #44 bus from UBC at 10am, when will you arrive at Davie?

B9 If you take the #44 bus from UBC at 10am, when will you arrive at Waterfront station?

B10 If you take the #44 bus from UBC at 5pm, when will you arrive at Macdonald?

B11 If you take the #44 bus from UBC at 5pm, when will you arrive at Davie?

B12 If you take the #44 bus from UBC at 5pm, when will you arrive at Waterfront station?

C1 When is the movie "L'Enfant" playing at the Ridge Theatre?

C2 When is the movie "L'Enfant" playing at Fifth Avenue Cinemas?

C3 When is the movie "L'Enfant" playing at Empire Oakridge?

C4 What rating did the movie "L'Enfant" receive in its review in the Los Angeles Times?

C5 What rating did the movie "L'Enfant" receive in its review in the New York Post?

C6 When is the movie "The Promise" playing at the Ridge Theatre?

C7 When is the movie "The Promise" playing at the Pacific Cinematheque?

C8 When is the movie "The Promise" playing at the Silvercity Riverport?

C9 What rating did the movie "The Promise" receive in its review in the Globe and Mail?

C10 What rating did the movie "The Promise" receive in its review in the Washington Post?

Home

- What's New
- Weather
- Transit
- Movies
- Finance
- Sports
- News
- Manage Account

Browser-content version 29-mar-06
Terms of Service Privacy Statement

Weather

SPRING OUTLOOK

Canadians eagerly anticipate spring each year. Find out how soon you can shed the winter layers and expect the warmer weather to arrive.
>> More...

LOCAL WEATHER

Vancouver [Change]

13°C	15°C	10°C
Today	Tomorrow	Tuesday

TRAVEL FORECAST

- North America
- Europe
- Asia
- Africa
- South America
- Antarctica

Back to Home

North America

NORTH AMERICA WEATHER

- Boston
- Calgary
- Chicago
- Denver
- Edmonton
- Los Angeles
- Montreal
- Ottawa
- San Francisco
- Seattle
- Toronto
- Vancouver
- Washington, D.C.
- Winnipeg

Back to Weather
Back to Home

Weather - Travel Forecast

SAN FRANCISCO, CALIFORNIA

12°C	12°C	11°C
Today	Tomorrow	Tuesday

>> Extended Forecast

Back to North America
Back to Weather
Back to Home

TRANSIT INFORMATION

- News
- Planned Maintenance
- Company Information

BUSES AT UBC

004 - Phibbs Exch/Powell/Downtown/UBC
009 - Bdry/Bway Stn/Gran/Alma/UBC

017 - Oak/Downtown/UBC
025 - Brentwood Stn/UBC
041 - Joyce Stn/Crown/UBC
043 - Joyce Stn/UBC
044 - UBC/Downtown
049 - Metrotown Stn/Dunbar Loop/UBC

099 - Broadway Stn/UBC (B-Line)

258 - UBC/West Vancouver
480 - UBC/Richmond Centre

N17 - Downtown/UBC Nightbus

Back to Home

Transit - 44 Downtown

DEPARTURE TIME

9:00 am
10:00 am
11:00 am

12:00 pm
1:00 pm
2:00 pm

3:00 pm
4:00 pm
5:00 pm

6:00 pm
7:00 pm
8:00 pm
9:00 pm

Back to Transit
Back to Home

Transit - 44 Stops

44 UBC/Downtown

UBC BUS LOOP BAY 7
5:04 p
ALMA ST
5:18 p
MACDONALD ST
5:22 p
VINE ST
5:24 p
BURRARD ST / 3RD AV
5:26 p
PACIFIC ST
5:30 p
DAVIE ST
5:31 p
ROBSON ST
5:33 p
PENDER ST
5:34 p
WATERFRONT STATION
5:35 p

Back to 44 UBC/Downtown
Back to Transit
Back to Home

Showtimes - L'Enfant

L'Enfant

RIDGE THEATRE
1:30, 4:30, 8:00, 10:30

FIFTH AVENUE CINEMAS
5:00, 7:30

EMPIRE OAKRIDGE
12:30, 6:00, 8:00

>> MORE THEATRES

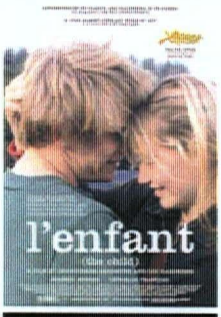
Back to L'Enfant
Back to Movies
Back to Home

Movies

News
Trailers

Featured Films

Basic Instinct 2

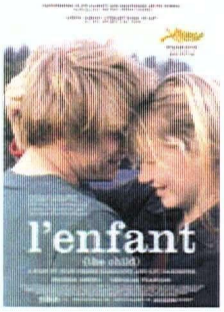


L'Enfant

Idlewild

The Promise

Movies - L'Enfant



Dispossessed twenty-year old Bruno (Jeremie Renier) lives with his eighteen-year-old girlfriend Sonia (Deborah Francois) in Seraing, an eastern Belgian steel town. They live off Sonia's unemployment benefits along with the panhandling and petty thefts committed by Bruno and his gang. Their lives change forever when Sonia gives birth to their child, Jimmy.

NOW PLAYING

Get Showtimes
View Trailer
Reviews

Foreign, Drama
Rating: Not yet rated
Jean-Pierre, Luc Dardenne (dir.)
Jeremie Renier
Deborah Francois

Back to Movies
Back to Home

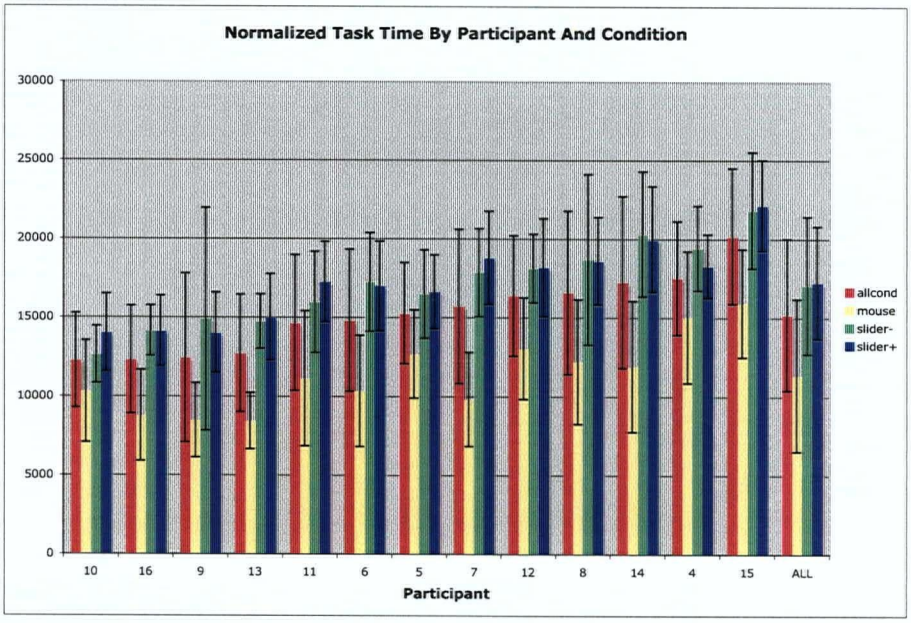
Appendix B

Browser User Study Supplemental Data

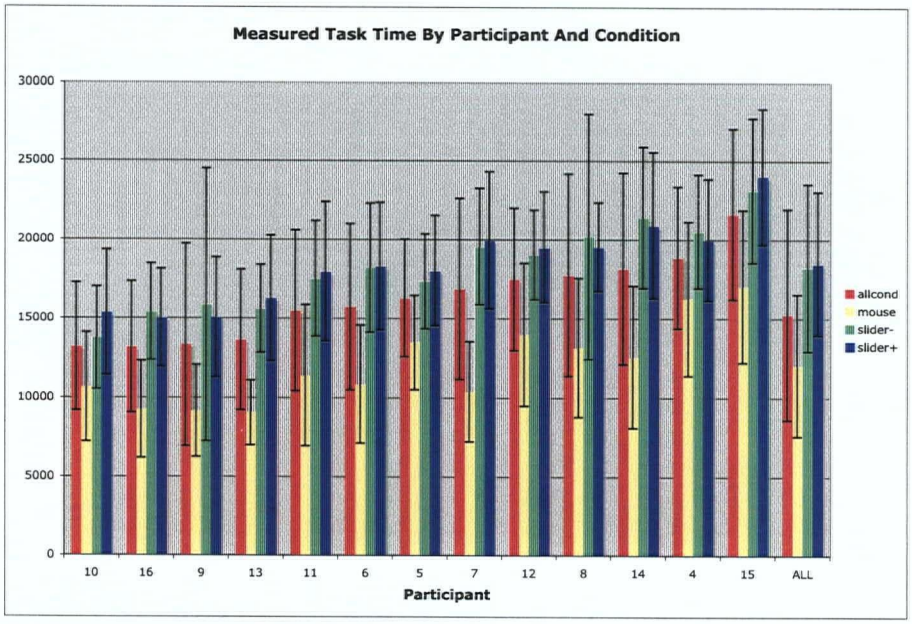
This appendix contains supplemental data charts related to the statistical analysis of the browser user study. The following are included:

- Variance information for the individual participant analysis, including charts and standard deviation tables for both the measured and normalized data. Normalized means are also provided. The means for the measured data were reported in Table 7.3.
- Data tables for task difficulty analysis, including scores used for the normalization of measured task times.

Normalized Task Time vs Condition					
Participant	**N**	**allcond**	**mouse**	**slider-**	**slider+**
10	51	12280	10307	12644	14027
16	50	12316	8788	14144	14124
9	53	12442	8480	14918	14016
13	48	12727	8434	14738	15008
11	46	14653	11130	15969	17268
6	53	14817	10341	17252	16994
5	54	15270	12683	16499	16628
7	49	15722	9817	17858	18795
12	49	16390	13050	18126	18181
8	47	16592	12205	18689	18596
14	46	17270	11912	20302	19975
4	49	17523	15036	19414	18275
15	50	20170	15933	21820	22142
ALL	645	15211	11354	17056	17247



Std.Dev. Norm. Task Time vs Cond.					
Participant	**N**	**allcond**	**mouse**	**slider-**	**slider+**
10	51	2987	3218	1821	2451
16	50	3414	2879	1595	2232
9	53	5329	2336	7056	2533
13	48	3709	1772	1726	2731
11	46	4302	4266	3201	2564
6	53	4505	3527	3147	2855
5	54	3195	2798	2807	2344
7	49	4900	2983	2797	2977
12	49	3827	3209	2190	3121
8	47	5183	3920	5397	2793
14	46	5450	4141	3968	3361
4	49	3609	4184	2721	2041
15	50	4311	3428	3704	2880
ALL	645	4822	4822	4358	3573



Std.Dev. Task Time vs Condition					
Participant	**N**	**allcond**	**mouse**	**slider-**	**slider+**
10	51	4032	3445	3238	3966
16	50	4152	3058	3064	3096
9	53	6388	2907	8625	3799
13	48	4440	2059	2772	3979
11	46	5095	4452	3671	4445
6	53	5265	3712	4109	4063
5	54	3726	2990	3021	3530
7	49	5734	3148	3708	4349
12	49	4533	4516	2867	3528
8	47	6386	4386	7791	2825
14	46	6058	4490	4476	4632
4	49	4504	4872	3596	3850
15	50	5431	4849	4622	4339
ALL	645	6692	4489	5298	4566

Task Difficulty Analysis

Group	Task	all cond	mouse	slider-	slider+	faster	all cond (N)	mouse (N)	slider- (N)	slider+ (N)	DIFFICULTY	NORMSCORE
A	0	14105	7642	14609	14723	-	12	1	3	8	0.923	1.084
A	1	15130	8460	17460	16361	+	13	3	7	3	0.990	1.010
A	2	11290	9213	11765	14169	-	12	6	2	4	0.739	1.354
A	3	16020	5421	15918	17607	-	13	1	5	7	1.048	0.954
A	4	13828	9729	14966	16031	-	11	3	5	3	0.905	1.105
A	5	14290	12400	15160	15069	+	13	4	6	3	0.935	1.070
A	6	13457	10860	19570	14067	+	15	8	3	4	0.880	1.136
A	7	14585	11987	17347	17074	+	10	5	2	3	0.954	1.048
A	8	12873	10269	12430	17764	-	14	7	3	4	0.842	1.188
A	9	13051	6062	13583	14140	-	11	1	7	3	0.854	1.171
A	10	12394	9419	11924	14462	-	11	3	3	5	0.811	1.233
A	11	13425	11441	13747	14591	-	11	3	4	4	0.878	1.139
A	12	13929	9124	17414	14590	+	16	4	4	8	0.911	1.097
A	13	11946	9096	#DIV/0!	15367		11	6	0	5	0.781	1.280
A	14	10371	8593	15921	12077	+	10	6	1	3	0.678	1.474
A	15	12541	7084	15892	15081	+	11	4	5	2	0.820	1.219
A	16	11969	10274	13087	15378	-	11	6	3	2	0.783	1.277
A	17	17094	#DIV/0!	15171	19499	-	9	0	5	4	1.118	0.894
B	0	16751	11788	19046	18657	+	17	5	5	7	1.096	0.913
B	1	16523	15389	16333	20368	-	19	9	7	3	1.081	0.925
B	2	17983	14212	18840	16508	+	19	1	13	5	1.176	0.850
B	3	19974	15766	21093	21317	-	22	5	8	9	1.307	0.765
B	4	17419	15240	17916	19023	-	18	5	9	4	1.139	0.878
B	5	18478	17095	21160	18541	+	13	6	3	4	1.209	0.827
B	6	15074	11025	17238	14395	+	22	4	10	8	0.986	1.014
B	7	14660	11789	17014	16918	+	18	8	4	6	0.959	1.043
B	8	14319	8909	17161	17834	-	16	6	4	6	0.937	1.068
B	9	16058	14538	16979	17064	-	18	7	5	6	1.050	0.952
B	10	17933	14586	21638	18546	+	20	7	5	8	1.173	0.852
B	11	15894	11692	#DIV/0!	21296		16	9	0	7	1.040	0.962
C	0	16145	8927	18562	19743	-	17	5	6	6	1.056	0.947
C	1	16862	12564	19454	19554	-	21	8	6	7	1.103	0.907
C	2	18329	13415	19318	22010	-	20	7	5	8	1.199	0.834
C	3	17694	13788	18568	19956	-	21	5	12	4	1.157	0.864
C	4	20835	16939	20161	24032	-	23	6	8	9	1.363	0.734
C	5	17623	13707	19154	20566	-	22	8	7	7	1.153	0.867
C	6	17879	11938	24994	23064	+	21	11	7	3	1.170	0.855
C	7	20011	14478	20831	23391	-	23	7	6	10	1.309	0.764
C	8	18112	9892	26462	21334	+	24	9	5	10	1.185	0.844
C	9	20869	14363	24941	23303	+	21	7	7	7	1.365	0.733
ALL	ALL	**15287**	12056	18245	18499	-	**645**	216	210	219	1.000	1.000
A	ALL	13479	9753	15171	15472	-	214	71	68	75	0.882	1.134
B	ALL	16790	13475	18495	18354	+	218	72	73	73	1.098	0.911
C	ALL	18508	12897	21010	21846	-	213	73	69	71	1.211	0.826

Appendix C

gif2hapticon Code

```
// GIF2HAPTICON
// Utility for converting (optionally, animated) GIF files to haptic icon XML
// format for the THMB device
// (c) 2006, Joseph Luk


// Compile command: gcc -I$HOME/ImageMagick-6.1.9 -L$HOME/ImageMagick-6.1.9/
     magick/.libs -lMagick gif2hapticon.c -o gif2hapticon
// ensure LD_LIBRARY_PATH=/Volumes/Usr/admin/ImageMagick-6.1.9/ltdl/.libs:/
     Volumes/Usr/admin/ImageMagick-6.1.9/magick/.libs

#include <unistd.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <magick/api.h>

bool outputTactileMovie (FILE *file, Image *image, unsigned char *pixels, int
     numframes);
bool outputHapticIcon (FILE *file, Image *image, unsigned char *pixels, int
     numframes);


int main( int argc, char* argv[] )
{
    // ImageMagick stuff
    ExceptionInfo exception;
    Image *image;
    ImageInfo *image_info;
    unsigned char *pixels;
  int x,y,i;
  FILE *file;
  unsigned long numframes;
  unsigned long framesize;

  enum output_modes { tactilemovie, hapticicon } output_mode;
  int ch;
  char *outfile;
  char defaultoutfile[] = "out.xml";
  output_mode = hapticicon;

  if (argc > 1 && !strcmp (argv[1],"--help"))
  {
    printf ("Usage: gif2hapticon <output mode> <animated GIF filename> <XML
         output filename>\n");
    printf ("\t<output mode> options can be:\n");
    printf ("\t\t -t Output tactile movie\n");
    printf ("\t\t -i (lowercase letter i) Output haptic icon (DEFAULT)\n");
    printf ("\tdefaults to <in.gif> and <out.xml> if no arguments are given.\n");
    printf ("\tNote: all frames must be the same size (no crazy optimized GIFs).\
         n");
    printf ("\tIf you have problems, go to ImageReady animation palette->Optimize
          Animation... and deselect everything.\n");
    exit(1);
  }


  while ((ch = getopt (argc, argv, "ti")) != -1)
```

```
  switch (ch)
  {
    case 't':
     output_mode = tactilemovie;
     break;
    case 'i':
     output_mode = hapticicon;
     break;
    case '?':
     if (isprint (optopt))
       fprintf (stderr, "Unknown option `-%c'.\n", optopt);
     else
       fprintf (stderr,
          "Unknown option character `\\x%x'.\n",
          optopt);
     return 1;
    default:
     abort ();
  }
argc -= optind;
argv += optind;


// Load the image
  InitializeMagick(*argv);
  GetExceptionInfo(&exception);
  image_info=CloneImageInfo((ImageInfo *)NULL);
  if (argc >= 1)
    (void) CopyMagickString(image_info->filename,argv[0],MaxTextExtent);
  else
    (void) CopyMagickString(image_info->filename,"in.gif",MaxTextExtent);

  image=ReadImage(image_info, &exception);
  if (exception.severity != UndefinedException)
      CatchException (&exception);
  if (image == (Image *) NULL)
  {
      fprintf (stderr, "Couldn't open the image file. Please ensure you have an
           image named '%s' in the current directory.\n", image_info->filename)
          ;
      exit(1);
  }

  // Deal with the animated GIF
  numframes = GetImageListLength(image);
  framesize = (image->rows)*(image->columns)*sizeof(unsigned int);

  if (!(pixels = (unsigned char *)malloc((image->rows)*(image->columns)*sizeof(
       unsigned int))))
  {
      fprintf (stderr, "Couldn't allocate image memory %u by %u.\n",image->
          columns,image->rows);
      exit(1);
  }


// Done loading image.
printf ("Loaded image %s (%u x %u) pixels by %u frames.\n", image_info->
     filename, image->columns, image->rows, numframes);


// Open output file
if (argc == 2) outfile = argv[1];
else outfile = defaultoutfile;
file = fopen(outfile,"w");
if (!file)
{
  fprintf (stderr, "Couldn't open output file %s for writing.\n", outfile);
  exit (1);
}


// Dump the XML
switch (output_mode)
{
```

```
    case tactilemovie:
      printf ("Exporting to file \"%s\", tactile movie format...\n", outfile);
      outputTactileMovie (file, image, pixels, numframes);
      printf ("Done.\n");
      break;
    case hapticicon:
      printf ("Exporting to file \"%s\", haptic icon format...\n", outfile);
      outputHapticIcon (file, image, pixels, numframes);
      printf ("Done.\n");
      break;
    default:
      abort();
  }


  fclose (file);
  free (pixels);
  DestroyImageInfo(image_info);
  DestroyExceptionInfo(&exception);
  DestroyMagick();
  return 1;
}


bool outputTactileMovie (FILE *file, Image *image, unsigned char *pixels, int
    numframes)
{
  assert (file);
  assert (image);
  assert (pixels);


  Image *workingimage;
  ExceptionInfo exception;
  float scaledpixelvalue;


  fprintf (file, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
  fprintf (file, "<!DOCTYPE TactileMovie SYSTEM \"TactileMovie-0.2.dtd\">\n");
  fprintf (file, "<TactileMovie Version=\"0.2\" FrameSizeY=\"%u\" FrameSizeX=\"%u
      \" NbFrames=\"%u\">\n",
    image->rows, image->columns, numframes);

  for (int i=0;i<numframes;i++)
  {
    fprintf (file, "  <Frame Index=\"%u\">\n",i);

    workingimage=GetImageFromList(image, i);  // get the ith image

    if (!(ExportImagePixels (workingimage, 0,0, image->columns, image->rows,
                             "I", CharPixel, (void *)pixels, &exception)))
    {
        fprintf (stderr, "Couldn't acquire image pixels for frame %d.\n",i);
        exit(1);
    }

    for (int y=0; y<workingimage->rows; y++)
    {
      fprintf (file, "  <Row r=\"%u\">",y);

      for (int x=0; x<workingimage->columns; x++)
      {
        scaledpixelvalue = (*( pixels + y*workingimage->columns + x )) / 255.0;
        fprintf (file, "%1.3f", scaledpixelvalue);
        if (x<(workingimage->columns-1)) fprintf(file," ");
      }

      fprintf (file, "</Row>\n");
    }

    fprintf (file, "  </Frame>\n");
  }

  fprintf (file, "</TactileMovie>\n");
}
```

```
bool outputHapticIcon (FILE *file, Image *image, unsigned char *pixels, int
    numframes)
{
  assert (file);
  assert (image);
  assert (pixels);

  ExceptionInfo exception;
  float scaledpixelvalue;
  Image *workingimage;


  if (image->columns != 1)
  {
    fprintf (stderr, "Warning: only the first column of pixels will be exported.\
        n");
  }


  fprintf (file, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
  fprintf (file, "<!DOCTYPE HapticIcon>\n\n");
  fprintf (file, "<icon>\n");

  if (numframes == 1)
  {
    // static icon

    if (!(ExportImagePixels (image, 0,0, image->columns, image->rows,
                             "I", CharPixel, (void *)pixels, &exception)))
    {
        fprintf (stderr, "Couldn't acquire image pixels.\n");
        exit(1);
    }

    fprintf (file, "   <type>SpatialIcon</type>\n");
    fprintf (file, "   <height>%d</height>\n",image->rows);

    int x=0;
    for (int y=0; y<image->rows; y++)
    {
      fprintf (file, "   <element>\n");
      fprintf (file, "    <index>%d</index>\n", y);

      // scale pixel value to float between 0 and 1
      scaledpixelvalue = (*( pixels + y*image->columns + x )) / 255.0;
      // scale pixel value to 0 -> (-50V),  1 -> (+50V)
      scaledpixelvalue = scaledpixelvalue * 100 - 50.0;

      fprintf (file, "    <value>%.f</value>\n", scaledpixelvalue);
      fprintf (file, "   </element>\n");

    }
  }
  else
  {
    // animated icon
    fprintf (file, "   <type>TemporalIcon</type>\n");
    fprintf (file, "   <height>%d</height>\n",image->rows);

    for (int i=0;i<numframes;i++)
    {
      fprintf (file, "   <Frame>\n");
      fprintf (file, "    <FrameIdx>%d</FrameIdx>\n",i);

      workingimage=GetImageFromList(image, i);  // get the ith image

      if (!(ExportImagePixels (workingimage, 0,0, image->columns, image->rows,
                               "I", CharPixel, (void *)pixels, &exception)))
      {
          fprintf (stderr, "Couldn't acquire image pixels for frame %d.\n",i);
```

```
            exit(1);
        }


        int x=0;
        for (int y=0; y<image->rows; y++)
        {
          fprintf (file, "     <element>\n");
          fprintf (file, "      <index>%d</index>\n", y);

          // scale pixel value to float between 0 and 1
          scaledpixelvalue = (*( pixels + y*image->columns + x )) / 255.0;
          // scale pixel value to 0 -> (-50V),  1 -> (+50V)
          scaledpixelvalue = scaledpixelvalue * 100 - 50.0;

          fprintf (file, "      <value>%.f</value>\n", scaledpixelvalue);
          fprintf (file, "     </element>\n");
        }


        fprintf (file, "   </Frame>\n");
      }
    }

    fprintf (file, "</icon>\n");

}
```

Appendix D

Browser Prototype Code

D.1 Tactile I/O Loop

The browser prototype application software layer for the tactile I/O loop consists of several modules, which are listed below. Input and output functions are handled by the STReSS Library [29] by Vincent Levesque.

- `DataUpdateThread` – The input/output loop thread.
- `HapticPageMap` – Functions related to the haptic page map and its haptic icons.
- `BrowserShared` – Shared memory data structure for inter-thread communication.
- `BrowserXMLBits` – Functions related to parsing the XML haptic icon and page map files.
- `main` – the main executable.

D.1.1 DataUpdateThread.h

```
#ifndef DATAUPDATE_THREAD_H
#define DATAUPDATE_THREAD_H

#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <math.h>
#include <assert.h>
#include <sys/time.h>

#include "stressd/FirmwareInfoXML.h"
```

```
#include "stressd/DeviceInfoXML.h"
#include "stressd/StreamDevice.h"
#include "stressd/PacketLogger.h"
#include "stressd/init.h"

#include "BrowserShared.h"

/* TODO

Move file I/O routines into yet another thread, running at lower priority...

this thread should:
- Check if there is a new page map
- If so, create a new instance of the page map variable and load it in
- Handle putting the position back to the browser?
*/


class DataUpdateThread
{
public:
  DataUpdateThread(BrowserShared* s,
                   stressd::StreamDevice* d);

  virtual ~DataUpdateThread();
  static void *Start(void *p);

protected:
  void Start_();

private:
    BrowserShared *shared;

    stressd::StreamDevice* dev;

    SliderSmoother smoother;

};

#endif
```

D.1.2 DataUpdateThread.cpp

```
// DataUpdateThread.cpp
// Joseph Luk, July 2006.  Based on code by Vincent Levesque and Shannon Little.

#include "DataUpdateThread.h"
#include "BrowserXMLBits.h"


DataUpdateThread::DataUpdateThread(BrowserShared* s,
                   stressd::StreamDevice* d )
{
    shared = s;
    dev = d;
}

DataUpdateThread::~DataUpdateThread()
{
}

void *DataUpdateThread::Start(void *p)
{
    ((DataUpdateThread *) p)->Start_();
    return 0;
}



void DataUpdateThread::Start_()
{
```

```
  float slider_pos;
  int map_pos = 0;
    float norm_map_pos;
    int last_map_pos = 0;
    int click_count;

  stressd::VoltImg img (dev->NbActuatorsX(), dev->NbActuatorsY());
  stressd::StatusInfo info;

    // slider parameters
    const float slider_min = 1700; //was 1615
    const float slider_max = 2770; //was 2799

  printf("Starting Data update thread...\n");

    while (1) {

      // read slider
        dev->ReadStatus(info); // currently takes exactly 3 ms

// (for calibration)
//        printf ("%d\n",info.sliderData_);

        if(info.sliderData_ < slider_min)
        {
            slider_pos = 0.0;
        }
        else if(info.sliderData_ > slider_max)
        {
            slider_pos = 100.0;
        }
        else
        {
            slider_pos = (info.sliderData_ - slider_min) /
                         (slider_max - slider_min) * 100.0;
        }
        slider_pos = 100.0 - slider_pos;
        slider_pos = smoother.SmoothSliderPos(slider_pos);


        if (shared->GetPageMapReady())
        {
            map_pos = shared->pagemap->CurrentMapPos(slider_pos);

          // Get click count
            click_count = info.clickCount_;

          // Export data to shared memory
           shared->SetClicks(click_count);

    //          if (map_pos != last_map_pos)
    //          {

                 // Export data to shared memory
                 shared->SetMapPos(map_pos);

                 // write to device
    //             printf ("Compute image\n");
                 shared->pagemap->ComputeImage(&img, map_pos);
    //             printf ("Write Image\n");
                 dev->WriteImage(&img);
    //             printf ("Done.\n");


                 last_map_pos = map_pos; // cache last value
    //     }

        } // if (shared->GetPageMapReady())


  } // while(1)
}
```

D.1.3 HapticPageMap.h

```
#ifndef HAPTICPAGEMAP_H
#define HAPTICPAGEMAP_H

class HapticPageMap;  // defined below
class HapticIcon; // defined below


#include "stressd/FirmwareInfoXML.h"
#include "stressd/DeviceInfoXML.h"
#include "stressd/StreamDevice.h"
#include "stressd/PacketLogger.h"
#include "stressd/init.h"

#include "BrowserXMLBits.h"


// Support functions for time stuff

// return the time (in seconds) since timeStamp, with microsecond precision
double TimeSince (const struct timeval *timeStamp, struct timeval *
    currentTimeStamp = NULL);

// returns the difference in the timestamps, in milliseconds (tv1-tv2)
double TimeDiff (const struct timeval *tv1, const struct timeval *tv2);

// stamp the time into a timeval struct
void TimeStamp (struct timeval *timeStamp);



class HapticIconContent
{
    public:

    HapticIcon *iconparent;

    char *type;
    char *name;

    virtual bool RenderIcon (double *buf_ptr) = 0;
        // Write this haptic icon to the buffer at the zero position

    virtual bool AllocateData (int size) = 0;
    virtual double GetData (int index) = 0;
    virtual bool SetData (int index, double value) = 0;
};

class SpatialIconContent : public HapticIconContent
{
    public:
    SpatialIconContent( int height = 0 );
    virtual ~SpatialIconContent();

    double *data;

    bool RenderIcon (double *buf_ptr);

    bool AllocateData (int size);
    double GetData (int index);
    bool SetData (int index, double value);
};

class AnimatedIconContent : public HapticIconContent
{
    public:
    AnimatedIconContent( int height = 0, int frames = 1 );
    virtual ~SpatialIconContent();

    double **data;

    bool RenderIcon (double *buf_ptr);

    bool AllocateData (int size);
```

```
    double GetData (int index);  // index = frame*height
    bool SetData (int index, double value);

    int numFrames;
    int currentFrame;
    struct timeval initTimeStamp;
    int fps;
};


class HapticIcon
{
    public:
    HapticIcon( const char *filename );
    virtual ~HapticIcon();

    int at;  // Y position of the icon
    int height;   // extent of the icon in Y taxels

    HapticIconContent *content;

    HapticIcon *next;

    virtual bool RenderIconToMap (HapticPageMap *map_ptr);
        // Write this haptic icon to the page map "at" its location
};


class HapticPageMap {
    public:
    HapticPageMap(const char *filename);
    virtual ~HapticPageMap();

    int CurrentMapPos (const float slider_pos);
        // update with the slider position and return the current position

    float ComputeDetentPosFromVelocity ();
        // return the new detent position based on the current velocity.


    bool ComputeImage (stressd::VoltImg *outputImg, int map_pos);

    void RenderAllIcons (void);
        // pre-render all icons to the page map, mostly useful for testing
            purposes

    HapticIcon *icons;

    double* voltmap;  // in volts, temporary while we only support spatial icons
    int height;  // size of the map array (in taxel units)


    private:
    // velocity model stuff
    bool simpleControlMode;  // configuration parameter; set to TRUE if you want
        simple position control

    enum VControlStates { pControl, vControlWaiting, vControlActive,
        vControlFinished };
    VControlStates vControlState;
    struct timeval vControlStateTimeStamp;   // time entered current
        vControlState
                                        // exception: always zero when in
                                            pControl


    int currentVelocity;  // UNITS: map units per second; up is negative, down is
         positive
    float lastDetentPos;
    int pOffset;
    struct timeval lastTimeStamp;


    struct timeval startTimeStamp;
```

```
    float lookahead[8];
        #define LOOKAHEAD_NOVALUE -1000
    float lookback[8];


    int CropPos (int inputPos);  // helper function to crop the position to the
        bounds of the map
    float CropPos (float inputPos);
};



class SliderSmoother {
    // Used to reduce noise and jitter from the slider.
    // Caches the last three values of the slider and averages them.
    // If the new slider position is close to the average, don't update.
    // Otherwise, return the new position.

    // currently only set up for float slider positions
    // probably should be templatized in the future.

    public:
    SliderSmoother();
    virtual ~SliderSmoother();

    float SmoothSliderPos (float rawpos);

    private:
    int cache_size;
    float threshold;
    float *slider_cache;
    int startidx;
};

#endif
```

D.1.4 HapticPageMap.cpp

```
// Haptic Page Map support functions
// July 2006, Joseph Luk


#include "HapticPageMap.h"

// #define DEBUG

// #define CVTEST 70
    // constant velocity test flag


// #define CLICKONVCONTROL
    // create "clicking" tactile feedback for velocity control

// #define SUBTAXELDOUBLE
    // use bilinear scaling to reduce image size by 1/2 (slows down tactile flow,
        makes an effectively 16-taxel wide display)
#define SUBTAXELTRIPLE
    // use bilinear scaling to reduce image size by 1/3 (slows down tactile flow,
        makes an effectively 24-taxel wide display)


// Time arithmetic support functions (C-style)

// return the time (in seconds) since timeStamp, with microsecond precision
// also puts the current time in currentTimeStamp, if provided
double TimeSince (const struct timeval *timeStamp, struct timeval *
    currentTimeStamp)
{
```

```
    double timediff;

    struct timeval temp_tv;
    struct timeval *tv = currentTimeStamp ? currentTimeStamp : &temp_tv;

    gettimeofday ( tv, NULL);

    // subtract the current time from the cached time
    double usecdiff;
    timediff = tv->tv_sec - timeStamp->tv_sec;
    if ( 0.0 > (usecdiff = (double)(tv->tv_usec - timeStamp->tv_usec) / 1000000.0
         ) )
    {
        usecdiff = 1.0 + usecdiff;
        timediff--;
    }
    timediff += usecdiff;

    return (timediff);
}

// returns milliseconds tv1-tv2
double TimeDiff (const struct timeval *tv1, const struct timeval *tv2)
{
    double usecdiff;
    double timediff;

    timediff = tv1->tv_sec - tv2->tv_sec;
    if ( 0.0 > (usecdiff = (double)(tv1->tv_usec - tv2->tv_usec) / 1000000.0 ) )
    {
        usecdiff = 1.0 + usecdiff;
        timediff--;
    }
    timediff += usecdiff;

    return (timediff);
}

// stamp the time into a timeval struct
void TimeStamp (struct timeval *timeStamp)
{
    struct timeval tv;
    gettimeofday ( &tv, NULL);
    timeStamp->tv_sec = tv.tv_sec;
    timeStamp->tv_usec = tv.tv_usec;
}


HapticIcon::HapticIcon( const char *filename = NULL )
{
    height = 0;
    at = 0;

    if (filename)
    {
       RawIconParser *parser = new RawIconParser (filename, this);
       delete parser;

       if (content)
       {
           content->name = new char[strlen(filename)];
           strcpy (content->name, filename);
       }
    }
    else content = NULL;
}

HapticIcon::~HapticIcon()
{
    if (content) delete content;
}


bool HapticIcon::RenderIconToMap (HapticPageMap *map_ptr)
{
//    if (!content) return false;  // no icon!
```

```
    assert (content);
    assert (map_ptr);

    if (at > map_ptr->height) return false;

    int bottom = at + height;

    if ( bottom > (map_ptr->height) )
    {
        #ifdef DEBUG
//            cout << "Cropping haptic icon of type " << content->type << " at "
    << at << " height " << height << endl;
        #endif
        // needs cropping
        bool retval;
        int diff = map_ptr->height - at;
        // render into a temp buffer first and then copy to the page map
        double *buf = new double[this->height];
        if (retval = this->content->RenderIcon(buf))
        {
            for (int i=0; i<diff; i++)
            {
                // assert, remove later if necessary
                assert ( at+i < map_ptr->height );

                map_ptr->voltmap[at+i] = buf[i];
            }
        }
        delete [] buf;
        return retval;
    }
    else
        return (this->content->RenderIcon( (map_ptr->voltmap) + at));
}


SpatialIconContent::SpatialIconContent( int height )
{
    type = new char[12];
    strcpy (type, "SpatialIcon");

    if (height) data = new double[height];
    else data = NULL;

    name = NULL;
    iconparent = NULL;
}

SpatialIconContent::~SpatialIconContent()
{
    #ifdef DEBUG
        printf("SpatialIconContent Destructor\n");
    #endif
    delete[] data;
    delete[] type;
    delete[] name;
}

bool SpatialIconContent::RenderIcon (double *buf_ptr)
{
    assert (iconparent);
    for (int i=0; i< iconparent->height; i++)
        buf_ptr[i] = data[i];
    return true;
}

bool SpatialIconContent::AllocateData (int size)
{
    data = new double [size];
    return true;
}

double SpatialIconContent::GetData (int index)
{
    return data[index];
}
```

```
bool SpatialIconContent::SetData (int index, double value)
{
    assert (data);
    data[index] = value;
    return true;
}


AnimatedIconContent::AnimatedIconContent( int height, int frames )
{
    type = new char[12];
    strcpy (type, "AnimatedIcon");

    numFrames = frames;

    if (height) data = new double[height*frames];
    else data = NULL;

    name = NULL;
    iconparent = NULL;
    currentFrame = 0;
    fps = 30;  // default value
    gettimeofday (&initTimeStamp);
}

AnimatedIconContent::~AnimatedIconContent()
{
    #ifdef DEBUG
        printf("AnimatedIconContent Destructor\n");
    #endif
    delete[] data;
    delete[] type;
    delete[] name;
}

bool AnimatedIconContent::RenderIcon (double *buf_ptr)
{
    assert (iconparent);

    currentFrame = (fps / TimeSince(initTimeStamp)) % numFrames;

    for (int i=0; i< iconparent->height; i++)
        buf_ptr[i] = data[i+(height*currentFrame)];
    return true;
}

bool AnimatedIconContent::AllocateData (int size)
{
    data = new double [size];
    return true;
}

double AnimatedIconContent::GetData (int index)
{
    return data[index];
}

bool AnimatedIconContent::SetData (int index, double value)
{
    assert (data);
    data[index] = value;
    return true;
}


HapticPageMap::HapticPageMap (const char *filename = NULL)
{
    icons = NULL;

    if (filename)
    {
       PageMapParser *parser = new PageMapParser (filename, this);
       delete parser;
    }
```

```
    else height=100;    // default volt map height

    // init the array with base value
    voltmap = new double[height];
    for (int i=0; i<height; i++)
        voltmap[i] = -50.0;

    lastDetentPos = 0;
    lastTimeStamp.tv_sec = 0;
    lastTimeStamp.tv_usec = 0;
    currentVelocity = 0;
    vControlState = pControl;
    vControlStateTimeStamp.tv_sec = 0;
    vControlStateTimeStamp.tv_usec = 0;

    for (int i=0; i<8; i++) lookahead[i] = LOOKAHEAD_NOVALUE;
    for (int i=0; i<8; i++) lookback[i] = 0;

    simpleControlMode = false;

    TimeStamp (&startTimeStamp);

}


HapticPageMap::~HapticPageMap ()
{
    delete [] voltmap;

    HapticIcon *icon = icons;
    HapticIcon *nexticon;
    while (icon)
    {
        nexticon = icon->next;
        delete icon;
        icon = nexticon;
    }
}


int HapticPageMap::CurrentMapPos (float slider_pos)
{
    // return the cursor position in the haptic map
    // if using velocity mode, may depend on previous values!

    // slider_pos should be expressed as float from 0.0 to 100.0
    /*

    |--|--|-----------------------------|
       |
     0.0 = slider_pos

    |-----------------------------|--|--|
                                  |
                                 1.0

    |<------------ map size ------------>|
       |<--------- range ---------->|

    |<--->|
      display window

    */

    if (false)    // automated movement program for demo movie
    {
        if (TimeSince(&startTimeStamp) < 0.1) lastDetentPos =        0;
        else if (TimeSince(&startTimeStamp) < 3.0) currentVelocity =   62;
        else if (TimeSince(&startTimeStamp) < 3.5) currentVelocity =   50;
        else if (TimeSince(&startTimeStamp) < 4.0) currentVelocity =    0;
        else if (TimeSince(&startTimeStamp) < 5.6) currentVelocity =  -17;
        else if (TimeSince(&startTimeStamp) < 6.0) currentVelocity =    0;
        else if (TimeSince(&startTimeStamp) < 7.6) currentVelocity =   17;
        else if (TimeSince(&startTimeStamp) < 8.0) currentVelocity =    0;
        else if (TimeSince(&startTimeStamp) < 9.6) currentVelocity =  -17;
        else if (TimeSince(&startTimeStamp) < 10.0 ) currentVelocity = 0;
```

```
        else if (TimeSince(&startTimeStamp) < 11.6 ) currentVelocity = 17;
        else if (TimeSince(&startTimeStamp) < 12.0 ) currentVelocity = 0;
        else if (TimeSince(&startTimeStamp) < 13.0 ) currentVelocity = -50;
        else if (TimeSince(&startTimeStamp) < 14.9 ) currentVelocity = -80;
        else if (TimeSince(&startTimeStamp) < 17 ) TimeStamp(&startTimeStamp);

        ComputeDetentPosFromVelocity();

        return (CropPos((int)round(lastDetentPos)));

    }

    if (simpleControlMode)
    {
        float display_window = 8.0 / (float)height;
            // display_window is the percentage of the page "visible" through the
            // tactile display.

        int range = height - 16;
            // range is the portion of the map where the centre of the slider
            // may be placed

        int window_start = (int)((100.0-slider_pos)/100.0 * (float)range);
            // start index in the haptic page map

        lastDetentPos = (float)window_start;

        return (window_start);
    }
    else      // velocity control
    {
        float normSliderPos = ( 1.0 - (slider_pos / 50.0) );
            // normalized to between -1.0 and +1.0
        float currentAcceleration;

        // A few convenient macros
        // #define SIMPLEVELOCITY
         #define VELOCITYATSTOPS

        // constants
        #if defined VELOCITYATSTOPS
            int maxP = 50;
            float maxV = 200.0;
            float posControlZone = 0.75;  // Use this many percent of slider area
                 for pos control zone
            float timeBeforeVControl = 0.05;  // Number of seconds before
                velocity control activates
            float vControlAcceleration = 1;  // Number of seconds before velocity
                 reaches maxV after starting V-control
            float timeAfterVControl = 0.05;  // Number of seconds after
                disengaging velocity control, during which time user input is
                ignored so that they can return the slider to centre
        #elif defined SIMPLEVELOCITY
            int maxP = 10;
            float maxV = 300.0;
            float posControlZone = 0.20;  // Use this many percent of slider area
                 for pos control zone
            float timeBeforeVControl = 0.0;  // Number of seconds before velocity
                 control activates
            float vControlAcceleration = 0.2;  // Number of seconds before
                velocity reaches maxV after starting V-control
            float timeAfterVControl = 0.01;  // Number of seconds after
                disengaging velocity control, during which time user input is
                ignored so that they can return the slider to centre
        #else
            int maxP = 50;
            float maxV = 100.0;
            float posControlZone = 0.95;  // Use this many percent of slider area
                 for pos control zone
            float timeBeforeVControl = 0.01;  // Number of seconds before
                velocity control activates
            float vControlAcceleration = 1;  // Number of seconds before velocity
                 reaches maxV after starting V-control
            float timeAfterVControl = 0.01;  // Number of seconds after
                disengaging velocity control, during which time user input is
                ignored so that they can return the slider to centre
```

```
#endif


if (fabs(normSliderPos) < posControlZone)
{
    // Position Control Zone
    currentVelocity = 0;

    switch (vControlState)
    {
        case vControlWaiting:
            // abort waiting for vControl, return to pControl
            vControlState = pControl;
            // continue to pControl
        case pControl:
            pOffset = (int)(round(normSliderPos / posControlZone * (float
                )maxP));
            break;
        case vControlActive:
            // enter vControlFinished; start timer
            vControlState = vControlFinished;
            TimeStamp (&vControlStateTimeStamp);
            break;
        case vControlFinished:
            // check timer; if passed, then enter pControl
            if (TimeSince(&vControlStateTimeStamp) > timeAfterVControl)
            {
                // re-zero detent position to avoid moving the highlight
                int newPOffset = (int)(round(normSliderPos /
                    posControlZone * (float)maxP));
                lastDetentPos += pOffset - newPOffset;
                pOffset = newPOffset;

                vControlState = pControl;
            }
            break;
    } // switch (vControlState)
}
else {
    // Velocity Control Zone

    switch (vControlState)
    {
        case pControl:
        case vControlFinished:
            // enter waiting state, start timer
            vControlState = vControlWaiting;
            TimeStamp (&vControlStateTimeStamp);
            pOffset = (int)(round(normSliderPos / posControlZone * (float
                )maxP));
            break;
        case vControlWaiting:
            // check to see if the timer has exceeded the
                timeBeforeVControl, if so, go to next state
            pOffset = (int)(round(normSliderPos / posControlZone * (float
                )maxP));
            if (TimeSince(&vControlStateTimeStamp) > timeBeforeVControl)
            {
                TimeStamp (&vControlStateTimeStamp);
                vControlState = vControlActive;
            }
            break;
        case vControlActive:
            if (normSliderPos > 0)
            {
                pOffset = maxP;
                #ifdef CVTEST
                currentVelocity = CVTEST;
                #else
                currentVelocity = (int)(round((normSliderPos -
                    posControlZone) /
                                (1.0 - posControlZone) *
                                maxV));
                #endif

            }
```

```
                    else
                    {
                        pOffset = -maxP;
                        #ifdef CVTEST
                        currentVelocity = -CVTEST;
                        #else
                        currentVelocity = (int)(round((normSliderPos +
                            posControlZone) /
                                (1.0 - posControlZone) *
                                maxV));
                        #endif
                    }

                    // add acceleration factor
                    currentAccceleration = (TimeSince(&vControlStateTimeStamp)) /
                        (vControlAcceleration);
                    if (currentAcceleration < 1.0)
                        currentVelocity = (int)(round)((float)currentVelocity *
                            currentAccceleration);

                    break;
                } // switch (vControlState)

            } // velocity control zone

            if (pOffset > maxP) pOffset = maxP;
            else if (pOffset < -maxP) pOffset = -maxP;

            ComputeDetentPosFromVelocity();

            #ifdef DEBUG
                printf ("CurrentMapPos: velocityControl pOffset=%d, currentVelocity=%
                    d, detentPos=%f, mapPos=%d\n", pOffset, currentVelocity,
                    lastDetentPos, CropPos((int)round(lastDetentPos)+pOffset));
            #endif

            #ifdef CVTEST
            if (lastDetentPos >= height-8) lastDetentPos = 0.0;
            #endif

            return (CropPos((int)round(lastDetentPos) + pOffset));
        }

}


float HapticPageMap::ComputeDetentPosFromVelocity ()
{
    // return the cursor position in the haptic map

    if (simpleControlMode) return lastDetentPos;
    else
    {
        // compute the current position based on the last heading
        float newDetentPos;
        double timediff;


        struct timeval tv;
        gettimeofday ( &tv, NULL);

        // subtract the current time from the cached time
        double usecdiff;
        timediff = tv.tv_sec - lastTimeStamp.tv_sec;
        if ( 0.0 > (usecdiff = (double)(tv.tv_usec - lastTimeStamp.tv_usec) /
            1000000.0 ) )
        {
            usecdiff = 1.0 + usecdiff;
            timediff--;
        }
        timediff += usecdiff;
        lastTimeStamp.tv_sec = tv.tv_sec;
        lastTimeStamp.tv_usec = tv.tv_usec;


//      if timediff is specified
```

```
//          {
              // UNTESTED!!!
//              lastTimeStamp.tv_sec += (time_t)trunc(timediff);
//              lastTimeStamp.tv_usec += (suseconds_t)( (modf(timediff,NULL)) *
     1000000 );
//              if (lastTimeStamp.tv_usec >= 1000000)
//              {
//                  lastTimeStamp.tv_usec -= 1000000;
//                  lastTimeStamp.tv_sec++;
//              }
//          }
//


        newDetentPos = lastDetentPos + timediff * (float)currentVelocity;
        newDetentPos = CropPos (newDetentPos);

        lastDetentPos = newDetentPos;
        return newDetentPos;
    }
}

int HapticPageMap::CropPos (int inputPos)
{
    if (inputPos < 0) return(0);
    if (inputPos > height-8) return(height-8);
    return inputPos;
}

float HapticPageMap::CropPos (float inputPos)
{
    if (inputPos < 0.0) return(0.0);
    #if defined SUBTAXELTRIPLE
        if (inputPos > (float)(height-8)) return((float)(height-24));
    #elif defined SUBTAXELDOUBLE
        if (inputPos > (float)(height-8)) return((float)(height-16));
    #else
        if (inputPos > (float)(height-8)) return((float)(height-8));
    #endif

    return inputPos;
}



bool HapticPageMap::ComputeImage (stressd::VoltImg *outputImg, int map_pos)
{
    // compute the tactile output image at cursor position map_pos
    // and place the result in outputImg

    // returns true if everything was dandy; false otherwise

    // uncomment one block of code below
        // crop to map bounds
        map_pos = CropPos (map_pos);

        // or assert map bounds
        // assert (map_pos >= 0 && map_pos < height-8);

        // or return false if outside of map bounds
        // if (map_pos < 0 || map_pos > height-8) return false;

    double timediff;
    float valueAt;
    register int index;

    HapticIcon *icon;

    icon = icons;
    while (icon)
    {
        // render only the area that's "visible"
        #if defined SUBTAXELTRIPLE
            const int visiblearea = 24;
        #elif defined SUBTAXELDOUBLE
            const int visiblearea = 16;
```

```
        #else
            const int visiblearea = 8;
        #endif


        if ( (icon->at <= map_pos+visiblearea) &&
             (icon->at+icon->height >= map_pos) )
        {
            icon->RenderIconToMap(this);
        }
            icon = icon->next;
    }

    // copy data from the page map into the outputImg
    for (int x=0; x<8; x++)
    {
        #if defined SUBTAXELTRIPLE
            index = x*3 + map_pos;
            valueAt = (voltmap[index] + voltmap[index+1] + voltmap[index+2]) /
                3.0;
        #elif defined SUBTAXELDOUBLE
            index = x*2 + map_pos;
            valueAt = (voltmap[index] + voltmap[index+1]) / 2;
        #else
            valueAt = voltmap[ x + map_pos ];
        #endif


        // Do the lookahead and lookback smoothing
        if (lookahead[x] != LOOKAHEAD_NOVALUE) valueAt = (valueAt + lookahead[x])
            / 2.0;

        if (fabs(valueAt - lookback[x]) > 50.0)  // threshold value
        {
            lookahead[x] = valueAt;
            valueAt = (valueAt + lookback[x]) / 2.0;
        }
        else lookahead[x] = LOOKAHEAD_NOVALUE;

        lookback[x] = valueAt;


        // optionally, provide tactile feedback of transition between velocity
            control and position control modes
        #ifdef CLICKONVCONTROL
            if (vControlState == vControlWaiting)
            {
                timediff = TimeSince(&vControlStateTimeStamp);

                if (timediff < 0.005) valueAt += 40;
                else if (timediff < 0.01) valueAt -= 40;

                if (valueAt < -50) valueAt = -50;
                else if (valueAt > 50) valueAt = 50;
            }
       #endif

        outputImg->Set(7-x, 0, valueAt);

    }



    return true;
}

void HapticPageMap::RenderAllIcons (void)
{
    HapticIcon *icon;

    icon = icons;
    while (icon)
    {
        icon->RenderIconToMap(this);
        icon = icon->next;
    }
```

```
}


SliderSmoother::SliderSmoother()
{
    cache_size = 10;
    threshold = 5.0;
        // when slider is moved a large distance (above this threshold),
        // abandon averaging and just jump to the new slider position

    slider_cache = new float [cache_size];
    for (int i=0; i<cache_size; i++)
        slider_cache[i] = 0.0;
    startidx=0;

}

SliderSmoother::~SliderSmoother()
{
    delete [] slider_cache;
}


float SliderSmoother::SmoothSliderPos (float rawpos)
{
    // Takes the current slider pos and returns the "smoothed" value of the
        slider pos
    // Uses simple averaging of the last (cache_size-1) pos's, and the current
        rawpos

    float avgpos = 0.0;
    float retval;
    int i;

    // add the value to the array, clobbering the oldest value
    startidx++;
    startidx %= cache_size;
    slider_cache[startidx] = rawpos;

    for (i=0; i<cache_size; i++)
        avgpos += slider_cache[i];
    avgpos /= cache_size;

    if ( fabs(rawpos - avgpos) >= threshold )
    {
        retval = rawpos;
        for (i=0; i<cache_size; i++)
            slider_cache[i] = rawpos;
    }
    else
        retval = avgpos;

    return (retval);
}
```

D.1.5 BrowserShared.h

```
#ifndef BROWSERSHARED_H
#define BROWSERSHARED_H

#include "acc/Mutex.h"

#include "HapticPageMap.h"
#include "BrowserXMLBits.h"


class BrowserShared {

    public:

    BrowserShared();
```

```
    int GetMapPos();
    void SetMapPos(int pos);

    int GetClicks();
    void SetClicks(int click_count_param);

    bool GetPageMapReady();
    void SetPageMapReady (bool ready_flag);

    // PUBLIC haptic page map structure
    // Rules: before reading, check if GetPageMapReady() is true or not
    // before writing, SetPageMapReady(false)
    HapticPageMap *pagemap;


    private:
    int click_count;
    int map_pos;

    bool pagemap_ready;

    ACE_Mutex mutex;
};

#endif
```

D.1.6 BrowserShared.cpp

```
// BrowserShared.cpp

#include "BrowserShared.h"

BrowserShared::BrowserShared ()
{
    click_count = 0;
    map_pos = 0;
    pagemap_ready = 0;
    pagemap = NULL;
}


int BrowserShared::GetMapPos ()
{
    mutex.acquire();
    int retval = map_pos;
    mutex.release();
    return retval;
}

void BrowserShared::SetMapPos (int pos)
{
    mutex.acquire();
    map_pos = pos;
    mutex.release();
    return;
}

int BrowserShared::GetClicks ()
{
    mutex.acquire();
    int retval = click_count;
    mutex.release();
    return retval;

}

void BrowserShared::SetClicks (int click_count_param)
{
    mutex.acquire();
    click_count = click_count_param;
```

```
    mutex.release();
    return;
}


bool BrowserShared::GetPageMapReady()
{
    mutex.acquire();
    int retval = pagemap_ready;
    mutex.release();
    return retval;
}

void BrowserShared::SetPageMapReady (bool ready_flag)
{
    mutex.acquire();
    pagemap_ready = ready_flag;
    mutex.release();
    return;
}
```

D.1.7 BrowserXMLBits.h

```
// Browser XML Bits, modified from original file by Vincent Levesque (GPL)
// Joseph Luk, July 2006

#ifndef BROWSERXMLBITS_H
#define BROWSERXMLBITS_H

#include <string>
#include <vector>
#include "ACEXML/common/DefaultHandler.h"


class PageMapParser : public ACEXML_DefaultHandler
{
public:
   PageMapParser(std::string file, HapticPageMap* data);
   virtual ~PageMapParser();

   void ParseFile(std::string fileName);

   virtual void characters (const ACEXML_Char *ch, int start, int length
       ACEXML_ENV_ARG_DECL)
    ACE_THROW_SPEC((ACEXML_SAXException));

   virtual void endDocument (ACEXML_ENV_SINGLE_ARG_DECL)
    ACE_THROW_SPEC((ACEXML_SAXException));

   virtual void endElement (const ACEXML_Char *namespaceURI, const ACEXML_Char *
       localName, const ACEXML_Char *qName ACEXML_ENV_ARG_DECL)
    ACE_THROW_SPEC((ACEXML_SAXException));
   virtual void setDocumentLocator (ACEXML_Locator *locator)
    ACE_THROW_SPEC((ACEXML_SAXException));

   virtual void startDocument (ACEXML_ENV_SINGLE_ARG_DECL)
    ACE_THROW_SPEC((ACEXML_SAXException));

   virtual void startElement (const ACEXML_Char *namespaceURI, const ACEXML_Char
       *localName, const ACEXML_Char *qName, ACEXML_Attributes *atts
       ACEXML_ENV_ARG_DECL)
    ACE_THROW_SPEC((ACEXML_SAXException));

private:
      ACEXML_Locator* locator_;
      std::vector<std::string> tags_;
    HapticPageMap *data;
    int currAt;
};
```

```
class RawIconParser : public ACEXML_DefaultHandler
{
public:


    RawIconParser(std::string file, HapticIcon* iconptr);
    virtual ~RawIconParser();

    void ParseFile(std::string fileName);

    virtual void characters (const ACEXML_Char *ch, int start, int length
        ACEXML_ENV_ARG_DECL)
     ACE_THROW_SPEC((ACEXML_SAXException));

    virtual void endDocument (ACEXML_ENV_SINGLE_ARG_DECL)
     ACE_THROW_SPEC((ACEXML_SAXException));

    virtual void endElement (const ACEXML_Char *namespaceURI, const ACEXML_Char *
        localName, const ACEXML_Char *qName ACEXML_ENV_ARG_DECL)
     ACE_THROW_SPEC((ACEXML_SAXException));
    virtual void setDocumentLocator (ACEXML_Locator *locator)
     ACE_THROW_SPEC((ACEXML_SAXException));

    virtual void startDocument (ACEXML_ENV_SINGLE_ARG_DECL)
     ACE_THROW_SPEC((ACEXML_SAXException));

    virtual void startElement (const ACEXML_Char *namespaceURI, const ACEXML_Char
        *localName, const ACEXML_Char *qName, ACEXML_Attributes *atts
        ACEXML_ENV_ARG_DECL)
     ACE_THROW_SPEC((ACEXML_SAXException));

private:
      ACEXML_Locator* locator_;
      std::vector<std::string> tags_;
    HapticIcon* icon;
    int currIndex;
    int currFrame;
};


#endif //BROWSERXMLBITS_H
```

D.1.8 BrowserXMLBits.cpp

```
// Browser XML Bits, modified from original file by Vincent Levesque (GPL)
// Joseph Luk, July 2006

#include <iostream>
#include <sstream>
#include <assert.h>

#include "ace/ACE.h"
#include "ace/Log_Msg.h"

#include "ACEXML/common/FileCharStream.h"
#include "ACEXML/common/HttpCharStream.h"
#include "ACEXML/common/StrCharStream.h"
#include "ACEXML/parser/parser/Parser.h"
#include "ace/Get_Opt.h"
#include "ace/Auto_Ptr.h"

#include "HapticPageMap.h"
#include "BrowserXMLBits.h"



int average = -1;

void PageMapParser::ParseFile(std::string fileName)
{
    ACEXML_FileCharStream *fstm = new ACEXML_FileCharStream;
```

```
    if (fstm->open (fileName.c_str()) != 0)
    {
        std::string str("Failed to open XML file: ");
        str += fileName;
        //STRESSD_THROW_EX_DESC(FileEx,"PageMapParser::ParseFile", str);
    }

    // TODO: throw the correct exception
    ACEXML_TRY_NEW_ENV
    {
        ACEXML_Parser parser;
        ACEXML_InputSource input(fstm);
        parser.setContentHandler (this);
        parser.setFeature("http://xml.org/sax/features/validation", false); // todo
        parser.parse (&input ACEXML_ENV_ARG_PARAMETER);
        ACEXML_TRY_CHECK;
    }
    ACEXML_CATCH (ACEXML_SAXException, ex)
    {
        ACE_UNUSED_ARG(ex);
        ACE_DEBUG ((LM_ERROR, ACE_TEXT ("Exception occurred. Exiting...\n")));
        ex.print();
    }
    ACEXML_ENDTRY;

}

void
PageMapParser::characters (const ACEXML_Char *cdata,
                           int start,
                           int end ACEXML_ENV_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
    int ltag = tags_.size() - 1;
  //printf("ltag: %d\n", ltag);

    if (tags_.back() == "at"){
    if (tags_.at(ltag-1) == "icon")
        {
        currAt = atoi(cdata);
            #ifdef DEBUG
            printf ("Tag <at> value %d\n", currAt);
            #endif
        }
    }
  else if (tags_.back() == "name"){
        if (tags_.at(ltag-1) == "icon")
        {
            #ifdef DEBUG
            printf ("Tag <name> value %s,  currAt = %d\n", cdata, currAt);
            #endif

            HapticIcon *temp = data->icons;
            char *pathname = new char [strlen("icons/") + strlen (cdata)];
            strcpy (pathname, "icons/");
            strcat (pathname, cdata);

            data->icons = new HapticIcon(pathname);
            delete[] pathname;

            data->icons->next = temp;
            data->icons->at = currAt;

        currAt = 0;
        }
    }
    else if (tags_.back() == "height")
    {
        int height = atoi(cdata);

        #ifdef DEBUG
        printf ("Tag <height> value %d\n", height);
        #endif

     data->height = height;
    }
```

```
    else
    {
        #ifdef DEBUG
        std::string thetag = tags_.back();
        printf ("Unknown tag %s\n", thetag.c_str());
        #endif
    }
}

void
PageMapParser::endDocument (ACEXML_ENV_SINGLE_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
}

void
PageMapParser::endElement (const ACEXML_Char *uri,
                           const ACEXML_Char *name,
                           const ACEXML_Char *qName
                           ACEXML_ENV_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
   tags_.pop_back();
}

void
PageMapParser::setDocumentLocator (ACEXML_Locator * locator)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
  this->locator_ = locator;
}


void
PageMapParser::startDocument (ACEXML_ENV_SINGLE_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
}

void
PageMapParser::startElement (const ACEXML_Char *uri,
                             const ACEXML_Char *name,
                             const ACEXML_Char *qName,
                             ACEXML_Attributes *alist
                             ACEXML_ENV_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
   tags_.push_back(name);

}

PageMapParser::PageMapParser(std::string file, HapticPageMap* data)
{
  this->data = data;
  currAt = -1;
  #ifdef DEBUG
  printf ("PageMapParser about to call parsefile\n");
  #endif
    ParseFile(file);
}

PageMapParser::~PageMapParser()
{
}




/* ************************************************************************** */

void RawIconParser::ParseFile(std::string fileName)
{
```

```
    ACEXML_FileCharStream *fstm = new ACEXML_FileCharStream;

    if (fstm->open (fileName.c_str()) != 0)
    {
        std::string str("Failed to open XML file: ");
        str += fileName;
        //STRESSD_THROW_EX_DESC(FileEx,"RawIconParser::ParseFile", str);
    }

    // TODO: throw the correct exception
    ACEXML_TRY_NEW_ENV
    {
        ACEXML_Parser parser;
        ACEXML_InputSource input(fstm);
        parser.setContentHandler (this);
        parser.setFeature("http://xml.org/sax/features/validation", false); // todo
        parser.parse (&input ACEXML_ENV_ARG_PARAMETER);
        ACEXML_TRY_CHECK;
    }
    ACEXML_CATCH (ACEXML_SAXException, ex)
    {
        ACE_UNUSED_ARG(ex);
        ACE_DEBUG ((LM_ERROR, ACE_TEXT ("Exception occurred. Exiting...\n")));
        ex.print();
    }
    ACEXML_ENDTRY;

}

void
RawIconParser::characters (const ACEXML_Char *cdata,
                           int start,
                           int end ACEXML_ENV_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
   int ltag = tags_.size() - 1;
  //printf("ltag: %d\n", ltag);

   if (tags_.back() == "index"){
//    if (tags_.at(ltag-1) == "element")
      {
      currIndex = atoi(cdata);
      assert (icon->height);
          assert (currIndex < icon->height);
      }
   }
  else if (tags_.back() == "value"){
//     if (tags_.at(ltag-1) == "element")
      {
          assert (icon->content);

          if (!strcmp(icon->content->type, "SpatialIcon"))
          {
              #ifdef DEBUG
              printf ("Tag <value> = %f, currIndex=%d\n",atof(cdata), currIndex);
              #endif

              icon->content->SetData(currIndex, atof(cdata));
          }
          if (!strcmp(icon->content->type, "AnimatedIcon"))
          {
              #ifdef DEBUG
              printf ("Tag <value> = %f, currIndex=%d\n",atof(cdata), currIndex);
              #endif
              if (tags_.at(ltag-1) == "frame")
              {
                  icon->content->SetData(currIndex*currFrame, atof(cdata));
              }
              else {
                  #ifdef DEBUG
                  printf("ERROR: value given outside of frame context in
                       AnimatedIcon.\n");
                  #endif
              }
          }
          else {
```

```
            #ifdef DEBUG
            printf("Unknown icon type for element/value %s\n", icon->content->type)
                ;
            #endif
            }
        }
    }
    else if (tags_.back() == "frames")
    {
        if (!strcmp(icon->content->type, "AnimatedIcon"))
        {
            icon->content->numFrames = atoi(cdata);
        }
    }
    else if (tags_.back() == "fps")
    {
        if (!strcmp(icon->content->type, "AnimatedIcon"))
        {
            icon->content->fps = atoi(cdata);
        }
    }
    else if (tags_.back() == "frame")
    {
        if (!strcmp(icon->content->type, "AnimatedIcon"))
        {
            currentFrame = atoi(cdata);
        }
    }
    else if (tags_.back() == "type")
    {
        if (!strcmp(cdata, "SpatialIcon"))
        {
            icon->content  = new SpatialIconContent();
            icon->content->iconparent = icon;
        }
        else if (!strcmp(cdata, "AnimatedIcon"))
        {
            icon->content  = new AnimatedIconContent();
            icon->content->iconparent = icon;
        }
        else
        {
            #ifdef DEBUG
            printf("Unknown icon type %s\n", cdata);
            #endif
        }
    }
    else if (tags_.back() == "height")
    {
        assert (icon->content);

        icon->height = atoi(cdata);

        if (!strcmp(icon->content->type, "SpatialIcon"))
        {
            #ifdef DEBUG
              printf ("Allocating data for height=%d\n",icon->height);
            #endif
            icon->content->AllocateData (icon->height);
        }
        else if (!strcmp(icon->content->type, "AnimatedIcon"))
        {
            #ifdef DEBUG
              printf ("Allocating data for height=%d\n",icon->height);
            #endif
            icon->content->AllocateData (icon->height * icon->content->
                numFrames);
        }
        else {
        #ifdef DEBUG
        printf("Unknown icon type for element/value %s\n", icon->content->type)
            ;
        #endif
        }
    }
}
```

```
void
RawIconParser::endDocument (ACEXML_ENV_SINGLE_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
}

void
RawIconParser::endElement (const ACEXML_Char *uri,
                           const ACEXML_Char *name,
                           const ACEXML_Char *qName
                           ACEXML_ENV_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
   tags_.pop_back();
}

void
RawIconParser::setDocumentLocator (ACEXML_Locator * locator)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
  this->locator_ = locator;
}


void
RawIconParser::startDocument (ACEXML_ENV_SINGLE_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
}

void
RawIconParser::startElement (const ACEXML_Char *uri,
                             const ACEXML_Char *name,
                             const ACEXML_Char *qName,
                             ACEXML_Attributes *alist
                             ACEXML_ENV_ARG_DECL_NOT_USED)
  ACE_THROW_SPEC((ACEXML_SAXException))
{
   tags_.push_back(name);

}

RawIconParser::RawIconParser(std::string file, HapticIcon* iconptr)
{
  this->icon = iconptr;
  currIndex = -1;
    ParseFile(file);
}

RawIconParser::~RawIconParser()
{
}
```

D.1.9 main.cpp

```
// Browser V2.0
// Rewrite to reduce overhead - single thread only
// Joseph Luk, July 2006
// based on code by Vincent Levesque and Shannon Little


#include "stressd/FirmwareInfoXML.h"
#include "stressd/DeviceInfoXML.h"
#include "stressd/StreamDevice.h"
#include "stressd/PacketLogger.h"
#include "stressd/init.h"
#include <stdio.h>
#include <iostream>
#include <signal.h>
#include <sys/stat.h>
#include "ace/Thread.h"
```

```
#include "DataUpdateThread.h"


#define FIRMWARE_FILE "default.sdf"
#define HARDWARE_FILE "default.sdh"

// These must be kept synchronized with MyComponent.cpp!!
#define WORKING_DIR "/home/luk/browser-working-dir"
#define PAGEMAP_FILENAME "pagemap.xml"
#define PAGEMAP_FLAGFILENAME "newpagemap"



int termflag;

void sigint_handler (int signum)
{
    termflag = 1;
}


// I/O routines to communicate with browser
// currently uses file I/O, should probably be improved later
bool WriteSliderPosition (int map_pos, HapticPageMap *pageMap);
bool WriteClicks (int click_count);
bool WasPageReloaded (void);

bool fileExists (char * fileName);


int main(int argc, char* argv[])
{
    struct BrowserShared sharedData;
    int click_count, last_clicks;
    int map_pos;
    float norm_map_pos;

    termflag = 0;
    signal (SIGINT, &sigint_handler);

  try
  {


    stressd::init();

    std::cout << "Loading firmware definition from " << FIRMWARE_FILE << "...\n";
    stressd::FirmwareInfoXML fwInfo(FIRMWARE_FILE);

    std::cout << "Loading hardware definition from " << HARDWARE_FILE << "...\n";
    stressd::DeviceInfoXML devInfo(HARDWARE_FILE);

    std::cout << "Creating streaming STReSS device with buffer size of 1...\n";
    stressd::StreamDevice dev(devInfo, fwInfo, devInfo, 1);

    std::cout << "Connecting and programming device (this may take a few seconds)
        ...\n";
    dev.Connect();


        WasPageReloaded();  // delete page update file, if it exists


        std::cout << "Starting high-priority Data UPDATE thread...\n";
        DataUpdateThread dataUpdThread(&sharedData, &dev);
        ACE_hthread_t hThread;
        ACE_thread_t t_id;
        ACE_Thread::spawn (
                DataUpdateThread::Start,
                &dataUpdThread,
                THR_NEW_LWP | THR_JOINABLE,
                &t_id,
                &hThread,
                ACE_Sched_Params::priority_max(ACE_SCHED_OTHER)
```

```
            );


        do  // this loop runs once per new page map
        {
            sharedData.SetPageMapReady(false);  // stop thread from reading from
                page map

            char pagemap_path[100];
            sprintf (pagemap_path, "%s/%s", WORKING_DIR, PAGEMAP_FILENAME);
            printf ("Loading page map file %s.\n", pagemap_path);
            HapticPageMap pageMap(pagemap_path);

            printf ("Done loading page map. Rendering icons...\n");
            pageMap.RenderAllIcons();
            printf ("Done rendering icons.\n");

            printf ("\nStatic pagemap dump:\n");
            for (int pos=0; pos<pageMap.height; pos++)
            {
                printf ("%2.0f,",pageMap.voltmap[pos]);
            }
            printf ("\n\n");


            sharedData.pagemap = &pageMap;
            sharedData.SetPageMapReady(true);  // start thread reading from page
                map
            usleep(500);


            last_clicks = sharedData.GetClicks();
            // this should probably be moved into a low-priority thread...

            printf ("Type CTRL-C to quit.\n");

            while (termflag == 0)
            {
                usleep (100);

                map_pos = sharedData.GetMapPos();
                click_count = sharedData.GetClicks();

                #ifdef DEBUG
                    printf ("Current position at %d\n", map_pos);
                #endif

                WriteSliderPosition(map_pos, &pageMap);


                // WriteClicks(click_count);

                if (WasPageReloaded()) break;

            } // while (termflag == 0)


            // clean up page map stuff here, if necessary

    } while (termflag == 0);


        printf ("Killing tactile loop thread.\n");
        ACE_Thread::cancel(hThread);

        std::cout << "Disconnecting device...\n";
    dev.Disconnect();
}
catch(stressd::Exception& ex)
{
    std::cout << "libstressd threw an exception:\n";
    std::cout << ex.ErrStr() << "\n";
}
stressd::fini();

return 0;
```

```
}




bool WriteSliderPosition (int map_pos, HapticPageMap *pageMap)
{
    //delete all existing .pos files
    DIR* pdir = opendir(WORKING_DIR);
    struct dirent *pent;

    if (!pdir){
      printf ("opendir() failure; terminating");
      exit(1);
     }

    std::string icon_type = "";
    while ((pent=readdir(pdir))){
        std::string file = pent->d_name;
        int length = file.length();
         //filter out .pos (pos) files only
         if(file.rfind(".pos", length) == length-4){
            //delete file
            char type_str[50];
            strcpy (type_str,WORKING_DIR);
            strcat (type_str,"/");
            strcat (type_str,file.c_str());
            remove(type_str);
        }
     }
     closedir (pdir);

    //write new pos file
    char type_str[50];
    strcpy (type_str,WORKING_DIR);
    strcat (type_str,"/");
    char slider_str[10];


//     float norm_map_pos = (float)map_pos / (float)pageMap->height * 100.0;
//     sprintf(slider_str, "%f", norm_map_pos);

    sprintf (slider_str, "%d", map_pos);

    strcat (type_str,slider_str);
    strcat (type_str,".pos");

    std::ofstream outputFile;
    outputFile.open (type_str, std::ofstream::out);
    outputFile.close();
    return true;
}



bool WriteClicks (int new_clicks)
{
    static int last_clicks = 0;

    printf ("last_clicks = %d, new_clicks = %d\n", last_clicks, new_clicks);

    int click_diff = new_clicks - last_clicks;
    if (click_diff > 0){
        for (int c = 0; c < click_diff; c++){
            //write new click file
            char type_str[50];
            strcpy (type_str,WORKING_DIR);
            strcat (type_str,"/");
            char slider_str[10];
            sprintf(slider_str, "%d", last_clicks + c);
```

```
            strcat (type_str,slider_str);
            strcat (type_str,".click");
            std::ofstream outputFile;
            outputFile.open (type_str, std::ofstream::out);
            outputFile.close();
        }
    }
    last_clicks = new_clicks;
}


// Check the pagemap_flagfile to see if it exists; if it does, delete it.
bool WasPageReloaded (void)
{
//      DIR* pdir = opendir(WORKING_DIR);
//      struct dirent *pent;
//
//   if (!pdir){
//     printf ("opendir() failure; terminating");
//     exit(1);
//    }
//
//   bool found = false;
//   while ((pent=readdir(pdir)) && found == false){
//      std::string file = pent->d_name;
//      int length = file.length();
//      //filter out only the PAGEMAP_FLAGFILENAME
//      if(file.rfind(PAGEMAP_FLAGFILENAME, length) != std::string::npos){
//     found = true;
//     //delete file
//     char pathname[100];
//     sprintf (pathname, "%s/%s", WORKING_DIR, PAGEMAP_FLAGFILENAME);
//     remove(pathname);
//      }
//   }
//   closedir (pdir);
//
//   return (found);

    // Let's try a more efficient way
    static char path[100];

    sprintf (path, "%s/%s", WORKING_DIR, PAGEMAP_FLAGFILENAME);

    if (fileExists(path))
    {
        remove (path);
        return true;
    }
    else return false;
}


bool fileExists (char * fileName)
{
    struct stat buf;
    int retval = stat( fileName, &buf );
    return (retval == 0 ? true : false);
}
```

D.2 Visual Browser Component

The visual browser component extends the Mozilla platform using a combination of custom JavaScript, XUL, and XPCOM files. It is based partially on code developed by Shannon Little [33]. In the interest of brevity, some

automatically generated and trivial configuration files are omitted.

- `browser.js` – the main interactive JavaScript routine.
- `webscroller.css` – Cascading Style Sheet description of the default browser presentation settings.
- `browser.xul` – onscreen user interface description file.
- `readydialog.xul` – user interface description for a modal dialog used as a prompt during user evaluation.
- `IMyComponent.idl` – IDL (Interface Description Language) stub header for the XPCOM I/O routines.
- `MyComponent.cpp` – the file input and output routines, implemented in the XPCOM framework.

D.2.1 browser.js

```
// browser.js
// Joseph Luk, July 2006
// Based on framework code by Shannon Little

// Globals

// uncomment one of the following input modes
// var input_mode = "mouse";
// var input_mode = "slider-";
 var input_mode = "slider+";


var init_page = "file:///home/luk/browser-content/index.html"


const tactile_scalefactor = 1;
    // should be 1 (full-scale) or 2 (half-scale)

var myBrowser = null;
var appCore = null;
var top = 0;
var current = 0;
var searchRange;
var startPt;
var endPt;
var linkTexts;
var linkURLs;
var highlighted = null;
var pageheight = 0;
var pagewidth = 0;
var addheight = 0;
var links;
```

```
var numlinks = 0;  // total number of links in the page
var num_links_at_ypos;
var componentlinker;


// study logging globals
var log_resetCount = 0;
var log_startTime = "";
var log_endTime = "";
var log_pages = new Array();  // page load links
var log_times = new Array();  // page load times



var firstload = true;
var debug = true;

const PREFS_CID = "@mozilla.org/preferences;1";
const PREFS_I_PREF = "nsIPref";
const PREF_STRING = "browser.dom.window.dump.enabled";

try{
    var Pref = new Components.Constructor(PREFS_CID, PREFS_I_PREF);
  var pref = new Pref();
  pref.SetBoolPref(PREF_STRING, true);
}catch(e){}

function initBrowser() {
        if (debug) loadPage_("javascript:");  // load debug page if applicable

      myBrowser = document.getElementById("browser-content");

      appCore = Components.classes["@mozilla.org/appshell/component/browser/
          instance;1"]
                .createInstance(Components.interfaces.nsIBrowserInstance);
      appCore.setWebShellWindow(window);

      var content = document.getElementById("browser-content");
      if(content){
        content.addEventListener("load", setup, true);
      }

  try {

    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

    const cid = "@mydomain.com/XPCOMSample/MyComponent;1";

    componentlinker = Components.classes[cid].createInstance();

    componentlinker = componentlinker.QueryInterface(Components.interfaces.
        IMyComponent);

  } catch (err) {

    alert(err);

    return;

  }


    // Go full-screen mode
/*      netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
 */
/*      const MEDIATOR_CONTRACTID="@mozilla.org/appshell/window-mediator;1"; */
/*      const nsIWindowMediator=Components.interfaces.nsIWindowMediator; */
/*      var windowManager= */
/*          Components.classes[MEDIATOR_CONTRACTID].getService(nsIWindowMediator);
 */
/*      try { */
/*        netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect")
 ; */
/*        mainWindow = windowManager.getMostRecentWindow("navigator:browser"); */
/*      } */
/*      catch(e) { */
```

```
/*        alert(c); */
/*      } */
/*      if(mainWindow.sidebar_is_hidden( )) */
/*        hideSidebar=false; */
/*      if(hideSidebar) */
/*        mainWindow.SidebarShowHide( ); */
/*      mainWindow.BrowserFullScreen( ); */
/*      window.fullScreen=true; */
/*      window.locationbar.visible=false; */
/*      window.toolbar.visible=false; */

    var browserWin = document.getElementById("browserwin");

  if (input_mode == "mouse"){
    //attach listeners
    browserWin.addEventListener('DOMMouseScroll', scroll, true);
    browserWin.addEventListener('click', PushButtonHandler, true);
  }

  // attach keyboard select event listener (workaround for nonfunctional select
      button)
  browserWin.addEventListener('keydown', PushButtonHandler, true);

    // Load the page, triggering the setup function
    loadPage_(init_page);

}

function loadPage_(page)
{
   // glob only the last part of the URL
   var pagestring = page.toString();
   var pagefilename = pagestring.substring(pagestring.lastIndexOf("/") + 1,
        pagestring.length);

//   dump ("loadPage_: " + pagefilename + "\n");

   // push the page load into the log array
   log_pages[log_pages.length] = pagefilename;
    var dateObj = new Date();
   log_times[log_times.length] = dateObj.getTime();

   const nsIWebNavigation = Components.interfaces.nsIWebNavigation;
   getBrowser().webNavigation.loadURI(page, nsIWebNavigation.LOAD_FLAGS_NONE,
        null, null, null);
}


function setup(event){



    myBrowser = event.originalTarget;
  myBrowser.collapse = true;
    current = 0;
  highlighted = null;
    document.commandDispatcher.focusedWindow = window;
    document.commandDispatcher.focusedElement = myBrowser.links.item(current);

  if (input_mode.indexOf("slider") != -1){

     pageheightalt = content.innerHeight + content.scrollMaxY;
     pagewidthalt = content.innerWidth + content.scrollMaxX;
     if (debug) dump ("Page dimensions (via content.innerHeight+scrollMaxY)" +
          pageheight + " X " + pagewidth + "\n");

       pageheight = getBrowser().contentDocument.body.scrollHeight;
       pagewidth = getBrowser().contentDocument.body.scrollWidth;
     if (debug) dump ("Page dimensions (via ScrollHeight) " + pageheightalt + " X
          " + pagewidthalt + "\n");


     if (debug) dump ("window.innerHeight = " + window.innerHeight + "\n");
     if (debug) dump ("content.innerHeight = " + content.innerHeight + "\n");
```

```
     numlinks = myBrowser.links.length;
     if (debug) dump ("Number of links = " + numlinks + "\n");

    // pageheight = getBrowser().contentDocument.firstChild.offsetHeight;
    // pagewidth = getBrowser().contentDocument.firstChild.offsetWidth;

    num_links_at_ypos = new Array(pageheight);

    //find rows where there is more than one link and the same y pos
    for (j = 0; j < numlinks; ++j){
      y = findPosY(getBrowser().contentDocument.links.item(j));
      if (!num_links_at_ypos[y]){
        num_links_at_ypos[y] = 1;
      }
      else{
        num_links_at_ypos[y] ++;
        pageheight += getBrowser().contentDocument.links.item(j).offsetHeight;
      }
    }
    //create links array
    links = new Array(pagewidth);
    for (j = 0; j < pagewidth; j++){
      links[j] = new Array(pageheight);
      for (k = 0; k < pageheight; k++){
        links[j][k] = -1;
      }
    }


        // Start generating the pagemap file
        componentlinker.DeletePageMap();
        componentlinker.WritePageMap ('<?xml version="1.0" encoding="UTF-8"?>\n\n
            ');
        componentlinker.WritePageMap ('<pagemap>\n');
        var scaledPageHeight = pageheight / tactile_scalefactor;
        componentlinker.WritePageMap ('    <height>' + scaledPageHeight + '</
            height>\n');


    i = 0;
    //iterate through all the page links
    for (i = 0; i < numlinks; i++)
    {
      link = getBrowser().contentDocument.links.item(i);
      y = findPosY(link);
      x = findPosX(link);
      range = link.offsetHeight;

/*      if (link.name) */
/*          iconname = link.name; */
/*        else */
        iconname = "bigsquare.xml";

      if (debug) dump ("Link name=" + link.name + " ");
      if (debug) dump ("Y = ");
      if (debug) dump (y);
      if (debug) dump (" Range = ");
      if (debug) dump (range);
      if (debug) dump ("\n");

      var scaledY = y / tactile_scalefactor;
      componentlinker.WritePageMap ("    <icon>\n");
      componentlinker.WritePageMap ("        <at>" + scaledY + "</at>\n");
//      componentlinker.WritePageMap ("        <range>" + range + "</range>\n");
      componentlinker.WritePageMap ("        <name>" + iconname + "</name>\n");
      componentlinker.WritePageMap ("    </icon>\n");


      found = false;
      for (j = 0; (j < pagewidth && !found); j++){
        //if there is already a link at this y, extend new link's y coord by its
            height
        if (links[j][y] >= 0){
          y = y + range;
          found = true;
        }
```

```
      }
      for (k = y; k < y + range; k++){
        links[x][k] = i;
      }
    }

    componentlinker.WritePageMap ('</pagemap>\n');

    // trigger reload from tactile loop
    // deprecated, wait until user starts the task before doing this
  }
  //otherwise, assume mouse
  else {
    highlighted = getBrowser().contentDocument.links.item(current);
    getBrowser().contentDocument.getElementById("highlightdiv").style.visibility
        = "hidden";
  }


  // scale the highlight image as needed to represent the cursor size
  var imagetag = "<img src='highlight-image.png' id='highlightimage' width=";
  imagetag += pagewidth;
  imagetag += " height=";
//  imagetag += 8 * tactile_scalefactor;    (without bilinear interpolation)
  imagetag += 16 * tactile_scalefactor;
  imagetag += " />";
    getBrowser().contentDocument.getElementById("highlightdiv").innerHTML =
        imagetag;




   if (firstload)
   {


    window.openDialog("chrome://webscroller/contents/readydialog.xul","showmore",
                  "chrome,modal");

    firstload = false;
    log_resetCount++;
    log_pages = new Array();
    log_times = new Array();
    log_endTime = 0;

    if (input_mode.indexOf("slider") != -1){
        componentlinker.SetPageMapFlag();
        update_from_device();
    }


    var dateObj = new Date();
    log_startTime = dateObj.getTime();


   }
   else
   {
    var dateObj = new Date();
    if (input_mode.indexOf("slider") != -1){
        componentlinker.SetPageMapFlag();
        update_from_device();
    }
     if (debug) dump ("Page loaded at " + dateObj.getTime() +  "\n");
   }



  //needs focus in order to process keyboard
  getBrowser().contentDocument.links.item(0).focus();

    //set bottom text bar
```

```
    curr = document.getElementById("current-link");
}


function goBack()
{
    var webNavigation = getBrowser().webNavigation;
    if (webNavigation.canGoBack)
        webNavigation.goBack();
}

function goForward()
{
    var webNavigation = getBrowser().webNavigation;
    if (webNavigation.canGoForward)
        webNavigation.goForward();
}

function UpdateBackForwardButtons()
{
    var backBroadcaster = document.getElementById("canGoBack");
    var forwardBroadcaster = document.getElementById("canGoForward");
    var webNavigation = getBrowser().webNavigation;

    var backDisabled = (backBroadcaster.getAttribute("disabled") == "true");
    var forwardDisabled = (forwardBroadcaster.getAttribute("disabled") == "true")
        ;

    if (backDisabled == webNavigation.canGoBack)
        backBroadcaster.setAttribute("disabled", !backDisabled);

    if (forwardDisabled == webNavigation.canGoForward)
        forwardBroadcaster.setAttribute("disabled", !forwardDisabled);
}

function getBrowser(){
    return document.getElementById("browser-content");
}

function BrowserContent() {
    return getBrowser().contentDocument;
}



// Implement push-to-scroll function
function ScrollWindow (cursorYPos)
{
    // Constants
    var margintop = 100;  // Scrolling begins when cursor hits margin
    var marginbottom = margintop;

    // Local vars
    var currtop = getBrowser().contentDocument.body.scrollTop;
    var currbottom = currtop + content.innerHeight;
    var newtop = currtop;

    // Code

    if (cursorYPos < currtop + margintop)
    {
        newtop = cursorYPos - margintop;
        if (newtop < 0) newtop = 0;
    }
    else if (cursorYPos > currbottom - marginbottom)
    {
        newtop = cursorYPos + marginbottom - content.innerHeight;
        if (newtop > content.scrollMaxY) newtop = content.scrollMaxY;
    }

    getBrowser().contentDocument.body.scrollTop = newtop;

}
```

```
//for input_mode = slider
function update_from_device(){

  if (input_mode.indexOf("slider") == -1){
    alert("ERROR: method \'update_from_device\' should not be called unless using
          slider device.\n Set string in line 2 of browser.js to \"slider\".");
    return;
  }
  else{
    click = componentlinker.GetClick();

    if(click && highlighted != null){
/*      document.getElementById("URLBar").insertItemAt(0, highlighted,
    highlighted); */
/*          document.getElementById("URLBar").selectedIndex = 0; */
      myHighlight(highlighted, "yellow");
        // goTo();
    }
    else{

        var scaledYPos = componentlinker.ReadSliderPos();
        var ypos = scaledYPos * tactile_scalefactor;

//      var ypos = Math.round( (componentlinker.ReadSliderPos() / 100) *
    pageheight);

/*              var canvas = document.getElementById("canvas"); */
/*              var ctx = canvas.getContext("2d"); */
/*              ctx.fillStyle = "rgba(150, 150, 0, 0.5)"; */
/*              ctx.fillRect (0, ypos-1, pagewidth, ypos+1); */

    // move the highligh-div element to show the ypos
    // TODO: need to adjust to correspond to tactile window
    var cursortop = ypos - 4;
    getBrowser().contentDocument.getElementById("highlightdiv").style.top=
         cursortop+"px";


    ScrollWindow(ypos);


      var i = 0;
      var found = false;
      for (i; (i < pagewidth && !found); i++){
        if (links[i][ypos] >= 0){
          // check if highlighted is different from previous; if so, play icon..
/*          if (getBrowser().contentDocument.links.item(links[i][ypos]) !=
    highlighted) */
/*          { */
/*            componentlinker.WriteIcon("link");              */
/*          } */

          if (highlighted != null){
            myHighlight(highlighted, null); //clear previous highlighting
          }
          else
          {
        /*      componentlinker.WriteIcon("link");  */
          }

          highlighted = getBrowser().contentDocument.links.item(links[i][ypos]);
          highlighted.focus();
          myHighlight(highlighted, "#f55");
          found = true;
        }
      }
      if (!found && highlighted != null){
        myHighlight(highlighted, null);
        highlighted = null;
        // componentlinker.WriteIcon("null");
      }
    }



    setTimeout('update_from_device()', 20);  //TODO - tweak this value
```

```
  }
}

//for input_mode = mouse
function scroll(event){
    var direction = event.detail;


  //direction > 0 --> UP
  //direction < 0 --> DOWN
  //if we're not already at the top or bottom, then scroll

  if ((direction > 0 && current < (myBrowser.links.length - 1)) || (direction < 0
      && current > 0)){
    myHighlight(highlighted, null); //clear previous highlighting
    if (direction > 0)
        current = current + 1;
    else
        current = current - 1;  0
    highlighted = getBrowser().contentDocument.links.item(current);
    highlighted.focus();
    myHighlight(highlighted, "#f55");
    setCurrLabel();
    }
  event.preventDefault(); //don't actually scroll the page - focus() will do that
      for us
}


function PushButtonHandler(event) {
    var linkSelected = false;


    if (event.type == "click")
    {
        button = event.which;

        event.preventDefault(); // cancel default event handler
    }
    else if (event.type == "keydown")
    {
        var dateObj = new Date();
        if (String.fromCharCode(event.keyCode) == " ")
        {
            log_endTime = dateObj.getTime();
            if (debug) dump ("User signalled task completion at: " + dateObj.
                getTime() +  "\n");
            StudyLog();
        }
        else if (String.fromCharCode(event.keyCode) == "B") getBrowser().history.
            back();  // untested
        else if (String.fromCharCode(event.keyCode) == "R")
        {
            if (debug) dump ("Reset at: " + dateObj.getTime() +  "\n");
            firstload = true;
            loadPage_(init_page);
        }

        else  // TODO: test for return key press here
        {
           linkSelected = true;

        }
        event.preventDefault(); // cancel default event handler
    }

    if (linkSelected)
    {

    /*    document.getElementById("URLBar").insertItemAt(0, highlighted,
        highlighted); */
    /*          document.getElementById("URLBar").selectedIndex = 0; */
        var dateObj = new Date();
        if (debug) dump("Link selected at " + dateObj.getTime() + ": " +
            highlighted + "\n");
```

```
        myHighlight(highlighted, "yellow");
        loadPage_(highlighted);
    }

}


// ********************************************************

function setCurrLabel(){
//   curr = document.getElementById("current-link");
//   text = highlighted;
}

function myHighlight(node, color) {
//   if (debug) dump(node);
  if (color == null){
    node.removeAttribute("style");
  }
  else {
    node.setAttribute("style", "background-color: " + color + ";");
  }
}

//not used
function removeImages(){
  var images = getBrowser().contentDocument.getElementsByTagName('img');
  for(var i = 0;i < images.length; i++) {

    parent = images[i].parentNode;
    parent.removeChild(images[i]);
  }

}
function shrinkImages() {
  //for small screen viewing
  var screen_height = 160;
  var screen_width = 120;
  var l = getBrowser().contentDocument.getElementsByTagName('img');
  for(var i = 0;i < l.length; i++) {
    if(l[i].width > screen_width) {
      l[i].height *= screen_height /l[i].width;
      l[i].width   = screen_width;
    }
    else if(l[i].naturalWidth > screen_width) {
      var e = screen_width / l[i].naturalWidth;
      l[i].height = l[i].naturalHeight * e;
      l[i].width  = screen_width;
    }
  }
}

function findPosX(obj)
{
  var curleft = 0;
  if (obj.offsetParent)
  {
    while (obj.offsetParent)
    {
      curleft += obj.offsetLeft
      obj = obj.offsetParent;
    }
  }
  else if (obj.x)
    curleft += obj.x;
  return curleft;
}

function findPosY(obj)
{
  var curtop = 0;
  if (obj.offsetParent)
  {
    while (obj.offsetParent)
    {
      curtop += obj.offsetTop
```

```
      obj = obj.offsetParent;
    }
  }
  else if (obj.y)
    curtop += obj.y;
  return curtop;
}


// Dump log for study
// TODO: add file output would probably be best
function StudyLog()
{
    // log file format:
    // LOG: subj,block#,cond,task#,tasktype,task,starttime,endtime,elapsedtime,
         page1addr,page1time,page2addr,page2time, ....
    // aka LOG: input_mode,log_resetCount,,,log_startTime,log_endTime,(calculated
         ), contents of log_pages and log_titmes arrays

    var logoutput = "LOG: ";

    logoutput += input_mode;
    logoutput += ",";
    logoutput += log_resetCount;
    logoutput += ",";
    logoutput += log_startTime;
    logoutput += ",";
    logoutput += log_endTime;
    logoutput += ",";
    if (log_startTime && log_endTime)
    {
        logoutput += log_endTime - log_startTime;
    }

  for(var i = 0;i < log_pages.length; i++) {
        logoutput += ",";
        logoutput += log_pages[i];
        logoutput += ",";
        logoutput += log_times[i];
    }

    logoutput += "\n";
    dump (logoutput);

}
```

D.2.2 webscroller.css

```
window
{
 background-color: white;
}
.highlight
{
 background-color: #CCFFFF;
}
.normal
{
 background-color: white;
}
.selected
{
 background-color: red;
}
```

D.2.3 browser.xul

```
<?xml version="1.0"?>

<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<?xml-stylesheet href="chrome://navigator/skin" type="text/css"?>
<?xml-stylesheet href="file:///home/luk/browser-ui/webscroller/skin/webscroller.
     css" type="text/css"?>

<window id="browserwin" title="Tactile browser"
     xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     width="240"
     height="320"
     toolbar="yes"
     scroolbars="yes"
     onload="initBrowser();" >

<script type="application/x-javascript" src="browser.js"/>

<broadcasterset id="browserBroadcasters">
<!--
    <broadcaster id="canGoBack"     disabled="true"/>
    <broadcaster id="canGoForward" disabled="true"/>
 -->
</broadcasterset>

<browser id="browser-content" type="content-primary"
            src="about:blank" flex="1" />


</window>
```

D.2.4 readydialog.xul

```
<?xml version="1.0"?>

<?xml-stylesheet href="chrome://global/skin/global.css" type="text/css"?>

<dialog id="donothing" title="Ready"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
        buttons="accept"
        ondialogaccept="return doOK();">

<script>
function doOK()
{
  return true;
}
</script>

<description value="Ready?"/>

</dialog>
```

D.2.5 IMyComponent.idl

```
#include "nsISupports.idl"

[scriptable, uuid(008c030b-1f71-4b51-9df9-31915d567103)]
interface IMyComponent : nsISupports
{

    long ReadSliderPos();
    long WriteIcon(in string type);
    boolean GetClick();

    boolean DeletePageMap();
    boolean WritePageMap(in string type);
```

```
    boolean SetPageMapFlag();
};
```

D.2.6 MyComponent.cpp

```
#include "MyComponent.h"
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <string.h>
#include <dirent.h>


NS_IMPL_ISUPPORTS1(MyComponent, IMyComponent)



// These must be kept synchronized with the tactile loop's main.cpp!!
#define WORKING_DIR "/home/luk/browser-working-dir"
#define PAGEMAP_FILENAME "pagemap.xml"
#define PAGEMAP_FLAGFILENAME "newpagemap"



MyComponent::MyComponent()
{
  /* member initializers and constructor code */
}

MyComponent::~MyComponent()
{
  /* destructor code */
}

/* long ReadSliderPos (out long pos); */
NS_IMETHODIMP MyComponent::ReadSliderPos(PRInt32 *_retval)
{
  DIR* pdir = opendir(WORKING_DIR);
      struct dirent *pent;

    std::string::size_type startpos;

  if (!pdir){
    printf ("opendir() failure; terminating");
    exit(1);
   }

  std::string pos = "";
  while ((pent=readdir(pdir))){
     std::string file = pent->d_name;
     int length = file.length();
     //filter out .pos (position) files only


     if((startpos = file.rfind(".pos", length)) != std::string::npos){
         pos = file.substr(0, startpos);
    break;
     }
  }
  if (pos == "") {
     *_retval = -1;
  }
  else {
     *_retval = atoi(pos.c_str());
  }

  closedir (pdir);

//     printf("Slider post returning: %d on string %s\n", *_retval, pos.c_str());

  return NS_OK;
```

```
}

/* long WriteIcon (in string type); */
// 15-Dec-05 DEPRECATED!!!!
NS_IMETHODIMP MyComponent::WriteIcon(const char *type, PRInt32 *_retval)
{
  //delete all existing icon files
      DIR* pdir = opendir(WORKING_DIR);
      struct dirent *pent;

  if (!pdir){
    printf ("opendir() failure; terminating");
    exit(1);
   }

  while ((pent=readdir(pdir))){
      std::string file = pent->d_name;
      int length = file.length();
       //filter out .icon (icon) files only
       if(file.rfind(".icon", length) == (unsigned int)length-5){
      //delete file
      char type_str[50];
        strcpy (type_str,WORKING_DIR);
        strcat (type_str,"/");
        strcat (type_str,file.c_str());
      remove(type_str);
      }
  }
  closedir (pdir);

  //write new icon file
  char type_str[50];
    strcpy (type_str,WORKING_DIR);
    strcat (type_str,"/");
    strcat (type_str,type);
    strcat (type_str,".icon");
  std::ofstream outputFile;
  outputFile.open (type_str, std::ofstream::out);
  //std::assert (outputFile); //check for output stream errors
  outputFile.close();
  return NS_OK;
}

/* long GetClick (out boolean clicked); */
NS_IMETHODIMP MyComponent::GetClick(PRInt32 *_retval)
{
  DIR* pdir = opendir(WORKING_DIR);
      struct dirent *pent;

  if (!pdir){
    printf ("opendir() failure; terminating");
    exit(1);
   }

  bool found = false;
  while ((pent=readdir(pdir)) && found == false){
     std::string file = pent->d_name;
     int length = file.length();
     //filter out .click (click) files only
     if(file.rfind(".click", length) == (unsigned int)length-6){
    found = true;
    //delete file
    char type_str[50];
      strcpy (type_str,WORKING_DIR);
      strcat (type_str,"/");
      strcat (type_str,file.c_str());
    remove(type_str);
     }
  }

  *_retval = found;

  closedir (pdir);
      //printf("Slider post returning: %d\n", *_retval);
  return NS_OK;
}
```

```
// ---------------------------------------------------------------------
// Page map / reloading support routines

// long DeletePageMap (out boolean successFlag)
// Removes the page map file
NS_IMETHODIMP MyComponent::DeletePageMap(PRBool *_retval)
{
  char filename[50];
    strcpy (filename,WORKING_DIR);
    strcat (filename,"/");
    strcat (filename,PAGEMAP_FILENAME);

    remove (filename);

    *_retval = true;
    return NS_OK;
}



// long WritePageMap (in string outputString)
// Appends a line to the PageMap file
NS_IMETHODIMP MyComponent::WritePageMap(const char *outputString, PRBool *_retval
    )
{
    // TODO: should check to see if the pagemap reload flag is set (i.e.,
        existence of the file)
    // and wait until the file is deleted (i.e., the tactile loop is done loading
        it) before
    // clobbering the pagemap file

  //write new icon file
  char filename[50];
    strcpy (filename,WORKING_DIR);
    strcat (filename,"/");
    strcat (filename,PAGEMAP_FILENAME);
  std::ofstream outputFile;
  outputFile.open (filename, std::ofstream::app);
//  std::assert (outputFile); //check for output stream errors

  outputFile << outputString;

  outputFile.close();
    *_retval = true;
  return NS_OK;
}


// long SetPageMapFlag (out boolean successFlag)
// Sets the page map flag file, causing the tactile loop to reload the page map
NS_IMETHODIMP MyComponent::SetPageMapFlag(PRBool *_retval)
{
  //write new icon file
  char filename[50];
    strcpy (filename,WORKING_DIR);
    strcat (filename,"/");
    strcat (filename,PAGEMAP_FLAGFILENAME);
  std::ofstream outputFile;
  outputFile.open (filename, std::ofstream::out);
//  std::assert (outputFile); //check for output stream errors
  outputFile.close();
    *_retval = true;
  return NS_OK;
}
```

Appendix E

Browser Experiment Software Code

Software that was created to support the browser user evaluation is included here. The application is based on web standards and can run in versions of Microsoft Internet Explorer, Mozilla Firefox, and Apple Safari that were current as of this writing. The files are included as follows. In the interest of brevity, some files that are small modifications of the files shown here (for example, for the "training" cases) are omitted.

- `taskloop.js` – JavaScript control logic for the experiment task software. There are also files for the "training" cases, which are nearly identical except for a reduction in the number of tasks and non-randomized task presentation order; thus, they have been omitted.

- `taskloop.html` – Presentation layer (including embedded Cascading Style Sheet) for the experiment task software.

- `ajaxcomponent.js` – Logic related to the AJAX (Asynchronous JavaScript and XML) communication method, which allows the page to interactively communicate with and send data to the server without exposing the user to a page reload.

- `reinforce.js` – Logic related to providing feedback to the user on their performance on the distraction task.

E.1 taskloop.js

```
// PC Task program
// (c) 2006 Joseph Luk
// Functionality:
// 1. Display the task to the user
// 2. Display the distractor task to the user
// 3. Accept user input on the distractor task
// 4. Produce log output

const tasksPerBlock = 6;


var currentBlockNumber = 0;
var currentTaskNumber = 0;

var blockGroupOrder = new Array();

var groupOrder = new Array();
// all possible permutations of group order
groupOrder[0] = new Array(0, 1, 2);
groupOrder[1] = new Array(0, 2, 1);
groupOrder[2] = new Array(1, 0, 2);
groupOrder[3] = new Array(1, 2, 0);
groupOrder[4] = new Array(2, 0, 1);
groupOrder[5] = new Array(2, 1, 0);

var taskCounterByGroup = new Array(0,0,0);
var selectedTaskGroup = 0;
var selectedTask = 0;
var currentPedalTask = 0;

var tasks = new Array();
tasks[0] = new Array();
tasks[1] = new Array();
tasks[2] = new Array();

tasks[0][0] = "What is the weather in London today?";
tasks[0][1] = "What will the weather be like in London tomorrow?";
tasks[0][2] = "What will the weather be like in London the day after tomorrow?";
tasks[0][3] = "What is the weather in Paris today?";
tasks[0][4] = "What will the weather be like in Paris tomorrow?";
tasks[0][5] = "What will the weather be like in Paris the day after tomorrow?";
tasks[0][6] = "What is the weather in Tokyo today?";
tasks[0][7] = "What will the weather be like in Tokyo tomorrow?";
tasks[0][8] = "What will the weather be like in Tokyo the day after tomorrow?";
tasks[0][9] = "What is the weather in Hong Kong today?";
tasks[0][10] = "What will the weather be like in Hong Kong tomorrow?";
tasks[0][11] = "What will the weather be like in Hong Kong the day after tomorrow
    ?";
tasks[0][12] = "What is the weather in San Francisco today?";
tasks[0][13] = "What will the weather be like in San Francisco tomorrow?";
tasks[0][14] = "What will the weather be like in San Francisco the day after
    tomorrow?";
tasks[0][15] = "What is the weather in Toronto today?";
tasks[0][16] = "What will the weather be like in Toronto tomorrow?";
tasks[0][17] = "What will the weather be like in Toronto the day after tomorrow
    ?";

tasks[1][0] = "If you take the 99 B-line from UBC at 1pm, when will you arrive at
     Broadway station?";
tasks[1][1] = "If you take the 99 B-line from UBC at 1pm, when will you arrive at
     Granville?";
tasks[1][2] = "If you take the 99 B-line from UBC at 1pm, when will you arrive at
     Main?";
tasks[1][3] = "If you take the 99 B-line from UBC at 9pm, when will you arrive at
     Broadway station?";
tasks[1][4] = "If you take the 99 B-line from UBC at 9pm, when will you arrive at
     Granville?";
tasks[1][5] = "If you take the 99 B-line from UBC at 9pm, when will you arrive at
     Main?";
tasks[1][6] = "If you take the #44 bus from UBC at 10am, when will you arrive at
    Macdonald?";
tasks[1][7] = "If you take the #44 bus from UBC at 10am, when will you arrive at
    Davie?";
```

```
tasks[1][8] = "If you take the #44 bus from UBC at 10am, when will you arrive at
    Waterfront station?";
tasks[1][9] = "If you take the #44 bus from UBC at 5pm, when will you arrive at
    Macdonald?";
tasks[1][10] = "If you take the #44 bus from UBC at 5pm, when will you arrive at
    Davie?";
tasks[1][11] = "If you take the #44 bus from UBC at 5pm, when will you arrive at
    Waterfront station?";

tasks[2][0] = "When is the movie &quot;L&#8217;Enfant&quot; playing at the Ridge
    Theatre?";
tasks[2][1] = "When is the movie &quot;L&#8217;Enfant&quot; playing at Fifth
    Avenue Cinemas?";
tasks[2][2] = "When is the movie &quot;L&#8217;Enfant&quot; playing at Empire
    Oakridge?";
tasks[2][3] = "What rating did the movie &quot;L&#8217;Enfant&quot; receive in
    its review in the Los Angeles Times?";
tasks[2][4] = "What rating did the movie &quot;L&#8217;Enfant&quot; receive in
    its review in the New York Post?";
tasks[2][5] = "When is the movie &quot;The Promise&quot; playing at the Ridge
    Theatre?";
tasks[2][6] = "When is the movie &quot;The Promise&quot; playing at the Pacific
    Cinematheque?";
tasks[2][7] = "When is the movie &quot;The Promise&quot; playing at the
    Silvercity Riverport?";
tasks[2][8] = "What rating did the movie &quot;The Promise&quot; receive in its
    review in the Globe and Mail?";
tasks[2][9] = "What rating did the movie &quot;The Promise&quot; receive in its
    review in the Washington Post?";

var taskJumble = new Array();


var pedalTask = new Array();
pedalTask[0] = "PRESS LEFT PEDAL ONCE";
pedalTask[1] = "PRESS LEFT PEDAL TWICE";
pedalTask[2] = "PRESS RIGHT PEDAL ONCE";
pedalTask[3] = "PRESS RIGHT PEDAL TWICE";
pedalTask[4] = "PRESS LEFT, THEN RIGHT PEDAL";
pedalTask[5] = "PRESS RIGHT, THEN LEFT PEDAL";
var currentPedalTask;
var prevPedalTask = -1;
var prevPedalCorrect = true;
var leftPedalCount = 0;
var rightPedalCount = 0;

var pedalCorrect = 0;
var pedalTotal = 0;


var lastTimeout;


function init() {
    document.getElementById("taskInfo").style.visibility = "hidden";
    document.getElementById("mainTask").style.visibility = "hidden";
    document.getElementById("pedalTask").style.visibility = "hidden";

}

function StartExpt()
{
    var i=0;
    var j=0;

//  document.getElementById("log").value = "";
//  document.getElementById("data").value = "";

    document.getElementById("mainTask").style.visibility = "visible";
    document.getElementById("pedalTask").style.visibility = "hidden";
    document.getElementById("setupScreen").style.visibility = "hidden";
    document.setupForm.beginexpt.disabled = true;

//    document.getElementById("mainTask").focus();

    Log ("Experiment started.  Subject # = " + document.setupForm.subject.value);
```

```
    // randomize the task lists within groups
    for (i=0; i<tasks.length; i++)
    {
        taskJumble[i] = new Array(tasks[i].length);
        for (j=0; j<tasks[i].length; j++)
        {
            taskJumble[i][j] = j;
        }

    }
    Permute(taskJumble[0]);
    Permute(taskJumble[1]);
    Permute(taskJumble[2]);


    // initialize the blockGroupOrder array
    var numBlocksMaxSix = document.setupForm.blocks.value;
    if (numBlocksMaxSix > 6) numBlocksMaxSix = 6;

    for (i=0; i<numBlocksMaxSix; i++)
    {
        blockGroupOrder[i] = i;
    }
    // randomize it
    Permute(blockGroupOrder);

    var logthis;
    for (i=0; i<document.setupForm.blocks.value; i++)
    {
        logthis = "Task Groups used in Block #" + (i+1) + ": ";
        logthis += GroupAsAlpha(groupOrder[blockGroupOrder[i%6]][0]);
        logthis += GroupAsAlpha(groupOrder[blockGroupOrder[i%6]][1]);
        logthis += GroupAsAlpha(groupOrder[blockGroupOrder[i%6]][2]);
        Log(logthis);
    }


  StartBlock();

    // add event handler to wait for keypress to begin task
}

function StartBlock()
{
    currentBlockNumber ++;
    Log ("-----");
    Log ("Block #" + currentBlockNumber);
    currentTaskNumber = 0;

    StartTask();
}


function EndExpt()
{
    document.getElementById("taskInfo").style.visibility = "hidden";
    document.getElementById("mainTask").style.visibility = "hidden";
    document.getElementById("pedalTask").style.visibility = "hidden";


    AjaxSendForm("FINAL");

    // deprecated, we're using Ajax to send the form after every block now
/*      document.outputForm.subject.value = "Study log results (FINAL after
     completing " + currentBlockNumber + " blocks)";      */
/*      document.outputForm.submit(); */

    alert ("Thank you for participating in this experiment.\nHave a nice day.");

    return;
}



function EndBlock()
```

```
{

    if (currentBlockNumber == document.setupForm.blocks.value) EndExpt();
    else {
        AjaxSendForm("INTERIM");
        document.getElementById("taskInfo").style.visibility = "hidden";
        document.getElementById("mainTask").style.visibility = "hidden";
        document.getElementById("pedalTask").style.visibility = "hidden";
        alert ("Done with block " + currentBlockNumber + ".\nTake a one-minute
            break.");
        document.getElementById("taskInfo").style.visibility = "visible";
        document.getElementById("mainTask").style.visibility = "visible";
        StartBlock();
    }
}


function StartTask() {
    var logthis;

    document.getElementById("pedalTask").style.visibility = "hidden";

    currentTaskNumber++;
    if (currentTaskNumber > tasksPerBlock) { EndBlock(); return; }

    document.getElementById("taskInfo").style.visibility = "visible";
    document.getElementById("taskInfo").innerHTML = "Task " + currentTaskNumber +
        " of " + tasksPerBlock;


    selectedTaskGroup = groupOrder[blockGroupOrder[(currentBlockNumber-1) % 6]][(
        currentTaskNumber-1) % 3];

    selectedTask = taskJumble[selectedTaskGroup][taskCounterByGroup[
        selectedTaskGroup] % tasks[selectedTaskGroup].length];
    taskCounterByGroup[selectedTaskGroup] ++;

    logthis = "Task #" + currentTaskNumber;
    logthis += ", using " + GroupAsAlpha(selectedTaskGroup);
    logthis += selectedTask + ": ";
    logthis += tasks[selectedTaskGroup][selectedTask];
    Log(logthis);


    // display the task box
    document.getElementById("mainTask").innerHTML = tasks[selectedTaskGroup][
        selectedTask];

    // wait for spacebar press
//    addEventListener('keydown', StartPedalTask, true);
    window.onkeypress = StartPedalTask;

}

function StartPedalTask(e)
{
    var evtobj=window.event? event : e //distinguish between IE's explicit event
        object (window.event) and Firefox's implicit.
    var unicode=evtobj.charCode? evtobj.charCode : evtobj.keyCode
    var actualkey=String.fromCharCode(unicode)


    if (actualkey != " ")
    {
        evtobj.preventDefault();
        return;
    }

    // spacebar pressed, proceed
    document.getElementById("pedalTask").style.visibility = "visible";

    prevPedalTask = -1;
    prevPedalCorrect = true;
    pedalCorrect = 0;
    pedalTotal = 0;
```

```
    // wait 3 seconds before starting the pedal task
    lastTimeout = setTimeout (NewPedalTask, 3000);

    window.onkeypress = EndPedalTask;

    evtobj.preventDefault();
}

function EndPedalTask(e)
{
    var evtobj=window.event? event : e //distinguish between IE's explicit event
        object (window.event) and Firefox's implicit.
    var unicode=evtobj.charCode? evtobj.charCode : evtobj.keyCode
    var actualkey=String.fromCharCode(unicode)

    var logthis;

    if (actualkey != " ")
    {
        evtobj.preventDefault();
        return;
    }

    // spacebar pressed, proceed
    clearTimeout(lastTimeout);
    window.onkeypress = null;
    document.onmousedown = null;

    document.getElementById("pedalTask").innerHTML = "";

    logthis = "";
    logthis += document.setupForm.subject.value + ", ";
    logthis += currentBlockNumber + ", ";
    logthis += currentTaskNumber + ", ";
    logthis += GroupAsAlpha(selectedTaskGroup) + ", ";
    logthis += selectedTask + ", ";
    logthis += pedalTotal + ", ";
    logthis += pedalCorrect;
    Data (logthis);


    StartTask();

    evtobj.preventDefault();
}


function NewPedalTask() {
    // choose a pedal task randomly, but one that is different from the current
        task

    if (currentBlockNumber < 3)
    {
        // ez pedals, first 4 pedal tasks only
        do
        {
            currentPedalTask = GetRandom(0,3);
        } while (currentPedalTask == prevPedalTask);
    }
    else
    {
        do
        {
            currentPedalTask = GetRandom(0,pedalTask.length-1);
        } while (currentPedalTask == prevPedalTask);
    }


    document.getElementById("pedalTask").innerHTML = pedalTask[currentPedalTask];


    leftPedalCount = 0;
    rightPedalCount = 0;
```

```
    document.onmousedown = HandlePedals;

    lastTimeout = setTimeout(PedalTimeout, 7000);

}

function PedalTimeout()
{
    var logthis;
    var correct = PedalsCorrect();


    logthis = "PEDAL: Task " + currentPedalTask + ": ";
    if (correct) logthis += "CORRECT, ";
    else logthis += "WRONG, ";
    logthis += "Left=" + leftPedalCount;
    logthis += ", Right=" + rightPedalCount;
    Log(logthis);

    pedalTotal++;
    if (correct) pedalCorrect++;

    if (correct) document.getElementById("pedalTask").style.backgroundColor = "#
        fff";
    else document.getElementById("pedalTask").style.backgroundColor = "#f55";

    if (correct) prevPedalCorrect = true;
    else prevPedalCorrect = false;

    if (correct) CorrectPedalResponse();
    else WrongPedalResponse();


    prevPedalTask = currentPedalTask;
    NewPedalTask();
}


function PedalsCorrect()
{
    var correct = false;

    switch (currentPedalTask)
    {
        case 0:
            if (leftPedalCount == 1 && rightPedalCount == 0) correct=true;
            break;
        case 1:
            if (leftPedalCount == 2 && rightPedalCount == 0) correct=true;
            break;
        case 2:
            if (leftPedalCount == 0 && rightPedalCount == 1) correct=true;
            break;
        case 3:
            if (leftPedalCount == 0 && rightPedalCount == 2) correct=true;
            break;
        case 4:
        case 5:
            if (leftPedalCount == 1 && rightPedalCount == 1) correct=true;
            break;
    }

    return correct;
}


function HandlePedals(e) {
    var evtobj=window.event? event : e //distinguish between IE's explicit event
        object (window.event) and Firefox's implicit.

    button = evtobj.which;

    switch (button)
    {
        case 1:
            leftPedalCount++;
```

```
            Log ("Left Pedal");
            break;
        case 2:
            break;
        case 3:
            rightPedalCount++;
            Log ("Right Pedal");
            break;
    }

    var correct = PedalsCorrect();
    if (correct)
    {
        var content = "<font color='";
        if (prevPedalCorrect)
            content += "#ddd";
        else content += "#c77";

        content += "'>";

        content += pedalTask[currentPedalTask];
        content += "</font>";
        document.getElementById("pedalTask").innerHTML = content;
    }

    evtobj.preventDefault();
}


function GetRandom(low, high)
{
    return (Math.floor(Math.random() * (high-low+1)) + low);
}

function FormatTwoDigits (number)
{
    var retval = "";
    if (number < 10) retval = "0";
    retval += number;
    return retval;
}

function FormatThreeDigits (number)
{
    var retval = "";
    if (number < 100) retval = "0";
    if (number < 10) retval += "0";
    retval += number;
    return retval;
}

function Log (text)
{
  var myDate = new Date();
  var logoutput = "";

    logoutput += myDate.getFullYear().toString().substring(2,4);
    logoutput += FormatTwoDigits(myDate.getMonth()+1);
    logoutput += FormatTwoDigits(myDate.getDate()) + " ";
    logoutput += myDate.getHours() + ":";
    logoutput += FormatTwoDigits(myDate.getMinutes());
    logoutput += ":";
    logoutput += FormatTwoDigits(myDate.getSeconds()) + ".";
    logoutput += FormatThreeDigits(myDate.getMilliseconds());
    logoutput += " - ";

    logoutput += text;
    logoutput += "\n";

  document.getElementById("log").value += logoutput;
}

function Data (text)
{
  var myDate = new Date();
  var logoutput = "";
```

```
    logoutput += myDate.getFullYear().toString().substring(2,4);
    logoutput += FormatTwoDigits(myDate.getMonth()+1);
    logoutput += FormatTwoDigits(myDate.getDate()) + " ";
    logoutput += myDate.getHours() + ":";
    logoutput += FormatTwoDigits(myDate.getMinutes());
    logoutput += ":";
    logoutput += FormatTwoDigits(myDate.getSeconds()) + ".";
    logoutput += FormatThreeDigits(myDate.getMilliseconds());
    logoutput += " - ";

    logoutput += text;
    logoutput += "\n";

  document.getElementById("data").value += logoutput;
}


function GroupAsAlpha (number)
{
    if (number == 0) return "A";
    if (number == 1) return "B";
    if (number == 2) return "C";

}



function Permute(the_array)
{
    var loop;
  var temp_array = new Array();
  for (loop = 0; loop < the_array.length; loop++)
  {
    temp_array[loop] = the_array[loop];
  }

  var new_array = new Array();
  var random_num = 0;
  for (loop = 0; loop < the_array.length; loop++)
  {
    random_num = Math.round(Math.random() * (temp_array.length-1));
    new_array[loop] = temp_array[random_num];
    temp_array[random_num] = temp_array[temp_array.length-1];
    temp_array.length--;
  }
  for (loop = 0; loop < new_array.length; loop++)
  {
    the_array[loop] = new_array[loop];
  }

  return;

}
```

E.2 taskloop.html

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
    /TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Tactile Browser - PC Task</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<script type="text/javascript" src="reinforce.js" />
<script type="text/javascript" src="ajaxcomponent.js" />
<script type="text/javascript" src="taskloop.js" />
<style type="text/css">
<!--
body
{
```

```
text-align: center;
margin-left: 10%;
margin-right: 10%;
}
div.setupScreen {
  padding: 20px;
  border: 1px solid #666666;
  position: relative;
  top: 10px;
  background-color: #AACCAA;
}div.mainTask {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 36px;
  font-weight: bold;
  background-color: #EEEE77;
  padding: 20px;
  position: relative;
  top: 0px;
  background-position: center;
  text-align: center;
  border: none #999999;
}
div.taskInfo {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 12px;
  font-weight: bold;
  background-color: #ee7;
  padding: 5px;
  position: relative;
  top: 0px;
  background-position: center;
  text-align: center;
  border: none #999999;
  color: #333333;
  margin: auto;
  width: auto;
}
div.pedalTaskContainer {
  position: relative;
  top: 100px;
}
div.pedalTask {
  font-family: "Times New Roman", Times, serif;
  font-size: 48px;
  padding: 20px;
  border: 1px solid #666666;
  position: relative;
  top: 0px;
}
div.pedalReinforce {
  padding: 10px;
  border: none;
  position: relative;
  top:5px;
  margin: auto;
  width: auto;
}
div.logArea {
  padding: 10px;
  border: 1px solid #000000;
  position: absolute;
  top: 1000px;
}-->
</style>
</head>

<body oncontextmenu="return false" onload="init();">
<div id="setupScreen" class="setupScreen">
  <h1>Welcome!</h1>
  <form name="setupForm">
    <p>Subject Number:
      <input name="subject" type="text" id="subject" size="4" />
    </p>
    <p>Number of Blocks:
      <input name="blocks" type="text" id="blocks" value="9" size="3" />
    </p>
```

```
    <p align="center">
      <input name="beginexpt" type="button" onclick="StartExpt();" value="Begin
          Experiment" />
    </p>
  </form>
</div>

<div id="taskInfo" class="taskInfo">Task n of n</div>
<div id="mainTask" class="mainTask">Ready (main).</div>

<div id="pedalTaskContainer" class="pedalTaskContainer">
    <div id="pedalTask" class="pedalTask"></div>
    <div id="pedalReinforce" class="pedalReinforce"></div>
</div>

<div id="logArea" class="logArea">
  <p> </p>
  <form action="http://choberi.com/cgi-bin/FormMail.pl" method="post" name="
      outputForm" id="outputForm">
    <input name="recipient" type="hidden" value="joe@josephluk.com" />
    <input name="subject" type="hidden" value="Study log output" />
    <textarea id="log" name="log" cols="100" rows="10" wrap="off">
    Log contents:

</textarea>
    <textarea id="data" name="data" cols="100" rows="10" wrap="off">
    Data contents:

</textarea>
    <br />
    <input name="Submit" type="submit" />
  </form>
</div>


<p class="mainTask"> </p>
</body>
</html>
```

E.3 ajaxcomponent.js

```
// ajaxcomponent.js
// Joseph Luk, July 2006
// send the interim form data to the server without reloading the page (for
    safety purposes)


var req;

function AjaxSendForm(typeOfBlock) {

    // this line pretty much means you have to run the script locally from file
        ://
    // need to sign the script to have it work over http
    netscape.security.PrivilegeManager.enablePrivilege('UniversalBrowserRead');


    // use ajax call to send form without reloading the page
   if (window.XMLHttpRequest) {
       req = new XMLHttpRequest();
   } else if (window.ActiveXObject) {
       req = new ActiveXObject("Microsoft.XMLHTTP");
   }

   req.open("POST", "http://choberi.com/cgi-bin/FormMail.pl", true);
    // inline callback function to wait for server response

    req.onreadystatechange = function() {
        if (req.readyState == 4) {
            // don't need to do anything, just continue asynchronously
```

```
        }
    }

    document.outputForm.subject.value = "Study log results (" + typeOfBlock + "
        after block " + currentBlockNumber + ")";

    req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    req.send(formData2QueryString(document.outputForm));
}



/*
 * Copyright 2005 Matthew Ernisse (mde@fleegix.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License..
 *
 * Original code by Matthew Ernisse (mde@fleegix.org)
 * Additional bugfixes by Mark Pruett (mark.pruett@comcast.net)
 *
*/

// The var docForm should be a reference to a <form>

function formData2QueryString(docForm) {

  var submitContent = '';
  var formElem;
  var lastElemName = '';

  for (i = 0; i < docForm.elements.length; i++) {

    formElem = docForm.elements[i];
    switch (formElem.type) {
      // Text fields, hidden form elements
      case 'text':
      case 'hidden':
      case 'password':
      case 'textarea':
      case 'select-one':
        submitContent += formElem.name + '=' + escape(formElem.value) + '&'
        break;

      // Radio buttons
      case 'radio':
        if (formElem.checked) {
          submitContent += formElem.name + '=' + escape(formElem.value) + '&'
        }
        break;

      // Checkboxes
      case 'checkbox':
        if (formElem.checked) {
          // Continuing multiple, same-name checkboxes
          if (formElem.name == lastElemName) {
            // Strip of end ampersand if there is one
            if (submitContent.lastIndexOf('&') == submitContent.length-1) {
              submitContent = submitContent.substr(0, submitContent.length - 1);
            }
            // Append value as comma-delimited string
            submitContent += ',' + escape(formElem.value);
          }
          else {
            submitContent += formElem.name + '=' + escape(formElem.value);
          }
          submitContent += '&';
```

```
            lastElemName = formElem.name;
          }
          break;

      }
    }
    // Remove trailing separator
    submitContent = submitContent.substr(0, submitContent.length - 1);
    return submitContent;
}
```

E.4 reinforce.js

```
// Positive reinforcement script for pedal task

var pedalCorrectRun = 0;
    // run of correct pedal responses

function CorrectPedalResponse() {
    pedalCorrectRun ++;

    var filename = pedalCorrectRun;

    if (pedalCorrectRun > 5) filename ="5";
    if (pedalCorrectRun > 7 && (pedalCorrectRun % 2) ) filename="goyou";

    document.getElementById("pedalReinforce").innerHTML = "<img src='" +
        pedalCorrectRun + ".gif' />";

    return;
}

function WrongPedalResponse() {
    pedalCorrectRun = 0;
    document.getElementById("pedalReinforce").innerHTML = "";
    return;
}
```