

# Tactile Editor

*A Prototyping Tool to  
Design and Test  
Vibrotactile Patterns*

Diploma Thesis at the  
Media Computing Group  
Prof. Dr. Jan Borchers  
Computer Science Department  
RWTH Aachen University



by  
Markus Jonas

Thesis advisor:  
Prof. Dr. Jan Borchers

Second examiner:  
Prof. Dr. Ulrik Schroeder

Registration date: Nov 30th, 2007  
Submission date: May 30th, 2008



# Contents

<b>Abstract</b>	<b>xv</b>
<b>Überblick</b>	<b>xvii</b>
<b>Acknowledgements</b>	<b>xix</b>
<b>Conventions</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Haptics . . . . .	1
1.1.1 Why Is Haptics Important? . . . . .	2
1.1.2 Haptic Feedback . . . . .	3
1.2 The DIA-cycle . . . . .	3
1.3 Thesis Structure . . . . .	4
<b>2 Background</b>	<b>5</b>
2.1 The Somatic Sensory System . . . . .	5
2.2 The Sense of Touch . . . . .	6

---

2.2.1	The Cutaneous System and the Tactile Sense . . . . .	7
	Sensory Physiology . . . . .	7
2.3	Perception of Vibrotactile Stimuli . . . . .	8
2.3.1	Temporal Order . . . . .	10
2.3.2	Vibrotactile Patterns . . . . .	10
2.3.3	Tactile Spacial Resolution and Tactile Acuity . . . . .	11
2.3.4	Tactile Illusions . . . . .	14
2.4	Summary . . . . .	16
2.4.1	Haptic Devices . . . . .	16
2.4.2	Tactile Stimulators and Vibrotactile Technology . . . . .	17
<b>3</b>	<b>Related Work</b>	<b>19</b>
3.1	Haptic Feedback . . . . .	20
3.1.1	Perception of Haptic Feedback . . . . .	22
3.2	Editors to Specify Haptic Icons . . . . .	22
3.2.1	Hapticon Designer, Editor and Displayer (HDED) . . . . .	23
3.2.2	VITAL . . . . .	24
3.2.3	Immersion Studio . . . . .	25
3.2.4	VibeTonz . . . . .	27
3.2.5	Skinscape . . . . .	27
3.3	HCI Prototyping Toolkits . . . . .	28

---

3.3.1	iStuff . . . . .	28
3.3.2	iStuff Mobile . . . . .	29
3.3.3	Suede . . . . .	30
3.3.4	d.tools . . . . .	31
3.3.5	Exemplar . . . . .	32
3.3.6	Discussion . . . . .	33
3.4	Track-based Editors . . . . .	33
3.4.1	Music Sequencing Software . . . . .	34
3.4.2	Video Editing Software . . . . .	34
3.5	Toolkit Evaluation . . . . .	35
3.6	Summary . . . . .	36
<b>4</b>	<b>Requirements</b>	<b>39</b>
4.1	Overview . . . . .	39
4.2	Problem Definition and System Objective . . . . .	40
4.3	Tactile Editor . . . . .	40
4.4	Functionality . . . . .	41
4.4.1	Necessary Properties . . . . .	42
4.4.2	Desirable Properties . . . . .	43
4.4.3	Conceivable Requirements . . . . .	44
4.5	User Characterization . . . . .	44
4.6	Development and Runtime Environment . . . . .	45
4.7	Supported Hardware Platforms . . . . .	45

---

4.7.1	Arduino . . . . .	46
4.7.2	Make Controller Kit . . . . .	47
<b>5</b>	<b>Conceptual Design</b>	<b>51</b>
5.1	System Architecture . . . . .	51
5.2	Concepts of Interaction . . . . .	54
<b>6</b>	<b>Implementation Details</b>	<b>57</b>
6.1	Overview . . . . .	57
6.2	Model-View-Controller Paradigm . . . . .	58
6.3	Class Structure . . . . .	59
6.4	Tactile Editor GUI . . . . .	62
6.4.1	The Canvas . . . . .	63
6.4.2	The Manage Devices Drawer . . . . .	66
6.4.3	The Status Bar . . . . .	69
6.4.4	The Menu Bar . . . . .	70
6.5	Undo Functionality . . . . .	72
6.6	Tactile Editor Help System . . . . .	72
6.6.1	Tooltips . . . . .	72
6.6.2	Help Book . . . . .	74
6.6.3	Keyboard Shortcuts . . . . .	75
6.7	Communication with Hardware Platforms . . . . .	76
6.7.1	Arduino . . . . .	76

---

6.7.2	Make Controller Kit . . . . .	78
<b>7</b>	<b>Evaluation</b>	<b>81</b>
7.1	User Testing . . . . .	82
7.1.1	Setup . . . . .	83
7.1.2	Users . . . . .	84
7.1.3	Tasks . . . . .	84
7.1.4	Results . . . . .	86
<b>8</b>	<b>Summary and Future Work</b>	<b>93</b>
8.1	Summary and Contributions . . . . .	93
8.2	Future Work . . . . .	94
8.2.1	Export Function for Patterns . . . . .	94
8.2.2	Communication with Other Tools . . . . .	95
8.2.3	Control Points . . . . .	95
8.2.4	Perception of Tactile Patterns . . . . .	96
<b>A</b>	<b>Basic Data Structures for Tactile Editor</b>	<b>97</b>
<b>B</b>	<b>List of Keyboard Shortcuts</b>	<b>99</b>
<b>C</b>	<b>Questionnaire and Task List Tactile Editor User Study</b>	<b>101</b>
<b>D</b>	<b>GUI Development</b>	<b>107</b>
	<b>Bibliography</b>	<b>111</b>

**Index**

**117**



# List of Figures

1.1	DIA-cycle . . . . .	3
2.1	Mechanoreceptors . . . . .	8
2.2	Vibrotactile patterns . . . . .	11
2.3	Tactile acuity . . . . .	13
2.4	Sensory humunculus . . . . .	14
2.5	Sensory saltation temporal . . . . .	15
2.6	Sensory saltation location . . . . .	15
2.7	Phantom haptic devices . . . . .	17
2.8	Vibrotactile stimulators . . . . .	18
3.1	Haptic feedback system for robotic surgery .	21
3.2	Haptic feedback devices . . . . .	21
3.3	Hapticon Editor . . . . .	23
3.4	Software interface for VITAL . . . . .	25
3.5	Immersion Studio GUI . . . . .	26
3.6	VibeTonez Studio GUI . . . . .	27

3.7	Quartz Composer GUI . . . . .	28
3.8	Quartz Composer GUI . . . . .	29
3.9	Suede GUI . . . . .	30
3.10	d.tools editor GUI . . . . .	31
3.11	Exemplar GUI . . . . .	32
3.12	GarageBand GUI . . . . .	34
3.13	Jahshaka GUI . . . . .	35
3.14	Existing systems . . . . .	37
4.1	Tracks . . . . .	41
4.2	Example motor vibrations . . . . .	42
4.3	Pulse Width Modulation . . . . .	47
4.4	Arduino box . . . . .	49
5.1	System architecture . . . . .	52
5.2	Feedback - tooltips . . . . .	56
6.1	MVC diagram Cocoa . . . . .	59
6.2	UML class diagram . . . . .	60
6.3	Tactile Editor GUI . . . . .	63
6.4	Canvas . . . . .	64
6.5	Creating patterns . . . . .	65
6.6	Editing patterns . . . . .	66
6.7	"Manage Devices" drawer . . . . .	67

---

6.8	Sensor window . . . . .	68
6.9	Status bar . . . . .	70
6.10	Menu bar . . . . .	70
6.11	Undo . . . . .	72
6.12	Tooltips . . . . .	73
6.13	Tactile Editor Help . . . . .	75
6.14	Arduino protocols . . . . .	77
7.1	Changes made to GUI . . . . .	82
7.2	Setup user test . . . . .	83
7.3	User working with Tactile Editor . . . . .	85
7.4	Statistical results task completion . . . . .	87
7.5	Results task completion . . . . .	87
7.6	Results toolkit evaluation . . . . .	88
7.7	Results GUI evaluation . . . . .	88
7.8	Statistical results toolkit evaluation . . . . .	89
7.9	Patterns task 3 . . . . .	90
8.1	Patterns using polygons . . . . .	96
C.1	Questionnaire user study - page 1 . . . . .	102
C.2	Questionnaire user study - page 2 . . . . .	103
C.3	Questionnaire user study - page 3 . . . . .	104
C.4	Task list user test . . . . .	105

D.1	Development GUI 1/2 . . . . .	108
D.2	Development GUI 2/2 . . . . .	109
D.3	Tactile Editor GUI - final design . . . . .	110

# List of Tables

2.1	Definitions of terminology . . . . .	6
2.2	Characteristics of the four types of mechanoreceptors responsible for perceiving tactile sensations . . . . .	9
2.3	Spacial resolution evaluated with two-point thresholds - from [Kandel et al., 2000] and [Grunwald and Beyer, 2001] . . . . .	12
2.4	Anatomical and perceptive characteristics relevant for evaluating vibrotactile feedback	16
6.1	Protocol for Arduino and MakeController . .	79
B.1	List of keyboard shortcuts . . . . .	100



# Abstract

The haptic sense is one of the most important human senses: it provides a valuable source of information for the human. In situations where our auditory or visual senses are occupied, touch can become a particularly important communication channel.

While visual and auditory feedback is nowadays standard in most human-computer-interfaces, the use of haptic feedback has only been recognized and appreciated recently. Devices such as haptic mice or haptic joysticks provide additional haptic feedback; primarily this feedback is restricted to computer games. Haptic feedback is still widely a research area.

Therefore we present a prototyping tool to design and test vibrotactile patterns in this work. Tactile patterns can convey important information in a subtle way, where auditory or visual messages are inadequate. Tactile Editor is a software tool that provides a track-based graphical user interface and enables end-users to design patterns using direct manipulation techniques. The program uses an evident interaction metaphor representing motors, which produce vibrations, as rectangles on tracks and provides common drag and drop techniques that allow end-users to work effectively with the editor.

For testing vibrotactile feedback, users can send patterns from the editor to supported hardware devices that activate vibration motors to render these patterns. Tactile Editor explicitly supports Arduino and MakeController devices - two well-known and wide-spread hardware toolkits for creating physical prototypes.

Tactile Editor was tested in a first-use study at the Media Space Lab of the Media Computing Group at RWTH Aachen University. Evaluation and analysis of these tests show that Tactile Editor can be used effectively to design vibrotactile patterns and to test those patterns with vibration motors.





# Überblick

Die haptische Wahrnehmung ist mit Sicherheit einer der wichtigsten menschlichen Sinne; der Tastsinn ist eine der wichtigsten Informationsquellen für den Menschen. Besonders in Situationen, in denen visuelle und auditive Wahrnehmung beschäftigt sind, kann dem haptische Sinn eine große Bedeutung zukommen.

Während visuelles und auditives Feedback heutzutage Standards in Mensch-Maschine-Schnittstellen sind, wird die Bedeutung von haptischem Feedback erst nach und nach erkannt. Geräte wie haptische Mäuse oder haptische Joysticks bieten dem Nutzer zusätzliches haptisches Feedback; hauptsächlich ist dieses aber bisher auf Computerspiele beschränkt. Haptisches Feedback bleibt nach wie vor größtenteils ein Forschungsgebiet.

Deshalb stellen wir in dieser Arbeit ein Prototyping-Tool vor, das es Benutzern ermöglicht, vibro-taktile Feedback-Patterns zu erstellen und zu testen. Taktile Patterns können in Situationen, wo auditives oder visuelles Feedback unzureichend oder unangemessen ist, unaufdringlich Informationen vermitteln. Tactile Editor ist eine Software, die auf einer track-basierten grafischen Benutzeroberfläche aufbaut und es dem Nutzer erlaubt, Patterns mittels direkter Manipulation zu erstellen. Das Programm bietet seinen Nutzern dazu eine eingängige Interaktionsmetapher, die Vibrationen erzeugende Motoren als Rechtecke auf Tracks darstellt und bekannte Drag-and-Drop-Techniken zur Verfügung stellt, damit auch Erstbenutzer erfolgreich mit dem Editor arbeiten können.

Die erstellten Patterns können zum Testen des vibro-taktilen Feedbacks vom Editor an unterstützte Hardwaregeräte geschickt werden, die diese Patterns dann über verbundene Vibrationsmotoren rendern. Tactile Editor unterstützt explizit die Arduino und MakeController Plattformen - zwei bekannte und weit verbreitete Hardware Toolkits für physikalische Prototypen.

Tactile Editor wurde in einer Erstbenutzerstudie im Media Space der Media Computing Group der RWTH Aachen getestet. Bewertung und Analyse der Tests zeigen, dass Tactile Editor erfolgreich eingesetzt werden kann, um vibro-taktile Patterns zu erstellen und dieses Patterns zu testen.



# Acknowledgements

This thesis would never have been possible without the help and support of many people, all of whom I owe my gratitude.

First of all, I want to thank Prof. Dr. Jan Borchers for introducing me to the field of HCI, for teaching a different perspective on many issues, and for always keeping a collegial touch around the department.

Many thanks to Daniel Spelmezan, my thesis advisor, for always taking the time for meetings, suggestions, and constructive criticism. The countless reviews of both the software and the draft of this thesis were very helpful and often guided me towards the right direction.

Thanks again to everyone, who took part in the user studies and helped me gain valuable feedback for improving Tactile Editor.

Special thanks go to Christina, who was always there for me, especially at times, when I needed her emotional support.

Finally I want to thank my family, who made it possible for me to focus on my studies. They always supported me both in my private and my educational life.

Thank you!



# Conventions

Throughout this thesis, we use the following conventions:

The plural “we” will be used throughout this thesis instead of the singular “I”, even when referring to work that was primarily or solely done by the author.

Unidentified third persons are always described in male form. This is only done for purposes of readability.

Definitions of technical terms or short excursus are set off in coloured boxes.

Source code and implementation symbols are typeset using a monospace font.

The whole thesis is written in American English.



# Chapter 1

## Introduction

*“Everything should be made as simple as possible, but not one bit simpler.”*

— *Albert Einstein, US (German-born) physicist*  
(1879 - 1955)

Our ability to interact with the environment by touching objects or receiving information through objects touching us, is one of the fundamental human senses.

### 1.1 Haptics

In psychology and physiology, the word *haptic* refers to the ability to experience the environment through active exploration, as when palpating an object in order to experience its physical shape and material [Haptics, 2008].

*Haptics* is commonly used today to refer to the study of touch in real and virtual environments. This includes the development of engineering systems to (re-)create virtual haptic environments and is also known as *computer haptics*.

Haptics: Study of touch in real and virtual environments.

As today's interfaces get more and more complex,

Sensory overload is increasing.

sensory overload is becoming a common problem, especially in non-desktop environments and for portable devices. Therefore the haptic sense can absorb some of the load that auditory and visual senses place on the user of contemporary user interfaces.

Contributions

In this thesis, we present a toolkit to design and test vibrotactile feedback. Our goal is to provide users with a simple, effective, and intuitive way of achieving this - without the need for extensive soldering, wiring, or programming. Therefore, we aim to contribute the following:

- A toolkit that enables testing vibrotactile feedback.
- A toolkit that supports various vibrotactile devices that can easily be attached to various parts of the body for rapid prototyping.
- Tactile Editor, a graphical user interface for the toolkit, that uses well-known interaction metaphors such as a timeline-based canvas and a drag-and-drop interface.

Touch is the most underrated human sense.

### 1.1.1 Why Is Haptics Important?

Touch and touch-related capabilities such as kinesthesia are probably the most underrated human senses and abilities, as most of those capabilities operate effortlessly and without our conscious awareness. However those capabilities are critical for normal human functioning:

Loss of touch can have severe consequences.

Without our sense of touch, it would be virtually impossible to walk, stand, or even sit upright and we could not skillfully handle objects such as tools. Robles-De-Le-Torre [2006] cites two cases, where patients lost their sense of touch on a permanent basis and shows which extensive consequences this loss had.



### 1.1.2 Haptic Feedback

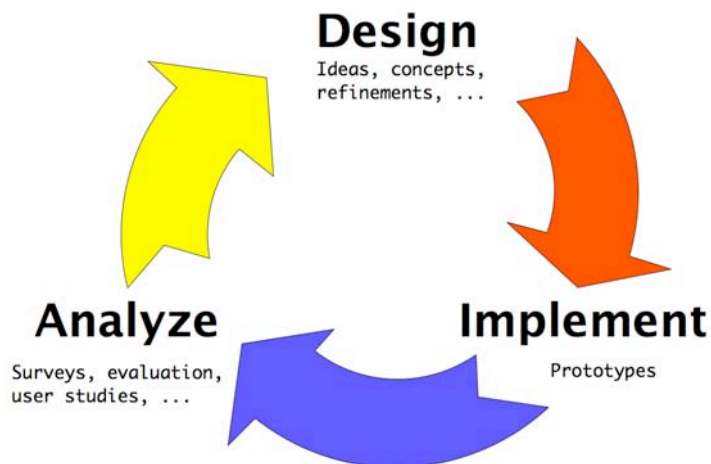
Haptic Feedback has only recently been recognized as an important communication channel in user interfaces. The term "haptic feedback" includes tactile as well as kinesthetic or force feedback. These two types of haptic feedback refer to the sensory subsystem they activate: the cutaneous or the kinesthetic sense.

In this work, we will focus on tactile feedback as the toolkit developed in this thesis stimulates a human's tactile perception.

Haptic feedback includes tactile and force feedback.

## 1.2 The DIA-cycle

When designing our tool, we followed an iterative design approach, the *DIA-cycle*, that consists of three phases within each iteration: *design*, *implement*, and *analyze*, as shown in figure 1.1



**Figure 1.1:** The DIA-cycle: one iteration consisting of design, implement, and analyze

Iterative design approaches substantially improve the usability of a system as developers can react to user demands gathered from user testing in the analysis phases.

The DIA-cycle usually starts with an idea in the design phase, a user survey as a first analysis, and a simple storyboard as the initial implementation. During each iteration of the cycle, the artifacts get more and more refined.

DIA-cycle adapted to develop Tactile Editor

We developed Tactile Editor in several DIA iterations - starting with a very basic paper prototype before gradually refining our system by adding, removing or modifying certain features of our design. Appendix D shows, how Tactile Editor's graphical user interface evolved over the different DIA-cycle iterations.

### 1.3 Thesis Structure

This thesis is organized as follows

- The chapter *Background* gives background information on physiological facts that are relevant to our research into vibrotactile feedback patterns.
- In *Related work*, we name similar research work on vibrotactile patterns, track-based editors, and HCI toolkits that relate to our work in this thesis.
- The chapter *Requirements* outlines the requirements for our system to design and test vibrotactile patterns.
- In *Conceptual design*, we present our concept for Tactile Editor.
- The chapter *Implementation details* describes the code for the tool and how it was implemented.
- *Evaluation* elaborates on the process of evaluating Tactile Editor and making changes based on the feedback gathered from users working with the tool.
- In *Summary and future work*, we summarize our contributions and give an outlook on improving the tool in the future.

## Chapter 2

# Background

*“If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas.”*

—George Bernard Shaw, Irish literary Critic,  
Playwright and Essayist, 1856-1950

### 2.1 The Somatic Sensory System

For designing and testing vibrotactile feedback, it is essential to understand the principles of how our sensory system perceives and processes vibration stimuli. Therefore we will only focus on one of the five senses (sight, hearing, taste, smell, touch) in this work: the sense of touch.

Five human senses

The human somatic sensory system includes those parts of the nervous system involved in receiving and processing information about stimuli on the body surface and inner structures such as muscles and joints. It is responsible for the perception by transmitting information of physical stimuli conveyed by receptors within the body.

Modern studies of sensation show that the sensory

Humans can perceive four attributes of a stimulus.

system processes four elementary attributes of a stimulus: modality, intensity, duration, and location of a stimulus [Kandel et al., 2000]. We will discuss each of those in conjunction with perceiving tactile feedback below.

## 2.2 The Sense of Touch

Touch can be divided into kinesthetic and cutaneous perception.

Touch can be defined as "...the sensation evoked by mechanical, thermal, chemical, or electrical stimulation of the skin and body" [Mazzone, 2004]. Our sense of touch covers two main sensory systems: the kinesthetic and the cutaneous (or tactile) system.

Kinesthetic perception constitutes awareness of one's body state - the ability to sense the position and movement of one's limbs - and covers all sensations originating in muscles, tendons, and joints.

The cutaneous system however responds to all sensations applied to the surface of the skin, such as pressure, pain, or temperature. In particular, the tactile sense refers to the sensation of pressure, rather than temperature or pain. For an overview of these definitions see table 2.1 (adapted from [Oakley et al., 2000]).

Overview: haptic sensations

Term	Definition
Haptic	Relating to the sense of touch.
Kinesthetic	Meaning the feeling of motion. Relating to sensations originating in muscles, tendons and joints.
Cutaneous	Pertaining to the skin itself or the skin as a sense organ. Includes sensation of pressure, temperature, and pain.
Tactile	Pertaining to the cutaneous sense but more specifically the sensation of pressure rather than temperature or pain.

**Table 2.1:** Definitions of terminology

As the kinesthetic system is not involved in processing vibrotactile stimuli, we will not discuss it any further in this work and concentrate on the tactile sense for the remainder of this chapter instead. A detailed discussion of kinesthetic perception can be found in [Goldstein, 2002].

We focus on the tactile sense in this work.

### 2.2.1 The Cutaneous System and the Tactile Sense

The cutaneous system reacts to stimuli pertaining the surface of the skin, including pressure, vibration, thermal, electrical, and pain stimuli. We will focus on skin deformation caused by pressure and vibration here, as only these address the tactile sense, and refer to them as vibrotactile stimuli.

Tactile sense reacts to temperature, vibration, pain, and pressure.

#### Sensory Physiology

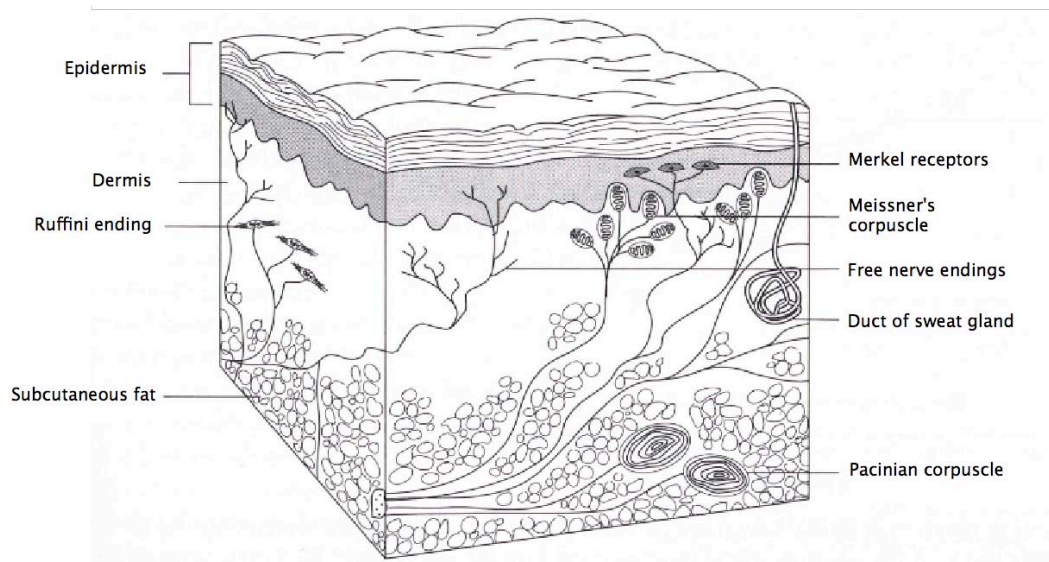
Sensor physiology examines physiological and neural consequences of a stimulus, both how it is transduced by receptors and processed by the brain. We will not go into detail about stimulus procession and rather focus on the various types of receptors relevant to the sense of touch here.

Different classes of receptors are responsible for different touch sensations: *mechanoreceptors* for tactile sensations, *muscle afferent fibers* for limb movements, *nociceptors* for pain, and *thermal receptors* for temperature sensations. In the following we will concentrate on mechanoreceptors as those are pertinent to detecting vibrotactile stimuli.

Mechanoreceptors are responsible for tactile sensations.

Four types of mechanoreceptors responsible for tactile perception are located in and between the two skin layers Epidermis (outer) and Dermis (inner), as well as below those in the hypodermis (see figure 2.1): Meissner Corpuscles, Merkel's Disks, Pacinian Corpuscles, and Ruffini Corpuscles.

Those types can be divided into two groups, based on their response to skin deformation: Rapidly adapting receptors



**Figure 2.1:** Location of the mechanoreceptors - cross section of glabrous skin; taken from [Goldstein, 2002]

Fast and slow receptors respond to different tactile stimuli.

Pacian corpuscles are most relevant for tactile feedback.

(Meissner and Pacinian) respond quickly to a stimulus onset (or offset), but are not activated during sustained skin deformation. Slowly adapting mechanoreceptors (Merkel's and Ruffini) however respond continuously to a persistent stimulus. Mechanoreceptors can further be categorized according to the sizes of their respective reception fields, which can vary from 1 or 2 millimeters up to 45 or 60 millimeters. All four types of mechanoreceptors and their properties are summarized in table 2.2.

Most relevant for our research are the Pacinian corpuscles as they are fast-responding receptors responding to vibration stimuli. In addition they offer a wide spatial range and detect a broad range of vibrations of up to one kilohertz.

### 2.3 Perception of Vibrotactile Stimuli

Modality, intensity, duration, and location define a stimuli.

To adequately understand the effect of tactile sensations, we have to take a closer look at the four attributes the tactile sense extracts from a stimulus: modality, intensity,

Receptors	Meissner Corpuscles	Merkel's Disks	Pacinian Corpuscles	Ruffini Corpuscles
Location	Dermis	Epidermis	Dermis	Epidermis, Dermis
Adaption	Rapid	Slow	Rapid	Slow
Spatial range	Small 12mm	Small 12mm	Large 100mm	Large 60mm
Function	Movement, Velocity	Vibrations, Pressure	Vibrations, Pressure	Pressure, Skin shear Thermal changes
Frequency	20-100 Hz	0-10 Hz	100 Hz-1 kHz	0-10 Hz

**Table 2.2:** Characteristics of the four types of mechanoreceptors responsible for perceiving tactile sensations

duration, and location.

### Modality

The modality identifies the type of stimulus presented. As mentioned above, touch (as one of the five main senses) can be divided into distinct submodalities: cutaneous, proprioceptive, pain, and thermal sense.

### Intensity

The intensity of a vibrotactile stimulus is defined by the strength or magnitude of its vibration. Intensity is measured in Decibel relatively to a detection threshold. This threshold is adaptable and depends on a variety of parameters such as actuator used, location of the stimulation, person, or frequency of the vibration.

Geldard [1960] discovered that while we can discriminate 15 levels of intensity when receiving vibrotactile stimuli, no more than three intensity levels can be identified absolutely.

We can identify three levels of stimulus intensity absolutely.

### Duration

The term duration refers to the length of the tactile stimulus - from onset to the termination of the stimulus

- and is generally measured in milliseconds. There exist various reference values for duration and timings of vibrotactile stimuli; more on this in the section "Temporal Order" below.

### Location

Our ability to locate the site of a vibrotactile stimulus is important. The accuracy with which we can detect the location of a stimulus highly depends on various parameters as we will outline in the section "Tactile Spatial Resolution and Tactile Acuity" below.

### 2.3.1 Temporal Order

Our ability to distinguish two stimuli depends on the time between those.

When our haptic system is given a sequence of tactile stimuli on spatially different parts of the body, our ability to distinguish the order of presentation depends on the length of the interval between two stimuli.

Hirsh and Sherrick [1961] found out that the threshold for distinguishing two brief stimuli is about 20 ms, but increases significantly with the number of stimuli we receive. In order to correctly identify a temporal sequence of five or six stimuli, the intervals between the individual stimuli may have to be as long as 500 ms.

Stimuli duration between 0.1 and two seconds is reasonable.

It is important to ensure that the individual vibrotactile stimuli are within a reasonable range of durations, which Geldard ([Geldard, 1960]) found to be roughly between 0.1 and two seconds. Impulses shorter than 0.1 seconds may feel like unwanted pokes, while stimuli that lasted longer than the upper limit of two seconds lead to delays resulting in very slow communication.

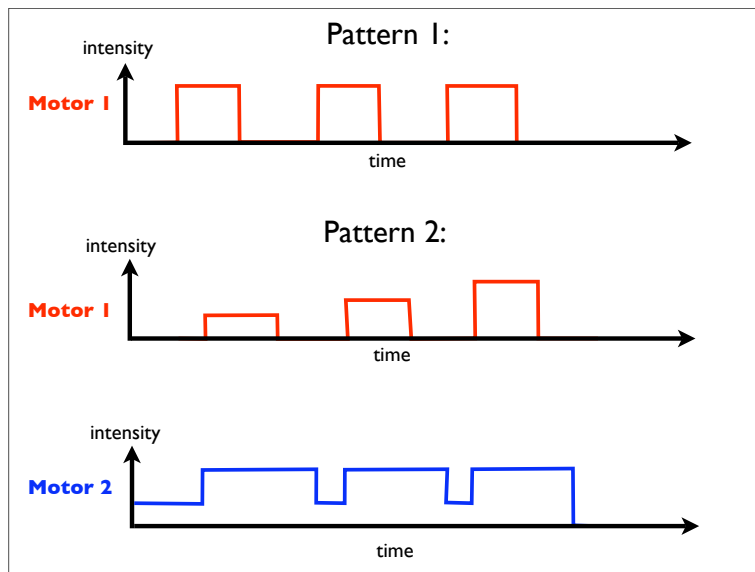
### 2.3.2 Vibrotactile Patterns

Vibrotactile patterns are temporal sequences of stimuli.

A sequence of vibrotactile stimuli and gaps of different durations can be combined to form a temporal pattern, similar to rhythm for the auditory sense. We will call these tempo-



ral sequences *vibrotactile pattern* for the remainder of this work. Figure 2.2 shows two examples of such a pattern; one with only one actuator receiving on/off signals and another with two actuators receiving signals with three levels of intensity.



**Figure 2.2:** Two sample vibrotactile patterns; pattern 1 with one actuator and one level of stimulus intensity, pattern 2 consisting of 2 actuators with varying levels of stimulus intensity

Vibrotactile patterns form a fundamental concept for the toolkit we present in this work - we later on present a tool to design and test these patterns very conveniently using a graphical user interface.

Patterns form a central concept for this work.

### 2.3.3 Tactile Spatial Resolution and Tactile Acuity

In addition to temporal order, stimuli have to be identified spatially. When presented a given tactile stimulus, one has to determine where this stimulus originated. The level of precision, with which we can locate the origin of a tactile stimulus is called the *tactile spacial resolution*, see [Boven et al., 2000] for details.

Level of precision for locating stimuli is defined as spatial resolution.

Spatial resolution depends on stimulus and body area.

Spatial resolution is not an absolute measure, but rather depends on the type of stimulus and the area of the body where it is applied (see table 2.3 and figure 2.3).

It is proportional to the spacing of the different types of mechanoreceptors in the skin and formally defined by the so-called *two-point threshold*, "defined as the smallest separation at which two points applied simultaneously to the skin can be clearly distinguished from a single point." [Colman, 2001]

Body part	Two-point threshold
Tongue tip	1.1 mm
Finger (inside)	1-2 mm
Palm	10 mm
Upper arm	39 mm
Thigh	45 mm
Middle of the neck	67 mm

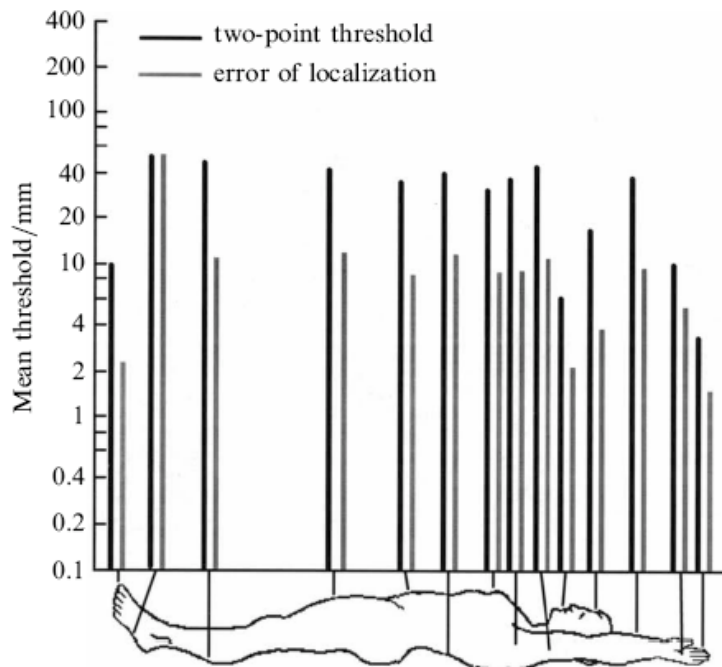
**Table 2.3:** Spatial resolution evaluated with two-point thresholds - from [Kandel et al., 2000] and [Grunwald and Beyer, 2001]

Two-point threshold ranges from 1 mm to 60 mm.

This threshold is highest on the fingerpad, where two stimuli that are only separated by 1 mm can be distinguished [Johnson and Phillips, 1981]. On other parts of the body, this resolution is much lower and can be as high as 60 millimeters on upper arm or back - table 2.3 lists sample two-point thresholds for other parts of the human body.

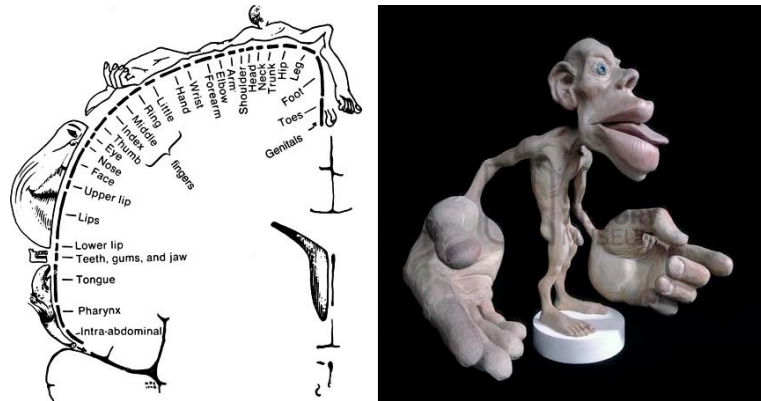
Another measure of tactile acuity is the so-called *error of localization*. It measures whether the same point on the skin was touched twice or whether two different points were touched. Cholewiak [Cholewiak, 1999] noted that the error of localization is generally lower or equal to the two-point threshold; figure 2.3 shows these two values for different parts of the human body.

As shown, different parts of the body offer different spatial resolutions. This is also supported by the physiological fact, that different areas of the body are mapped



**Figure 2.3:** Two-point threshold and error of localization for the human body - taken from [Cholewiak, 1999]

onto specific regions of the brain. The sizes of these areas are proportional to the importance of the sensory perception of that particular body part. Figure 2.4, often called *sensory homunculus*, shows that, for example, hand, fingers, and lips are mapped onto relatively large parts in the brain, compared to trunk, legs or arms.



**Figure 2.4:** Sensory humunculus - detailed view (left, adapted from [Schiffman, 2000]) and 3-D overview (right, copyright National History Museum, London)

### 2.3.4 Tactile Illusions

When the sensory system misinterprets the stimuli being presented to it, sensory illusions occur.

Tactile illusions are misinterpretations of tactile stimuli.

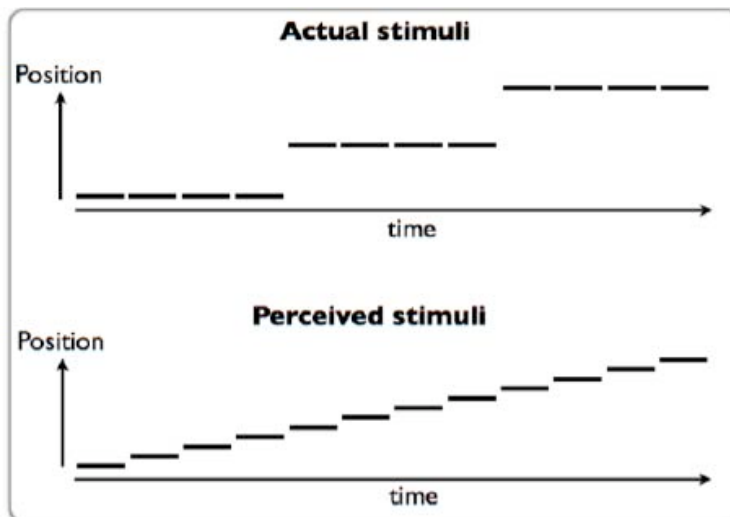
These illusions affect vision, audition and the cutaneous sense - well known are, for example, the *Phi phenomenon* [Graham, 1965] or the *Tau effect* [Helson, 1930]. We are focussing on illusions pertaining the cutaneous sense here: tactile illusions.

Sensory saltation:  
We perceive taps as movement.

One well known tactile illusion is called *sensory saltation* (also called "Cutaneous Rabbit"), discovered in 1972 by psychologist Frank Geldard [Geldard and Sherrick, 1972]. It occurs when a number of taps is being presented to three spatially separated points on the skin and results in the sensation of a continuous movement between the two points as shown in figures 2.5 and 2.6.

Gerald discovered that the time between the taps can vary - intervals between 25 and 200 ms delivered movement sensations. Furthermore, the two tapping points could be as close together as 2 cm or as far apart as 35 cm.

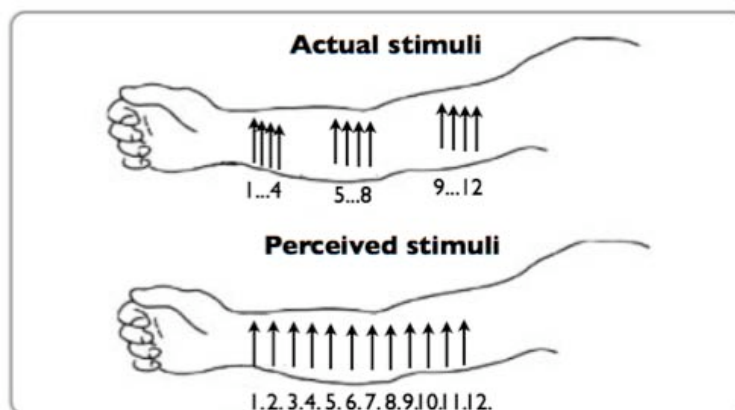
This phenomenon has been studied repeatedly since



**Figure 2.5:** The cutaneous rabbit illusion - actual and perceived stimuli as a temporal pattern

and results indicated that it applies equally to various different parts of the body [Eimer et al., 2005]. When exploring vibrotactile feedback patterns, the sensory saltation phenomenon can be of great interest.

Sensory saltation is of great interest for exploring tactile feedback.



**Figure 2.6:** The cutaneous rabbit illusion - actual and perceived stimuli as locations one the forearm (below)

## 2.4 Summary

The topics covered in this chapter are extensive and innumerable literature is available on the human haptic sense, its physiological requirements, and its capabilities. Nevertheless, we presented the basic physiological and perceptive principles that are relevant for designing vibrotactile feedback patterns. These principles have to be understood in order to generate realistic vibrotactile feedback that is equivalent to real stimuli.

Table 2.4 summarizes the most important anatomical requirements that are involved in researching vibrotactile feedback.

Characteristic	Relevant value
Tactile spatial resolution	1- 60 mm
Frequency range	1-2 mm
Optimal frequency	10 mm
Temporal distinction threshold	39 mm
Optimal stimulus distinction	45 mm

**Table 2.4:** Anatomical and perceptive characteristics relevant for evaluating vibrotactile feedback

Devices engaging the sense of touch can be grouped in two categories: *tactile stimulators* and *haptic devices*.

### 2.4.1 Haptic Devices

Haptic devices move limbs or body parts.

Haptic devices activate receptors in joints, muscles, and tendons in order to stimulate the kinesthetic sense and allow users to touch, explore, or manipulate a virtual object. The best know example for haptic devices is the [PHANTOM range by SensAble Technologies](#)<sup>1</sup>, which includes small haptic desktop devices as well as large haptic devices (see figure 2.7).

<sup>1</sup>see <http://www.sensable.com/products-haptic-devices.htm>



**Figure 2.7:** Two haptic devices from the Phantom series by SensAble Technologies - a Phantom Desktop device on the left and large Phantom Premium device on the right; photos from SensAble Technologies

We will not elaborate on haptic devices in detail here as they are addressing the kinesthetic sense, which is not relevant for the work in this thesis. Therefore we focus on tactile stimulators in the following and give a brief summary of current technology in this field. A comprehensive overview of haptic devices can be found in [Berkley, 2003] and [Burdea and Coiffet, 2003].

### 2.4.2 Tactile Stimulators and Vibrotactile Technology

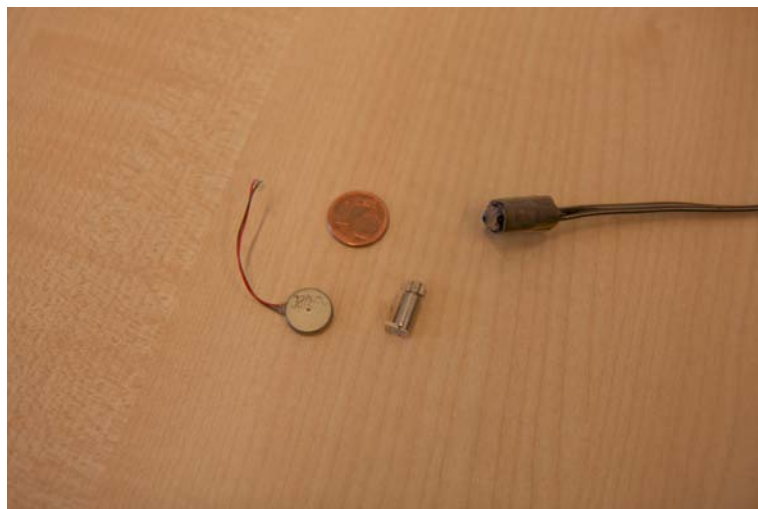
Tactile stimulators typically stimulate the effect of the skin touching a surface - often using vibrotactile stimulators, devices that are in contact with the skin and vibrate at a certain frequency, as used for example, in Lindeman's TactaPack [2006] or TactaBoard devices. Others include devices using a mechanism for skin deformation, such as Mazzone's SmartMesh [2004].

Vibrotactile devices stimulate the skin through vibration.

There is a wide range of vibrotactile devices available - most common are so-called *inertial actuators*. These actuators cause vibrations by moving an internal mass that causes the entire case in which the moving mass is encased, to vibrate. Two types of actuators can be distinguished: cylindrical and coin/pancake vibration motors, see figure

2.8 for examples of both types. One actuator is the VBW32 Tactor (Skin Transducer), which is commercially used in the [Tactaid devices](#)<sup>2</sup>, that offer tactile hearing aids.

For testing and evaluating our toolkit, we used vibration motors as used in the Nokia 3210 mobile phones.



**Figure 2.8:** Vibrotactile stimulators: A coin vibration motor (left) and a Nokia 3210 cylindrical vibration motor - bare (middle) and with cable attached (right)

We used Nokia 3210 vibration motors for testing purposes.

These motors are comparably cheap and easy to get - approximately five dollars in designated electronic stores - which makes them well suited for our intended purpose of testing vibrotactile feedback patterns in the lab. Figure 2.8 shows a bare Nokia 3210 vibration motor (middle) and the same motor with cable attached for easily connecting the motor to a hardware board (right).

Jones [2004] gives an excellent survey of current vibrotactile actuator technologies and evaluates these technologies in regard to designing a vest containing vibrotactile actuators.

---

<sup>2</sup>see <http://www.tactaid.com/>



## Chapter 3

# Related Work

*“Research is to see what everybody else has seen,  
and to think what nobody else has thought.”*

— *Albert Szent-Gyorgyi (Hungarian Biochemist,  
1937 Nobel Prize for Medicine, 1893-1986)*

The field of haptic interface research is less developed than its visual or auditory counterparts, yet it has become a fast-growing and promising research area. Various projects attempt to make the most of the potential, the haptic sense offers.

In the first section of this chapter, we will examine the research fields where haptic feedback has been used recently. These include virtual reality, medical simulations, and motor skill training.

Even though there are numerous works that explore vibrotactile feedback patterns, only very few of those mention ways of easily designing and testing those patterns. The second section of this chapter examines systems that allow for limited designing and testing of vibrotactile patterns; although they are not fully suitable for the purpose of this thesis, these systems still offer valuable ideas and suggestion for an own design.

Very few existing  
tools to design  
patterns

Beyond that, we will examine a few selected HCI toolkits

Track-based editors  
as examples for our  
GUI

to show how ideas from design and evaluation of these toolkits can be adopted for our work.

Finally, we look at track-based editors from other fields of application, such as music or video editing, to assess the extent to which we can adopt approaches from these fields for our work presented here.

### 3.1 Haptic Feedback

Haptic feedback is  
often used in virtual  
reality.

In various research areas, haptic feedback is augmenting its visual and auditory counterparts already - the ComTouch device [Chang et al., 2002], for example, uses vibrotactile stimuli to augment remote voice communication.

Medical training  
scenarios often rely  
on haptic feedback.

In virtual reality environments, haptic feedback is often used to provide a more authentic experience - numerous examples exist here, but we refer to [Burdea, 1996] and [Burdea and Coiffet, 2003] for more information on tactile and force feedback in virtual reality environments.

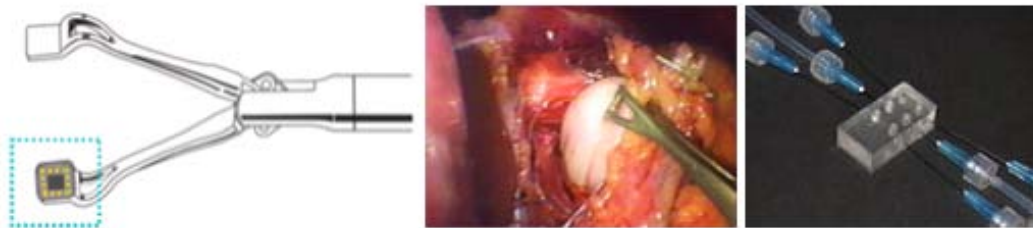
Various medical training scenarios use haptic feedback for simulating surgical operations, among others by Chen [1998], Eltaib [2003], and Schostek [2006], or providing feedback for minimally invasive surgery, where the surgeon has no "natural" haptic feedback compared to conventional, open surgery - see figure 3.1 for an example.

Riener et al. [2002] designed an orthopedic training simulator that provides haptic feedback through a haptic display, compare figure 3.2 (right).

Several research projects use haptic feedback devices to assist users in training a certain skill:

Haptic feedback  
assists users in  
training motor skills.

Adams et al. [2001] deployed haptic feedback for training users in a manual assembly task while Morris et al. [2007] explore the use of haptic feedback to teach an abstract motor skill that requires recalling a sequence of forces.



**Figure 3.1:** Sensor mounted on a grasper (left), grasping motion (center), pneumatic balloon actuator array that transmits haptic feedback to the surgeon's hand (right); adapted from CASIT [King et al., 2007]

Yano et al. [2003] designed a portable device that used a haptic interface to guide a user's movement of his body and suggested this can be applied to sports training or rehabilitation (see figure 3.2, left).

All of these projects showed that haptic feedback increased the learning rate for the specific task - users that received haptic feedback while training the skill showed reduced learning times compared to users that received training without haptic feedback or did not receive any training at all.



**Figure 3.2:** Yano's haptic interface (left) providing guidance in body movement [2003]; multi-modal training simulator containing a haptic feedback display [Riener et al., 2002]

Haptic sense can replace visual or auditory channel.

In some cases the haptic sense can even replace visual or auditory feedback - as in situations where the visual and auditory channels are occupied otherwise (while driving a car or concentrating on a similar, mentally challenging task).

Braille displays

There are situations where visual or auditory senses are impaired and haptic feedback can provide a valuable aid for the user, as in braille displays or similar devices. Numerous research projects cover the topic of haptic interfaces for visually or auditory impaired users [Sjostrom, 2001].

For a detailed discussion on haptic feedback, see [Stone, 2000].

### 3.1.1 Perception of Haptic Feedback

Numerous works have discussed perception of haptic feedback over the last 50 years - a comprehensive overview can be found in [Loomis, 1981].

We will not go into too much detail about studies regarding the perception of vibrotactile feedback patterns here - in chapter 2, we gave examples on perceiving vibrotactile stimuli already, such as the tactile illusion called sensory saltation.

One interesting approach was pursued by Chan, MacLean and McGrenere [2005], who tested how subjects perceived haptic feedback while under workload.

## 3.2 Editors to Specify Haptic Icons

In this work we create a tool to design and test vibrotactile patterns. Therefore we now review existing editors that allow users to specify haptic content.

### 3.2.1 Hapticon Designer, Editor and Displayer (HDED)

Enriquez [2003] has developed the HDED (Hapticon Designer, Editor and Displayer), a tool that allows the user to create, edit, and display haptic icons (also called "Hapticons") in a simple and understandable manner using a graphic display.

HDED is linked to one hardware device, a single degree of freedom haptic device configured as a haptic knob. All haptic forces are displayed on this knob via a single, direct-driven DC motor. Hapticons are displayed as waveforms with time on the horizontal and vibration frequency on the vertical axis (compare figure 3.3) and can be edited by modifying the waveform via simple drag and drop operations.

Haptic force is displayed as a waveform.

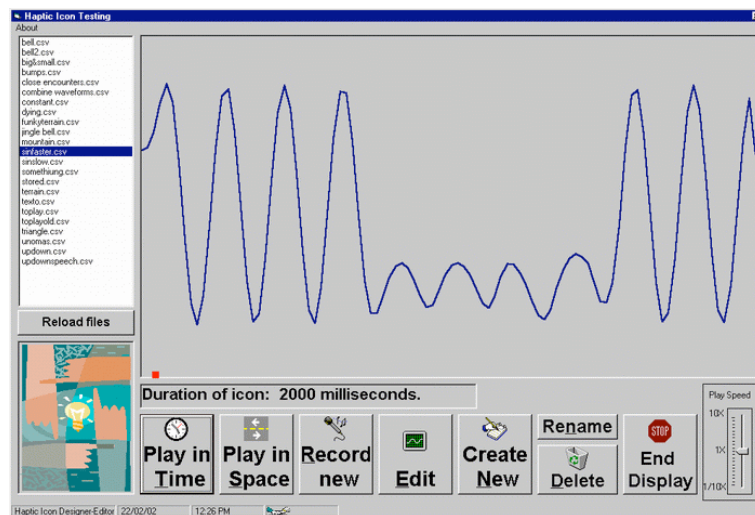


Figure 3.3: Hapticon Editor Main Screen

The main drawback of the Hapticon Editor is the very limited number of output devices it supports. Unfortunately, users can only send hapticons to one single output device, the haptic knob. While Enriquez interface is appropriate for his setting, we need a more variable setup - including more than one hardware device that displays our haptic

HDED only supports one haptic output device.

HDED not suitable for our research.

output. For our research into vibrotactile feedback patterns, we want to connect multiple vibration motors at the same time and distribute those motors over the human body, so using the Hapticon Editor is not an option.

Furthermore, the editor's editing metaphor is based on waveforms, which assumes, the user has at least a basic understanding of waveforms and the underlying mathematical principles. We propose a different editing metaphor, based on switching motors on and off for certain durations and at certain intensities. This on/off metaphor is easier to understand and does not require any previous knowledge.

### 3.2.2 VITAL

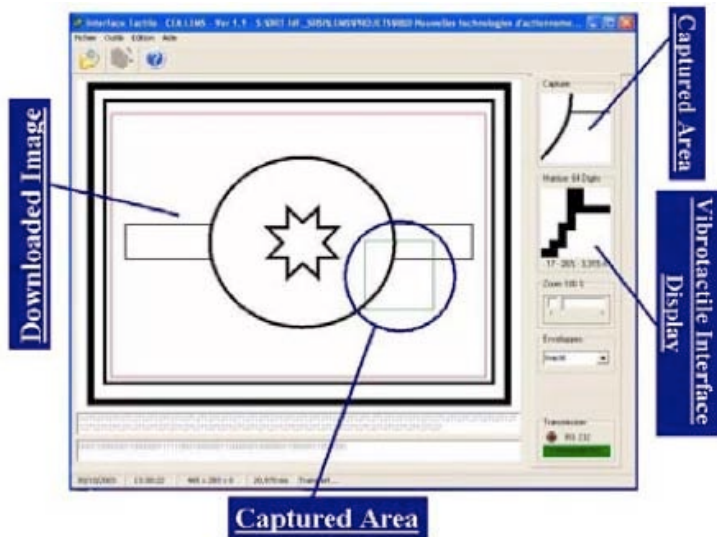
Digital black and white images are translated into tactile figures.

Khoudja and Hafez [2004] developed a vibrotactile display system, composed of a 8x8 matrix of vibrotactile actuators including a software interface to activate the display. Their main focus was translating digital black and white images into tactile figures, where a white pixel corresponds to an actuator being off and a black pixel to an on-state of the corresponding actuator.

Using their software interface, which connected to the vibrotactile matrix via a RS232 serial interface or USB, Khoufia and Hafez could choose the motor to actuate and specify wave form and frequency of the signal that represents a black pixel of the image.

Temporal dimension of patterns is neglected.

The main advantage of the VITAL system is that it can address each actuator of the array independently to display haptic images. On the other hand, its editing capabilities are too basic and limited (compare figure 3.4) for our purposes as vibrotactile stimuli are connected to black and white pixels, which does not allow exploration of vibrotactile patterns; the software interface focusses on spatial patterns only and disregards the temporal dimension of tactile feedback.



**Figure 3.4:** Software Interface for VITAL - taken from [Khoudja and Hafez, 2004]

### 3.2.3 Immersion Studio

Chan et. al [Chan et al., 2005] have used [Immersion Studio](#)<sup>1</sup>, a GUI-based haptic editor, to design and test a set of haptic icons.

This editor contains predefined haptic effects and allows the user to map those effects to standard devices such as tactile feedback mice, joysticks, or gamepads. Immersion Studio is mainly used for commercial purposes when designing touch sensations that match certain gaming events and environments.

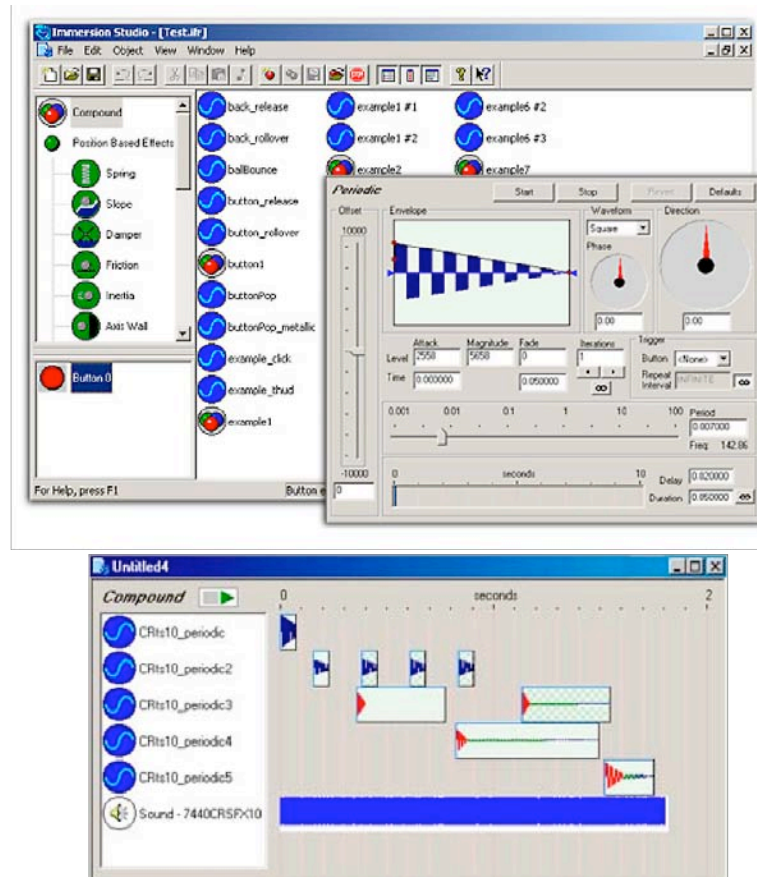
Predefined haptic effects can be mapped to standard haptic devices.

Chu [2002] also used the Immersion SDK to add haptic sensations to sound editing tasks and [Evreinovea](#)<sup>2</sup> designed vibrotactile patterns for a haptic mouse.

Immersion Studio is the only commercially available application to generate haptic feedback for standard haptic devices. It provides a compound effect view (compare figure 3.5), where different pre-defined haptic effects can

<sup>1</sup>see <http://www.immersion.com>

<sup>2</sup>see [http://www.cs.uta.fi/~e\\_tg/Tactons.htm](http://www.cs.uta.fi/~e_tg/Tactons.htm)



**Figure 3.5:** Immersion Studio graphical user interface - haptic effects library (top) and compound effect view with timeline (bottom) - from Immersion Corp.

SDK aimed at game developers designing haptic content.

be arranged on a timeline-based view. Unfortunately, Immersion's APIs only work with selected hardware devices and are aimed at developers designing haptic content for commercial computer games. Furthermore, the development kit is only available for the Windows platform and at a price of \$299, which eliminates Immersion Studio for our research purposes.



### 3.2.4 VibeTonz

Immersion have also produced a system called *VibeTonz* that adds haptics as a channel for mobile phone use, see figure 3.6. The system includes an SDK containing *VibeTonz Studio*, a haptic editor used to design haptic effects that can be assigned to mobile phone handsets - similar to aforementioned Immersion Studio. On this account, we will not discuss VibeTonz any further here - more information on the system can be found in [Immersion, 2008].

Haptic effects can be assigned to mobile phones.



**Figure 3.6:** Immersion VibeTonz Studio - composition environment to design haptic effects for mobile devices - from Immersion Corp.

### 3.2.5 Skinscape

Skinscape, Eric Gunther's master thesis at MIT, includes a software composition environment that allows a user to design tactile compositions for a connected device along a previously recorded audio track [Gunther, 2001]. Haptic sensations are designed on one vibration track that displays the vibration as appropriate waveform information with amplitude on the vertical and time on the horizontal axis - just like Enriquez's HDED.

Skinscape's restrictions are similar to those with Enriquez' Hapticon Editor - haptic sensations are limited to one haptic output device. For this reason, we cannot use

this composition environment for designing and testing vibrotactile patterns.

### 3.3 HCI Prototyping Toolkits

As this thesis introduces a toolkit to design and test vibrotactile feedback patterns, it makes sense to look at other toolkits from the research area of human computer interaction.

Observing these toolkits may suggest useful ideas for design and evaluation of our system.

#### 3.3.1 iStuff

iStuff (see figure 3.7), a toolkit consisting of physical devices and a flexible software infrastructure, was designed to simplify and facilitate the exploration of novel interaction techniques [Ballagas et al., 2003].



Figure 3.7: Quartz Composer graphical user interface

Toolkit allows rapid prototyping of physical user interfaces.

The toolkit is aimed at HCI researchers to allow quick prototyping of physical user interfaces without the need for extensive wiring, soldering, or programming. Figure 3.7 (taken from [Ballagas et al., 2003]) shows sample physical input components for the toolkit. iStuff is a physical

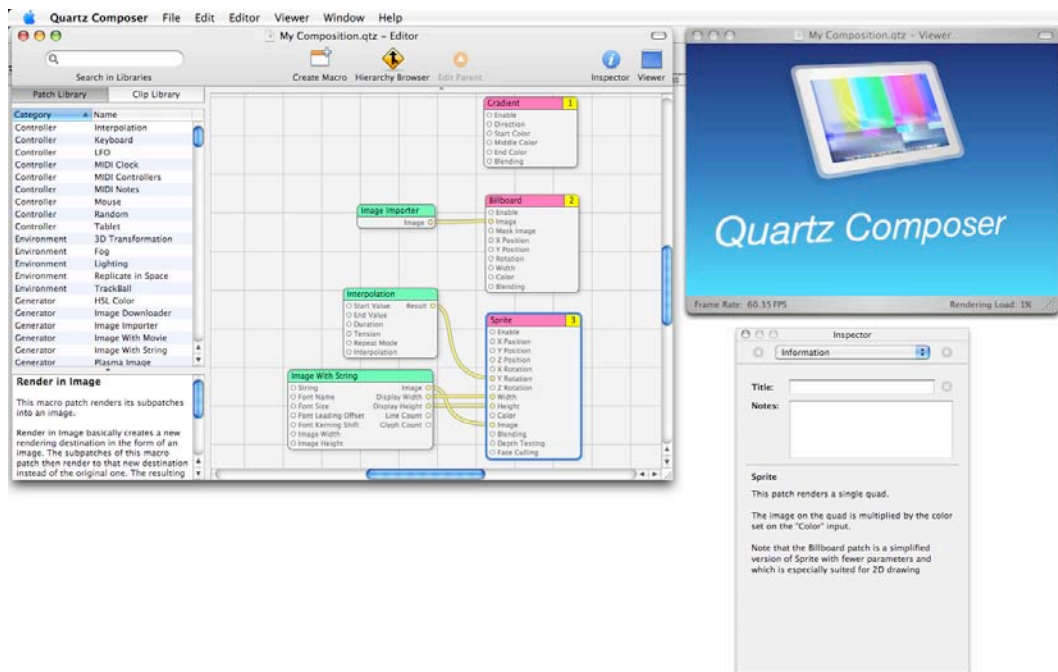


Figure 3.8: Quartz Composer graphical user interface

toolkit, and therefore, the demands on hardware compatibility and communication between devices are similar to those demands for our work, although the application areas are quite different.

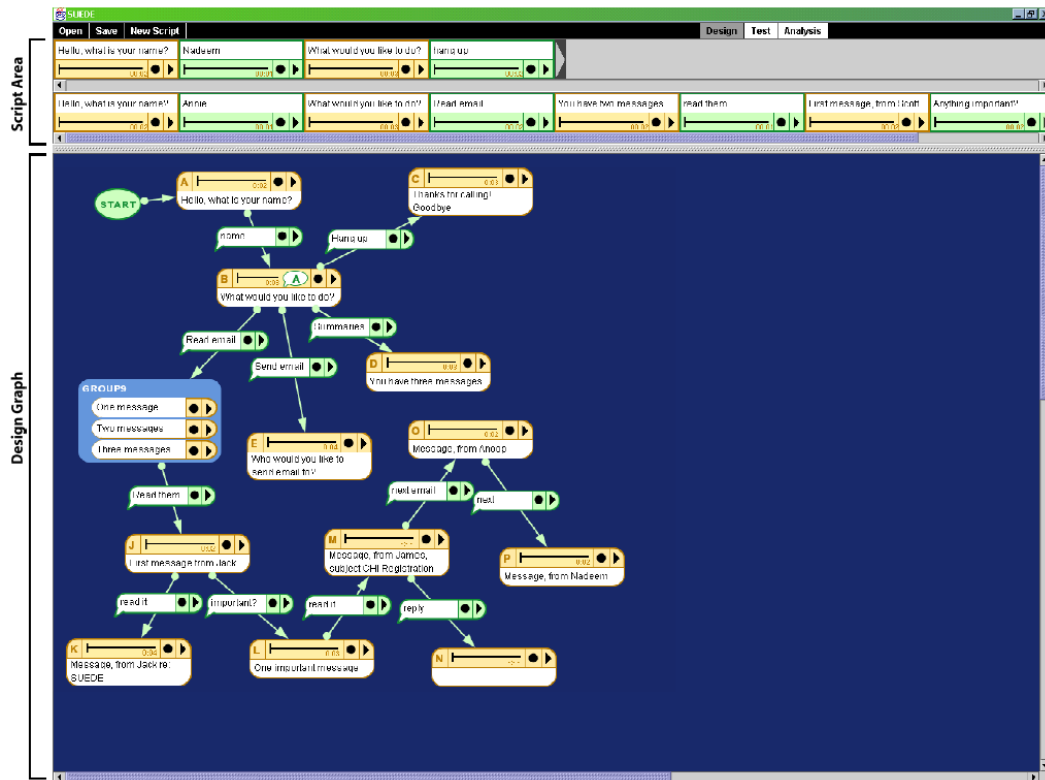
### 3.3.2 iStuff Mobile

iStuff mobile (see figure 3.8), an extension and refinement of the original iStuff toolkit, supports rapid prototyping of mobile phone interactions [Ballagas et al., 2007]. The framework includes mobile phone software, sensor boards, and a rapid prototyping framework that allows interaction designers to quickly create and test functional prototypes without the need for changing the phone's hardware or software components.

iStuff mobile allows rapid prototyping by providing a new interface for physical prototyping that extends Apple's Quartz Composer (see figure 3.8), a visual programming environment based on a cable patching metaphor instead

Rapid prototyping for mobile phone interactions.

Visual programming environment based on Quartz Composer



**Figure 3.9:** Suede tool to design speech user interfaces - script mode (top) and state transition mode (bottom)

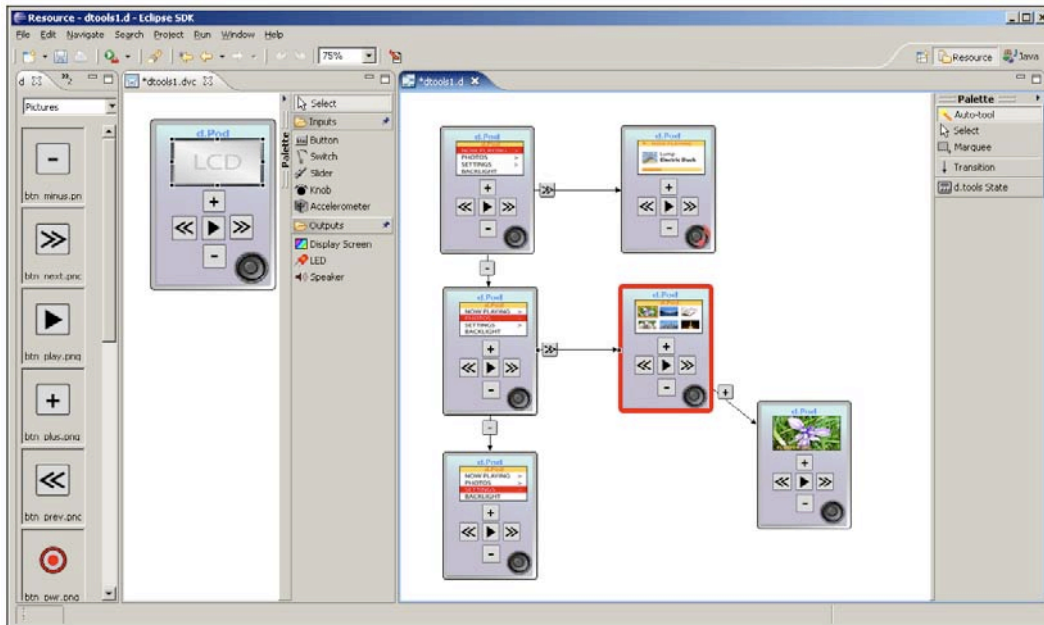
of textual programming. This metaphor guarantees both that the system for prototype construction is easy to learn (low threshold) and that much can be achieved with the system (high ceiling).

### 3.3.3 Suede

Suede allows rapidly prototyping speech interfaces.

Suede, a tool for prototyping speech interfaces, allows designers to rapidly develop prompt/response speech interfaces and enables user interface designers, even non-experts, to easily create, test, and analyze speech user interface systems [Klemmer, 2000], compare 3.9.

The tool offers an interesting approach to designing alternative user interfaces that purely rely on auditory



**Figure 3.10:** d.tools editor using statecharts to develop interactive prototypes (taken from [Hartmann et al., 2005])

input and output. Users can prototype speech user interfaces based on concepts of example-based scripts or prompt/response state transition graphs ( see figure 3.9 for an example).

### 3.3.4 d.tools

d.tools was developed at Stanford University as a design tool for prototyping physical user interfaces [Hartmann et al., 2005], compare figure 3.10. It enables designers to quickly build functional, interactive prototypes without special programming or engineering knowledge.

Prototypes are constructed using a visual authoring environment that represents interaction models as state transition networks (STN). Designers can lay out the interaction model as a STN and then execute interaction on the physical device that is connected to the computer running the authoring environment. The interaction model

Interactive prototypes are designed using statecharts.

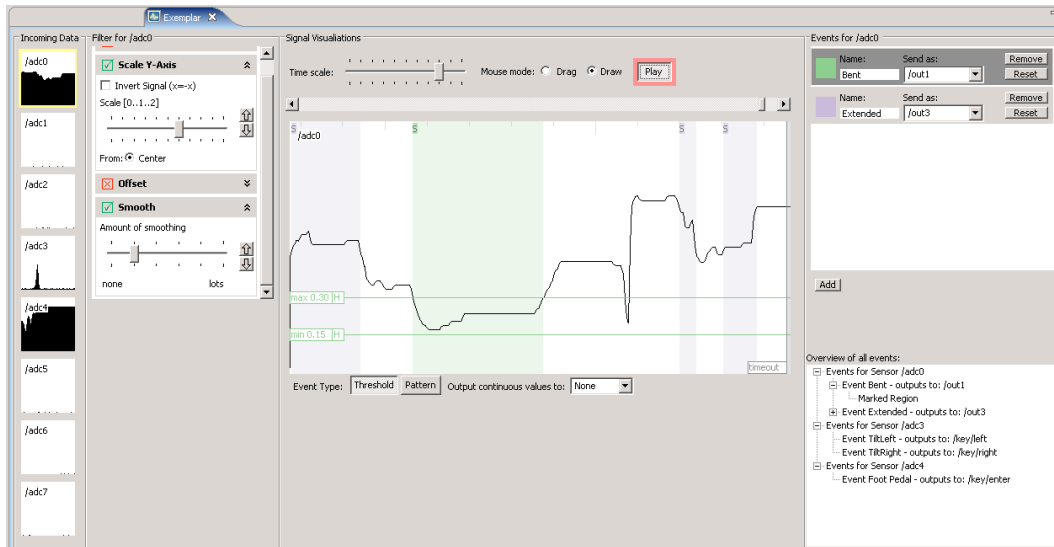


Figure 3.11: Exemplar GUI

is “live”, which means designers can test interactions with the physical device while observing the model on the screen (see [Hartmann et al., 2005] for details).

Figure 3.10 shows an interaction model represented by a STN in d.tools.

### 3.3.5 Exemplar

Exemplar is a rapid prototyping tool that enables a wider audience of designers to process raw sensor data. Figure 3.11 shows a screenshot of its user interface. The system was developed at Stanford University by Hartmann et al. [Hartmann et al., 2007] and allows designers to focus on how interactions with sensors work instead of forcing them to understand technical sensor signal processing details.

Sensor data is displayed graphically.

While the designer demonstrates a sensor-based interaction to the system, sensor data is captured and thereupon displayed graphically by Exemplar. The designer can then edit that visual representation and review the result by performing the interaction again.

### 3.3.6 Discussion

We reviewed the HCI prototyping toolkits mentioned above to extract principles, concepts, and techniques for our work. In the following, we will now briefly summarize these points:

Exemplar, d.tools, Suede, and iStuff mobile all provide a *visual programming metaphor* to make their toolkits amenable to non-expert users. These metaphors all include a meaningful graphical representation of underlying concepts such as state charts to model interactions (d.tools), visualization of sensor data (Exemplar), or the cable patching metaphor for mobile phone prototypes (iStuff mobile). In this work, we aim to find such a strong visual programming metaphor for designing vibrotactile feedback patterns.

Visual programming

Another important paradigm that these toolkits implement is *direct manipulation*. Objects can be manipulated directly, which highly facilitates the user's work. As an example, sensor visualizations in Exemplar can be edited directly by clicking on the visualized sensor data graph. For Tactile Editor, we aim to provide such direct manipulation techniques as well to provide the user with an easy-to-use graphical user interface.

Direct manipulation

## 3.4 Track-based Editors

In other application domains there are some track-based editors, whose approaches can be transferred to our application domain, tactile feedback. Music sequencing software often relies on a track-based user interface and video editing software also borrows from this paradigm. In the following, we will briefly touch on these areas and give a few examples of applications that realize track-based interfaces successfully.

### 3.4.1 Music Sequencing Software

Track-based paradigm is natural analogy for music editors.

Most music editors such as [Apple's GarageBand](#)<sup>3</sup>, [Fruity-loops](#)<sup>4</sup>, or the open-soure project [Audacity](#)<sup>5</sup> use a track-based graphical user interface for creating, editing and mixing songs or audio files (see figure 3.12 as an example). This editing metaphor is a natural analogy for the music domain, is well-known to most users, and can be adopted easily for our purpose of editing vibrotactile feedback patterns.

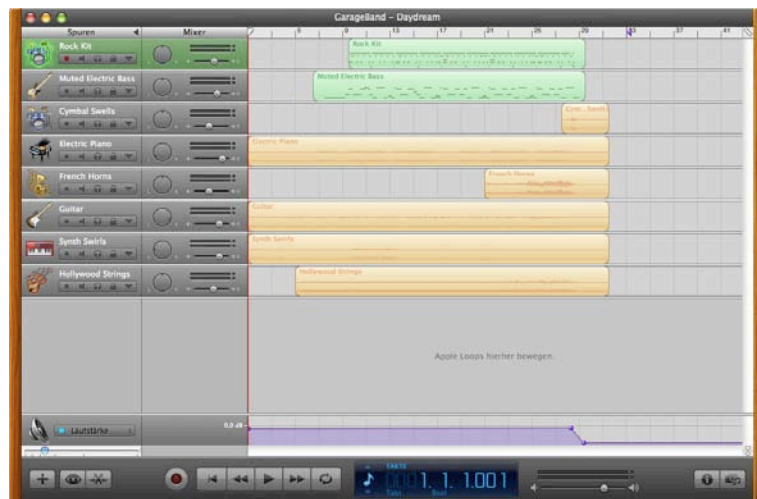


Figure 3.12: GarageBand - track-based GUI

### 3.4.2 Video Editing Software

Most editors use drag and drop timeline interface.

A common feature in commercial video editing application is an intuitive drag and drop timeline interface. This principle can be found in professional, high-end products such as [Avid's Media Composer](#)<sup>6</sup>, [Adobe's Premier Pro](#)<sup>7</sup>, or [Apple's Final Cut suite](#)<sup>8</sup>, just as in open source products such

<sup>3</sup>see <http://www.apple.com/ilife/garageband/>

<sup>4</sup>see <http://www.fruityloops.com/>

<sup>5</sup>see <http://audacity.sourceforge.net/>

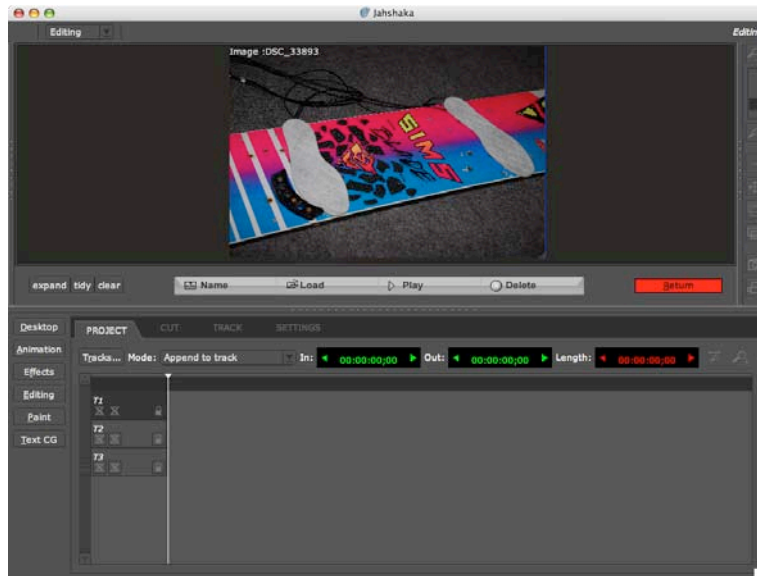
<sup>6</sup>see <http://www.avid.com/products/732.htm>

<sup>7</sup>see <http://www.adobe.com/products/premiere/>

<sup>8</sup>see <http://www.apple.com/finalcutstudio/>



as [Jahshaka](#)<sup>9</sup> or applications for non-professional users like [iMovie](#)<sup>10</sup>. Figure 3.13 shows an example for a track-based video editing environment.



**Figure 3.13:** Jahshaka - track-based open source video editor

## 3.5 Toolkit Evaluation

While it is rather difficult to formally evaluate a toolkit, there are certain properties that can measure the quality of a toolkit. Brad Myers et al. [2000] identified, among others, two key characteristics or “themes” for evaluating toolkits - threshold and ceiling of a system:

The *threshold* of a system defines how difficult it is to learn using it, and the *ceiling* refers to how much can be achieved by using the system. Most existing system are either low-threshold and low-ceiling or high-threshold and high-ceiling. However, Myers et al. emphasize that it is

Threshold and ceiling are measures for toolkit evaluation.

<sup>9</sup>see

<sup>10</sup>see <http://www.apple.com/ilife/imovie/>

highly desirable to find a way to design systems with both a low initial threshold and a high ceiling at the same time.

We will revisit these two themes in chapter 7, where we evaluate this work and analyze, how our toolkit helps users to create and test vibrotactile patterns.

### 3.6 Summary

Three dimensions to classify existing systems.

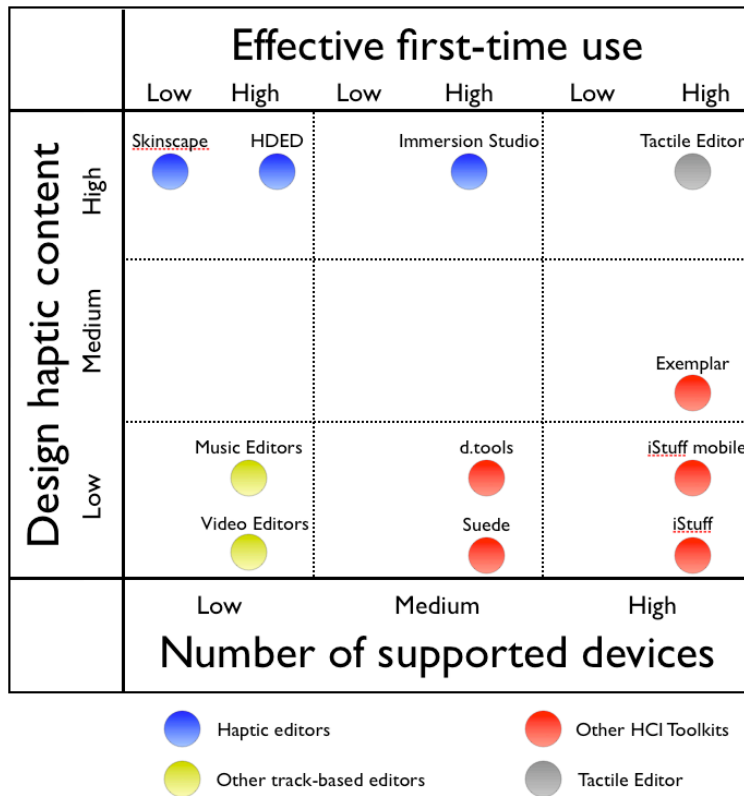
We reviewed several editors and toolkits in this section and pointed out the contributions and shortcomings of each of those systems for our own work. To conclude, we can classify these systems along three dimensions:

- **To what extent can the system be used to design tactile content?** How suitable is the system for testing tactile feedback prototypes?
- **What number of input and output devices/media does the system support?** Is it possible to design for multiple devices at a time?
- **How easily can a first-time user start using the system effectively?** To what extent enables the system a non-expert user to create and edit content? How fast can a user design a working prototype with the system?

When categorizing the systems presented in this chapter along these three dimensions, we observe that none of the systems meets all three requirements stated above - Figure 3.13 shows this graphically.

Tactile Editor, the tool we propose in this thesis, is the only system occupying a spot in the upper left section of the diagram.

Existing systems can only partially meet the demands, we place on a toolkit to design and test vibrotactile patterns. The Haptic Editors presented in section 3.2 were essentially



**Figure 3.14:** Comparison of existing systems - classified along three dimensions

too specifically designed for one particular haptic device and did not offer the range of possibilities, we expect from our toolkit. While the HCI toolkits we discussed in section 3.3 supported a wide range of output devices, we cannot use them to design vibrotactile patterns. Finally, music and video editors we introduced in section 3.4 provide a well designed interaction metaphor well worth investigating; they were designed for different application areas though.

Tactile Editor should aim to provide the advantages of all these benefits: support for multiple haptic output devices, effective first-time use, and the possibility to design haptic content. In the following chapters, we will discuss, how this can be achieved.



## Chapter 4

# Requirements

*“The beautiful rests on the foundations of the necessary.”*

—*Ralph Waldo Emerson (American Poet, Lecturer and Essayist, 1803-1882)*

As outlined in the last section, there yet exists no toolkit that allows end-users to rapidly design and test vibrotactile feedback patterns. Existing systems are either too limited regarding the tactile output devices they support or do not allow for unrestricted editing of vibrotactile feedback patterns.

In this chapter, we will now outline the requirements of the system we design and reveal the constraints that affect its design and development process.

Requirements and constraints for the system

### 4.1 Overview

This section outlines the required characteristics of the tool build for this thesis. It shows the objective of the tool, what functionality it must encompass, and what constraints must be taken into account during the design phase. Requirements are ranked according to their priority for the project; priorities are defined as either “must”, “should”,

Priorities assigned to requirements

or “could”.

Furthermore, this section characterizes the users of the editor, gives details about development and testing environments, and lists hardware platforms that are supported.

## 4.2 Problem Definition and System Objective

Most modern interfaces without tactile feedback

iPhone lacks system for tactile feedback.

Haptics widely a research area

Tactile feedback is still widely absent from today’s communication channels as most interactions focus on our visual and auditory senses. Even some modern devices that use haptics as an input channel neglect haptics as an output channel - thus focussing on the visual and auditory senses for transmitting information to the user. The most prominent example is Apple’s iPhone that purely relies on tactile input via the built-in touch-screen, but lacks a system for tactile feedback. For details, compare [Hoggan et al., 2008].

As mentioned above, the field of haptics is still widely a research area; for this reason we need efficient methods to support testing the effects of haptic feedback. Therefore, the toolkit we develop enables users to easily create and test vibrotactile patterns on a variety of hardware platforms.

## 4.3 Tactile Editor

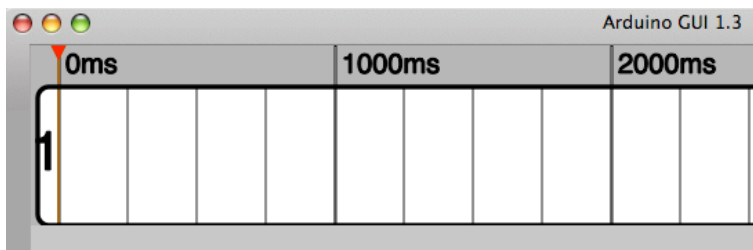
Tactile Editor: GUI to design and test vibrotactile patterns

To allow users rapid, fast, and effective design of vibrotactile patterns, our toolkit needs to provide a suitable way of achieving this - a tool that enables users to design and test these patterns. Therefore, we designed *Tactile Editor*, a timeline-based editor including a graphical user interface allowing users to create and modify vibrotactile patterns via simple drag and drop operations.

We need a suitable representation of the vibrating motors within the editor and decide upon a track-based canvas with a timeline along which representations of vibrating motors can be arranged.

Track-based canvas  
with timeline

The canvas can contain one or more vibration tracks, see figure 4.1. Each track can contain vibrotactile patterns and consists of motor objects representing individual motors that vibrate at specified times.



**Figure 4.1:** Canvas containing an empty track

Instances of vibrating motors are displayed as colored rectangles, where the properties of these rectangles correspond to the properties of the motor vibration they represent: each motor gets associated with a certain color - therefore, the color of the rectangle represents a motor number. The position of the rectangle along the timeline equals the time at which the motor vibrates, rectangle width equals vibration duration, and rectangle height corresponds to the intensity with which the motor vibrates. Figure 4.2 shows an example of two motor rectangles on a timeline:

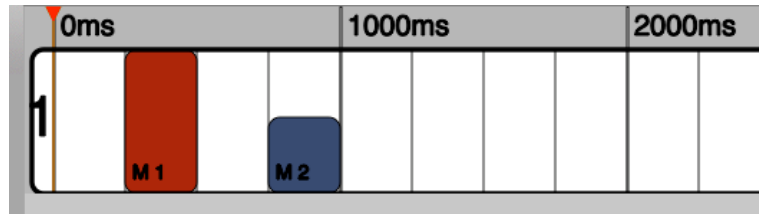
Motor vibrations  
represented as  
colored rectangles

Motor 1 vibrates with 100% intensity for 250 ms, starting at time 250 ms; motor 2 vibrates with 50% intensity from time 750 ms to 1000 ms.

Sample pattern in  
Tactile Editor

## 4.4 Functionality

Tactile Editor has to provide certain features to allow users rapid and effective design of vibrotactile patterns. This section states these requirements for Tactile Editor;



**Figure 4.2:** Example of two rectangles representing motor vibrations

we sort them by priority and define the following priorities:

Definition:  
*Priorities for requirements*

#### PRIORITIES FOR REQUIREMENTS:

- *"Must"*: Requirement is absolutely essential for the system.
- *"Should"*: Requirement should be fulfilled by the final system that gets implemented.
- *"Could"*: Requirement is nice to have, but no primary function of the final system.

We will now elaborate more on the range of functions, the editor has to provide and will group the properties according to the priorities we defined above.

#### 4.4.1 Necessary Properties

Basic editing  
functions

Most importantly, Tactile Editor has to provide basic editing functions to the user. This includes adding or removing an empty track and adding or removing a motor rectangle. Besides that, users should be able to modify motor rectangles by simple, intuitive drag and drop operations - resizing the motor rectangle or dragging a motor from one track to another are examples of those. Alternatively, users can edit rectangle properties by selecting a rectangle and providing the desired values via keyboard input.



In addition to creating and editing vibrotactile patterns, Tactile Editor has to provide a testing environment for these patterns. On this account, users must be able to send the patterns to vibration motors connected to hardware devices such as Arduino boards. User should be presented the opportunity to choose between sending selected tracks only or sending all patterns on all tracks.

Testing of vibrotactile patterns with Tactile Editor

Finally, Tactile Editor has to support standard file operations such as saving a pattern or loading a previously created pattern document. In addition, the editor has to enable users to export the patterns to a format, that can be read and processed by other devices.

Standard file operations

As an example, it should be possible to create a pattern using Tactile Editor, export the pattern to a file, send the file to a mobile phone, and send the pattern from the mobile phone to vibration motors.

#### 4.4.2 Desirable Properties

Apart from basic, compulsory editing functions, Tactile Editor should provide additional features that facilitate working with the editor:

Additional features

Multiple motors can be selected by drawing a selection rectangle around them or by holding down the shift-key while selecting them - as customary for drag and drop user interfaces. All basic editing functions can be accessed through keyboard shortcuts to provide an alternative interaction style that speeds up the editing process for expert users. The editor should provide a copy & paste function and a basic undo-functionality.

Keyboard shortcuts

Tactile Editor should allow users to connect to multiple hardware devices at the same time to allow scenarios where patterns including a large number of motors are required.

For our user tests to evaluate the tool, Tactile Editor should also be able to communicate directly with sensors that are connected to an appropriate hardware device.

Although in practice, communication with sensors takes place via a sensor environment such as Exemplar, we want to provide users in our tests with a possibility to test vibrotactile feedback patterns based on sensor input. Thus users should be able to build prototypes using both sensor data and motor control using Tactile Editor only.

### 4.4.3 Conceivable Requirements

Selection of partial tracks	Tactile Editor could allow users to select partial vibration tracks by providing markers on the timeline that allow selection of a certain region of a track. We could clearly mark the selected region and distinguish it from the rest of the track.
Undo	For increased usability, the tool could provide an extended Undo/Redo functionality, a mechanism that covers all editing functions such as inserting, deleting, moving a motor or track.
Help system	Furthermore, Tactile Editor could be accepting messages from sensing systems such as Exemplar [Hartmann et al., 2007] and play selected vibration tracks upon receiving these messages.  Finally, we could provide a comprehensive help system that explains usage of Tactile Editor in detail and answers all questions a user might encounter while using the system. The help system should be accessible from within the editor.

## 4.5 User Characterization

Tactile Editor designed for end-users	The system is to be deployed in various areas in order to allow users rapid, effective design and testing of vibrotactile feedback; primarily, the tool is to be used in research environments to test the effects of vibrotactile feedback. Therefore, the typical user is a designer or an end-user, rather than a highly trained, specialized programmer.
---------------------------------------	--

On this account, the tool provides a graphical user interface that allows users to design vibrotactile icons by direct manipulation and without the need for textual programming. Users are not required to write down start time, end time, duration, intensity, and motor number in source code, but rather specify these parameters graphically or via text fields in Tactile Editor's graphical user interface.

## 4.6 Development and Runtime Environment

Tactile Editor was initially developed and tested under Mac OS 10.4. Later on we modified and tested a version on Mac OS 10.5 as well, though it doesn't yet support connecting Make Controller Boards via OSC.

## 4.7 Supported Hardware Platforms

In order to enable users to use Tactile Editor for effectively developing and testing vibrotactile patterns, the system supports several different hardware platforms. Tactile Editor explicitly supports Arduino and Make Controller Toolkits (see sections below).

These two toolkits are well-known and in wide-spread use. Arduino also produces a bluetooth board, that allows mobile and outdoor use.

The tool can easily be extended to support any hardware platform that can be addressed via the OSC protocol.

Tactile Editor supports Arduino and MakeController platforms

### 4.7.1 Arduino

Arduino: electronics  
prototyping platform

[Arduino](#)<sup>1</sup> is an open-source electronics prototyping platform consisting of hardware devices (Arduino boards) and a software development environment (currently Arduino 0011 Alpha). It can be integrated with a variety of sensors and actuators such as lights or motors. Arduino's onboard micro-controller is programmed via the [Arduino development environment](#)<sup>2</sup>, that is based on the [Processing language](#)<sup>3</sup> and uses the Arduino programming language, which is based on [Wiring](#)<sup>4</sup>.

PWM

Output ports of Arduino boards can be addressed via analog or digital commands. Analog signals just offer the parameters 0 ("Off") or 1 ("On") for controlling devices connected to the board's output ports. Digital communication allows for a technique called *pulse width modulation (PWM)*, that basically encodes analog signal levels digitally by modulating the duty cycle of the signal, thus regulating the amount of power that gets sent to an output port. As an example, signal 1 is on for 20% of the time and off for 80 %of the time, signal 2 on for 50 %, off for 50 % and signal 3 on for 80% and of for 20 %. If the supply is 5V, signal 1 results in a 1V analog signal, signal 2 in a 2.5 V signal, and signal 3 in a 4V analog signal, compare figure 4.1. A comprehensive overview of PWM can be found in [Barr, 2001].

digitalWrite and  
analogWrite

Devices connected to output ports can be addressed from the Arduino programming language via "digitalWrite" (with parameters 0 and 1 for "off" and "on") or "analogWrite" messages with PWM parameters ranging from 0 to 255. The command "analogWrite (pin, 255)" means the motor connected to port "pin" vibrates with full intensity at 5V, and "analogWrite (pin, 128)" drives the motor with only 50 % intensity. Thus we can adjust the intensity with which the motors vibrate.

Arduino BT board

In this thesis, we focus on using the Arduino Blue-

<sup>1</sup>see <http://www.arduino.cc>

<sup>2</sup>see <http://www.arduino.cc/en/Main/Software>

<sup>3</sup>see <http://processing.org>

<sup>4</sup>see <http://www.wiring.org.co>

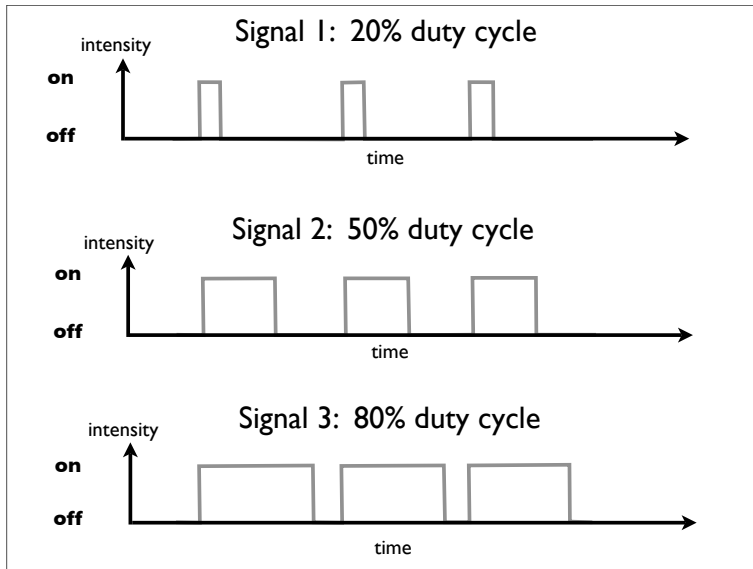


Figure 4.3: Three sample PWM signals

tooth board (Arduino BT) with several vibration motors that are connected to the output ports of the board.

#### 4.7.2 Make Controller Kit

[Make Controller Kit from MakingThings<sup>5</sup>](#) contains an input/output controller (Make Controller) and provides software and electronics tools to create projects that interact with the physical world. Various input devices (such as sensors or pushbuttons) and output devices (such as LEDs or DC motors) can be connected to the board and communicate with each other.

Therefore, the Make Controller board supports the Open Sound Control protocol (OSC), a message-based communication protocol that was designed as successor of the MIDI protocol. The OSC protocol allows communication among computers, audio controllers, and other multimedia devices.

MakeController supports OSC protocol.

<sup>5</sup>see <http://www.makingthings.com>

Sample OSC  
messages

OSC is designed as a simple client/server architecture, where data units called *packets* get sent from a client to a server. Basic data units in OSC are *messages*, which consist of three parts: an address pattern, a type tag string, and a parameter value. The address pattern defines the entity the message is directed to, the type tag defines the data type of the parameter, and the parameter defines the argument of the OSC message. As an example, `/server/port1 i 0` sends a message containing the parameter "0" of data type "Integer" to port1 of the receiving entity.

We focussed on  
working with Arduino  
boards.

In chapter 6, implementation details, we elaborate further on the OSC message format we are using to communicate with Make Controller boards. A comprehensive summary of the Open Sound Control communication protocol can be found in [Wright et al., 2003].

For our purpose, we connect several motors to the board's digital outputs and communicate with the board via simple OSC messages. While we tested Tactile Editor with Make Controller boards, we decided in later iterations to focus on using Arduino Boards as these were significantly smaller and lighter.

In addition, we could use a sensor/actuator box that was built at our group, which saved us the work of building the hardware ourselves. The box (see figure 4.4) contains an Arduino BT board with a custom built motor shield connected to the board. The motor shield can control six vibration motors that are connected to the actuator connectors of the box (compare figure 4.4). Furthermore, the box provides three connectors for sensors.

For our tests, we connected six pressure sensors (force-sensitive resistors) to the board, three to the first and three to the second connector. In addition, we connected six vibration motors to the output connectors of the board, thus providing a complete hardware set for testing vibrotactile feedback patterns.

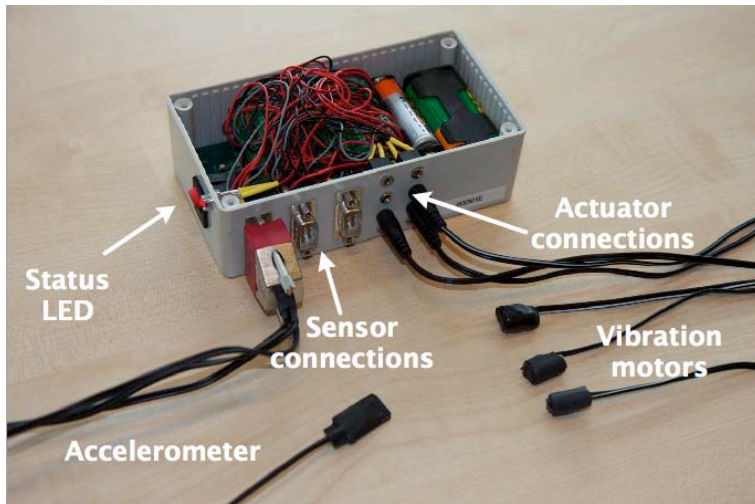


Figure 4.4: Arduino box designed at our group





## Chapter 5

# Conceptual Design

*“A common mistake people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.”*

—Douglas Adams (1952 - 2001), *“Mostly Harmless”*

Now the requirements for the system to be implemented are established, it is time to develop a suitable conceptual design - this shall later on guarantee a smooth implementation. We will establish a concept for implementing Tactile Editor and elaborate on this in more detail in the following section.

Concept for Tactile Editor

### 5.1 System Architecture

Fundamentally, the system to be developed consists of four different layers: visual layer, data layer, logic layer, and hardware. Figure 5.1 shows the relationships between these layers and what function each layer is assigned in our system architecture.

System consists of four layers.

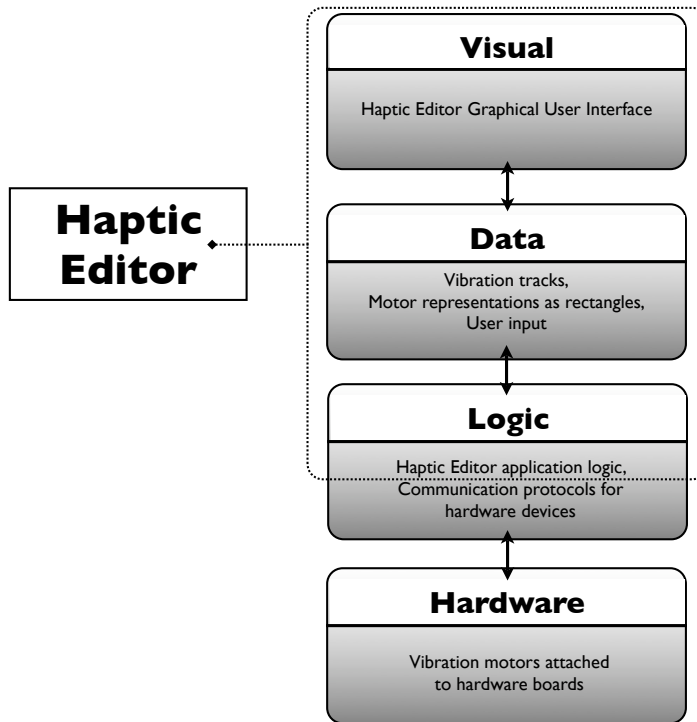


Figure 5.1: System architecture: Layers

### Visual Layer

GUI as visual layer

Everything that is visible to the user when he interacts with the system is combined into the visual layer of our system; primarily this refers to the graphical user interface of the editing tool in the toolkit - Tactile Editor.

The visual layer provides an accurate display of the underlying data structures and ensures that all user-created or modified data is displayed accordingly on Tactile Editor's "center stage", the canvas.

### Data Layer

This layer includes internal representations of user input or other data and all processes associated with those. All user-created patterns have to be handled internally - either to send them to connected vibration motors or to

permanently save them in files. Therefore each pattern file gets stored as a two-dimensional array; every file contains an array of tracks, each of which gets stored as an array of motor objects.

Data stored in two-dimensional array.

Each track contains one or more vibrotactile patterns, consisting of different motors vibrating at certain times. Motor vibrations are represented by motor objects containing start time, duration, frequency, and motor number of the vibration. Appendix A shows header files for pattern files, tracks, and motor objects to illustrate the data structures used in Tactile Editor.

### Logic Layer

Two central parts are included in this layer: the application logic for Tactile Editor and the protocols for communication between Tactile Editor and suitable hardware devices the toolkit supports.

Application logic will be defined as code in Tactile Editor's classes and ensure consistency between data and visual layer.

Application logic ensures consistency between visual and data layer.

The logic layer is also responsible for allowing Tactile Editor to communicate with suitable hardware devices and the other way round by providing appropriate protocols for this communication. Thus, the logic layer ensures that all data edited by a user in Tactile Editor gets converted to the appropriate format and sent to the hardware devices the user selects.

To ensure consistent execution of the commands Tactile Editor sends to connected hardware devices, these devices have to implement the communication protocols as well (see hardware layer below).

### Hardware Layer

This layer includes all hardware components of the toolkit, such as hardware boards which communicate with Tactile Editor or sensors and vibration motors which are connected to these hardware boards.

Hardware components

Firmware allows communication with Tactile Editor.

All of these devices have to be configured properly in order to integrate seamlessly with the software environment; hardware devices such as Arduino or Make Controller boards have to be equipped with suitable firmware that allow for communication with the software part of the toolkit. Therefore, as part of the toolkit, connected Arduino boards run a special firmware program that enables them to receive and execute Tactile Editor's commands.

## 5.2 Concepts of Interaction

Common, well known interaction concepts

Users shall interact with the toolkit by using common and well known interaction concepts. Thus the system uses standard interaction metaphors based on mouse and keyboard input.

### Mouse Interaction

Drag-and-drop mouse interactions

Most interactions with the toolkit are based on mouse interaction with Tactile Editor's timeline-based interface using an intuitive drag-and-drop concept as it is common in today's direct manipulation tools. Motor vibrations, represented by rounded rectangles on a track within the canvas, can be created and modified using common, well known interaction techniques:

Mouse interactions on motor rectangles

Rectangles can be selected by clicking on them, moved by dragging them along the timeline, and modified in shape by clicking on the border and dragging them to the desired size. If users require an alternative technique, properties of motor vibrations and their rectangles can be specified by keyboard input (see next section). The markers that specify the selected region of a track can be selected and dragged along the timeline as well.

Mouse cursor indicates available editing options.

All mouse interactions that are available to the user should be clearly visible; the mouse cursor should indicate the interaction that is available at the current cursor position. If the user moves the cursor over a rectangle border,

the cursor shape should change into the shape indicating the appropriate resizing option (`upDownCursor` for height resizing or `leftRightCursor` for width resizing).

In addition all actions a user executes should be indicated by instant and clearly visible feedback once they are completed (see section *Feedback* below).

### Keyboard Interaction

Several editing functions of the toolkit are provided by keyboard commands. This allows users to access frequently used commands faster and thus allows a more effective use of Tactile Editor.

Furthermore users can specify certain properties of motor vibrations and their rectangles via keyboard input, which allows for an alternative way of specifying start time, duration, or frequency of a certain motor vibration. Users should be able to choose whether they prefer mouse interactions or keyboard interaction for certain actions - thus Tactile Editor should provide both methods of data input for the user.

### Feedback

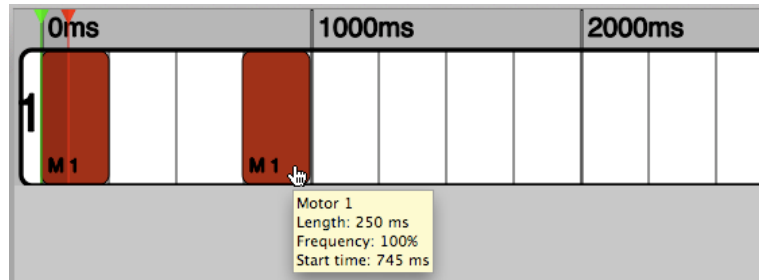
All actions a user performs using Tactile Editor shall be accompanied by instant, clear, and visible feedback. Therefore all modifications to a rectangle representing a motor vibration shall be visible as soon as a user performs them; this includes selection of a rectangle, dragging it along the timeline, modifying its start time, duration, or frequency, etc.

Instant and clearly visible feedback for all user actions

Tactile Editor shall provide several features that give users feedback regarding the progress of their work. This includes tooltips when a user drags a motor along the timeline (showing current start and end time) or resizes a motor (showing current duration or current frequency, when changing width or height). Furthermore, moving the mouse cursor over a motor rectangle should activate a tooltip showing details about that particular motor vibra-

Tooltips

tion, see figure 5.2 for an example.



**Figure 5.2:** Feedback: User moves mouse over existing motor rectangle - tooltip showing motor vibration details

## Chapter 6

# Implementation Details

*“Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.”*

— Rick Cook, *fantasy author, in “The Wizardry Compiled”*

The following chapter presents the tool we developed and elaborates on Tactile Editor’s implementation in detail, explaining the different parts of its graphical user interface and how they were implemented. In addition, we will give illustrated examples on how interactions with Tactile Editor work.

We illustrate the iterative design approach taken when developing the system and demonstrate how Tactile Editor’s design evolved over the different iterations.

Iterative design approach

### 6.1 Overview

The system is implemented as an editor that allows users to design tactile feedback patterns before testing those patterns with connected vibration motors. Therefore we

Editor to design and test patterns

needed a suitable representation of the vibrating motors within the editor and decided, as mentioned in the previous chapter, on a track-based canvas with a timeline along which representations of vibrating motors can be arranged. We will now discuss the editor in detail, starting with the structure of the underlying application, before going into detail about the different parts of Tactile Editor’s graphical user interface.

We need protocols for communication with hardware devices.

Finally, we will specify the communication protocols for sending commands to address vibration motors from Tactile Editor and for receiving commands from connected sensors.

## 6.2 Model-View-Controller Paradigm

MVC paradigm is an architectural pattern to isolate data, GUI, and application logic.

We designed Tactile Editor using the *Model-View-Controller paradigm (MVC)*, an architectural pattern from software engineering that isolates application data (model) from the visual presentation layer (View), usually the graphical user interface, and the application logic (Controller) that connects the two. Figure 6.1 illustrates the MVC paradigm graphically.

MVC originated during Smalltalk development.

The MVC paradigm originated back in 1979, when Trygve Reenskaug was working on Smalltalk at Xerox PARC [May 1979] and has also been adopted as a [recommended design pattern for Cocoa applications](#)<sup>1</sup>.

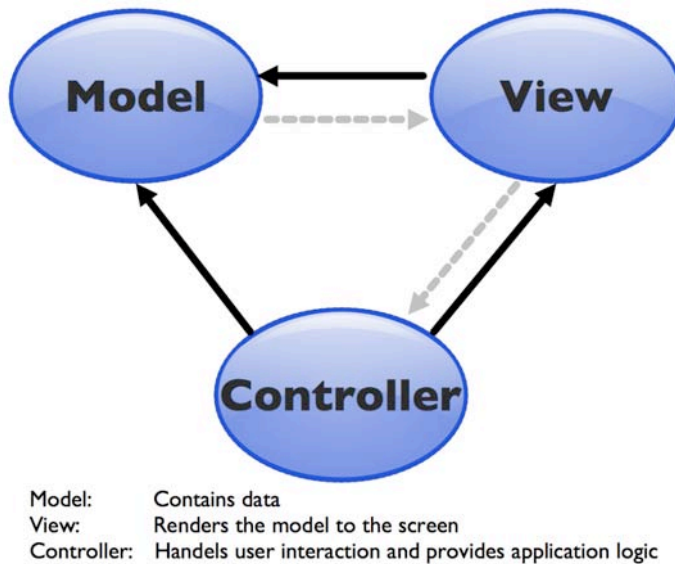
We apply MVC paradigm to Tactile Editor.

We implemented the MVC by providing structures to store the application data (model) and a custom view to display this data (View). In addition, we wrote a view controller that provides the application logic and ensures consistency between model and view of our application. Figure 6.2 shows which classes implement the three components of the MVC model.

The following section gives details about these components and the other classes we implemented for Tactile Editor.

<sup>1</sup>see Cocoa Fundamentals Guide, chapter 5, section 4





**Figure 6.1:** Model-View-Controller paradigm - adapted to Cocoa

## 6.3 Class Structure

Tactile Editor was implemented as a Cocoa application comprising a set of Objective-C-classes, which we will briefly describe here. Figure 6.2 shows the editor's main classes and their relationships.

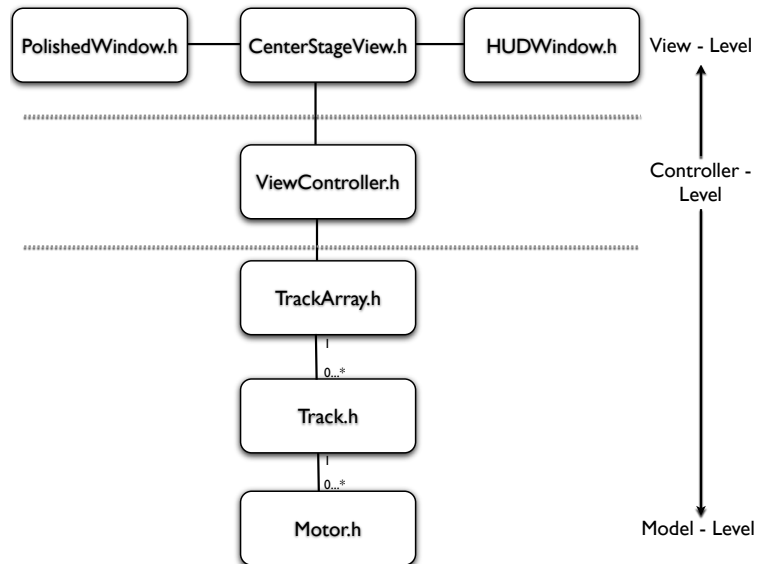
Tactile Editor is implemented as Cocoa application.

### **Main.m:**

This is the main class of the application; it just returns the `NSApplication` object.

### **CenterStageView.m:**

The center stage is the most important part and largest area of the user interface. `CenterStageView.m` displays the center stage and handles mouse or keyboard events. The view displays the application data



**Figure 6.2:** UML class diagram - main Tactile Editor classes

### ViewController.m:

This class serves as the controller for our application. It provides motor and track data for display in the center stage view and handles user interaction with the graphical user interface by providing interface actions for the application.

### TrackArray.m

This class provides the structural foundation for a pattern file and contains an array of track objects. In combination with the the motor and track classes, TrackArray.m forms the model of our application.

### Track.m

One single track is defined in this class and consists of an array of motor objects.

**Motor.m:**

A representation of a motor instance is defined within this class, along with the properties of a motor instance within the editor. Each motor has instance variables for start time, end time, intensity, and motor number.

**RoundedRect.m:**

Rounded rectangles are used to display motors and tracks in Tactile Editor. This class provides those rectangles for motor and track display in the view that contains the center stage.

**ToolTip.m:**

Tooltips are displayed, whenever a user drags a motor or one of the time indicators over the canvas. The tooltip class is responsible for setting up these tooltips.

**ToolTipTextField.m:**

Tooltips are implemented as separate, rectangular “mini” windows in Tactile Editor; they are displayed to the upper-right of the mouse cursor using the class `ToolTipTextField.m`.

**HUDWindow.m:**

Tactile Editor uses half-transparent, heads-up-display (HUD) windows to display additional information within its graphical user interface. These HUD windows are used for setting the preferences of motor representations in the canvas, when creating a new motor rectangle or when setting up the preferences for communication with sensors.

Objective-C header files for some of these classes can be found in appendix A.

In addition to these classes, I used source code provided by other programmers:

**PolishedWindow.m:**

Tactile Editor uses a custom window interface called "Polished Window", which was written by Matt Gemell and is provided by this class.

**Serial Port additions (AM\*.m):**

These are a number of classes, the main class being *AMSerialPort.m*, that provide serial port communication for Tactile Editor. Originally these were written by Andreas Mayer in 2001, but later on modified by several others. Sean McBride updates the classes in late 2007 and made the code 64bit compatible to allow use of the classes with Mac OS 10.5. We modified these classes to allow Tactile Editor bluetooth communication with Arduino devices via serial ports.

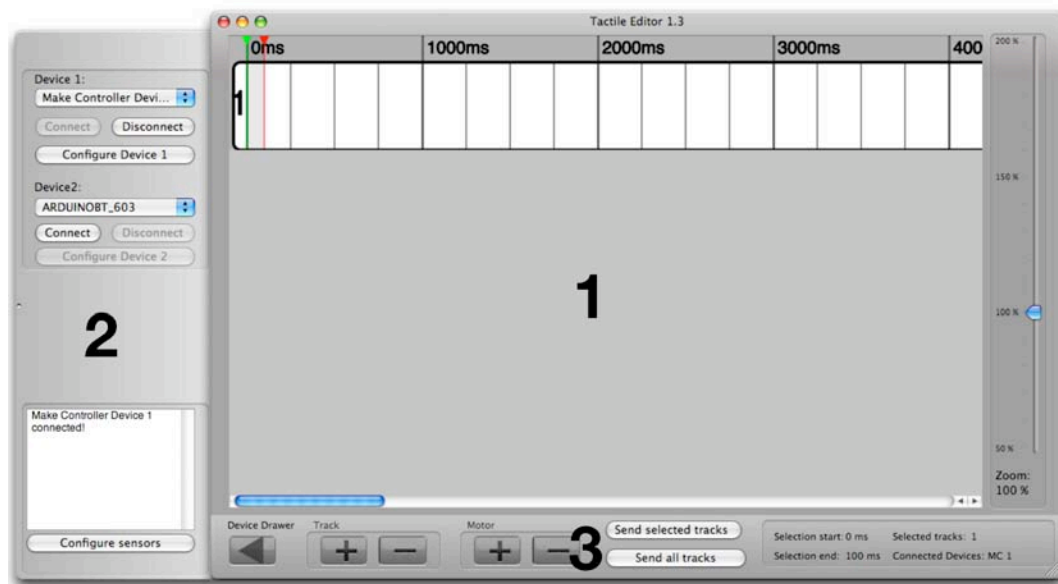
## 6.4 Tactile Editor GUI

The Tactile Editor GUI consists of four main parts:

GUI consists of four main parts: center stage, drawer, status bar, main menu.

1. The "center stage" or canvas - the main view of the editor which shows the tracks the user has created and edited
2. A "drawer" on the left-hand side that can be toggled via a button and contains controls to connect and configure different hardware devices
3. A bar at the bottom of the window that contains buttons for accessing frequently used editing functions and status information such as current time, connected devices, or selected tracks.
4. A main menu bar, which contains all available commands in several submenus

We will now discuss each of these parts in more detail and show how they effect a user working with Tactile Editor. As an overview, figure 6.3 shows Tactile Editor's user interface and how the different GUI parts are arranged.



**Figure 6.3:** Tactile Editor graphical user interface: center stage/canvas (1), drawer (2), status bar (3)

### 6.4.1 The Canvas

The central area of the application is called the center stage or the canvas - all editing tasks are focussed here. Taking in roughly 80 per cent of the editor's screen space, the canvas shows, along a timeline, user-created tracks containing motor representations as rectangles. As discussed in the previous chapter, these rectangles have several properties each representing a property of the associated motor: rectangle width representing time in milliseconds, height the PWM intensity in per cent, and color the motor number.

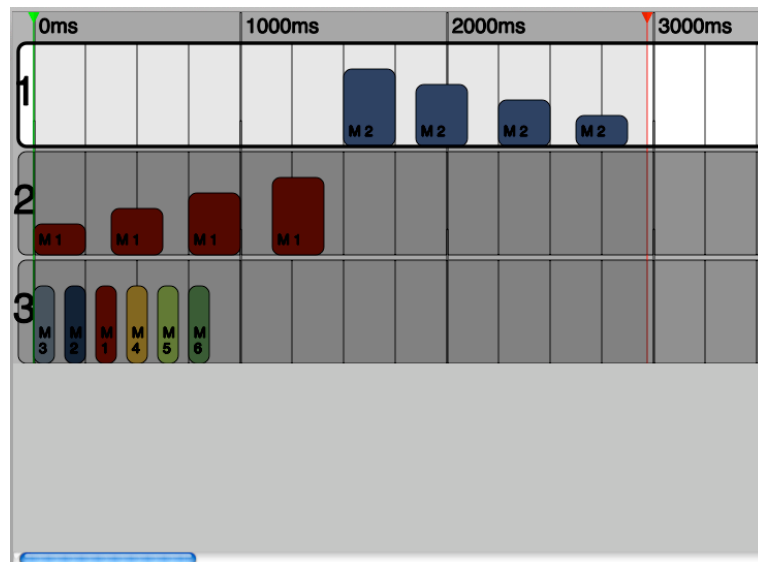
Figure 6.4 shows Tactile Editor's canvas while a user is working with the editor.

Central part of the GUI is called canvas or center stage.

We will focus on three typical editing tasks users will face when using Tactile Editor and show how these can be achieved by working on the canvas:

Three principal editing tasks

- Creating patterns
- Editing patterns



**Figure 6.4:** Canvas while user is designing vibrotactile patterns

- Sending patterns

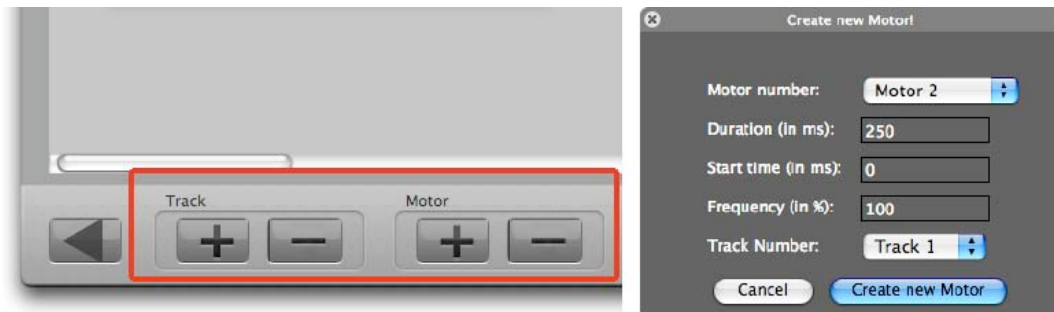
### Creating Patterns

Adding new tracks to the view

New tracks can be added to the editor by copying existing tracks or adding new tracks to the application. Existing tracks can be selected by clicking on the track in the view and copied via keyboard shortcut ("Option - C") or menu item ("Edit - Copy"). Several tracks can be selected at once by Shift-Click or dragging a selection rectangle over the required tracks. New tracks can be inserted via the "Add new track" - button or via the corresponding menu item.

Adding new motors to the view

Motors can be added by clicking the "Create new motor" - button or via the "Create new motor"- menu item. A transparent window appears, where motor number, start time, length, intensity, and track number can be specified, compare figure 6.5. Furthermore, motors can be added by selecting existing motors and copying or pasting them via keyboard shortcut or menu item respectively. Several motors can again be selected at once via shift-click or by



**Figure 6.5:** Creating patterns - buttons for adding/removing tracks and motors (left); HUD window for creating a new motor object (right)

dragging a selection rectangle around the motors to be copied.

### Editing Patterns

Existing patterns can be modified using a range of editing functions the system provides:

Motors can be modified by double-clicking on the specified motor rectangle within the center stage. Upon double-clicking, a half-transparent HUD window appears, in which the corresponding motor values can be specified (see figure 6.6).

Double-clicking a motor opens properties window.

Motors can be moved along the timeline or from one track to another by simply dragging and dropping the motor rectangle. Width and height of a motor rectangle can be resized by clicking on the appropriate border and dragging it to the desired position.

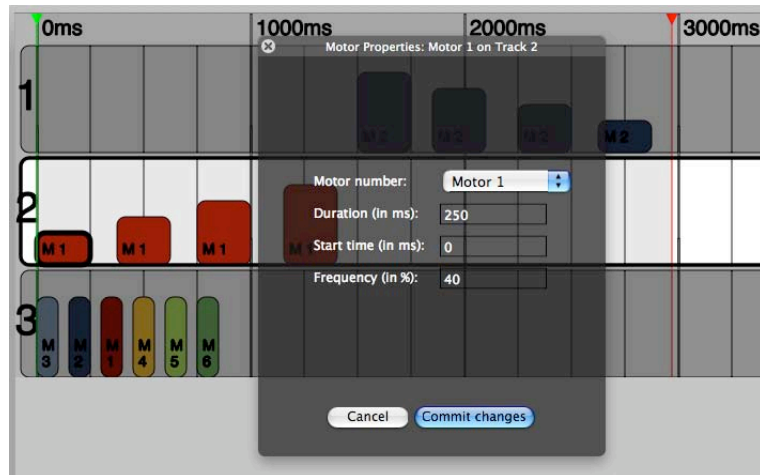
Motors can be resized via drag and drop.

As outlined above, new motors can be added by clicking the "Create new motor"-button, via the "Create new motor"-menu item or by copying and pasting selected motors.

### Sending Patterns

With Tactile Editor, there are two ways of sending patterns to the vibration motors attached to connected devices: Sending tracks as a whole or sending selected

Two ways to send patterns: whole tracks or selected regions.



**Figure 6.6:** Editing patterns - HUD window for modifying an existing motor vibration (first motor on track two was double-clicked)

regions only:

The button "Send selected tracks" sends all selected tracks to the devices where the corresponding vibration motors are connected. "Send all tracks" makes sure that all tracks within the center stage get sent to the vibration motors.

For sending certain parts of certain tracks only, users can delimit the area on the tracks by dragging the red and green markers along the timeline; the "Send marked patterns" button then sends the patterns within the selected area to the appropriate vibration motors.

Two markers indicate selected area of a track.

### 6.4.2 The Manage Devices Drawer

Drawer contains controls for managing hardware devices.

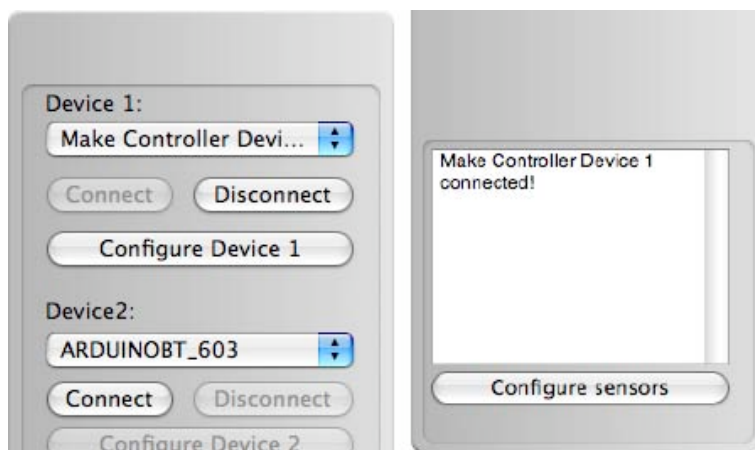
Hardware devices can be addressed from Tactile Editor via the drawer button on the left side of the status bar. This opens up a drawer to the left of the main window that contains controls to connect new hardware devices and disconnect or configure existing devices from Tactile Editor as shown in figure 6.3 before.



### Connecting and Disconnecting Hardware Devices

Users can select a device they want to connect to from the list within the popup button. If Tactile Editor is connected to one device already, this device is shown in the first popup and there appears another popup below where users can select a second device, as shown in figure 6.7 (left) below. Once a device is connected via the *Connect* button, Tactile Editor opens a serial port for sending and receiving messages between the editor and the connected device. The *Connect* button gets grayed out and the *Disconnect* button becomes available.

Tactile Editor opens serial port for communicating with a device.



**Figure 6.7:** *Manage Devices* drawer - upper part (left) with controls for (dis-)connecting devices, lower part (right) with status box and controls to address sensors

Disconnecting a device works similarly: For every device connected to Tactile Editor, a *Disconnect* button becomes available for that particular device. By clicking this button, Tactile Editor closes the port for that device and terminates the connection. The *Disconnect* button is greyed out and the connect button for that device becomes available again (see figure 6.7, left).

### Status Information

The text box within the control panel gives status information when connecting or disconnecting devices or

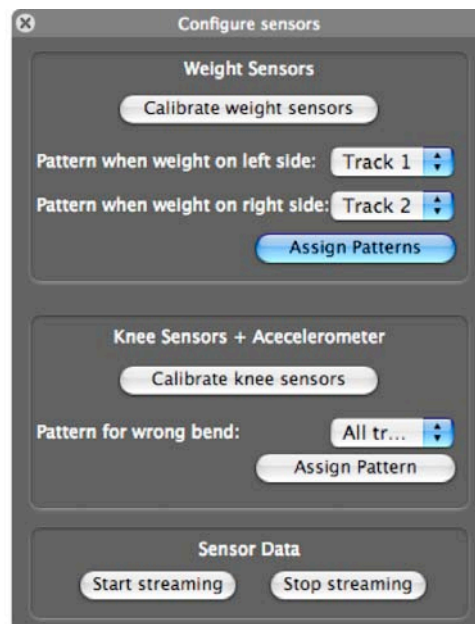
Text box gives information about device status.

when sending patterns to vibration motors, compare Figure 6.7 (right).

### Configuring Sensors

Tactile Editor can receive data from connected sensors. For our testing purposes, we configured Tactile Editor to receive data from six pressure sensors, two bend sensors, and one accelerometer. These sensors can be connected to an Arduino board, which then streams the sensor values to Tactile Editor. The editor can then trigger vibrotactile patterns in response to certain sensor values it receives.

Figure 6.8 shows the window that opens, when a user clicks the "Configure sensors" button in the drawer (compare figure 6.7, right).



**Figure 6.8:** HUD window to configure sensor details and stream sensor data

In this window, users can calibrate all connected sensors and start or stop streaming sensor data. In addition, patterns that should be triggered upon certain sensor values can be specified.

### How the Drawer Evolved

The drawer was added in a later iteration of Tactile Editor's design process: The first design put all controls related to connecting and disconnecting devices into a control panel that was visible all the time, which led to a rather crowded user interface and reduced the available space for the center stage. Users could not easily locate the individual controls, got confused by the many choices the interface offered and had less space available to edit patterns.

Drawer was added in a later iteration.

Compare Appendix B for earlier designs of Tactile Editor's graphical user interface.

As a result, we moved all controls related to managing hardware devices that Tactile Editor communicates with, into the "Manage Devices" drawer, that users can open whenever they need to access any of its controls. We deemed this appropriate, as these controls are used rather infrequently - probably just once during an editing task, when users want to decide, which devices should communicate with Tactile Editor.

Functions in drawer only needed occasionally.

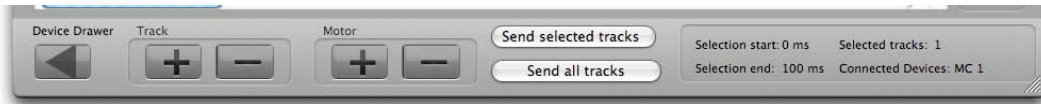
### 6.4.3 The Status Bar

Located at the bottom of the screen is a status bar that contains status information and gives users feedback about the status of the current editing process, see figure 6.9.

Status bar adds feedback.

This information includes the currently selected tracks, start and end time of the currently selected region between the red and green markers on the timeline, and the names of all hardware devices that are currently connected to Tactile Editor.

In addition, the bar contains buttons for opening the *Manage Devices* drawer and for adding or removing tracks and motors to or from the current pattern document.



**Figure 6.9:** Status bar of Tactile Editor

#### 6.4.4 The Menu Bar

Menu bar contents partly defined by Apple's HIG.

Customs menus: Pattern, Track, and Motor

As customary with Mac OS X applications, the main menu bar is located at the top of the screen and contains all available commands in several submenus. Some of them are determined by [Apples Human Interface Guidelines](#)<sup>2</sup>, others are specific for the Tactile Editor. According to these guidelines, "...the ordering of application-specific menus in the menu bar should reflect the natural hierarchy of objects in your application." As tracks contain motors, the ordering of menus from left to right is Apple, Tactile Editor, File, Edit, Track, Motor, Window and Help Menu, as shown in figure 6.10.



**Figure 6.10:** Tactile Editor's menu bar

Keyboard shortcuts for most menu items

Most commands within the menu structure correspond to a keyboard shortcut in order to increase the usability of the editor's interface (see section "Keyboard Shortcuts below").

We will not discuss the menu items in detail here and just give a short summary of the submenus, Tactile Editor provides:

##### *Tactile Editor Menu*

The Tactile Editor menu contains items that apply to the application as a whole rather than to a specific document or window, such as hiding or quitting the Tactile

<sup>2</sup>see Apple Human Interface Guidelines

Editor application.

#### *File Menu*

Menu items in the file menu apply to one single pattern file only; loading or saving a pattern file are examples for file menu items.

#### *Edit Menu*

The Edit menu provides commands that allow users to edit the contents of patterns and to share content via the clipboard, such as Cut, Copy, or Paste.

#### *Pattern Menu*

The Pattern menu contains commands that allow users to play or export patterns.

#### *Track Menu*

Important command for working with tracks, such as playing all or selected tracks, are included in this menu.

#### *Motor Menu*

The Motor menu contains important command for creating and editing motors; examples are adding or removing motors in the canvas.

#### *Window Menu*

The Window menu contains commands for organizing and managing Tactile Editor's windows.

#### *Help Menu*

Tactile Editor's help system (see section help book) can be accessed from this menu.

## 6.5 Undo Functionality

Undo for editing functions

Tactile Editor offers an undo system that covers editing functions such as adding objects, removing, or modifying tracks and motors. At the beginning of every editing operation, the current editing progress is stored and the title of the Undo menu item in the *Edit* menu is adjusted accordingly, as shown in figure 6.11.

Undo inserting motor	⌘Z	Undo deleting motor	⌘Z
Redo	⇧⌘Z	Redo	⇧⌘Z
Cut	⌘X	Cut	⌘X
Copy	⌘C	Copy	⌘C
Paste	⌘V	Paste	⌘V
Paste and Match Style	⇧⌘V	Paste and Match Style	⇧⌘V
Delete		Delete	
Select All Motors	⌘A	Select All Motors	⌘A
Find	▶	Find	▶
Spelling	▶	Spelling	▶
Special Characters...	⇧⌘T	Special Characters...	⇧⌘T

**Figure 6.11:** Undo String is adjusted - depending on the last user action; user inserts motor (left) and user deletes motor (right).

## 6.6 Tactile Editor Help System

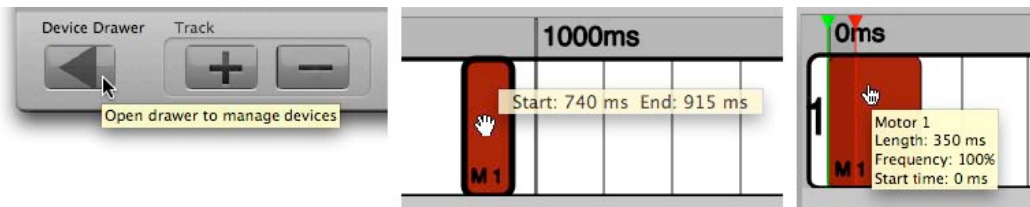
Several help features to assist the user.

In order to allow users to effectively use Tactile Editor, the system provides several help features that illustrate how to work with the editor. These include a complete help system and tooltips for all interface elements such as buttons, text-fields and sliders.

### 6.6.1 Tooltips

Two kinds of tooltips

Tactile Editor provides two kinds of tooltips to increase the usability of its interface: Tooltips for all interface elements



**Figure 6.12:** Tactile Editor tooltips - standard OS X tooltips (left), custom tooltips while dragging a motor (middle), motor tooltip when pointing at a motor rectangle (right)

and tooltips that appear within the center stage, offering users additional information about certain objects while editing patterns.

Whenever a user points at a certain control, such as a button or a slider, a short notice pops up that briefly summarizes the function of the associated element in three to eight words. These tooltips are implemented as OS X standard tooltips, that are associated with each user interface widget (see figure 6.12, left).

Standard OS widgets

Within the center stage, every rectangle representing a motor offers an associated tooltip (see figure 6.12, right). When the user points at a motor rectangle, a tooltip displays motor number, length, intensity, and start time of the motor. These tooltips were connected to cursor rectangles, which we calculated in the `resetCursorRects` method of our center stage

CursorRects used for motor tooltips.

Every time the user drags a motor across the canvas while editing a pattern, a tooltip displaying the start and end time of the moving motor appears to the top-right of the mouse cursor. When resizing a motor rectangle, the tooltip shows the current size (intensity when resizing the height, duration when resizing the width of a motor rectangle). In the same way, tooltips displaying the current time in milliseconds appear when users drag one of the two time markers along the time line (see figure 6.12, middle). These tooltips were implemented as separate windows that are connected to the `mouseDragged` method of our view.

Dragging tooltips implemented as separate windows.

## 6.6.2 Help Book

Help system can be accessed from within the editor.

Tactile Editor contains a comprehensive help system that explains how to use the toolkit. Help can be accessed from Tactile Editor via the "Tactile Editor Help" menu item from the "Help" menu or via the keyboard shortcut "Command - ?".

As shown in figure 6.13, the help book is divided into six parts:

Definition:  
*Help book structure*

### HELP BOOK STRUCTURE:

1. **Introduction** - Illustration of what Tactile Editor offers and how to get started
2. **Basic Structures** - How motors are represented in Tactile Editor and how to access those representations
3. **Hardware Devices** - Essential information on how to connect and access hardware devices from the editor
4. **Editing Patterns** - How to create, modify, load, and store vibration patterns with Tactile Editor
5. **Sending Patterns** - How to select the correct patterns and send them to connected devices
6. **Miscellaneous** - Frequently asked questions



[Tactile Editor Help Home](#) >



## Tactile Editor Help

As a starting point for first-time-users as well as a reference point for more experienced designers, these help pages offer a comprehensive overview about the capabilities of the Tactile Editor Toolkit.

The first part introduces the main functions of the toolkit and how to access them. Part two shows how the concept of vibrating motors is mapped to the Tactile Editor and part three shows how to connect different hardware devices and access them from the editor.

### [Part 1: Introduction](#)

Illustration of what Tactile Editor offers and how to get started.

### [Part 2: Basic Structures](#)

How motors are represented in Tactile Editor and how to access those representations.

### [Part 3: Hardware Devices](#)

Essential information on how to connect and access hardware devices from the editor.

### [Part 4: Editing Patterns](#)

How to create, modify, load, and store vibration patterns

### [Part 5: Sending Patterns](#)

How to select the correct patterns and send them to connected devices.

### [Part 6: Miscellaneous](#)

FAQ, etc.

Part four helps with problems occurring while editing patterns, and part five shows how to send the patterns to different hardware devices. Finally, part six deals with with FAQ, etc.

### Keyboard Shortcuts

View a list of [Keyboard Shortcuts](#) for the Haptic Editor

Figure 6.13: Tactile Editor help system - start page

### 6.6.3 Keyboard Shortcuts

All common editing functions can be easily accessed via keyboard shortcuts, thus allowing users faster, more efficient editing of vibrotactile patterns and improving their workflow. By using a keyboard shortcut, experienced users can speed up execution times for most tasks of the editor and reduce a series of mouse operations to just one single keyboard shortcut.

Keyboard shortcut can speed up editing tasks.

Some of the commands are common Mac OS X keyboard shortcuts for accessing menu commands such as "Command - C" for copying items (such as tracks or motors in our application), "Command - P" for pasting items or "Command - Q" for quitting the application.

Others are application-specific keyboard shortcuts to access certain functions of the editor such as adding a new track to the canvas (shortcut "Command - T"), adding a motor to an existing track within the canvas (shortcut "Command - Shift - M") or playing all tracks within the canvas ("Command - Shift - A").

Application specific shortcuts

All functions that can be accessed via shortcuts are listed in appendix B.

## 6.7 Communication with Hardware Platforms

Specific protocol for communication with hardware devices.

In order to send the patterns designed with the editor to attached hardware devices, we need a specified protocol for communication between the software and the hardware devices. As Tactile Editor explicitly supports the Arduino and the Make Controller hardware toolkits, we specified communication protocols for these two hardware platforms.

### 6.7.1 Arduino

Communication via serial ports

All communication with Arduino BT boards is established via a Bluetooth connection. Tactile Editor opens a serial port for every Arduino to send commands and listen for incoming messages from connected sensors or other software tools.

We defined the following protocol for communication with Arduino boards in order to turn motors on or off:

Definition:  
*Message Format Arduino*

#### MESSAGE FORMAT ARDUINO:

##### To control motors:

**M number [A|D] value \r\n**

M = motor

number = 1..6 (Motor number, not pin number)

A = analogWrite (PWM)

D = digitalWrite

value = duty at which motor is driven:

0..255 for analogWrite: 0 = off, 1..255 = on;

0..1 for digitalWrite: 0 = off, 1 = on

##### To send delays:

**D time**

D = delay

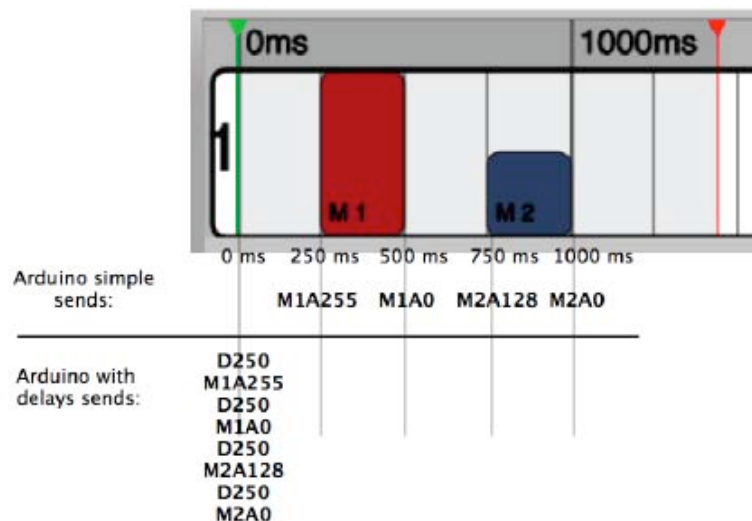
time = delay length in ms

We used two slightly different formats for sending patterns to Arduino devices: one for sending commands directly to the board from Tactile Editor (just called *simple Arduino protocol*) and for exporting the patterns to a file (called *Arduino with delays*).

Two Arduino protocols

When using the simple Arduino protocol, we start a timer within Tactile Editor and send the commands as the timer reaches the time, when a motor should be switched on or off. The Arduino protocol using delays on the other hand sends all commands at time 0, inserting delays to keep the motor vibrations synchronous with the patterns the user created. In case the patterns to be sent contain more than 16 commands, we have to buffer these commands and send them in blocks of 16. This is due to the buffer of the Arduino board being restricted to 128 Bytes - roughly the size of 16 commands.

Figure 6.14 shows an example pattern and how the two Arduino protocols send these patterns.



**Figure 6.14:** Comparison of the two Arduino protocols for sending patterns (simple) and exporting patterns (with delays)

### 6.7.2 Make Controller Kit

Communication via  
OSC messages

As mentioned before, Tactile Editor communicates with Make Controller boards via OSC messages. We realized the OSC protocol for Mac OS X using the [OBJCOSC framework](#)<sup>3</sup>, an Objective-C wrapper around the original [CNMAT Open Sound Control client code](#)<sup>4</sup>.

OSC messages sent from the tool to a Make Controller Board have the following format:

Definition:  
*OSC message  
format for  
MakeController*

#### OSC MESSAGE FORMAT FOR MAKECONTROLLER:

##### To control motors:

```
/pwmout/[number]/duty [value]
```

`pwmout` = PWM controller on Make Controller Board  
`number` = 0..3 - motor number  
`duty` = duty property of connected motor  
`value` = 0..1023 - duty at which motor is driven

##### To activate/deactivate PWM Out:

```
/pwmout/[number]/active [value]
```

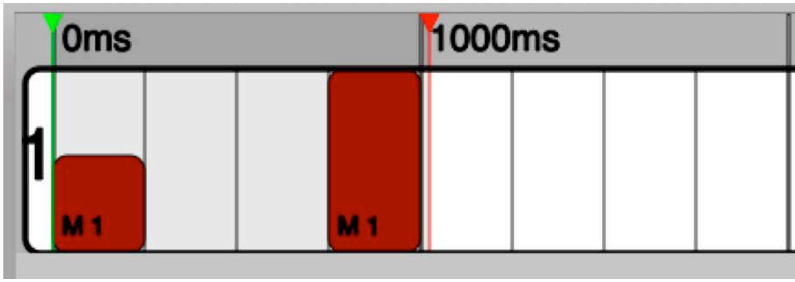
`pwmout` = PWM controller on Make Controller Board  
`number` = 0..3 - motor number  
`active` = active state property of PWM Out  
`value` = 0..1, 0 (false) = off, 1 (true) = on

To illustrate the use of these commands, we give a brief example that shows, how a pattern is "translated" into commands that are sent to Arduino or MakeController boards.

<sup>3</sup>see <http://www.mat.ucsb.edu/~c.ramakr/illposed/objcosc.html>

<sup>4</sup>see <http://archive.cnmat.berkeley.edu/OpenSoundControl/Kit/>

Table 6.1 below shows a simple sample pattern, along with the Arduino and Make Controller commands belonging to this pattern.



Time	Arduino simple	Make Controller	Function
0ms		/pwmout/0/active 1	Enable PWM for MakeController board
0ms	M1A127	/pwmout/0/duty 512	Motor 1 on
250ms	M1A0	/pwmout/0/duty 0	Motor 1 off
750ms	M1A255	/pwmout/0/duty 1024	Motor 1 on
1000ms	M1A0	/pwmout/0/duty 0	Motor 1 off
1000ms		/pwmout/0/active 0	Disable PWM for Make Controller board

**Table 6.1:** Protocol for Arduino and MakeController



## Chapter 7

# Evaluation

*“Experience is that marvelous thing that enables you to recognize a mistake when you make it again.”*

—Franklin P. Jones, american satirist, (1853-1935)

We evaluated Tactile Editor in order to measure the value and usefulness of the tool and in order to test and improve the usability of its interface.

Therefore, earlier versions of the editor were tested and changes were made - primarily to the editor’s user interface. As an example, we moved all controls related to connecting and managing hardware devices that are connected to the editor from the main window to a panel that can be accessed via a button in the main window (see figure 7.1). Likewise, certain infrequently used functions were moved from buttons in the main window to items in the main menu - thus freeing additional space for the center stage of our application (see figure 7.1).

Interface changed repeatedly

The evolution of Tactile Editor’s graphical user interface over the different iterations can be seen in appendix D.

Evolution of Tactile Editor’s GUI

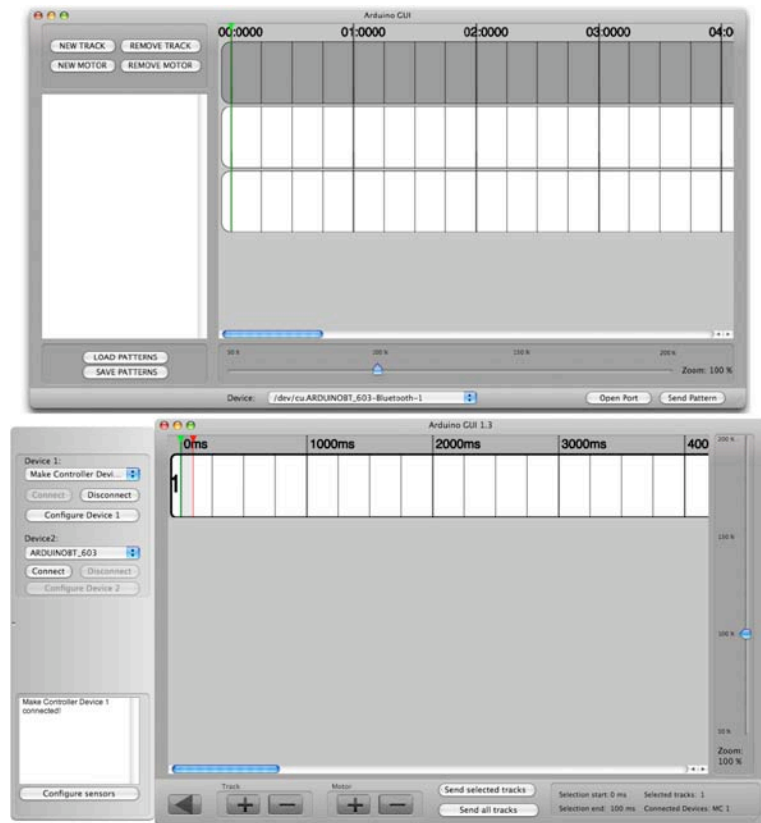


Figure 7.1: Changes made to Tactile Editor's interface

## 7.1 User Testing

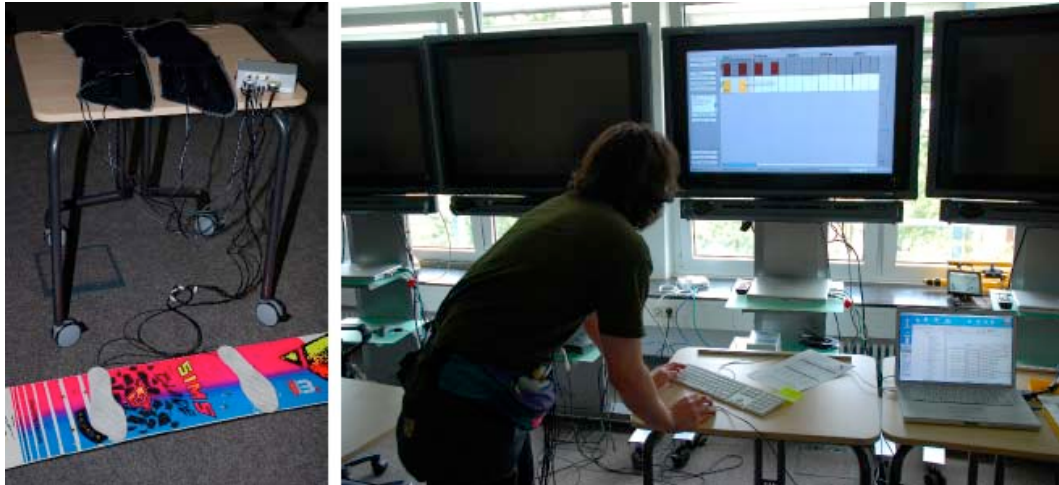
First-use study in our lab

To further improve Tactile Editor, the tool was tested in a first-use study in our Media Space Lab to determine threshold and ceiling for novice users, as well as to measure usability.

As mentioned in section 3.5—"Toolkit Evaluation", a low threshold means, the tool is easy to use - especially for first-time users that have not been exposed to the tool before. The ceiling limits the complexity of the systems and prototypes that are designed with the tool.

Our user tests included evaluating both the editor itself and the editor as part of a toolkit including Arduino boards, various sensors, and vibration motors.





**Figure 7.2:** Setup for the user test to evaluate Tactile Editor - hardware (left) and workstation (right)

### 7.1.1 Setup

Tests were conducted in the Media Space of our chair and scheduled to take 45 - 60 minutes per test. Participants were seated at a workstation with a keyboard, a mouse, and a screen, as shown in figure 7.2 (right). The hardware, consisting of two Arduino boxes, a snowboard with pressure sensors, and two barrel cuffs equipped with three vibration motors each, was placed next to the workstation (see figure 7.2, left).

We equipped the Arduino boxes with firmware programs that could process sensor data, stream the data to Tactile Editor and control the vibration motors upon commands sent from Tactile Editor. Users could focus on designing and testing the patterns and did not have to deal with technical implementation details.

All participants were given the same tasks they were asked to complete. We explained task and setup of the test to each user before the test to clarify the premises and goals of the user test. Furthermore, we pointed out to all participants that we tested the editor's interface, not their performance.

User test sessions

Hardware used in tests

### 7.1.2 Users

All participants had basic HCI knowledge.

The user group consisted of twelve students, all of which were postgraduate students, but with various educational backgrounds: four from Computer Science, four from Technical Communication, two from Media Informatics, and two from Engineering fields. All students had some HCI experience and were familiar with interaction design and the concept of prototyping.

Three of the students were female and the other nine were male; participants' ages ranged from 22 to 29 years. Eight participants stated they had previous experience using track-based editors.

### 7.1.3 Tasks

Users were given three tasks to complete.

Users were given a short introduction about Tactile Editor and the hardware they were about to use during the test session. After being able to play around with the system for five minutes, all users received three tasks they were asked to complete during the test.

Two of these tasks were about reproducing and testing predefined vibration patterns with the Tactile Editor interface. Users were given these predefined patterns as a table showing start times, durations, intensities, and motor numbers of the different motor objects (compare figure C.4 in Appendix C). A third task challenged users to come up with own patterns for a given situation and test those patterns for effectiveness.

First task

The first task was creating and testing a simple, predefined pattern that used just one track and two vibration motors.

Second task

The second task involved creating and testing a more complex, predefined pattern that used several tracks and all six motors, that were connected to an Arduino box.



**Figure 7.3:** User working with Tactile Editor

The last task required participants to devise patterns that alert a user that his weight is not distributed correctly, for example during a snowboard ride.

Third task: exploring and creating own patterns

Users were asked to think of a pattern that alerts the user when his weight is shifted too much towards the left foot and a pattern that tells a user his weight is shifted too much towards the right foot.

Subsequently, users had to create these patterns using Tactile Editor before finally testing the patterns using the pressure sensors attached to the snowboard and the vibration motors attached to their thighs (see figure 7.3, left). Users could then adjust and refine their patterns based on the vibrotactile feedback they received (compare figure 7.3, right).

### 7.1.4 Results

Questionnaire helps evaluating feedback.

After the test, we asked all participants to fill out a questionnaire to help us evaluating Tactile Editor (see Appendix C). Questions were split into three parts: The first section covered questions regarding the completion of the three tasks. The second section contained four questions (Q1 to Q4) about the usefulness of Tactile Editor:

- Q1 asked whether Tactile Editor reduced prototyping times.
- Q2 asked whether Tactile editor made users experiment more.
- Q3 asked whether Tactile Editor reduced the time to program individual vibration motors.
- Q4 asked whether Tactile Editor helped experiencing and understanding tactile feedback.

The third part included questions about problems that occurred during the test, features that participants found particularly helpful, and possible improvements to the toolkit.

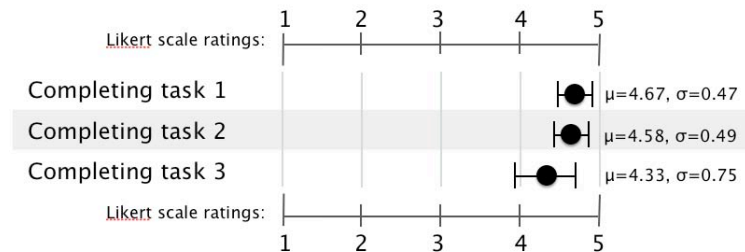
We will briefly sum up the results for each of those three parts.

#### Statistical analysis

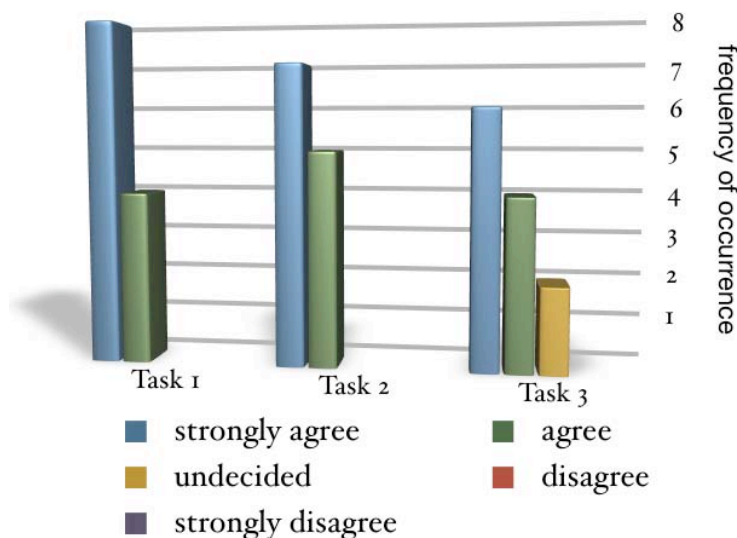
Completion of first, second, and third task

All participants stated they had no problems completing the first task (mean = 4.7 on a five-point Likert scale, median = 5, standard deviation  $\sigma = 0.47$ ) and the second task (mean = 4.6, median = 5,  $\sigma = 0.49$ ). The majority of participants also completed the third task easily (mean = 4.3, median = 4.5,  $\sigma = 0.75$ ), just two participants were uncertain while designing and testing their patterns. The results are displayed in figure 7.4 and figure 7.5.

When evaluating Tactile Editor itself and as part of a toolkit to design vibrotactile patterns, participants rated it highly

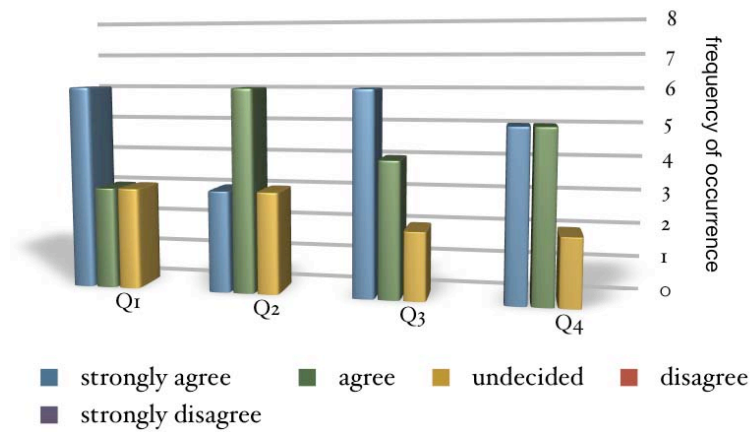


**Figure 7.4:** Statistical results for task completion



**Figure 7.5:** Questionnaire results for task completion

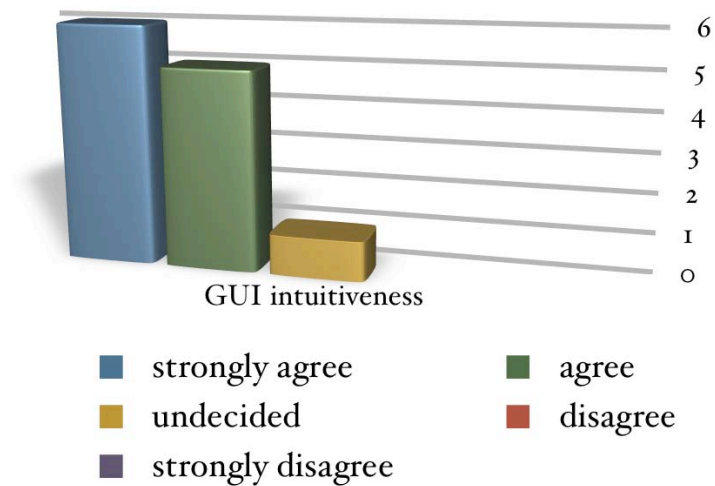
for understanding vibration motors and experiencing tactile feedback (Q 4: mean = 4.33, median = 4,  $\sigma = 0.47$ ). For reducing the time it takes to build a working prototype (Q1: mean = 4.25, median = 4.5) and the time to program individual vibration motors (Q3: mean = 4.33, median = 4.5), ratings were similar, but spread over a wider range (Q1:  $\sigma = 0.83$  and Q3:  $\sigma = 0.75$  respectively). Results were less conclusive when participants were asked if Tactile Editor helped them experiment more (Q2: mean = 4, median = 4,  $\sigma = 0.71$ ). Results for Q1 to Q4 are shown in figures 7.6.



**Figure 7.6:** Questionnaire results for toolkit evaluation

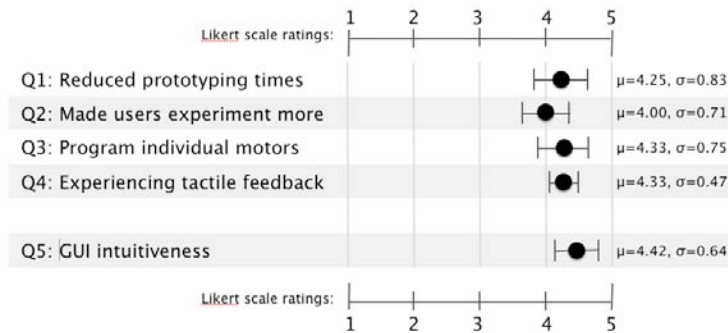
#### GUI evaluation

Participants rated Tactile Editor's user interface highly in terms of intuitiveness (Q5: mean = 4.42, median = 4.5,  $\sigma = 0.64$ ), as shown in figure 7.7. Figure 7.8 summarizes the statistical values for questions Q1 to Q5.



**Figure 7.7:** Questionnaire results for GUI evaluation

#### Suggestions and improvements



**Figure 7.8:** Statistical results for toolkit evaluation (Q1 - Q4) and GUI evaluation (Q5)

In addition to these statistical data, we gained valuable feedback from the open ended questions in the third part, both about what users liked and what should be improved. Users appreciated the resemblance to known track-based tools, the timeline metaphor, the drag and drop interactions, and the self-explanatory user interface.

Valuable feedback on improving Tactile Editor from open ended questions

Furthermore, users suggested a few slight improvements to Tactile Editor. Primarily, these regarded more intuitive interaction metaphors and clearer visible feedback. Based on these suggestions, we redesigned some parts of Tactile Editor to implement the features, users requested. As an example, we changed the properties window of a motor rectangle responding to a double-click instead of a single click as several users suggested this to be more intuitive. We also added visible feedback about the available editing options at any moment: The mouse cursor shape now indicates the editing option available; if, for example, a users moves the cursor over a motor rectangle border, the cursor shape changes to the appropriate resizing shape.

Users requested clearer visible feedback.

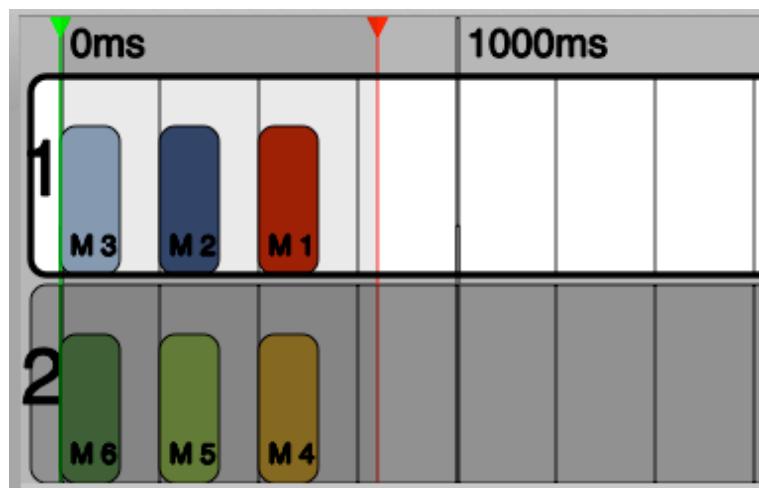
### User-created patterns

We analyzed and compared the patterns, users created during task three. Surprisingly, the resulting patterns were quite similar among all participants of the test.

As our main focus during the test was to observe par-

ticipants using Tactile Editor, we only discuss these patterns briefly here and give a few sample patterns that users designed during the weight balance task (task three) of the test.

All but one participant designed patterns, that gave feedback on the side, where too much weight was shifted to. The most common pattern can be seen in figure 7.9. The first track contains the pattern moving up the right thigh (motors 1, 2, and 3 were attached from top to bottom to the right thigh), track 2 contains the pattern for the left thigh (motors 4, 5, and 6 attached from top to bottom to the left thigh).



**Figure 7.9:** Patterns created by users in task 3 - for weight on the right side (Track1) and weight on the left side (Track2)

Most users created a variation of these patterns - some only using one vibration motor on each side to deliberately keep the patterns simple.

## Conclusions

There is an inherent bias in filling out post-test questionnaires like ours. Personal contact between tester and developer may result in a more positive result than any-



mous participation, for example by internet. Therefore, we need to take special care when interpreting the results of the survey.

Nevertheless, we are convinced, the predominantly positive feedback to Q1, Q3, and Q5 suggests that Tactile Editor is well suited to design and test vibrotactile feedback patterns.

Largely positive  
feedback

The fact that all users could instantly complete all three tasks without prior exposure to Tactile Editor, suggests that the tool offers a very low threshold for designers. This was supported by several user comments, who mentioned the timeline metaphor helped them design and test the vibrotactile patterns.



## Chapter 8

# Summary and Future Work

*“The day after tomorrow is the third day of the rest of your life.”*

—George Carlin (American stand-up comedian, actor, and author), *“Sometimes a Little Brain Damage Can Help”*, 1984

To conclude this thesis, we summarize the primary contributions in this work before giving an outlook on the future work that remains to be done.

### 8.1 Summary and Contributions

We introduced the field of haptics and, in particular, the research area of vibrotactile feedback as an alternative or supplementary communication channel. After outlining the physiological and perceptual foundations for tactile feedback, we reviewed previous work that had been done and related to the work in this thesis.

In chapters four to six, we designed and implemented Tactile Editor, our tool to design and test vibrotactile

Primary contributions feedback patterns, and adopted the DIA cycle for our work. Tactile Editor's primary contributions are:

- Allow creating haptic content in the form of vibrotactile patterns.
- Support multiple haptic output devices, address them simultaneously or consecutively and thus allow rapid prototyping of vibrotactile feedback systems.
- Provide a simple, track-based graphical user interface that offers an intuitive interaction metaphor and easy, intuitive drag and drop interactions to facilitate designing and testing vibrotactile patterns.

User tests

Finally, we tested our design in a first-use study with twelve participants in our Media Space Lab to evaluate and further improve the usability of Tactile Editor. These user tests gave us an insight on how users work with Tactile Editor and provided us with valuable feedback on how to improve the tool. We incorporated the suggestions for improvement gained from our user tests into the final version of Tactile Editor and present any open issues in the following section.

## 8.2 Future Work

Future work on Tactile Editor should focus on improving Tactile Editor and integrating the tool into an environment of hardware devices and software tools.

### 8.2.1 Export Function for Patterns

We started with this after evaluating the toolkit and added a preliminary export function for patterns, that exports a patterns to a file that contains commands according to the protocol we specified in chapter six.

### 8.2.2 Communication with Other Tools

Furthermore, Tactile Editor should be able to communicate with other tools via OSC messages. As an example, a sensor system such as Exemplar or iSense sends OSC messages to Tactile Editor upon receiving certain sensor values. Tactile Editor thereupon executes a certain pattern on one of its tracks.

We are currently working on implementing this feature; our current idea is to send an OSC message to Tactile Editor, for example `/TactileEdit/KneesBent`. The editor then searches for a track called "KneesBent" and triggers the pattern on that track. Alternatively, messages can address track by numbers (e.g. `/TactileEdit/Track1`).

OSC messages  
trigger patterns

### 8.2.3 Control Points

Another idea to enhance Tactile Editor is to loosen the strict mapping from rectangles to motor vibrations. Instead, we could represent motor vibrations in Tactile Editor by different polygons instead of rectangles only.

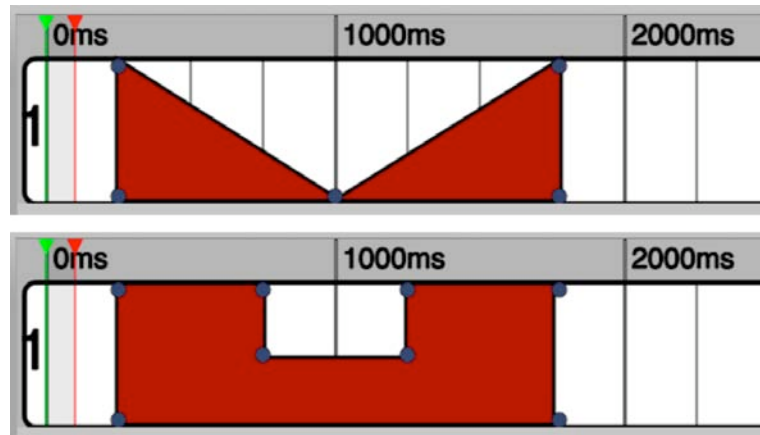
Polygons

This approach would broaden the range of patterns that can be created using Tactile Editor: users could create patterns, that can not be designed out of rectangles only. Whereas rectangles restrict changing the vibration intensity in discrete steps, using polygons would allow changing the intensity continuously, see figure 8.1 (top) for an example of such a pattern.

Furthermore, we could define shapes as seen in 8.1 (bottom). This pattern can be easily defined with one polygon instead of defining three rectangles.

Editing pattern becomes more complex, when we allow the mapping of motor vibrations to arbitrary polygons instead of rectangles only. A common technique to solve this problem is used in graphical editors: so-called *control points*.

Control points



**Figure 8.1:** Two sample patterns using polygons instead of rectangles. Control points are marked in blue.

Control points define the path of a shape. Lines, for example, have two control points - one at the start, one at the end - and rectangles are defined by four control points at the corners. By adding or removing control points, users can change the shape of, for example, a rectangle to a triangle or an arbitrary polygon.

#### 8.2.4 Perception of Tactile Patterns

During our user tests, we observed participants testing the patterns they created with vibration motors that were connected to Arduino boxes (see 7.1.1—“Setup”). Therefore, we could gain insight on how participants perceived the vibrotactile feedback patterns.

More work needs to be done in this direction. In particular, the question, whether humans instinctually react to certain vibrotactile patterns or whether these patterns have to be learned, remains an interesting research field.

## Appendix A

# Basic Data Structures for Tactile Editor

```
@interface TrackArray : NSObject
{
    NSMutableArray* tracks_;
}
-(NSMutableArray*)tracks;
-(void)setTracks:(NSMutableArray*)tracks;

+ (TrackArray*)createTrack;
- (int)count;
@end
```

```
@interface Track : NSObject
{
    NSMutableArray* motors_;
}
-(NSMutableArray*)motors;
-(void)setMotors:(NSMutableArray*)motors;

+ (Track*)newTrack;
- (int)count;
- (int)length;

@end
```

```
@interface Motor : NSObject
{
    Motor* motor;
    (int) startTime_;
    (int) endTime_;
    (int) intensity_;
    (int) motorNumber_;
}
-(int) startTime;
-(void) setStartTime: (int) startTime;

-(int) endTime;
-(void) setEndTime: (int) endTime;

-(int) intensity;
-(void) setIntensity: (int) intensity;

-(int) motorNumber;
-(void) setMotorNumber: (int) motorNumber;

- (Motor*) initWithStartTime: (int) startTime
    endTime: (int) endTime
    intensity: (int) intensity
    motorNumber: (int) motorNumber;

+ (Motor*) newMotorWithStartTime: (int) startTime
    endTime: (int) endTime
    intensity: (int) intensity
    motorNumber: (int) motorNumber;
```



## **Appendix B**

# **List of Keyboard Shortcuts**

Table B.1: List of keyboard shortcuts

Keyboard Shortcut	Description
Haptic Editor menu	
Command - H	Hide application
Command - Q	Quit application
File menu	
Command - N	New file
Command - O	Open saved file
Command - S	Save current file
Command - Shift - S	Save as...
Command -P	Print current patterns
Edit menu	
Command - Z	Undo last editing action
Command - A	Select all motors
Command - C	Copy
Command - P	Paste
Command - X	Cut
Pattern menu	
Command - Shift - P	Play selected tracks
Command - Shift - A	Play all tracks
Command - E	Export selected patterns
Command - Shift - E	Export all patterns
Track menu	
Command - T	New track
Command - Delete	Delete selected tracks
Command - Shift -Delete	Delete all tracks
Motor menu	
Command - M	New motor
Command - Delete	Delete selected motors
Help menu	
Command - ?	Open help book

## Appendix C

# Questionnaire and Task List Tactile Editor User Study

**Questionnaire - User Test Haptic Editor - Markus Jonas**Date/Time: **Part I - User Information**Age: Gender:  Male  FemaleField of Study: **Have you had prior experience with vibro-tactile feedback?** never  seldom  sometimes  often**Have you used track-based editors (such as GarageBand, Audacity, Final Cut etc.) to edit audio or video tracks before?** never  seldom  sometimes  often**Part 2 - Completing Tasks****I could complete the first task easily**
 strongly agree   
  agree   
  undecided   
  disagree   
  strongly disagree
**I could complete the second task easily**
 strongly agree   
  agree   
  undecided   
  disagree   
  strongly disagree
**I could complete the third task easily**
 strongly agree   
  agree   
  undecided   
  disagree   
  strongly disagree

Figure C.1: Questionnaire user study Tactile Editor - page 1

---

**Questionnaire - User Test Haptic Editor - Markus Jonas**

---

**Part 3 - Toolkit Evaluation**

**The toolkit decreases the time it takes to build a working system including sensors and vibration motors**

strongly agree     agree     undecided     disagree     strongly disagree

**The editor makes me experiment more**

strongly agree     agree     undecided     disagree     strongly disagree

**The editor reduces the time it takes to program individual vibration motors**

strongly agree     agree     undecided     disagree     strongly disagree

**The toolkit helps to understand how vibrotactile feedback works**

strongly agree     agree     undecided     disagree     strongly disagree

---

**Part 4 - GUI Evaluation**

**The user interface of the editor was easy to use and self-explanatory**

strongly agree     agree     undecided     disagree     strongly disagree

**Did you encounter any serious problems while using the editor?**

No     Yes, the following:

Figure C.2: Questionnaire user study Tactile Editor - page 2

**Questionnaire - User Test Haptic Editor - Markus Jonas**

---

**What did you like about the editor interface?**

**Do you have any other comment and/or suggestions on improving the toolkit?**

**Thank you for your participation!**

**Task List - User Test Tactile Editor - Markus Jonas**

## User Test Tasks

### Task 1 - Design and test a simple vibrotactile feedback pattern

Design the following simple one-track pattern using Tactile Editor:

Motor	Duration	Start Time	Frequency	Track
Motor 1	250 ms	0 ms	100%	Track 1
Motor 2	100 ms	250 ms	100%	Track 1
Motor 1	200 ms	400 ms	100%	Track 1
Motor 2	100 ms	650 ms	100%	Track 1

Test the pattern using the motors connected to the first box!

### Task 2 - Design and test a multi-track vibrotactile feedback pattern

Design the following multi-track pattern using Tactile Editor:

Motor	Duration	Start Time	Frequency	Track
Motor 1	250 ms	0 ms	100%	Track 1
Motor 1	250 ms	500 ms	100%	Track 1
Motor 1	250 ms	1000 ms	100%	Track 1
Motor 1	250 ms	1500 ms	100%	Track 1
Motor 3	100 ms	0 ms	100%	Track 2
Motor 2	100 ms	150 ms	100%	Track 2
Motor 1	100 ms	300 ms	100%	Track 2
Motor 4	100 ms	450 ms	100%	Track 2
Motor 5	100 ms	600 ms	100%	Track 2
Motor 6	100 ms	750 ms	100%	Track 2

Test the pattern using the motors connected to the first box!

### Task 3 - Develop and test a pattern for weight balance control

Think of a pattern that alerts the user when his weight is not distributed correctly, for example during a snowboard ride. Use the three vibration motors connected to each of the user's thighs to provide tactile feedback when the weight is shifted too much to the left or the right foot. The user should receive no feedback when the weight is distributed evenly between both feet.

Test the pattern using the weight sensors and motors attached to the first box!

Now design an alternative pattern for this task!

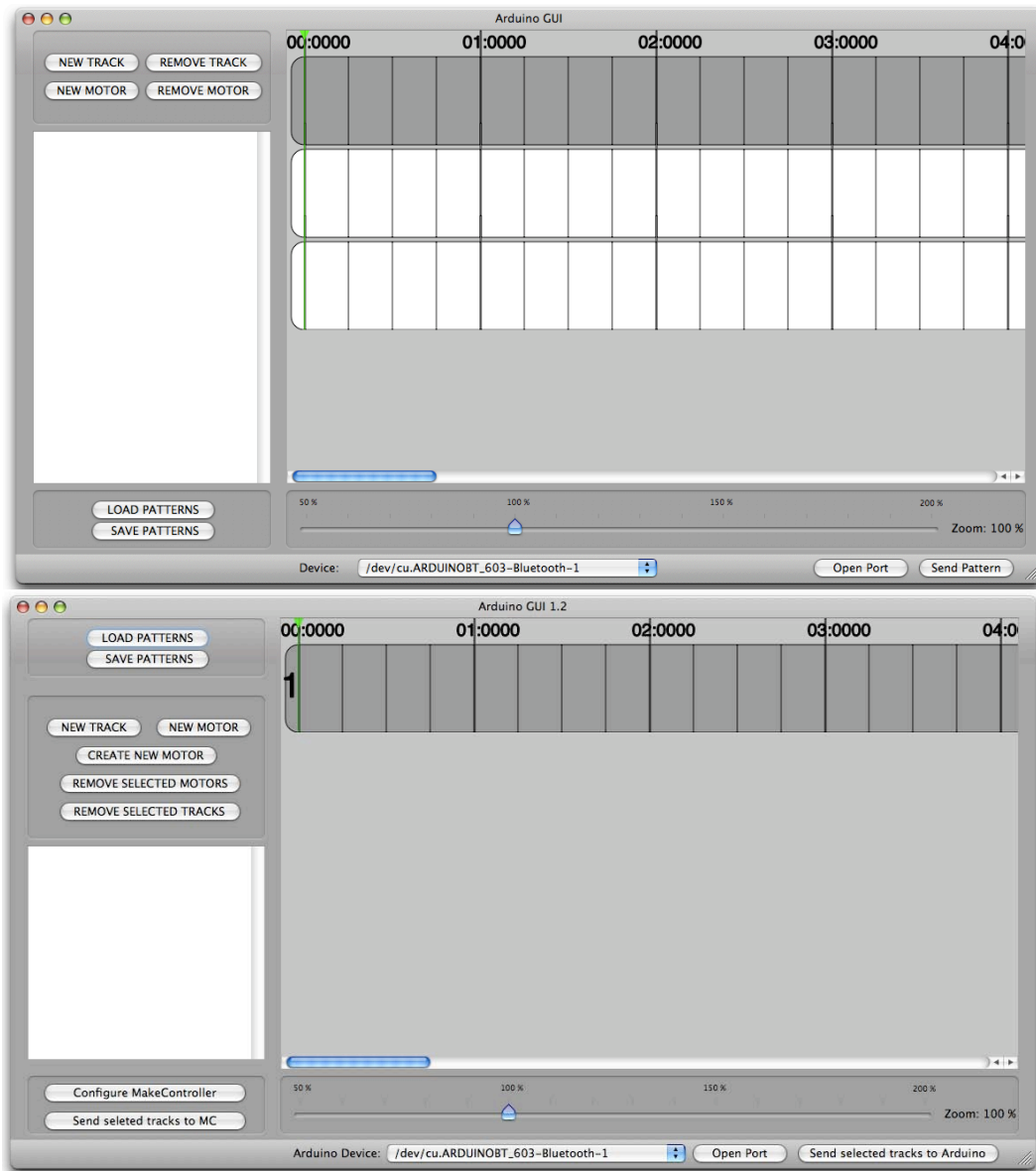
**Figure C.4:** Task list for the user study of Tactile Editor



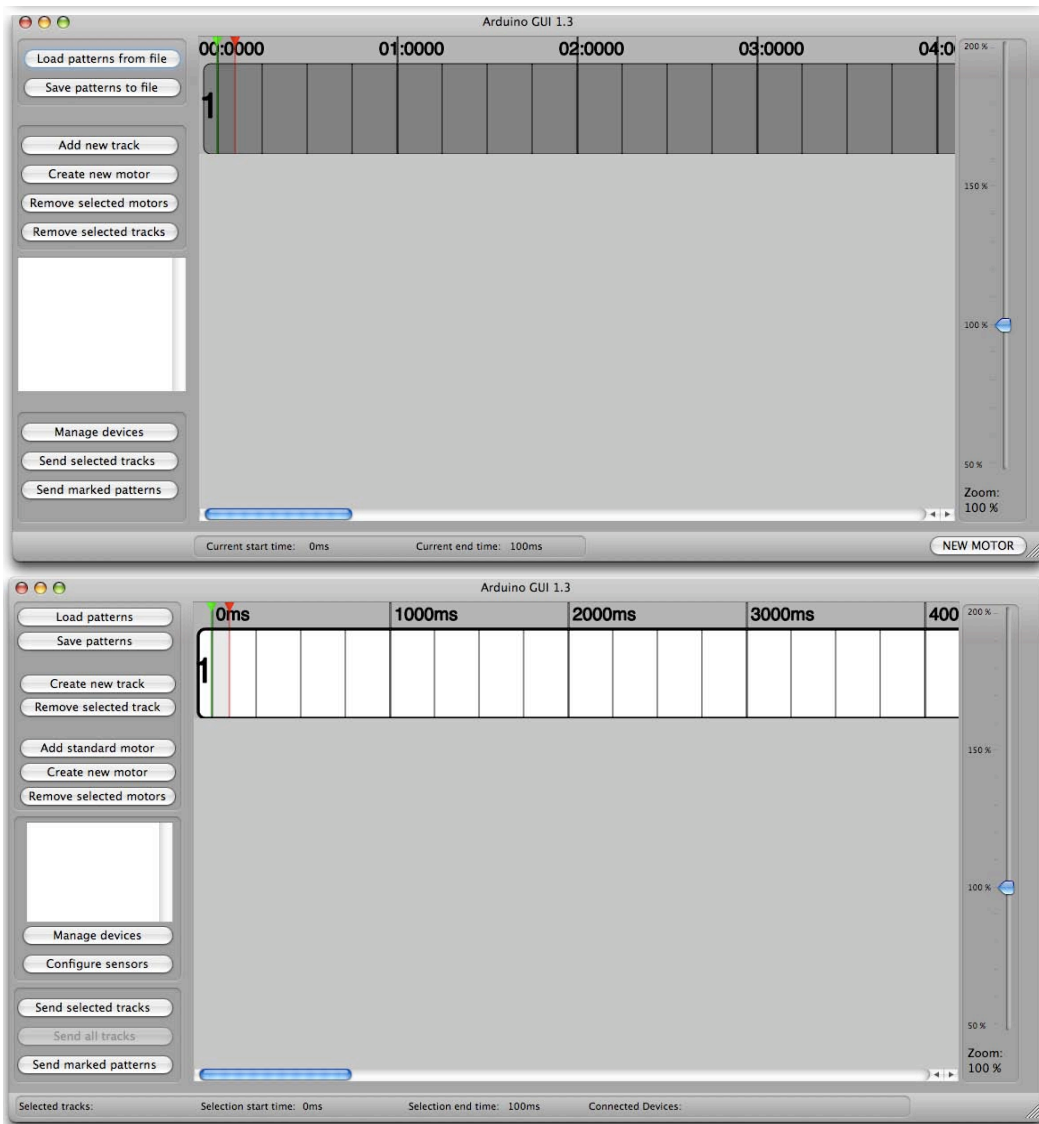


## **Appendix D**

# **GUI Development**



**Figure D.1:** Development of Tactile Editor's GUI through the different DIA-cycle iterations - figure 1/2



**Figure D.2:** Development of Tactile Editor's GUI through the different DIA-cycle iterations - figure 2/2

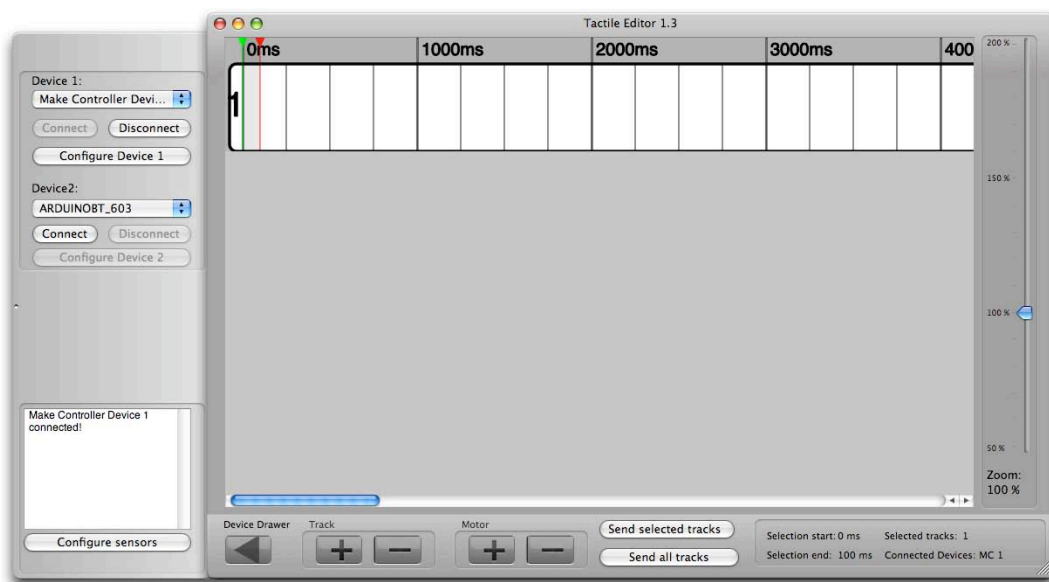


Figure D.3: Tactile Editor GUI - final design

## Bibliography

- Richard J. Adams, Daniel Klowden, and Blake Hannaford. Virtual training for a manual assembly task. *Haptics-e*, 2(2):1, 2001.
- R. Ballagas, M. Ringel, M. Stone, and Jan Borchers. istuff: A physical user interface toolkit for ubiquitous computing environments. *Proceedings of CHI: ACM Conference on Human Factors in Computing Systems*, pages 537–544, 2003.
- Rafael Ballagas, Faraz Memon, Rene Reiners, and Jan Borchers. istuff mobile: Rapidly prototyping new mobile phone interfaces for ubiquitous computing. *CHI 2007 Proceedings - Mobile Kits and Stuff*, 2007.
- Michael Barr. Pulse width modulation. *Embedded Systems Programming*, 14(10):103–104, 2001.
- Jeffrey J. Berkley. Haptic devices. White Paper, May 2003.
- Jan Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons, Ltd, 2001.
- R.W.V. Boven, R.H. Hamilton, T. Kauffman, J.P. Keenan, and A. Pascual-Leone. Tactile spatial resolution in blind braille readers. *Neurology*, 54(12):2230–2236, 2000.
- Grigore Burdea. *Force and touch feedback for virtual reality*. John Wiley & Sons, Inc. New York, NY, USA, 1996.
- Grigore Burdea and Phillipe Coiffet. *Virtual Reality Technology*. Wiley-IEEE Press, 2nd edition, 2003.
- Andrew Chan, Karon MacLean, and Joanna McGrenere. Learning and identifying icons under workload. *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2005.

- 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 4th conference on Designing interactive systems*, pages 312–320, New York, NY, USA, 2002. ACM. ISBN 1-58113-515-7. doi: <http://doi.acm.org/10.1145/778712.778755>.
- Elaine Chen and Beth Marcus. Force feedback for surgical simulation. *Proceedings of the IEEE*, 86(3):524–530, 1998.
- Roger W. Cholewiak. The perception of tactile distance: influences of body site, space, and time. *Perception*, 28(7): 851–875, 1999.
- Lonny L. Chu. User performance and haptic design issues for a force-feedback sound editing interface. *Conference on Human Factors in Computing Systems*, pages 544–545, 2002.
- Anrew M. Colman. *A dictionary of psychology*. Oxford University Press New York, 2001.
- Martin Eimer, Bettina Forster, and Jonas Vibell. Cutaneous saltation within and across arms: A new measure of the saltation illusion in somatosensation. *Perception & Psychophysics*, 67(3):458–468, 2005.
- M. E. H. Eltaib and J. R. Hewit. Tactile sensing technology for minimal access surgery - a review. *Mechatronics*, 13(10):1163–1177, 2003.
- Mario J. Enriquez and Karon E. MacLean. The hapticon editor: A tool in support of haptic communication research. *Proceedings of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2003.
- Frank A. Geldard. Some neglected possibilities of communication. *Science*, 131(3413):1583–1588, 1960.
- Frank A. Geldard and Carl E. Sherrick. The cutaneous rabbit: A perceptual illusion. *Science*, 178(4057):178–179, 1972.
- E. Bruce Goldstein. *Sensation & Perception*. Cole Publishing Company, 6th edition, 2002.
- C.H. Graham. Perception of movement. *Vision and visual perception*, pages 575–588, 1965.

- Martin Grunwald and Lothar Beyer. Der bewegte sinn. *Grundlagen und Anwendungen zur haptischen Wahrnehmung*, 2001.
- Eric Gunther. Skinscape tool for composition in the tactile modality. masters of engineering thesis in the department of electrical engineering and computer science. Master's thesis, Massachusetts Institute of Technology, 2001.
- International Society For Haptics. International society for haptics - what is haptics? <http://www.isfh.org/haptics.html>, 2008.
- Björn Hartmann, Scott R. Klemmer, Michael Bernstein, and Nirav Mehta. d.tools: Visually prototyping physical uis through statecharts. *Extended Abstracts of UIST 2005*, 2005.
- Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. *CHI 2007 Proceedings - Ubicomp Tools*, 2007.
- Harry Helson. The tau effect-an example of psychological relativity. *Science*, 71(1847):536–537, 1930.
- Ira J. Hirsh and Carl E. Sherrick. Perceived order in different sense modalities. *J Exp Psychol*, 62:423–32, 1961.
- Eve Hoggan, Stephen A. Brewster, and Jody Johnston. Investigating the effectiveness of tactile feedback for mobile touchscreens. *CHI 2008 Proceedings - Tactile and Haptic User Interfaces*, 2008.
- Immersion. Vibetonz. <http://www.immersion.com/mobility/>, 2008.
- K. O. Johnson and J.R. Phillips. Tactile spatial resolution. i. two-point discrimination, gap detection, grating resolution, and letter recognition. *Journal of Neurophysiology*, 46(6):1177–1192, 1981.
- Lynette A. Jones, Mealani Nakamura, and Brett Lockyer. Development of a tactile vest. *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004. HAPTICS '04. Proceedings. 12th International Symposium on*, pages 82–89, 2004. doi: 10.1109/HAPTIC.2004.1287181.

- Erich R. Kandel, James H. Schwartz, and Thomas M. Jessell. *Principles of Neural Science*. Appleton & Lange, 2000.
- Mohamed Benali Khoudja and Moustapha Hafez. Vital: A vibrotactile interface with thermal feedback. *Journee scientifique Internationale IRCICA*, 2004.
- C. H. King, A. T. Higa, M. O. Culjat, S. H. Han, J. W. Bisley, G. P. Carman, E. Dutson, and W. S. Grundfest. A pneumatic haptic feedback actuator array for robotic surgery or simulation. *Stud Health Technol Inform*, 125:217–22, 2007.
- Scott R. Klemmer. Suede: A wizard of oz prototyping tool for speech user interfaces. *CHI Letters, The 13th Annual ACM Symposium on User Interface Software and Technology*, 2000.
- Robert W. Lindeman, Yasuyuki Yanagida, Kenichi Hosaka, and Shinji Abe. The tactapack: A wireless sensor/actuator package for physical therapy applications. *Proceedings of the IEEE Virtual Reality Conference (VR 2006)-Volume 00*, 2006.
- Jack M. Loomis. Tactile pattern perception. *Perception*, 10(1):5–27, 1981.
- Andrea Mazzone. *Deformable mechanical structure for physical generation of objects and provision of wide area haptic feedback*. PhD thesis, Eidgenössische Technische Hochschule Zürich, 2004.
- Dan Morris, Hong Tan, Federico Barbagli, Timothy Chang, and Kenneth Salisbury. Haptic feedback enhances force skill learning. *Proceedings of the 2007 World Haptics Conference, Tsukuba, Japan, Mar*, pages 22–24, 2007.
- Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, 2000. ISSN 1073-0516. doi: <http://doi.acm.org/10.1145/344949.344959>.
- Ian Oakley, Marilyn Rose McGee, Stephen Brewster, and Philip Gray. Putting the feel in 'look and feel'. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 415–422, 2000.



- Trygve Reenskaug. Thing-model-view-editor: an example from a planning system. *Xerox PARC Technical Note*, May 1979.
- Robert Riener, Jens Hoogen, Mattja Ponikvar, Rainer Burgkart, Martin Frey, and Günther Schmidt. Orthopaedic training simulator with haptic feedback. *at-Automatisierungstechnik*, 50(6/2002):296, 2002.
- Gabriel Robles-De-La-Torre. The importance of the sense of touch in virtual and real environments. *IEEE MultiMedia*, 13(3):24–30, 2006.
- Harvey Richard Schiffman. *Sensation and Perception: An Integrated Approach*. Wiley, New York, 5th edition, 2000.
- Sebastian Schostek, Chi-Nghia Ho, Daniel Kalanovic, and Marc O. Schurr. Artificial tactile sensing in minimally invasive surgery—a new technical approach. *Minimally Invasive Therapy and Allied Technologies*, 15(5):296–304, 2006.
- Calle Sjostrom. Designing haptic computer interfaces for blind people. *Signal Processing and its Applications, Sixth International Symposium on. 2001*, 1:68–71 vol.1, 2001. doi: 10.1109/ISSPA.2001.949777.
- Robert J. Stone. Haptic feedback: A potted history, from telepresence to virtual reality. *The First International Workshop on Haptic Human-Computer Interaction*, pages 1–7, 2000.
- Matthew Wright, Adrian Freed, and Ali Momeni. Open-sound control: state of the art 2003. In *NIME '03: Proceedings of the 2003 conference on New interfaces for musical expression*, pages 153–160, Singapore, Singapore, 2003. National University of Singapore.
- Hiroaki Yano, Masayuki Yoshie, and Hiroo Iwata. Development of a non-grounded haptic interface using the gyro effect. *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings. 11th Symposium on*, pages 32–39, 2003.



# Index

Adobe Premier Pro .....	35
Apple Final Cut Suite .....	35
Apple GarageBand .....	34
Apple iMovie .....	35
Audacity .....	34
Avid Media Composer .....	35
braille displays .....	22
class structure .....	59
- Motor.m .....	61, 98
- Track.m .....	60, 97
- TrackArray.m .....	60, 97
Cocoa design pattern .....	58
communication protocols .....	76
- Arduino .....	76–77, 79
- Make Controller Kit .....	78, 79
concepts of interaction .....	54–56
- feedback .....	55
- keyboard interactions .....	55
- mouse interactions .....	54
- tooltips .....	55
contributions .....	94
control points .....	95
development environment .....	45
DIA-cycle .....	3, 57
DIA-iteration .....	4
direct manipulation .....	34, 45, 54
error of localization .....	<i>see</i> tactile spatial resolution
evaluation .....	81–91
Fruityloops .....	34
functionality .....	41
future work .....	94–96
- communication with other tools .....	95
- Export function .....	94
- perception of tactile patterns .....	96

- 
- haptic editors
    - Hapticon Editor ..... 23, 28
    - Immersion Studio ..... 25
    - Skinscape ..... 27
    - VibeTonz ..... 27
    - VITAL ..... 24
  - haptic feedback ..... 3
    - haptic devices ..... 16
    - in training motor skills ..... 21
    - in virtual reality ..... 20
    - tactile stimulators ..... 17
  - haptics ..... 1
  - hardware platforms ..... 45
    - Arduino ..... 46
    - configuring sensors ..... 68
    - connecting and disconnecting devices ..... 67
    - Make Controller Kit ..... 47
  - HCI toolkits
    - Exemplar ..... 32, 33, 44, 95
    - id.tools ..... 31, 33
    - iStuff ..... 29
    - iStuff mobile ..... 29, 33
    - Suede ..... 30, 33
  - Human Interface Guidelines ..... 70
  - implementation details ..... 57–79
  - iPhone ..... 40
  - Jashaka ..... 35
  - model-view-controller-paradigm ..... 58
  - music editors ..... 34
  - patterns ..... 10, 64–66, 90
    - creating patterns ..... 64
    - editing patterns ..... 65
    - sending patterns ..... 65
  - pulse width modulation ..... 46
  - PWM ..... *see* pulse width modulation
  - related work ..... 19–37
  - requirements ..... 39–49
    - conceivable requirements ..... 44
    - desirable properties ..... 43
    - necessary properties ..... 42
    - requirement priorities ..... 39
  - sense of touch ..... 6
    - cutaneous perception ..... 6
    - kinesthetic perception ..... 6
    - mechanoreceptors ..... 7

- perception ..... 8
- sensor physiology ..... 7
- tactile sense ..... 7
- sensory homunculus ..... 14
- Smalltalk ..... 58
- somatic sensory system ..... 5
- summary ..... 93
- system architecture ..... 51–54
  - data layer ..... 52
  - hardware layer ..... 53
  - logic layer ..... 53
  - visual layer ..... 52
- system objective ..... 40
  
- Tactile Editor GUI ..... 62, 81
  - canvas ..... 63
  - drawer ..... 66
  - Help book ..... 71, 74
  - help system ..... 72
  - keyboard shortcuts ..... 75
  - main menu ..... 70–71
  - status bar ..... 69
  - tooltips ..... 61, 72
- tactile illusions ..... 14
  - sensory saltation ..... 14
- tactile spatial resolution ..... 11
- toolkit evaluation ..... 35
  - ceiling ..... 30, 36, 82
  - threshold ..... 30, 36, 82
- two-point-threshold ..... *see* tactile spatial resolution
  
- user testing ..... 81, 94
  - results ..... 86–88
  - setup ..... 83
  - statistical analysis ..... 86
  - suggestions ..... 89
  - tasks ..... 84–86
  - users ..... 84
- users ..... 44
  
- vibrotactile patterns ..... *see* patterns
- video editors ..... 35
- visual programming ..... 33

