August 2005

Technical Report TR-2005-21

Theory, Software, and Psychophysical Studies for the Tactile Handheld Miniature Bimodal Device

Shannon Little

Department of Computer Science

University of British Columbia

2366 Main Mall

Vancouver, B.C. V6T 1Z4, Canada

shannon.little@gmail.com

Table of Contents

1. Introduction *6*

2. Basic Principles *7*
2.1. Hardware and Configuration Description 7
2.2. Hardware and Configuration Limitations 10
2.3. Definitions 12
2.4. Theory 14
2.4.1. Output modes 14
2.4.2. Direction 14
2.4.3. Stimulus Speed 15

3. Software *18*
3.1. THMB Library 19
3.1.1. Files 19
3.1.2. Dependencies 19
3.1.3. Compilation and Installation 19
3.1.4. Running the Program 19
3.1.5. Location in CVS 19
3.1.6. Recommendations and Future Work 20
3.2. THMB Designer 21
3.2.1. Files 21
3.2.2. Dependencies 21
3.2.3. Compilation and Installation 21
3.2.4. Running the Program 21
3.2.5. Location in CVS 21
3.2.6. Recommendations and Future Work 21
3.3. Speed Tester 22
3.3.1. Files 22
3.3.2. Dependencies 22
3.3.3. Compilation and Installation 22
3.3.4. Running the Program 22
3.3.5. Location in CVS 22
3.3.6. Recommendations and Future Work 22
3.4. Browser 23
3.4.1. Browser Graphical User Interface (browser-ui) 23
3.4.1.1. Files 23
3.4.1.2. Dependencies 24
3.4.1.3. Compilation and Installation 24
3.4.1.4. Running the Program 24
3.4.1.5. Location in CVS 25
3.4.2. Browser Communications Interface (browser-interface) 25
3.4.2.1. Files 25
3.4.2.2.Dependencies 25
3.4.2.3. Compilation and Installation 25
3.4.2.4. Running the Program 26
3.4.2.5. Location in CVS 26

3.4.3. Browser Server (browser-server)....26
3.4.3.1. Files....26
3.4.3.2. Dependencies....26
3.4.3.3. Compilation and Installation....26
3.4.3.4. Running the Program....26
3.4.3.5. Location in CVS....26
3.4.4. Recommendations and Future Work....27
3.5. MDS....28
3.5.1. MDS Server (MDS-server)....28
3.5.1.1. Files....28
3.5.1.2. Dependencies....28
3.5.1.3. Compilation and Installation....28
3.5.1.4. Running the Program....28
3.5.1.5. Location in CVS....28
3.5.2. MDS Tester....28
3.5.2.1. Files....28
3.5.2.2. Dependencies....29
3.5.2.3. Running the Program....29
3.5.3. Recommendations and Future Work....29
4. User Studies....*30*
4.1. Speed Study....30
4.1.1. Introduction....30
4.1.2. Method....30
4.1.2.1. Participants....30
4.1.2.2. Design....30
4.1.2.3. Task....31
4.1.2.4. Data....31
4.2. MDS Study....32
4.2.1. Introduction....32
4.2.2. Method....32
4.2.2.1. Participants....32
4.2.2.2. Design....32
4.2.2.3. Tasks....33
4.2.2.4. Data....34
5. Conclusions and Future Work....*35*
6. Acknowledgements....*36*
7. References....*37*

1. Introduction

The interface described in [2] uses piezo actuators arranged in a horizontal stack to create tactile stimuli on a handheld device. The THMB device described in this report uses piezo actuators arranged in a vertical array to create tactile stimuli on a handheld device. The primary difference between these two interfaces for handheld devices is that the interface described in [2] creates vibrotactile patterns, while the THMB device uses a lateral skin stretch method to create tactile patterns.

The intention of the work described in this report began as an exploratory effort to determine the basic principles and properties of the THMB device. The result of this experimentation is a description of the basic configuration and perceptual limits of the THMB device, terminology and theory to describe the output of the device, libraries that implement this theory, software that utilizes these libraries to interact with the THMB device, and two user studies, a speed study and a multi-dimensional scaling (MDS) study, that attempt to relate the terminology and theories developed to human perceptual limitations and capabilities.

The theory, software, and user studies described in the report are the result of iterative design and research done through May to August 2005 primarily by the author. The Canadian Distributed Mentor Project (CDMP) and the National Sciences and Engineering Research Council (NSERC) funded the research. See the Acknowledgments for more details on these awards and people involved in the research.

2. Basic Principles

2.1. Hardware and Configuration Description

The experiments and studies described in this technical report used a novel tactile device called the THMB (Tactile Handheld Miniature Bimodal device) developed at McGill University. The THMB uses distributed lateral skin deformation to display tactile stimuli to the thumb finger.

The THMB is a prototype of a tactile interface for a mobile device, such as a mobile phone, personal digital assistant (PDA), or an MP3 player. It is comprised of a miniature tactile display (TD) for the thumb tip, an LCD screen, and a sliding mechanism (see Figure 1). Only the tactile display component of the THMB was used for the studies reported here.

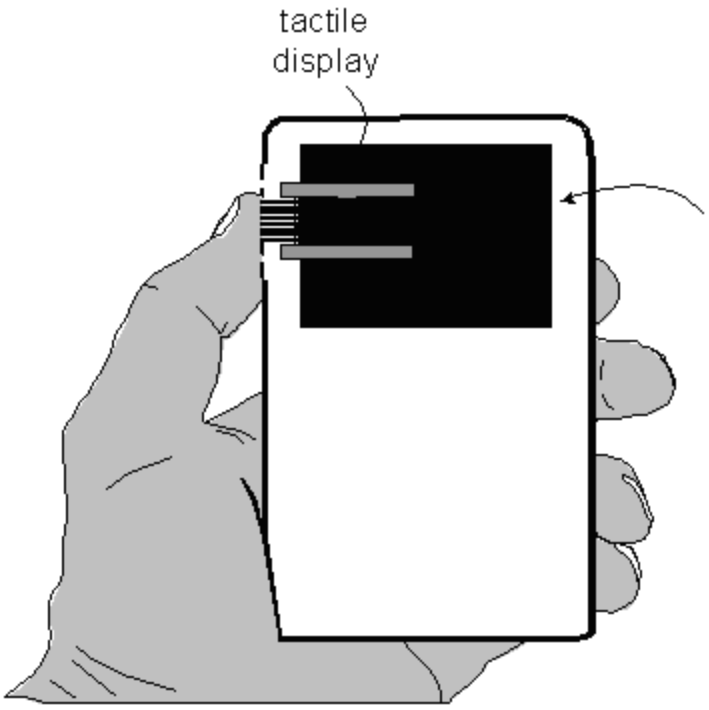

Figure 1: Overview of the THMB device

The tactile display is a miniaturized and improved version of the Virtual Braille Display (VBD) developed at McGill University. The VBD uses lateral skin stretch to create truncated Braille characters for visually impaired users. The THMB uses the same technique to create tactile stimuli for users of handheld devices. Refer to [1] for more details on the basic technology used.

Briefly, the mechanical assembly of the tactile display is about the size of a matchbox, with the contact area about 8.7 mm by 6.4 mm. It consists of 8 piezoelectric actuators that are intercalated in between pair of rods and held together by clamps (see Figure 2).

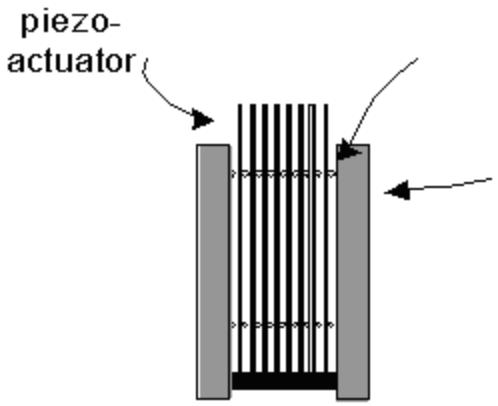

Figure 2: Tactile Display

The top of the tactile display (the top of the piezoelectric actuators) protrudes through the THMB device's case. Users hold the case in their left palm (see Figure 1) and rest their thumb tip across the top of the tactile display (see Figure 3).

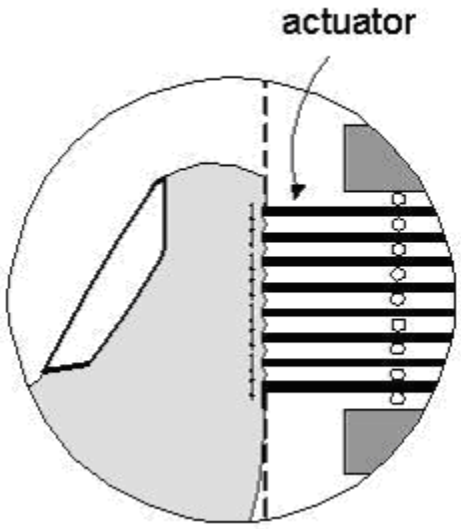

Figure 3: Interaction of the thumb tip with the tactile display

When activated, the tactile display causes lateral skin deformation to the fingertip through actuator tips that move sideways. The amount of bending of a particular piezoelectric actuator is controlled with the application of a voltage across its electrodes[1].

A PC host running Linux generates the 8 control voltages (one for each piezo-actuator) that control the bending. Referring to Figure 4, the control signals are sent through USB and connected to analog by a Field Programmable Gate Array (FPGA). They are then filtered and amplified by a custom built amplifier before being applied across the piezoelectric benders. The resulting control voltages range from ±50V and are updated every 320µs from the FPGA. They are encoded with a single byte and therefore can only take 256 different values.

[1] For more information on piezoelectric actuators, refer to [4]

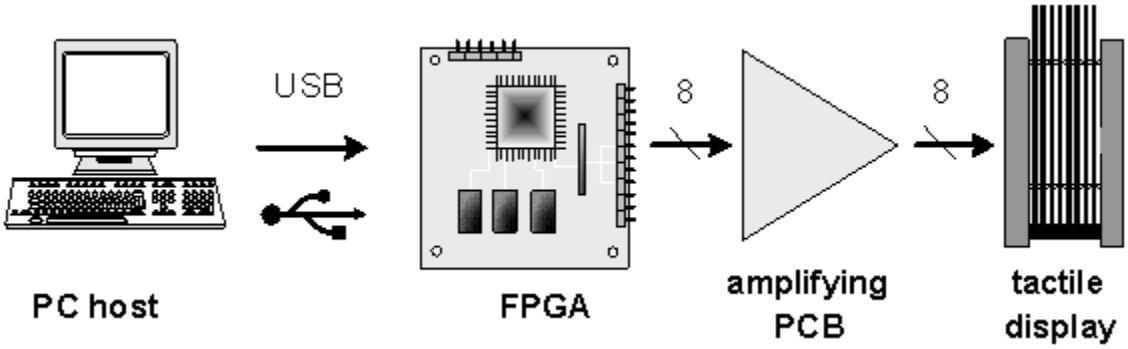

Figure 4: Control system for the THMB

The configuration described above allows for tight control of the timing. The PC host renders the tactile signals to be displayed on the TD and sends them to the FPGA, through USB communication. The FPGA accepts the input stream from the PC host and outputs it in parallel at a fixed rate (3125 samples/sec for each piezo-actuator). This strategy overcomes the poor reliability of USB communication under real-time requirements. The PC host sends data to the FPGA whenever it can. The FPGA then rectifies the timing discrepancies by outputting data at a fixed frequency with a modest but consistent delay.

2.2. Hardware and Configuration Limitations

The hardware and configuration were tested with an oscilloscope and an alternating +/-50V function to determine their characteristics and limitations.

As mentioned in the previous sections, the output sampling rate (parallel for all 8 actuators) of the FPGA board is 1/320 microseconds = 3.125 kHz. This limits us to a minimum phase of 320 microseconds for travelling waves and a similar, but yet undetermined limitation for navigable waves (travelling and navigable waves are defined in section 2.4.1). The limitation for navigable waves will be a combination of this sampling rate, the maximum slider speed possible, and basic human perceptual limits. This has not yet been precisely tested.

Initially, the amplified output from the FPGA board was not reaching +/-50 V within the 320-microsecond interval. To solve this, we doubled each sample (i.e. "stretched" it by 2), which has the effect of halving the resultant sampling rate to 1.5625 kHz and increasing the minimum phase to 640 microseconds. At this frequency, the amplified output displayed the expected square wave function. However, because the tactile device also acts as a capacitor and it was not connected to the circuit for these measurements, it may be necessary to "stretch" each sample by a factor more than 2 to ensure that the actuators are receiving the intended voltage signal. Preliminary tests suggest that this factor is at least 5.

When configuring the device for consistent real-time input and output, it is first necessary to understand the underflow and overflow states of the device. The FPGA outputs images from its buffer at a constant rate (3.125 kHz). The buffer has four "watermarks": very low, low, high, and very high. The normal range of operation is between very low and very high. The device will go into the underflow state if the number of images in the buffer goes below the very low mark and it will not output again until the number of images in the buffer reaches the low mark. Similarly, the device goes into overflow state if the number of images in the buffer goes above the very high mark, in which case it deletes images until it reaches the high mark.

A synchronous write operation was measured to take 8 milliseconds using the configuration described earlier. Operating at a constant rate of 3.125 kHz, the FPGA can output 25 images from its FIFO (First In First Out) buffer in 8 milliseconds. The output rate of this FIFO buffer is synchronous and always the same, which is not the case for the input into the buffer. The input/output operations work as follows: a 25-image package is written from the PC to the FPGA board, where it is added to the FPGA's buffer. The PC then reads from the FPGA, taking 8 ms. During the 8 ms read operation, the device outputs the first 25 images from its buffer. Immediately after outputting the 25th image, another package of 25 images arrives and is added the end of the FPGA buffer. The process then repeats. This process helps to ensure that the FPGA never goes into the underflow or overflow state; the buffer is always "topped up" with 25 images after the FPGA has written 25 images.

All experimentation and software in this report was done using the hardware and configuration described earlier, using (with the exception of the fastest speed in the speed study) at maximum a sampling rate 1.5625 kHz sampling rate (i.e. a stretch multiplier, as defined below, of 2). The software and user studies are designed for these specific hardware and configuration limitations.

With different hardware and/or under a different configuration, the output sampling rate, the read time, or other factors may be different, requiring adjustments of the image package size and other hardware- and configuration-specific theory. Refer to Vincent Levesque at McGill University for more information about adjusting the configuration specific parameters.

2.3. Definitions

For the most part, all stimuli described in this report are travelling pattern. Briefly, a travelling pattern is a pattern that is propagated across the actuators in the display and is dependent only on time, not slider position. Travelling waves are defined precisely in section 2.4.1. These definitions are both general and specific to travelling waves.

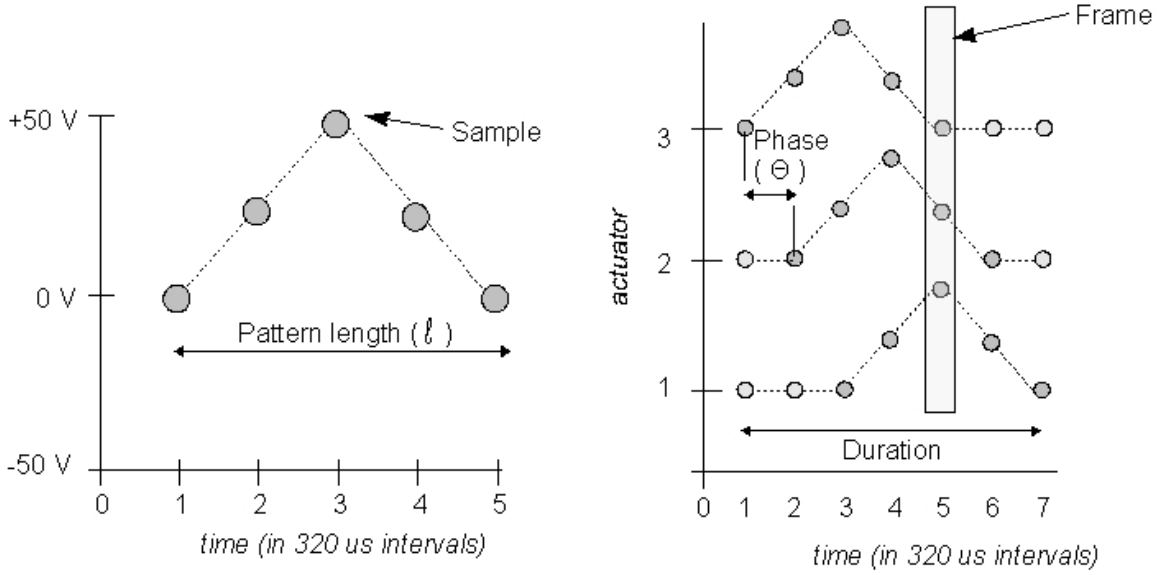

Figure 5 *Left*: A pattern, showing pattern length and a sample. *Right:* A travelling pattern, showing phase, duration, and a frame. Only 3 of 8 actuators are shown for simplicity.

Sample interval (i, measured in microseconds): an interval during which an actuator can only be a single, unchanging voltage. In this system: $i = 320$ μs, as restricted by the FPGA output rate described above.

Sample: the value of a single actuator in one sample interval. In Figure 5 *Left*, each dot represents a single sample in the pattern.

Frame: the set of values across all actuators in one sample interval. In Figure 5 *Right*, the frame shown at time 5 is made up of 3 samples from actuators 1, 2, and 3. On the THMB, a frame is composed of 8 samples from the 8 actuators.

Pattern: a set of samples on a single actuator measured over time (divided into sample intervals). Figure 5 *Left* shows a simple pattern.

Pattern length (ℓ, measured in samples): the number of samples intervals in an unstretched pattern (see below). The pattern length in Figure 5 *Left* is 5 samples.

Phase (Θ, measured in sample intervals): the amount of time between a particular sample of a pattern on one actuator and the same sample of the same pattern on the adjacent actuator. The phase in Figure 5 *Right* is one sample interval ($1i$).

Sample rate (w_s, measured in number of samples/sample interval): The number of samples per second on a single actuator. In the system described in this report, the maximum sample rate is 1/320 μs = 3125 samples/s.

Frame rate (w_f, measured in number of frames/sample interval): The number of frames per second on all actuators. In the system described in this report, the maximum sample rate is 1/320 μs = 3125 samples/s. In general, the frame rate is the same as the sample rate in this parallel configuration.

Stretch multiplier (m, measured in sample intervals per sample): The number of sample intervals over which each individual sample in a pattern is played. Stretching a pattern has the effect of reduces the frame rate by 1/m.

Duration (∂ measured in seconds): The total amount of time for a pattern to be played on all actuators. For a travelling pattern, duration is a function of the pattern length (ℓ), stretch multiplier (s), number of actuators (N) (on the THMB device N is 8), phase (Θ), and frame rate (320 μs):

$$\partial = \ell * m + (N - 1) * \Theta$$

The duration in Figure 5 *Right* in a simplified 8-actuator system is:

$$\begin{aligned} \partial &= (5 \text{ samples})*(i / \text{sample}) + (3 - 1) * 1i \\ &= 5i + 2i \\ &= 7 * 320 \text{ μs} \\ &= 2240 \text{ μs} \\ &= 2.24 \text{ ms.} \end{aligned}$$

Travelling speed: For a travelling wave, the speed at which the stimulus is perceived to travel across the display (described further in section 2.4.5.).

2.4. Theory

2.4.1. Output modes

The following modes are the way of creating a tactile stimulus on the THMB device. For the purposed of the user studies described below, only the first mode was used.

1.Travelling pattern (i.e. "movie"): a pattern is propagated from one end of the tactile display to the next. Each sample of the pattern is separated by a phase to create a frame across the actuators. The output frames are dependent only on time, not on slider position.

2.Navigable pattern: the output frames are dependent on the slider position and, optionally, the relative speed of the slider.

3.Static pattern: the output frames are not dependent on time or on slider position/speed. The output frames are produced on demand.

2.4.2. Direction

To propagate a travelling pattern "up" the display, the pattern is propagated from highest to lowest sample number, starting on the lowest actuator. To propagate a travelling pattern "down" the tactile display, the pattern is propagated from lowest to highest sample number, starting on the highest actuator. Table 1 shows the samples from the pattern shown in Figure 5 *Left* on each actuator for each time frame in the travelling pattern. It is shown in a 3-actuator system for simplicity. In this example, actuator 1 is the actuator located at the bottom (i.e. lowest position of the stack).

"Up" propagation samples			
	Actuator		
Time	1	2	3
1	5		
2	4	5	
3	3	4	5
4	2	3	4
5	1	2	3
6		1	2
7			1

"Down" propagation samples			
	Actuator		
Time	1	2	3
1			1
2		1	2
3	1	2	3
4	2	3	4
5	3	4	5
6	4	5	
7	5		

Table 1: Frames for a 3-actuator TD for travelling pattern shown in Figure 5 *Left*. Only 3 of the 8 actuators are shown for simplicity.

2.4.3. Stimulus Speed

The stimulus speed of a travelling pattern is affected by at least two parameters: the phase of the travelling pattern and the stretch multiplier of the pattern, as defined above. Figure 8 *Left* shows the effect of increasing the phase of the travelling pattern shown in Figure 5 *Left* from one 320 μ s interval to two 320 μs intervals (this halves the frame rate). A pattern is stretched by multiplying the time intervals of each sample by a constant factor. Figure 8 *Right* shows the effect of stretching the travelling pattern shown in Figure 5 *Left*. Increasing the phase has an additive effect on the duration, while stretching the pattern has a multiplicative effect.

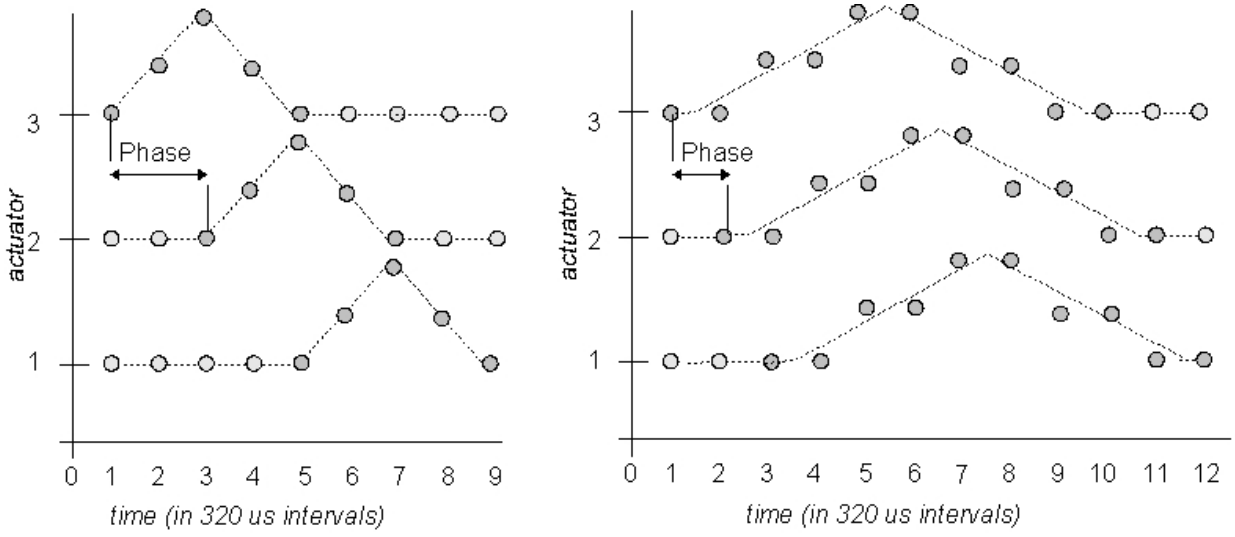

Figure 8 *Left*: the travelling pattern from Figure 5, shown with a phase of 2. *Right*: the travelling pattern from Figure 5 with a stretch multiplier of two.

Both changing the phase of a pattern and stretching a pattern may have unpredictable and unverified tactile effects because they both change the set of frames that the actuators receive. This may change the patterns of skin stretch created and therefore change the perceived shape and/or texture of the stimulus. In a pattern, a frame is read as shown in Figure 5 from top to bottom. Table 2 shows the frames of the patterns in Figures 5 and 8. The vertical frames from Figure 5 are rotated to horizontal in the table so that they can more easily be read from left to right. The values 1 through 5 indicate the sample number, as shown in the pattern in Figure 5. The character '-' indicates that the actuator voltage is set to the base value (a "neutral" actuator value that can be set in the THMB software library described below) and is not a component of the propagating pattern.

	Figure 5 *Left* (Phase 1)			Figure 8 *Left* (Phase 2)			Figure 8 *Right* (Stretch 2)		
	Actuator			Actuator			Actuator		
Time	1	2	3	1	2	3	1	2	3
1	1	-	-	1	-	-	1	-	-
2	2	1	-	2	-	-	1	1	-
3	3	2	1	3	1	-	2	1	1
4	4	3	2	4	2	-	2	2	1
5	5	4	3	5	3	1	3	2	2
6	-	5	4	-	4	2	3	3	2
7	-	-	5	-	5	3	4	3	3
8				-	-	4	4	4	3
9				-	-	5	5	4	4
10							5	5	4
11							-	5	5
12							-	-	5

Table 2: Frames for patterns appearing in Figures 5 and 8. Again, only 3 of the 8 actuators are shown for simplicity.

Notice that the frame "21-" at time 2 for the Phase 1 pattern doesn't occur in either of the other two patterns. This implies that the skin-stretch pattern created by this particular frame does not exist in the other two patterns, and so the tactile feeling of the patterns may be different.

One solution to this problem is to change the phase and the "stretch" of a pattern together by a constant factor, thus retaining the same frames but extending the time that the set of actuators spends at each of the frames. This is functionally equivalent to dividing the frame rate by the constant factor. Figure 9 shows the travelling pattern creating using this combined phase/stretch method with a constant factor of 2. Notice that the frame rate has been effectively halved.

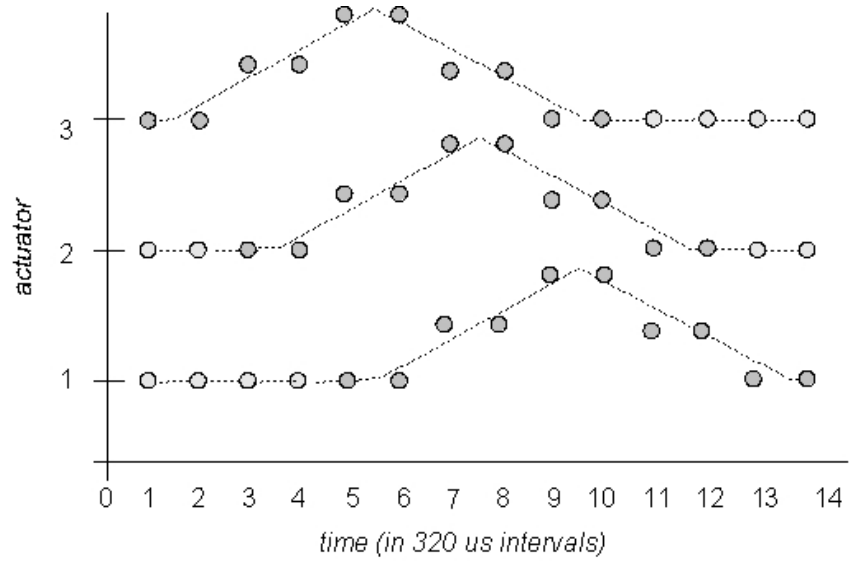

Figure 9: the travelling pattern from Figure 5 with a phase of 2 and a stretch multiplier of 2.

Table 2 shows the resulting frames in relation to the frames from the pattern in Figure 5.

	Figure 5 (Phase 1)			Figure 9 (Phase 2 + Stretch 2)		
	Actuator			Actuator		
Time	1	2	3	1	2	3
1	1	-	-	1	-	-
2	2	1	-	1	-	-
3	3	2	1	2	1	-
4	4	3	2	2	1	-
5	5	4	3	3	2	1
6	-	5	4	3	2	1
7	-	-	5	4	3	2
8				4	3	2
9				5	4	3
10				5	4	3
11				-	5	4
12				-	5	4
13				-	-	5
14				-	-	5

Table 2: Frames for patterns appearing in Figures 5 and 9

Notice that the set of frames between the original pattern in Figure 5 and the pattern in Figure 9 remain the same, but their duration doubles. This eliminates the potential skin-stretch inconsistency problem that occurs when using just phase or stretch to vary the stimulus speed of the travelling pattern. This combined phase/stretch technique is used in the Speed Study described later.

3. Software

The software designed for the THMB device is a work in progress. It is designed to be modular and to enable quick prototyping of different tactile patterns and applications based on tactile patterns for the THMB device.

Figure 10 shows an overview of the software hierarchy. With the exception of the stressd library, ACEXML, and MDS-tester, all software in the diagram was designed and written by the author. The stressd library was written by Vincent Levesque at McGill University (www.cim.mcgill.ca). ACEXML is a small and portable XML parser and can be found at www.dre.vanderbilt.edu/Doxygen/Beta/acexml. MDS-tester is a modified version of software written by Mario Enriquez in the UBC SPIN Lab (www.cs.ubc.ca/labs/spin).

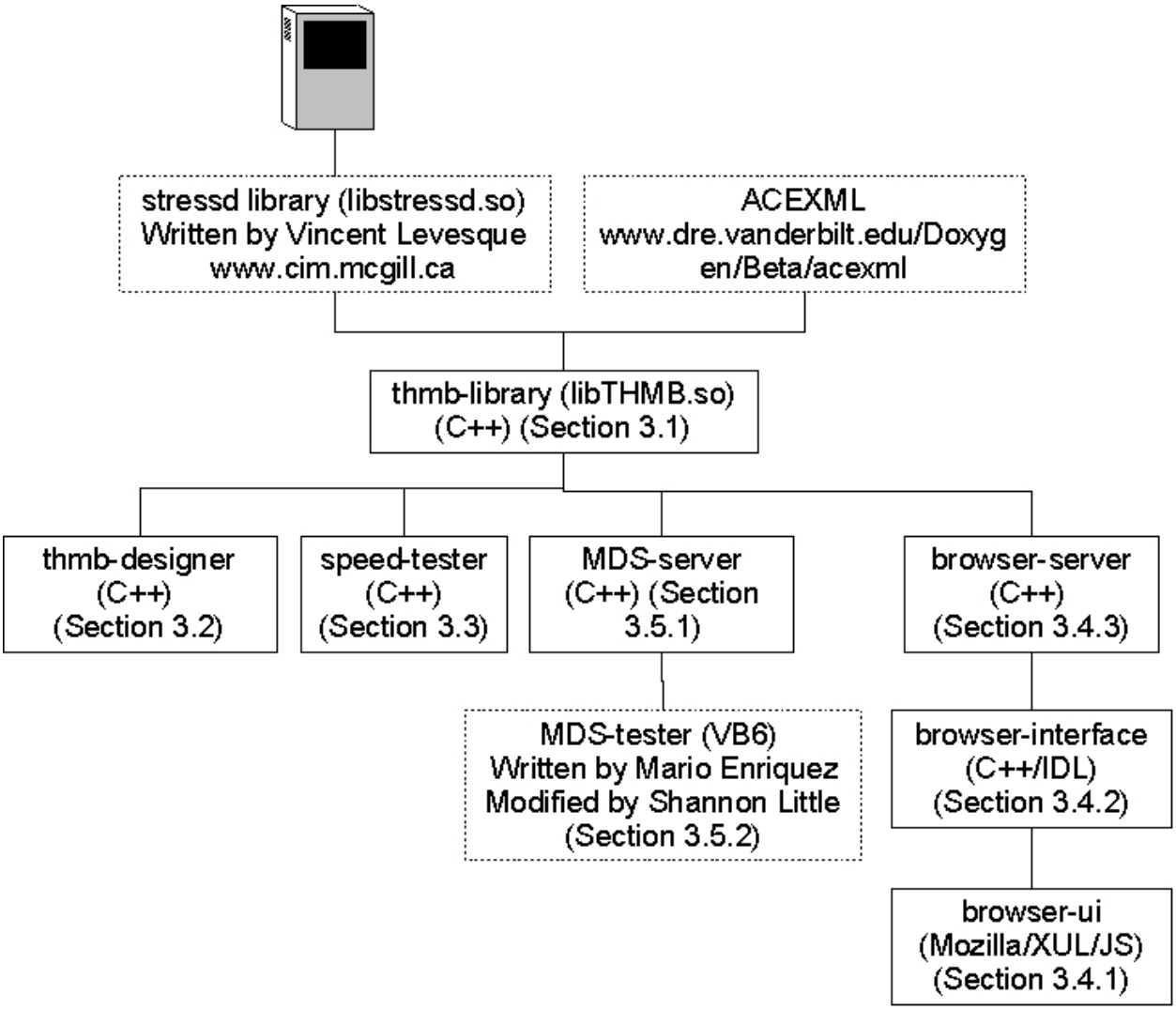

Figure 10: THMB software hierarchy

3.1. THMB Library

This library is an implementation of the output modes described above. It contains the logic to produce the raw data to create travelling patterns, navigable patterns, and static patterns and output these patterns to the device using methods in the stressd libraries. Its intention is to abstract this input/output logic from client applications and to define a finite set of data fields required to create these patterns. A client application using this library should never need to directly call any methods in the stressd library or communicate directly with the THMB device.

THMB-library logs all the output images it writes to an output file defined in DeviceUpdateThread.cpp as OUTPUT_FILE. Use this file to verify that the output you expected is the output created.

3.1.1. Files

DeviceUpdateThread.h
DeviceUpdateThread.cpp
This file contains a thread that reads slider position and click count from the THMB device and writes output images to the device. The output images are built from the patterns stored in SharedData.cpp using logic specific to the output mode of the pattern.

SharedData.h
SharedData.cpp
This file contains getter and setter functions for all data required to build and play an output pattern in any mode. It uses XMLParser.cpp to save the data in XML format that can be loaded later, enabling any pattern to be reused in any application.

XMLParser.h
XMLParser.cpp
This file uses ACEXML to parse patterns saved in XML format by SharedData.cpp and set the appropriate data values in SharedData.cpp

3.1.2. Dependencies

- Requires ACEXML (dynamic)
- Requires stressd >= version 0.1 (dynamic)
- Writes to OUTPUT_FILE (defined in DeviceUpdateThread.cpp), by default */usr/share/thmb-lib/wave-output.txt*.

3.1.3. Compilation and Installation

Run *make* as root (creates a dynamic library "libTHMB.so" in /usr/lib).

3.1.4. Running the Program

Not applicable. The library is dynamically loaded by client applications.

3.1.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/thmb-library

3.1.6. Recommendations and Future Work

The library currently implements only the “travelling pattern” output mode, but should be improved to support the other output modes (navigable pattern, static pattern) as well. It uses terminology that is inconsistent with the definitions in this report: “speed” refers to combined phase/stretch, “slider” refers to sample, “under” refers to a travelling pattern, “over” refers to a navigable pattern, and “static” refers to a static pattern. The concept of phase (independent of stretch) is not yet implemented.

All files in the library and all applications that use the library should be updated to consistently use the correct terminology.

3.2. THMB Designer

This is essentially a graphical user interface for the THMB library that allows a user to visually design a pattern and manipulate details relevant the pattern and how it is played on the device. During runtime all of the data is stored in an instance of SharedData.cpp from libTHMB.so. The program can save and load patterns to and from XML files in the formats defined in SharedData.cpp/XMLParser.cpp in libTHMB.so

3.2.1. Files

GUI.h
GUI.cpp
This implements the graphical user interface for the designer, written in gtkmm (a C++ interface to GTK+, the GIMP toolkit).

Main.cpp
This sets up the connection with the device and instantiates the GUI.

3.2.2. Dependencies

- Requires libTHMB.so
- Requires gtkmm >= 2.4
- Reads from FIRMWARE_FILE (defined in main.cpp), by default *default.sdf.*
- Reads from HARDWARE_FILE (defined in main.cpp), by default *default.sdh.*

3.2.3. Compilation and Installation

run *make*

3.2.4. Running the Program

> ./thmb-designer

3.2.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/thmb-programmer

3.2.6. Recommendations and Future Work

This program was developed in parallel with THMB library, and is likewise incomplete. Design interfaces for the “navigable pattern” and “static pattern” output modes need to be created and implemented. There also may be some old terminology used in the application that should be updated to reflect the terminology described in the “Definitions” section of this report.

3.3. Speed Tester

This is a simple application to run the Speed Study described later in this report. It consists of a simple GTK graphical interface and uses the THMB library to load and store pattern data and read and write to and from the device. This program can be used as a model for other programs that use the THMB library.

3.3.1. Files

GUI.h
GUI.cpp
This implements the graphical user interface for the designer, written in gtkmm (a C++ interface to GTK+, the GIMP toolkit). It generates random integers between 1 and 20 that determine the amount of phase/stretch used to vary the stimulus speed. It also saves experimental output to a file.

Main.cpp
This sets up the connection with the device and instantiates the GUI.

3.3.2. Dependencies

- libTHMB.so
- gtkmm >= 2.4
- Reads from input_file_dir (defined in gui.cpp), by default */home/slittle/device-lib/waves*.
- Reads from FIRMWARE_FILE (defined in main.cpp), by default *default.sdf.*
- Reads from HARDWARE_FILE (defined in main.cpp), by default *default.sdh.*
- Writes to output file defined in the command line arguments.

3.3.3. Compilation and Installation

run *make*

3.3.4. Running the Program

> ./speed-tester *output-file subject-number*

3.3.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/speed-tester

3.3.6. Recommendations and Future Work

This application is complete and tested, but can be modified for future similar studies.

3.4. Browser

This incomplete application is a simple web browser that uses the THMB device as an input and output interface. The user moves the THMB slider up and down to navigate through the hyperlinks on a web page. Moving the THMB slider up or down has two effects: it highlights links in the web page and scrolls the displayed portion of the page by focusing on the highlighted link. If the slider position matches a link position on the web page, the link is highlighted in the graphical display and a tactile stimulus is played on the tactile display; otherwise, nothing is highlighted and no tactile stimulus is played. Clicking the slider button when a link is highlighted will follow the link and load the new web page.

The indices of the links' relative positions, computed from the vertical and horizontal positions of the HTMLElement in the Document Object Model (DOM) for the webpage[2], are mapped to slider positions, resulting in a tactile mapping of the page that corresponds spatially to the visual layout of the links.

The application consists of three components: the graphical user interface, the communications interface, and the "server" component. The communication between the components is shown below in Figure 11.

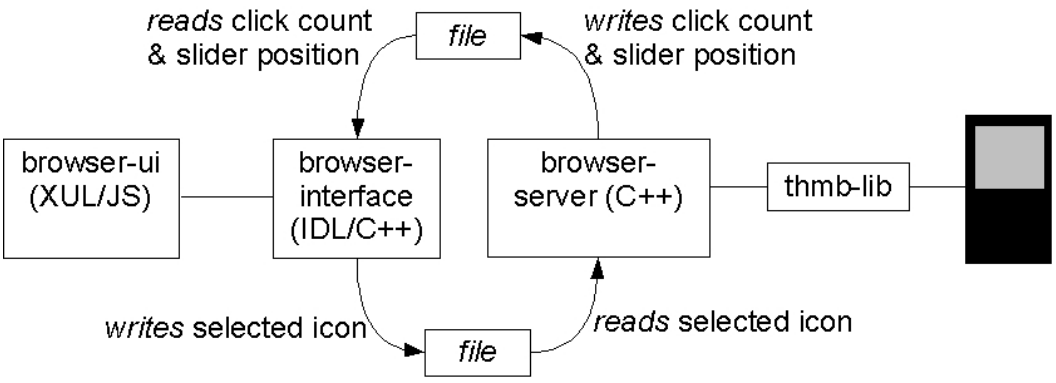

Figure 11: Browser software structure .

3.4.1. Browser Graphical User Interface (browser-ui)

The graphical user interface is implemented in XUL and JavaScript using the Mozilla framework. It runs the web browser using the Mozilla browser component (www.xulplanet.com/references/elemref/ref_browser.html), reads the current slider position from the communications interface, computes the highlighted link, and updates the UI and the communications-interface with the current highlighted link.

3.4.1.1. Files

content/browser.js

[2] See www.xulplanet.com and http://www.mozilla.org/docs/dom for information on the HTMLElement and the Mozilla DOM.

This implements the listeners for the graphical user interface components in browser.xul. It also instatiates the browser-interface and calls the methods implemented in MyComponent.cpp.

content/browser.xul
This creates the graphical user interface of the browser.

content/contents.rdf
Mozilla required file.

locale/contents.rdf
Mozilla required file.

skin/webscroller.css
Stylesheet for graphical user interface.

skin/contents.rdf
Mozilla required file.

3.4.1.2. Dependencies

- Requires Mozilla SDK (on SUSE Linux, you'll need the *mozilla* and *mozilla-devel* packages).

3.4.1.3. Compilation and Installation

1. Open installed-chrome.txt at *MOZILLA_PATH/lib/chrome*, where *MOZILLA_PATH* is the installation directory of Mozilla. This may differ on other configurations, but the default on SUSE Linux is */opt/mozilla*.
2. Add the following three lines to the end of installed-chrome.txt:
   content,install,url,file://*YOUR_FILE_PATH*/content/
   skin,install,url,file://*YOUR_FILE_PATH*/skin/
   locale,install,url,file:///*YOUR_FILE_PATH*/locale/
   Where *YOUR_FILE_PATH* is the location of the files described in the above *Files* section. Ensure that the last line in installed-chrome.txt is a newline.
3. Delete *MOZILLA_PATH/lib/chrome/chrome.rdf.*
4. Restart mozilla.
5. Run the extension as described in "Running the Program" below.

For more information, refer to: www.xulplanet.com/tutorials/xultu/packaging.html. If you need to change the application id, you'll need to change the three contents.rdf files as described in the "Quick Steps" section in the above URL.

Once you have installed the extension, there is no need to compile or re-install anything when you make a change to browser.js or browser.xul. Instead, just restart the application

3.4.1.4. Running the Program

> mozilla –chrome chrome://webscroller/contents/browser.xul

Note that browser-server must be started before browser-ui.

3.4.1.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/browser-ui

3.4.2. Browser Communications Interface (browser-interface)

The communications component is implemented in C++ and IDL (interface definition language). It is instantiated by the graphical user interface component. It provides methods for setting/getting the slider position, getting the click count, and indicating a highlighted link. It communicates with the server component via flat files ending in *.pos* for the slider position, *.click* for the click count, and *.icon* for the highlighted links (by default these files are in */usr/share/thmb-lib/browser-work*).

3.4.2.1. Files

MyComponent.cpp
Implementation of interface functions. Writes graphical interface data to a file and reads tactile interface data from a file. Both files are used by browser-server.

MyComponent.h
MyComponentModule.cpp
Module definitions file. Generated semi-automatically from IMyComponent.idl.

IMyComponent.h
Generated automatically by *xpidl* from IMyComponent.idl.

IMyComponent.idl
Interface definition file. Defines the methods and attributes of the interface.

IMyComponent.xpt
Generated automatically by *xpidl* from IMyComponent.idl.

3.4.2.2.Dependencies

- Reads from WORKING_DIR (defined in MyComponent.cpp), by default */usr/share/thmb-lib/browser-work.*

3.4.2.3. Compilation and Installation

For changes made only in MyComponent.cpp and/or MyComponentModule.cpp, run *make* as root (creates MyComponent.so in MOZILLA_PATH/lib/components/ and registers it with Mozilla).

For changes to IMyComponent.idl (i.e. to add or remove functions from the interface), refer to the "Creating the component" section of www.iosart.com/firefox/xpcom. Note that there should

be no need to install the gecko-sdk if you have installed the *mozilla-devel* package. The *xpidl* program it refers to can be found at *MOZILLA_PATH/lib*.

3.4.2.4. Running the Program

Not applicable. The component is instantiated by browser.js in the browser-ui.

3.4.2.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/browser-interface

3.4.3. Browser Server (browser-server)

The "server" component is implemented in C++. It runs a thread that checks for highlighted links from the communications interface, creating tactile output as necessary, and writes the slider position and click count to the communications interface.

3.4.3.1. Files

DataUpdateThread.h
DataUpdateThread.cpp
Reads slider position and click count from the device and writes them to a file for browser-interface. Reads the current highlighted link from a file created by browser-interface and writes corresponding output to the device.

main.cpp
Sets up the connection with the device and starts the DataUpdateThread.

3.4.3.2. Dependencies

- Requires libTHMB.so
- Reads from FIRMWARE_FILE (defined in main.cpp), by default *default.sdf.*
- Reads from HARDWARE_FILE (defined in main.cpp), by default *default.sdh.*
- Reads from WORKING_DIR (defined in DataUpdateThread.cpp), by default */usr/share/thmb-lib/browser-work*.
- Reads from LINK_ICON (defined in DataUpdateThread.cpp), by default */home/slittle/device-lib/waves/down-19-^.xml*.

3.4.3.3. Compilation and Installation

Run *make.*

3.4.3.4. Running the Program

> ./browser-server

Note that browser-server must be started before browser-ui.

3.4.3.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/browser-server

3.4.4. Recommendations and Future Work

The tactile feedback part of this application is not yet fully implemented. Other additions and improvements include the addition of tactile components other than just hyperlinks (i.e. images, text, etc.) and the improvement of the links' relative position calculation.

3.5. MDS

The MDS software was used to run the MDS Study described later in this report. The software is composed of two parts: the MDS Server and the MDS Tester.

3.5.1. MDS Server (MDS-server)

MDS-server is written in C++ and behaves similarly to browser-server. It checks a directory for files created by the MDS-tester that indicate a pattern to play and then plays the pattern using the THMB library.

3.5.1.1. Files

DataUpdateThread.h
DataUpdateThread.cpp
Reads the pattern to play from a file created by MDS-tester and writes corresponding output to the device.

main.cpp
Sets up the connection with the device and starts the DataUpdateThread.

3.5.1.2. Dependencies

- Requires libTHMB.so
- Reads from FIRMWARE_FILE (defined in main.cpp), by default *default.sdf.*
- Reads from HARDWARE_FILE (defined in main.cpp), by default *default.sdh.*
- Reads from WORKING_DIR (defined in DataUpdateThread.cpp), by default */home/slittle/device-data/waves*.
- Reads from WAVE_DIR (defined in DataUpdateThread.cpp), by default */home/slittle/device-lib/MDS-server/waves*.

3.5.1.3. Compilation and Installation

Run *make.*

3.5.1.4. Running the Program

> ./MDS-server (as root)

Note that MDS-server must be started before MDS-tester.

3.5.1.5. Location in CVS

/imager/project/spin/proj/THMB/cvsroot/src/MDS-server

3.5.2. MDS Tester

MDS-tester is a modified version of the Visual Basic 6.0 (VB6) software written by Mario Enriquez to run the MDS study explained in [1]. It is a graphical user interface consisting of a set of buttons and two to fifteen groups that the buttons can be sorted into. When one of the buttons is pressed, a file is created that indicates a pattern and phase, which is then read and played by MDS-server.

3.5.2.1. Files

The project files for the MDS Tester are located on the SPIN machine *garibaldi* on the Desktop in a folder called *MDS\THMB\MDS Test THMB Device*. These files should be placed in CVS as soon as possible (*garibaldi* was unavailable when this report was written).

3.5.2.2. Dependencies

- Requires Visual Basic 6.0

3.5.2.3. Running the Program

Start the project in Visual Basic 6.0 and press 'Play'.

Note that the MDS Server must be started first.

3.5.3. Recommendations and Future Work

This application is complete and tested, but can be modified for future similar studies.

4. User Studies

4.1. Speed Study

4.1.1. Introduction

The intention of this study was to determine the relationship between the stimulus speed of a travelling pattern and a user's accuracy in identifying the direction of the pattern at that speed. We hypothesized that as the stimulus speed of a travelling wave decreases, accuracy of direction identification increases. In the case of this study, the stimulus speed was varied using the combined phase/stretch method described in the "Stimulus Speed" section of this document.

4.1.2. Method

4.1.2.1. Participants

Two iterations of the study were performed. In total there were 8 participants, 5 male and 3 female. 6 participants participated in both iterations. 1 participant participated in only the first iteration and 1 participant participated in only the second iteration. In total, there were 7 participants in each iteration. The participants were between 20 and 40 years old and were all right handed. The participants were other students in the lab and neighbouring labs and they were not paid.

4.1.2.2. Design

The stimuli were formed by randomly selecting one of the two patterns shown in Figure 12, a direction ("up" or "down"), and a stimulus speed. The stimulus speed ranged from fast (1) to slow (20), where 1 indicates a phase/stretch of 1 and 20 indicates a phase/stretch of 20.
In total, there were 80 possible stimuli. The duration of the stimuli increased linearly with speed according to the following equation:

$$\partial = 17 * \text{speed} * 320\ \mu s$$

The two patterns used in this study are shown in Figure 12. They are both similar, with the only difference being that one begins at +50 volts and goes to –50 volts and the other does the opposite.

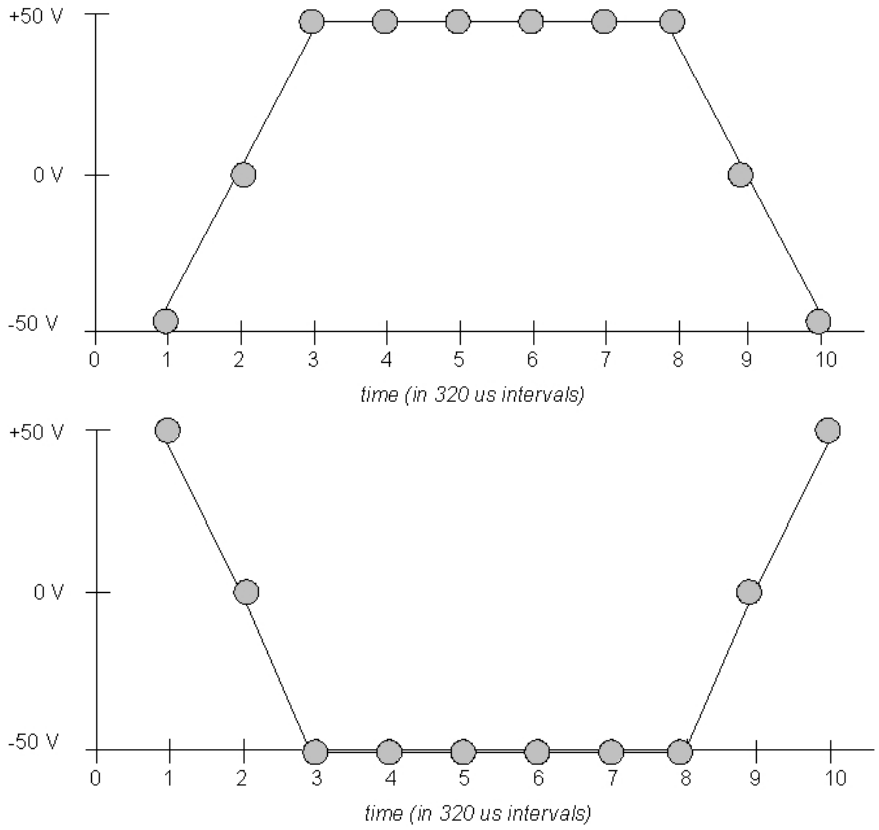

Figure 12: the two patterns used in the Speed Study

4.1.2.3. Task

Participants were presented with a simple graphical interface that asked them to feel a tactile stimulus, then to identify the direction of the stimulus ("up" or "down") and their confidence in their answer ("confident" or "guess").

In the first iteration participants were given a practice set of 5 stimuli for which they were given the correct answer ("up" or "down") after they entered their response. In the second iteration there was no practice set. Both iterations used a test set of 40 randomly selected stimuli from the set of 80 described above, for a total of 80 test stimuli per participant over the two trials (the results from the practice set was not recorded).

Each iteration of 40 stimuli took between 5 and 7 minutes with several days between iterations.

4.1.2.4. Data

The subject number, source file for the pattern, speed/phase (1 to 2), waveform ("^" or "v"), actual direction ("up" or "down"), subject's direction response ("up" or "down"), and subject's confidence ("confident" or "guess") were recorded for each trial.

4.2. MDS Study

4.2.1. Introduction

The intention of this study was to demonstrate the richness and variety of tactile stimuli possible on this novel tactile interface and to identify the structure and dependence of the multiple perceptual dimensions of tactile stimuli. We use the Multidimensional Scaling (MDS) technique described by MacLean *et al.* in [1] and adapted it to our handheld tactile display.

4.2.2. Method

4.2.2.1. Participants

We used 10 right-handed subjects (7 male, 3 female) between the ages of 19 and 31. Subjects were paid $10 for a 1-hour session.

4.2.2.2. Design

The study used a set of 30 stimuli that combined 5 travelling patterns, 2 amplitudes (50% and 100% of maximum), and 3 stimulus speeds (10, 15, 20). The stimulus speeds were produced using the combined phase/stretch method that was used the speed study. See the "Stimulus Speed" section earlier in this report for more details. All patterns were created with a stretch (independent of the phase/stretch used for stimulus speed) of 2.

Figure 13 shows the patterns used in the study and their relative pattern lengths and shape.

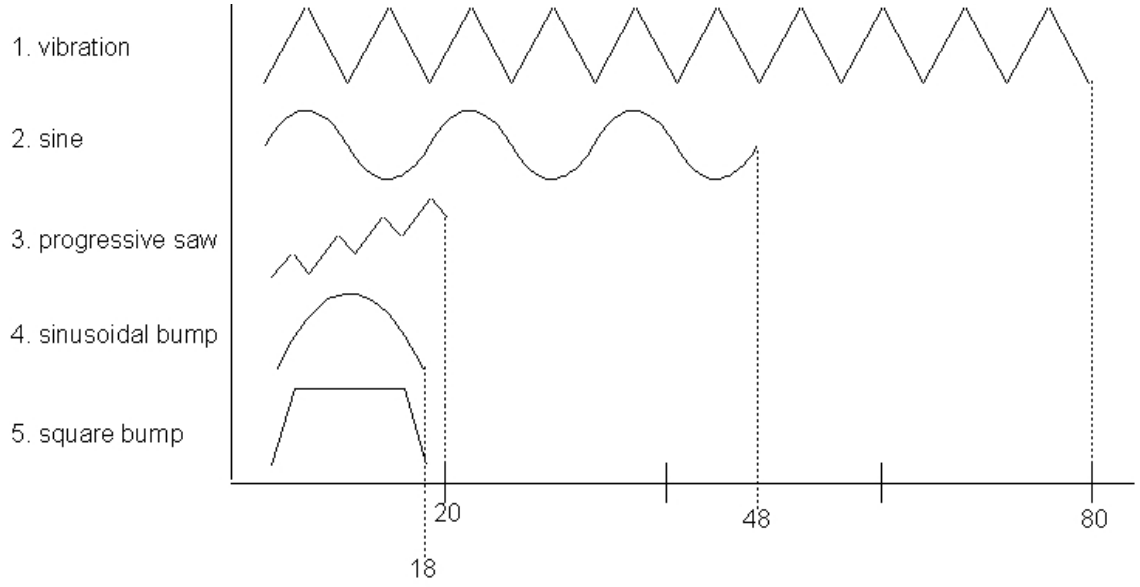

Figure 13: the patterns used in the MDS Study and their pattern lengths

We are redefining the wave shape parameter used by MacLean *et al.* in [1] to instead be a pattern, which defines both a shape and a length. The duration of a pattern (see Table 4 for the durations of each of the patterns at each phase/stretch), the tactile feeling (shape) of a pattern, and the stimulus speed of a pattern may all be independent dimensions identified by MDS.

Pattern	Pattern length	Duration at speed 10 (seconds)	Duration at speed 15 (seconds)	Duration at speed 20 (seconds)
vibration	80	0.28	0.42	0.56
sine	48	0.18	0.26	0.35
progressive saw	20	0.09	0.13	0.17
sinusoidal bump	18	0.08	0.12	0.16
square bump	18	0.08	0.12	0.16

Table 4: Durations of each of the patterns show in Figure 8 at each phase/stretch value.

Using patterns of variable length allows us to separate duration from speed and consider the following questions in our data analysis:

Duration: Does a short pattern played slowly feel similar to a long pattern played quickly? For example, does a vibration played at speed 10 (0.28 seconds) similar to a sinusoidal bump played at speed 20 (0.35 seconds)?

Tactile feeling (shape): Does a given pattern at any speed feel similar?

Apparent speed: Do two patterns feel similar only when played at the same speed? For example, are sinusoidal bumps and square bumps similar only when both are played at speed 10, 15, or 20?

Another dimension that may differentiate the stimuli is amplitude. It is possible that there are other perceptual dimensions; MDS will help us identify how many are needed to best design tactile icons that are differentiable and individually salient.

4.2.2.3. Tasks

The participants were told to hold the device in their left hand with their left thumb resting lightly on the tactile display. They were told not to press down hard with their thumb or rub their thumb over the display. They were told to take a break whenever needed and that they were required to take at least a one-minute break after each trial.

Participants were presented with a group of 30 buttons that were graphical representations of the 30 tactile sensations. They could play a stimulus by clicking the left mouse button on one of the 30 buttons. The stimuli could be played as many times as needed.

The buttons could be moved to another location of the screen individually by clicking the right mouse button on the button (picking it up) and right clicking again to drop the button. The buttons could only be dropped in one of two to fifteen boxes (groups) on the screen. The buttons could be moved between these boxes as many times as desired.

The study consisted of 5 sorting tasks each using the same set of stimuli. Participants were told that they could use any criteria they wanted to sort the tactile stimuli. They were told to take at least a one-minute break between each task.

Task 1 asked participants to sort the stimuli into as many (from 2 to 15) boxes that they wanted. By default they initially received 2 boxes for sorting and could add or remove boxes by clicking buttons labelled "+" and "-". The boxes initially had blank labels. Participants were required to label each box with a description that would help them to identify the boxes as they sorted the stimuli.

Tasks 2-5 were similar to Task 1 except participants could not choose the number of groups. Rather, the number of groups was randomly selected from a predetermined set of 3, 6, 9, 12, and 15 groups. The value in this predetermined set that was closest to the number of groups the participant chose freely in the first task was removed from the set. For example, if the participant chose 5 groups in Task 1, then the remaining four groups would be (in random order) 3, 9, 12, and 15.

4.2.2.4. Data

A similarity matrix was calculated and converted to a dissimilarity matrix in the same way described in [1]. The similarity matrix, dissimilarity matrix, stimulus groupings for each of the five tasks, participants' group labels was saved.

5. Conclusions and Future Work

The intention of the work described in this technical report was to determine the basic configuration and perceptual limits of the THMB device, develop terminology and theory about the output of the device, design libraries that implemented this terminology, implement applications that used these libraries to interact with the THMB device, and design user studies that would relate these theories and output to human perceptual limitations and capabilities.

Further work is necessary to improve our understanding of the theory of the formation of patterns on the device, what kind of skin stretch patterns these patterns create, and the perceptual effect of the different properties of patterns.

The libraries and software described should, for the most part, be considered as still in development. It has been designed and written to allow for easy future implementation and improvements. It is assumed that these improvements will be done for the software to continue to be useful.

The data analysis on the user studies described is not yet complete; once it is, the results should be related to the theory described in this report. This work is currently in preparation for two conference paper submissions.

6. Acknowledgements

The author would like to thank Professor Faith Ellen Fich, the rest of CRA-W (Committee on the Status of Women in Computing Research) and NSERC (National Sciences and Engineering Research Council) for their grants that supported this research, Professor Karon MacLean for the opportunity to participate in this research, and Joseph Luk, Jerome Pasquero, and Vincent Levesque for their indispensable help and support.

7. References

[1] MacLean, K., Enriquez, M., “Perceptual Design of Haptic Icons,” in Proc. of EuroHaptics 2003, Dublin, UK, July 2003.

[2] Poupyrev, I., Maruyama, S., Rekimoto, J., “Ambient Touch: Designing Tactile Interfaces for Handheld Devices,” in Proc. of UIST 2002, Paris, France, October 2002.

[3] V. Levesque, J. Pasquero, H. V., and M. Legault, “Display of virtual Braille dots by lateral skin deformation: Feasibility study.” ACM Transactions on Applied Perception, vol.2, no. 2, pp. 132-149, 2005.

[4] http://www.piezo.com/tech2intropiezotrans.html