Developing Maturity in DMIs and Mapping Tools

Brady Boettcher

Music Technology Area
Schulich School of Music
McGill University
Montreal, Canada

May 2023

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Arts.

2023/05/30

Abstract

This thesis explores solutions to smooth the challenging path for digital musical instruments (DMIs) and mapping tools to reach a state of maturity and usability by artists, discussing development and design practices with the goal of producing easy-to-set-up, stable, and well-documented academic projects. The concepts of technical maturity and stability are defined, using the user's perspective to introduce qualities of devices in music technology that support maturity. Barriers to these attributes are reviewed from literature, discussing the lifecycle tendencies of projects developed in research contexts based on design intents. Possible solutions to these barriers are then applied to two main case studies: 1) the development of *the Slapbox*, a responsive, reliable and replicable percussion DMI designed for intensive musical use, and 2) the expansion of *libmapper*, a distributed mapping framework created at the Input Devices and Music Interaction Laboratory (IDMIL), to address challenges to its setup, operation and integration with popular artistic tools. Evaluations with artists are then conducted for both projects to determine the effects of the proposed practices on the usability of novel DMIs and mapping tools. Finally, the implementations and results from each project are summarized, proposing paths for future work and describing applications to other projects.

Résumé

Cette thèse explore des solutions pour permettre aux instruments de musique numériques (DMIs) et aux logiciels de mappage d'atteindre un état de maturité et de faciliter leur utilisation par des artistes. Nous discutons plusieurs pratiques de conception et d'implémentation de systèmes dans le but de produire des outils faciles à configurer, stables et bien documentés. Les concepts de maturité technique et de stabilité sont définis, en utilisant la perspective de l'utilisateur pour introduire les qualités des appareils dans la technologie musicale pour qu'ils atteignent un niveau donné de maturité technique. Les obstacles à ces objectifs sont passés en revue à partir de la littérature, en discutant des tendances du cycle de vie des projets développés dans des contextes de recherche à partir des intentions de conception. Les solutions possibles à ces obstacles sont ensuite appliquées à deux études de cas principales: 1) la conception du *the Slapbox*, un DMI de percussion réactif, fiable et reproductible conçu pour une utilisation musicale intensive, et 2) l'expansion de *libmapper*, un logiciel de mappage distribué créé à le laboratoire de dispositifs d'entrée et d'interaction musicale (IDMIL), pour relever des défis de configuration, de fonctionnement et de l'intégration avec d'autres outils artistiques populaires. Des évaluations avec des artistes sont ensuite menées pour les deux projets afin de déterminer les effets des pratiques proposées sur l'usabilité des nouveaux DMI et des outils de mappage. Enfin, les implémentations et les résultats de chaque projet sont résumés, proposant des pistes pour les travaux futurs et décrivant les applications à d'autres projets.

Acknowledgments

I have received so much support by many people and organizations while entering this world of digital instruments and mappings. I would first like to express my appreciation for my supervisors Marcelo M. Wanderley and Joseph Malloch, as they understand where my passions and skills lie and have guided me to projects where they could be applied. Throughout the development of the Slapbox I have received an enormous amount of help from the Center for Interdisciplinary Research in Music Media and Technology (CIRMMT) and Yves Méthot, for which I am thankful. I have also been lucky to work with many talented artists willing to evaluate my projects, including Stuart Jackson, Martin Daigle, Angus MacMinn and Sara Adkins. I would like to thank Eduardo Meneses and Christian Frisson for their valuable encouragement and support throughout my internship at the Société des Arts Technologiques. Finally, I would like to thank my perfect one-to-one mapping, Miko Nore, for her constant love and support throughout this program.

Contents

1 Introduction **1**
1.1 Mature musical devices . . . 1
1.2 Digital musical instruments . . . 2
1.3 Mapping frameworks . . . 3
1.4 Entry and operation fees . . . 4
1.5 Structure of this thesis . . . 5
1.5.1 Adaptation of publications . . . 6

2 Maturity in NIMEs **7**
2.1 Qualities of a mature NIME . . . 7
2.2 Challenges and solutions from literature . . . 8
2.2.1 Usability and reliability . . . 8
2.2.2 Replication and availability . . . 10
2.2.3 Signal compatibility . . . 12
2.2.4 Community support . . . 12
2.3 A network of design and development priorities . . . 13
2.4 Summary . . . 14

3 DMI case study: the Slapbox **15**
3.1 Introducing the Tapbox . . . 15
3.1.1 The Tapbox design . . . 15
3.1.2 Challenges for maturity . . . 17
3.2 The Slapbox redesign . . . 18
3.2.1 Interaction methods . . . 18
3.2.2 Gesture extraction . . . 23
3.2.3 Sound synthesis . . . 26
3.3 Evaluations . . . 27

3.3.1 Performer backgrounds . . . 28
3.3.2 Instrument impressions . . . 28
3.4 Summary . . . 30

4 Mapping tools case study: libmapper 31
4.1 Introducing libmapper . . . 31
4.1.1 Framework description . . . 31
4.1.2 Challenges for maturity . . . 32
4.2 Addressing barriers for entry and operation . . . 33
4.2.1 Distribution and documentation . . . 34
4.2.2 Mapping session management . . . 36
4.2.3 Improving signal readability . . . 37
4.2.4 Mapper4Live: a libmapper to Ableton Live bridge . . . 39
4.2.5 Mapper4TD: a libmapper to TouchDesigner bridge . . . 43
4.3 Artistic evaluations . . . 44
4.3.1 Mapper4Live experimentation . . . 44
4.3.2 An interactive audiovisual installation . . . 46
4.4 Summary . . . 51

5 Conclusions and future work 53
5.1 Impact of the applied practices . . . 53
5.1.1 Applications to other projects . . . 54
5.2 Future work . . . 55
5.2.1 Slapbox improvements . . . 55
5.2.2 Libmapper improvements . . . 56

List of Figures

1.1 Relationship between gestural controllers and sound sources, adapted from Wanderley (2001). 2

2.1 A network of design and development priorities to support maturity. 13

3.1 The Tapbox, showing the Top panel with its piezo sensor and the front control panel with speakers, originally presented in (Sullivan, 2021). 16
3.2 The Slapbox design. *Left*: Top and front speaker/auxiliary control panels. *Right*: Top and guiro side. Note the (red) buttons and LED positions in the Top panel. . 19
3.3 Velostat sensors: Sandwich (top) and gap (bottom) configurations. 22
3.4 Details of the drum pad design showing individual layers of the sensor configurations. 22
3.5 Original guiro design showing individual layers of the sensor setup. 24
3.6 Slapbox GUI showing a flattened version of the instrument's interaction surfaces. . 28

4.1 Webmapper's List View during the audiovisual installation discussed in Section 4.3.2, showing gestural signals from T-Sticks on the left connected to modulation signals in TouchDesigner and SuperCollider on the right. 33
4.2 Left: the real time signal value plotter in webmapper, with the x-axis as time and the y-axis as the signal's range; right: a standalone signal plotter written in Python with the same axes as the webmapper plotter. 38
4.3 An example of signal metadata displayed as a tooltip to increase signal readability. The signal's name and length is displayed at all times, with its other properties appearing in a tooltip when the mouse hovers over the component. 39
4.4 The Mapper4Live plugin interface. Parameters from the Claverb instrument in Ableton Live are exposed on the libmapper network where gestural controllers can modulate their values in real time. 42

4.5 Internals of the Mapper4TD container, showing its inline setup and usage instructions at the top, components for source signals on the left (inSources), destination signals on the right (dstSignalNames) and a libmapper device written in Python that manages signal networking in the bottom left (MapperDevice). 43
4.6 A jam session using Probatio signals mapped to Ableton Live parameters using Mapper4Live. 45
4.7 Overview of the systems and interactions used in the interactive installation project. 48

List of Tables

3.1 Percussive sensor comparison 20
3.2 Potential signal space for the Slapbox 24

List of Acronyms

DMI	Digital Music Instrument
GUI	Graphical User Interface
HCI	Human Computer Interaction
IDMIL	Input Devices and Music Interaction Laboratory
MIDI	Musical Instrument Digital Interface
OSC	Open Sound Control
ISO	International Organization for Standardization
CI	Continuous Integration
BOM	Bill Of Materials
SUS	System Usability Scale
NIME	New Interface for Musical Expression
API	Application Programming Interface
LED	Light Emitting Diode
PLA	Polylactic Acid
JSON	Javascript Object Notation
LOM	Live Object Model
SDK	Software Development Kit
NDI	Network Device Interface
UDP	User Datagram Protocol

Chapter 1

Introduction

1.1 Mature musical devices

The rise of digital technology has transformed the way music is produced, recorded, performed and experienced. These developments have brought forth a new type of device, the digital musical instrument (DMI), that is able to separate the physical connection between gesture and sound that binds acoustic instruments (Wanderley, 2001). Today, research and commercial settings are overflowing with new and innovative digital musical tools, yet few are reaching the maturity to become viable for artists to include in their repertoire. As Jordà (2004) writes:

> "Many new instruments are being invented. Too little striking music is being made with them."

This observation is especially relevant in research contexts, where the majority of recently created DMIs are made for the researcher alone, often becoming unusable after the project's completion (Morreale & McPherson, 2017). Whether this is a result of design shortcomings, research intents or external influences, it poses a problem for future researchers aiming to build upon these works. Artistic research such as DMI design requires the ability to achieve long-term practice in order to establish objectivity in its findings (Hannula, 2008). If claims from DMI research are to gain the trust of other researchers, devices created in research require a stronger

focus on replicability and documentation (Calegario et al., 2021) as well as stability and reliability (Sullivan & Wanderley, 2018) to support their continued evaluations.

1.2 Digital musical instruments

In DMIs, gestural information captured by sensors is mapped to parameters in a sound-producing program (Wanderley & Depalle, 2004). The mapping between gesture and sound determines the behavior of the instrument, enabling a single control surface to produce many types of sounds depending on its mappings to the parameters of a synthesis program (Hunt et al., 2003). The simplest type of mapping between musical signals is a one-to-one connection, where an input signal directly drives an output. More complex relationships between signals such as convergent mappings (many-to-one) can result in a parameter depending on multiple inputs to compute its value. Divergent mappings (one-to-many) result in one signal controlling multiple audio parameters at once. Figure 1.1, presented in Wanderley (2001), visualizes the relationship between the gestural controller and the sound producer. Mapping can also be explored through a functional lens by defining mathematical expressions (functions) relating one or more input signal(s) to an output (Caramiaux et al., 2014). To relate gestural signals to audio signals using more meaningful representations, intermediate mapping layers can also be created (Hunt & Wanderley, 2002).

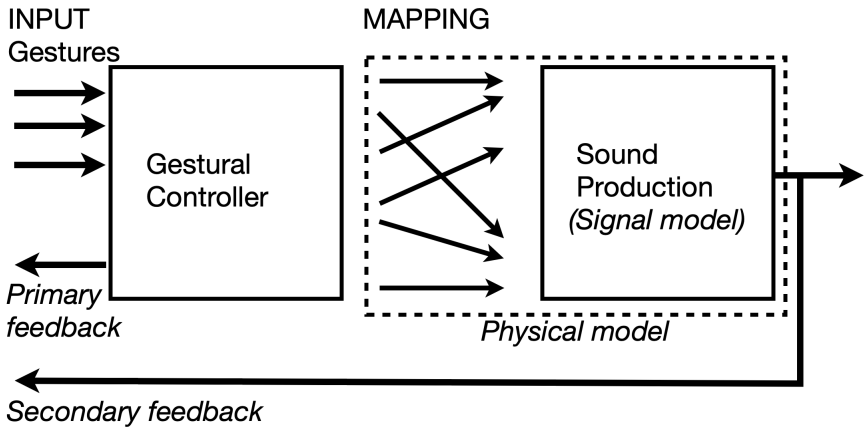

Fig. 1.1 Relationship between gestural controllers and sound sources, adapted from Wanderley (2001).

Signal mapping can be applied to computer-generated visuals, haptics and sound spatialization as well to enrich interactions made by performers or the audience. By connecting signals between sensory modalities, artists can more fully express their intents and immerse the audience in their pieces. This distribution and variety of signals used to create sound makes it difficult to define what a DMI is. Therefore, in this thesis, the term DMI will refer to any interactive digital system that produces sound in real time. In order to simplify the phrasing throughout this thesis, the well-established acronym NIME (New Interface for Musical Expression) will be used when referring to the category consisting of both DMIs and mapping tools.

1.3 Mapping frameworks

Connecting gestural controllers to sound sources with complex mappings can provide noticeable performance benefits when compared to one-to-one mappings (Hunt et al., 2003). Currently, support for user-designed convergent and functional mappings is not implemented in most commercial production environments, often restricting users to linearly scaled one-to-one connections.

Several standards and libraries have emerged for sending musical signals between gestural controllers and sound processors. The Musical Instrument Digital Interface standard (MIDI) is perhaps the most popular protocol for sending simple musical signals between devices. Though its adoption is widespread, MIDI lacks flexibility for data types and ranges outside of its standard, which can restrict the user's ability to express their sonic intents (Moore, 1988).

The networking protocol Open Sound Control (OSC) allows the design of custom namespaces for signals. This freedom is an attractive feature for researchers using complex gestural data, but results in one-off implementations for devices that are unable to be used for other purposes. A few libraries have been created to address some of OSC's limitations such as device connections (Dannenberg & Chi, 2016) or signal discovery[1], but still rely on the user to find their own method of creating dynamic mappings to devices. The original authors of OSC imagined that compatibility would be achieved through the use of common schemas defining signal names and data types.

[1] https://github.com/vidvox/oscqueryproposal

In development since 2007, the open-source library libmapper (Malloch et al., 2007, 2013) (further discussed in Section 4.1) aims to resolve the issues of compatibility in a different way. Instead of devices agreeing on a common signal representation, each device defines the signals that it sends and receives independently. Once registered with libmapper, devices, signals and other metadata can be discovered through multicast networking, and signal datastreams can be freely connected with libmapper handling any necessary translation, type coercion, and vector truncation or padding so that the destination always receives messages it knows how to process. Runtime connections between signals (called "maps") also embed configurable processing and other metadata, and can be managed over the network using OSC messaging or an arbitrary number of session managers such as webmapper (Wang et al., 2019) or mappersession[2]. Users of the libmapper application programming interface (API) are encouraged to use "strong semantics" and real units when designing device and signal representations; if they choose to define ranges for signals, new maps will default to linear scaling. Libmapper supports the design of many types of mappings not supported by other tools including convergent and functional mappings.

Some commercial music production environments have developed their own solutions for designing mappings, such as Ableton Live's Connection Kit[3] and Bitwig's modulation system[4]. These tools are capable of creating simple mappings, yet do not support more complex mapping types from literature (e.g., functional, convergent).

1.4 Entry and operation fees

According to Wessel and Wright (2002), musical devices should have "a low 'entry fee' with no ceiling on virtuosity." In other words, NIMEs should be simple and straightforward to set up and use, while having no discernible limit to the expertise that can be developed with the device. Though the requirement for no virtuosity ceiling has been challenged (Bowers & Archer, 2005), it is generally agreed-upon that NIMEs should be easy to pick up and play, which has been a

[2] https://github.com/libmapper/mappersession
[3] https://www.ableton.com/en/packs/connection-kit/
[4] https://www.bitwig.com/learnings/an-introduction-to-modulators-45/

challenge for many recent DMIs (Morreale & McPherson, 2017).

Instruments that embrace innovations in mapping methods often contain additional barriers for entry, requiring the installation of experimental applications that lack the stability to instill confidence in the user. Novel mapping design concepts and interfaces can also be unfamiliar to artists (West et al., 2020) as there is no established guide for designing mappings (Medeiros et al., 2014), leading to another competency condition to be fulfilled before any meaningful sounds can be produced. Gaps in system compatibility for preliminary software can render the NIME unusable over time, with poor documentation repressing any chance of its eventual repair. In this way, the advantages of distributing systems to favor mapping flexibility come at a cost to usability if the systems are not developed to a stable and mature state. This thesis explores challenges to this goal of maturity and applies potential solutions from literature to two projects to evaluate their impact.

1.5 Structure of this thesis

This thesis has five chapters:

1. **Introduction**.
2. **Maturity in NIMEs**, where the concept of maturity is introduced, citing challenges and solutions from literature.
3. **DMI case study: the Slapbox**, where I present a redesigned DMI with the goals of supporting reliability and replicability.
4. **Mapping tool case study: libmapper**, where I describe recent developments to a distributed mapping framework to promote usability by artists.
5. **Conclusions and future work**, where research results are summarized and suggestions for future work are provided.

1.5.1 Adaptation of publications

Parts of this thesis are adapted from three of my recent publications. The Slapbox design and evaluations presented in (Boettcher, Sullivan, & Wanderley, 2022) appear in Chapter 3. Chapter 4 includes recent developments to libmapper documented in (Boettcher, Malloch, et al., 2022) and (Boettcher et al., 2023).

Chapter 2

Maturity in NIMEs

This chapter begins by defining a set of attributes for DMIs and mapping tools that mark a state of maturity. Common challenges to achieving these qualities are then introduced, citing solutions provided in literature impacting the design and development process. The challenges are discussed in terms of their resolvability, narrowing our focus to particular design and development principles that support usability and stability.

2.1 Qualities of a mature NIME

Let us begin by imagining the goal of a **mature** NIME: a tool with a long-standing, far-reaching community built around its use, a simple and well-documented setup procedure and an expanse of publicly available resources demonstrating its capabilities. If attainable, its source code should be documented and extensible, following development practices to support future work. This NIME's performance is always reliable, it is compatible with all systems and is always available to be bought or built. Notably, these properties have little to do with performance (e.g, latency, expressivity), and instead focus on the NIME's usability by artists and stability for future work.

Granted, many of these expectations may be prioritized higher in commercial contexts, yet they also have value in research as discussed in Section 1.1. Much of the engineering work required to obtain them can be difficult to justify in research, and may not be primary requirements in

smaller projects that target a particular behavior. Ephemerality has also been claimed as an intrinsic modality of DMIs and their performance, having faith that great musical works will naturally emerge and be sustained (Goudard, 2019). Regardless, there is a trend of archived NIMEs that are in unusable states (Morreale & McPherson, 2017) that warrants an exploration into possible solutions to support their continued use and development. In this thesis, I have chosen to focus on particular qualities present in mature NIMEs that reinforce long-term use by artists and researchers: usability, replicability, availability, signal compatibility and community support.

2.2 Challenges and solutions from literature

In order to create a network of priorities that support maturity for NIMEs, I will begin by defining the qualities mentioned above in greater detail, exploring the difficulties involved in enforcing each of them over a long period. Methods of evaluating the success of each property are also presented from literature, introducing design and development practices to support maturity.

2.2.1 Usability and reliability

The term usability has many different definitions from various fields, with the most recognized coming from ISO 9241: "the extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (*Ergonomics of Human-System Interaction — Part 210: Human-Centred Design for Interactive Systems*, 2019). Often, the first goal specified for NIMEs is to get set up with the device and produce sounds. As discussed in Section 1.4, the separation of gestures, mapping and sound tools can result in unexpected entry barriers for artists setting up a new NIME. Communicating between gestural controllers and sound-producing programs in real time can produce its own technical and artistic issues as well (Staudt, 2016).

One approach for overcoming these barriers is to design standalone DMIs (i.e. self-contained instruments with embedded sound synthesis) (Sullivan, 2021). Instead of making the artist respon-

sible for designing mappings, standalone DMIs contain internal mappings between their control surfaces and their synthesis programs. This lowers the barrier for entry considerably, as the DMI can be picked up and played immediately. The internal mappings in standalone DMIs can be either fixed presets or modifiable by the user. If the presets are editable, the user benefits from the DMI's instant playability without losing flexibility in designing their own mappings.

Prototyping and iteration

A straightforward way of discovering how a tool is interacted with and perceived is by prototyping its design. Prototypes allow NIME designers to quickly create proofs of concept for their designs, using quick decisions to expedite the design process (Calegario et al., 2017). The modular DMI design framework *Probatio* was developed expressly for this purpose (Calegario, Wanderley, et al., 2020), enabling designers to rearrange modules containing various sensors and independently design mappings for the instrument. Non-functional DMI prototyping is useful as well and has been used to evaluate social and cultural design influences of musicians (Lepri & McPherson, 2019). After prototyping is complete, designs require additional work to improve their reliability.

Relating to prototyping, iterating on designs allows designers to continually address shortcomings of their product after receiving feedback from users (Preece et al., 2015). Evaluations should consider factors of practice (the performer's perspective) and interactions (the designer's perspective) to determine possible improvements to the design (O'Modhrain, 2011). This process can take years to reach the designer's state intended by the designer (Medeiros et al., 2014), stretching well beyond the timeline of many research projects. In order to verify progress in a NIME's usability, designers can have users complete a System Usability Scale (SUS) survey after each iteration to assess the progression of usability in a system (Bangor et al., 2008). The SUS survey, originally proposed by Brooke et al. (1996), is calculated by aggregating results from a quick 10-question interview to produce a quantitative estimate of the system's usability that can be used by future researchers aiming to improve the NIME. Relating to this objective of keeping projects alive, the next set of qualities supports continued work from other sources even after the original designer

steps away from the project.

2.2.2 Replication and availability

Replication has many advantages for NIMEs, including the exploration of new performance contexts and verification of previous evaluations (Calegario et al., 2021). Documentation supports this quality by providing instructions for rebuilding or redesigning a NIME. For hardware, this includes providing a bill of materials (BOM), fabrication files and detailed construction and usage instructions. The choice of sensors, fabrication materials and software libraries can also affect a DMI's replicability. Obsolete materials and software are common occurrences for NIMEs, and designers should choose stable, well-documented tools and parts for their devices (Calegario et al., 2021).

In software, it is important to utilize good development principles from the start, organizing code into modules and providing documentation to explain useful methods and algorithms (Agusa, 2004). Many synthesis programs and mapping tools are implemented as dynamically loaded *plugins*, capable of being operated in multiple applications. This modularity makes the tools more extensible and reusable, yet creates additional technical barriers for installation and distribution (Chiprianov et al., 2011).

DMI redesigns and rebuilds have appeared sparsely in literature addressing various aspects of the design process. Cook (2009) focuses on presenting important redesign principles such as building multiple copies, implementing backward compatibility and documenting electronics. While Cook redesigned their own instrument, Tom et al. (2019) took up the challenge of reconstructing the interactions from a flexible DMI called *The Sponge*. The authors were required to reinterpret much of the original instrument's fabrication, mapping and synthesis methods as there was no information available regarding its BOM or software patches. Despite these challenges, the authors state that they were able to learn more about the original instrument while finding ways to use new technology to improve its capabilities. Replicability makes it possible for iteration to take place without the original designer, continually improving designs and preventing abandonment. This

property is especially relevant for research projects, where there is a high turnover of researchers due to program lengths.

Open source and continuous integration

Making the hardware documentation and/or software sources publicly available, or open source, is an important factor in increasing replicability and extensibility for NIMEs (Calegario, Tragtenberg, et al., 2020). Contributions made by the public can be directly merged into the main repository by the project's maintainer(s) using software version control systems such as Git[1] and Subversion[2]. Responsibly open sourcing a project requires more than making the project available, though, and includes licensing, documentation, testing, packaging and distribution (McFee et al., 2018).

Open source projects typically place licensing restrictions on the use of the tool in commercial endeavors, encouraging the continuation of the open source philosophy. The issue of poor software documentation often arises from its tedious and persistent nature, making it difficult for the end user to work out its setup and operation for inadequately maintained repositories (Heron et al., 2013). Though not typically prioritized in NIME research, the creation of test programs can dramatically reduce debugging time and validate that new changes don't break existing functions (Harrold, 2000).

Continuous integration (CI) practices automate the compilation, testing and distribution of software, reducing long-term development costs and allowing for more frequent releases and contributions (Hilton et al., 2016). Commercial version control hosting toolchains such as Github[3] and Bitbucket[4] include many of these CI tools, allowing developers to verify, compile and distribute their changes as soon as they are made for all platforms. The technical barriers for entry to these tools are relatively low, and can leave the project in a much more workable state for future developers. CI tools are often able to automatically distribute the compiled software to package managers for new releases, making it easy for end users to retrieve updates. In order to encourage

[1] https://git-scm.com/
[2] https://subversion.apache.org/
[3] https://github.com/
[4] https://bitbucket.org/product/

many of these practices for maintaining open source DMIs, Calegario, Tragtenberg, et al. (2020) have proposed a certification process to rate the replicability of DMIs by external designers and developers.

2.2.3 Signal compatibility

An issue of signal compatibility between systems results from the separation of gestures and sound in DMIs. As discussed in Section 1.3, several communication and mapping frameworks such as MIDI, OSC and libmapper have been created to address compatibility issues in musical devices. Although MIDI is prized for its widespread use and simplicity, it contains data representation limitations and lacks internal signal mapping support that other frameworks address (Moore, 1988). Developers have the option of supporting multiple modes of communication with their NIME, yet each implementation adds to the development costs of the project. This presents NIME designers with a difficult choice between compatibility and capability. For this reason, DMIs commercialized through crowdfunding campaigns lean heavily toward MIDI for communication to make the device available to the most widespread audience (McPherson et al., 2019). Compatibility with systems is a continuous obstacle for NIMEs, as devices may be rendered obsolete while trends in preferred communication methods evolve.

2.2.4 Community support

One of the primary issues influencing the longevity of NIMEs built for research is the absence of a stable support community around the tool. Marquez-Borbon and Martinez Avila (2018) state:

> By taking a global perspective and moving away from the individual, it is expected that a shared knowledge and experience amongst the individuals within a group can yield both novel and lasting musical practices with a particular system.

Building a community around a NIME provides many benefits, such as increasing awareness and fostering the growth of its performance and pedagogical practices. Unfortunately, DMIs born

from research are often designed for the creator's artistic requirements, a quality that may constrain the instrument's performances to a particular style and prevent the engagement of other musicians (McPherson & Kim, 2012). Several studies have taken up the goal of building communities around a NIME with varying approaches. McPherson and Kim (2012) and Fukuda et al. (2021) organized the creation of performance communities around a DMI, while Morreale et al. (2017) built a maker community around a NIME-building hardware platform. Although these studies demonstrated the importance of continued performance with a community of artists, it can be difficult for NIME designers in research to prioritize community building as this often falls outside of the scope of the project.

2.3 A network of design and development priorities

In the previous section, several qualities were reviewed that support maturity in DMIs and mapping tools. Design intents and project scopes in research are often constrained to favor new developments (hence the N in NIME) rather than stability, benefiting the field's initial explorations but hindering verifiability. Instead of insisting on the implementation of all solutions presented above for every project, I present a network of priorities for NIME designers to support maturity as design intents expand. The following discussion assumes the perspective of a designer aiming to create a fully mature NIME, using Figure 2.1 as a guide for responsibilities over time.

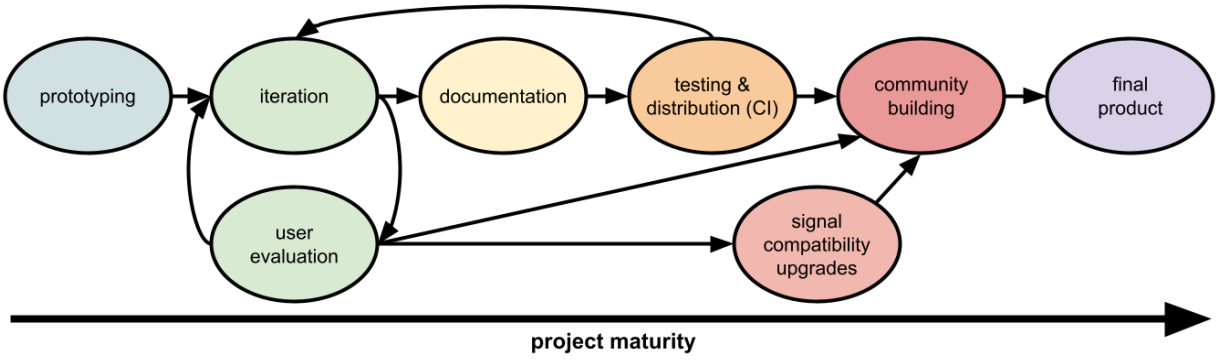

Fig. 2.1 A network of design and development priorities to support maturity.

As seen in Figure 2.1, the design process begins with the prototyping stage. Stable software platforms and libraries should be utilized to improve longevity and encourage replication. Hardware parts used for fabrication should similarly be chosen to prioritize availability and replicability. Next, the designer begins the loop of iteration and evaluation, making design changes and conducting user evaluations to receive feedback for subsequent iterations. Evaluations with artists can lead to the start of a community of users around the NIME as well. Though not an early priority, user feedback regarding the NIME's signal compatibility may lead to implementations of additional modes of communication or application bridges, which in turn may make the NIME available to more communities. After each iteration, the design's documentation should be updated to reflect the changes. This includes compilation and installation instructions for software and fabrication details for hardware, if applicable.

Next, the designer can utilize CI tools to automate the testing, building and distribution of their documentation and software components. The results from testing software may in turn lead to further iterations of fixing bugs and performance shortcomings. CI stabilizes the contribution process and can streamline many of the tedious steps involved in maintaining a large project such as cross-platform compilation and packaging, allowing designers to focus on more iteration loops. Following the project's distribution to users, community building can be fostered by providing consistent and reliable sources for software and documentation.

2.4 Summary

This chapter presented the qualities of a stable and mature NIME, focusing on design and development practices that support them. Next, an in-depth analysis of the challenges facing the attainment of each quality was organized, citing potential solutions from literature. Finally, I introduced a network of design and development priorities to guide designers in supporting their NIME's maturity. The priority network presented above will be applied to two NIMEs in the following two chapters, one DMI and one mapping tool, to evaluate their impact on supporting each project's maturity.

Chapter 3

DMI case study: the Slapbox

This chapter discusses the redesign of a standalone percussion DMI to support usability and replicability. The original instrument, the *Tapbox*, though completed, suffered from reliability issues that precluded it from being viable for performance. The new design, the *Slapbox*, addresses these shortcomings and expands the interaction possibilities of the instrument.

First, the design goals and shortcomings of the Tapbox are introduced, followed by a discussion of the redesign and its subsequent iterations. Evaluations with percussionists are conducted, receiving feedback regarding the new instrument's usability and responsiveness.

Parts of this chapter have been adapted from Boettcher, Sullivan, and Wanderley (2022), where the Slapbox redesign was first presented.

3.1 Introducing the Tapbox

3.1.1 The Tapbox design

The Tapbox (seen in Figure 3.1) was created as a part of a family of instruments called Noiseboxes, each with the goal of instant playability (pick it up and start playing), tightly coupled input to sound mapping, direct sound output, and wires-free operation (Sullivan, 2015). Several versions of Noiseboxes were constructed, all maintaining the same basic input controls (primarily buttons,

linear FSRs and IMU-based motion sensing) and sound output (FM synthesis-based polyphonic drones) (Sullivan et al., 2020).

Physical design

The interaction style of the Tapbox was inspired by the cajón, a box-shaped percussion instrument played with the hands. Each side of the box functioned as an interaction surface, detecting strikes from the musician and synthesizing sounds based on the interaction's properties.

One surface was reserved for necessary auxiliary controls and connection points, while the remaining five sides functioned as input devices that could be mapped to different sounds. To detect strikes from the player, a large piezoelectric element was attached to the underside of each acrylic side panel, while rubber bushings isolated the panels from the 3D-printed frame. Orientation sensing was also added to the Tapbox, with data modulating the sound in different ways.

Fig. 3.1 The Tapbox, showing the Top panel with its piezo sensor and the front control panel with speakers, originally presented in (Sullivan, 2021).

Sound synthesis and modulation

Two synthesis modes were developed for the Tapbox, including a conventional drum kit and a physical modeling synthesizer capable of generating a variety of sounds, from percussive bell-like tones to complex, multi-timbral drones. A unique feature of the instrument was the method of selecting and interpolating between the two synthesis modes which leveraged the Tapbox's IMU-based orientation sensing capability. Additional modulation parameters for both synthesis modes were mapped to rotational axes as well so that the instrument could produce a wide variety of percussive and non-percussive sounds depending on its position and movement.

3.1.2 Challenges for maturity

Once the Tapbox's original design was complete, it still suffered from issues that precluded it from being viable for artists. In particular, I focused on identifying shortcomings in its performance reliability and interaction methods, which ultimately led to the development of the new Slapbox instrument.

First, despite successful setup and performance with the instrument, I was disappointed with the Tapbox's ability to reliably detect interactions. Amplitude sensitivity was poor, and there was an abundance of missed detections when striking anywhere other than the direct center of a panel. While the piezoelectric sensors performed well in testing, the thickness and size of the panels dampened their signals too much to accurately detect strikes at any position.

Second, the types of physical interactions were similar in all panels, given the use of the same materials and sensors in all five sides, preventing exploration of other types of physical interactions (e.g., continuous control, positional sensing) with the instrument. Finally, the instrument's construction and software was sparsely documented, hindering its replicability. Iterating upon the instrument's design allowed me to improve both the performance and interaction shortcomings I had discovered throughout its evaluation.

To determine the work needed for an improved design, the Tapbox was evaluated in terms of

its enclosure and interaction surfaces to search for reusable components in the device. To retain the standalone functionality of the original design, the front panel's speakers and auxiliary control inputs were preserved. Much of the Tapbox's internal electronics including the microcontroller, audio and power circuits are reused, with both designs operating on the Bela[1] platform which enables low-latency audio and sensor acquisition.

3.2 The Slapbox redesign

The Slapbox is a percussive instrument that is a redesign of the Tapbox, offering several interaction surfaces to be tapped, brushed, and slapped. Interactions give users real-time control over modulation effects using the same drumming surfaces, aiming to further explore the uses of modulation gestures in percussive performances. The Slapbox was designed with two primary goals to address the limitations of the Tapbox: 1) accurately and reliably detect interactions, and 2) ensure replicability with thorough documentation of the design and its software which has been made available online[2].

3.2.1 Interaction methods

Panel layouts

The box's top panel contains two large position-sensing pads and two buttons that change the playback speed of the audio. To provide visual feedback to the user, light emitting diodes (LEDs) are positioned around the top panel to light up when the corresponding pad is struck or held. Each side panel functions similarly to the position sensing pads on the top panel, detecting the strike position relative to the side's center. A ridged guiro-inspired component lies in the middle of the back panel to enable rhythmic sliding gestures with small pressure pads on either side of it. Velocity, pressure and position are tracked by all components except for the small pads on either side of the guiro, which track only velocity and pressure. Each panel is covered with a 1mm layer

[1] https://bela.io
[2] https://gitlab.com/bboettcher31/velostat-drum

of cork, which provides a smooth tactile feeling without affecting the accuracy of sensor readings. The completed assembly can be seen from both sides in Figure 3.2.

Fig. 3.2 The Slapbox design. *Left*: Top and front speaker/auxiliary control panels. *Right*: Top and guiro side. Note the (red) buttons and LED positions in the Top panel.

Sensor evaluations

Capturing the position, continuous pressure and velocity of percussive gestures requires transducers that can sense extremely quick body movements. Although there exist many sensors that are capable of extracting percussive gestures (Tindale et al., 2005), the design goals and constraints of this project eliminate several options from consideration.

The use of rigid acrylic panels as interaction surfaces instead of flexible membranes rules out reflective optical sensors as candidates, which have been used to accurately detect strike positions (Sokolovskis & McPherson, 2014). As our design intends to be played with the user's hands, approaches that typically track stick motion such as cameras, electromagnetic sensors and accelerometers do not apply. Fiberoptic sensors have been used to sense multitouch pressure (Huott, 2002), but their lack of commercial availability and shape variations would impede the instrument's replicability. Microphones and piezoelectric sensors are able to extract and classify percussive excitation gestures as seen in (Jathal & Park, 2016) and (Sullivan et al., 2020) respectively, but fail to capture the continuous pressure of modulation gestures. Additionally, trials with the panel-

mounted piezoelectric sensors on the Tapbox resulted in leakage peaks when striking the frame or adjacent panels, and mounting the sensor between an additional acrylic layer also dampened its signal leading to unreliable velocity estimation.

This leaves us with force-sensitive resistors (FSRs), which are able to sense continuous pressure and come in a variety of shapes, sizes and sensitivities. A number of configurations were considered, including standalone FSRs from commercial sources and homemade sensors built from resistive materials that can be configured to detect pressure (Koehly et al., 2006).

Though both commercial and homemade FSRs are able to track pressure, the shape restrictions of commercial models do not allow for the position sensing capabilities afforded by homemade sensors. Pressure-resistive materials like Velostat can be configured to detect continuous position by arranging patterns of conductors above and below the material. Compared to other pressure resistive materials seen in (Koehly et al., 2006), Velostat's consistency and robustness led to its use for the Slapbox's sensors. It should be noted that a combination of different sensing types can be used to reinforce the accuracy and consistency of extracted gestures as seen in (Kapur et al., 2004), and these methods may be applied to future iterations of the Slapbox. The comparison of capabilities for commonly used percussive sensors can be seen in Table 3.1.

Table 3.1 Percussive sensor comparison

	Velocity	Continuous Pressure	Position
Reflective optical	✓	✓	✓
Video and IR	✓		✓
Electromagnetic	✓		✓
Accelerometer	✓		
Fiberoptic	✓	✓	✓
Microphone	✓		
Piezoelectric	✓		
Force Sensitive Resistor	✓	✓	
Velostat	✓	✓	✓

Sensor design

Velostat can be configured to detect taps using either of the circuits shown in Figure 3.3. Drum pads for the top and side panels are created with the sandwich circuit option by placing rings of copper tape concentrically around each other (shown in Figure 3.4), configuring each ring as a voltage divider with Velostat as the variable resistor. The concentric configuration was chosen to track the tap position relative to the center of the pad, imitating the behavior of a circular drum head. The top pads contain five rings each while the side pads only had room for three, resulting in slightly less resolution in the side panel's position estimations. Slots are laser cut into the panels to pass the top and bottom conductors through to the inside of the box for connection. The sensing range for the drum pads was increased by adding an additional layer of Velostat to each pad. This modification resulted in more accurate velocity estimations and added room for lighter impulses to be detected.

One important drawback to the concentric ring pads is their ability to detect multiple contact points on the same surface. While this configuration achieves concentric position sensing using only a few sensing inputs to the microcontroller, each ring is only able to detect one interaction at a time. This limits the types of detectable finger drumming techniques, and could be improved in a later iteration by using a different (and more dense) conductor configuration.

Variance in surface area in the conductive strips was found to lead to uneven steady-state values for each ring due to larger conductive areas allowing more charge through the resistive material. This inconsistency required normalization to detect the centroid accurately by distributing the signal values linearly between their minimum resting values and a common maximum shared by all pads, which is an available function in the graphical user interface presented later on.

Both the guiro and its neighboring small pads were tested using Figure 3.3's gap circuit, but experiments showed that this configuration was only accurate when pressing *directly* in the middle of the gap. Using the sandwich circuit for all components on the back panel (seen in Figure 3.5) resulted in usable signals, but required too much force to detect natural brushes across the ridges.

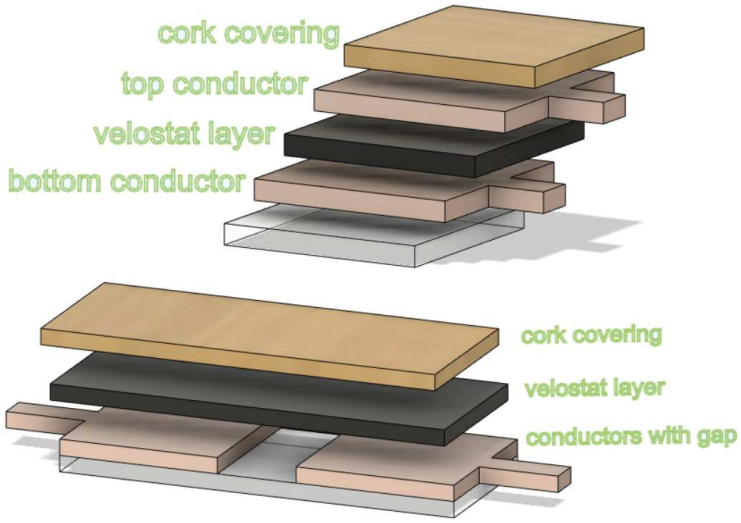

Fig. 3.3 Velostat sensors: Sandwich (top) and gap (bottom) configurations.

Fig. 3.4 Details of the drum pad design showing individual layers of the sensor configurations.

In a later iteration of the back panel, the guiro was replaced with 3D printed ridges of conductive Polylactic Acid (PLA) material[3] (seen in Figure 3.2). 3D printed conductive materials have many uses for creating custom resistive and capacitive sensors (Lazarus & Bedair, 2020). By connecting the conductive ridges to a Trill Craft capacitive touch sensing board[4], the ridges are able to detect touch interactions from the user. Capacitive sensing for the guiro proved much more reliable than the Velostat configuration, but limits guiro interactions to fingers or another conductive material. As the Slapbox is primarily designed for finger and hand interactions, this was an acceptable limitation.

The back panel's side pads were also iterated upon, using 3D printed conductive layers of PLA instead of copper tape to make the contact with the Velostat more consistent. Wires could be attached to the PLA for the side pads and guiro components cleanly on the other side of the panel by heating up the tips of male jumper wires with a soldering iron while pressing them into the material.

The Slapbox's construction details have been documented in its public repository[5] to support its replicability, including laser cutting settings, 3D printing files and sensor fabrication instructions. These details should also be useful for NIME designers looking to reuse its enclosure for their own designs.

3.2.2 Gesture extraction

The Slapbox's sides are able to detect continuous touch position and pressure as well as detect strikes using an impulse detection algorithm. The overall signal space for the instrument is presented in Table 3.2, and I discuss the details of each gesture detection algorithm below.

[3] https://www.proto-pasta.com/pages/conductive-pla
[4] https://shop.bela.io/products/trill-craft
[5] https://gitlab.com/bboettcher31/velostat-drum/-/blob/master/how_to_build_a_slapbox.md

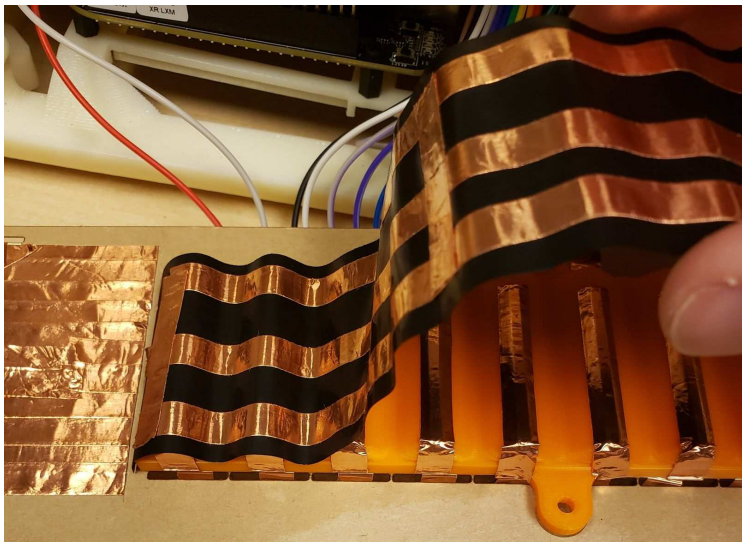

Fig. 3.5 Original guiro design showing individual layers of the sensor setup.

Table 3.2 Potential signal space for the Slapbox

Control surface	Signal Name	Type	Range	Value Interpretation
pressure pads	*_*_pressure (e.g., top_left_pressure)	float	0 - 1	0: pressure intensity
	*_*_position	float	0 - 1	pad center - outer edge
	*_*_impulse_velocity (instanced signal, one instance created per hit)	float	0 - 1	strike velocity
back panel	guiro_touch_position	float	0 - 1	left - right
top panel buttons	button_left	int	0, 1	0: not pressed, 1: pressed
	button_right	int	0, 1	0: not pressed, 1: pressed

Position calculation

The values for each ring can be compared to find the most likely position of the source of pressure being applied to a pad, and can also be used for analyzing continuous pressure changes. As pads are comprised of multiple rings each with their own pressure values, the centroid is first determined to estimate where the pressure source lies relative to the pad's center. A larger number of rings in a pad results in a finer resolution of this calculation, yet only three rings as in the side pads produced

accurate estimations. Ring thickness and distance between rings must also be considered, though the centroid calculation assumes equidistant rings with equal widths. For the array of ring values x, the floating point centroid index is calculated with

$$centroidIndex = \sum_{i=0}^{n} i * x[i]/m \tag{3.1}$$

where n represents the number of strips and m the sum of values.

Pressure estimation

A pad's centroid index (Equation 3.1) represents the floating point representation of the center of mass of the ring pressure array, indicating the relative position to the pad's center while pressure is applied. Once the index is calculated, a pad's pressure value is found by linearly interpolating between the ring array values around the centroid index. The centroid was found to be mostly noise until the pressure applied to the pad exceeded roughly $\frac{1}{10}$ of its maximum value, therefore a simple thresholding function was applied to determine its validity.

Impulse detection

Detecting pad strikes, seen as impulses in pressure values, is done by calculating the rate of change of pressure over time. If the pad's pressure and its rate of change exceed a threshold, an impulse is detected using a state machine. The impulse threshold determines the sensitivity of the instrument, and it is important to find a value that detects light impulses without resulting in false triggers. This pressure value at the time of an impulse being detected corresponds to the impulse's velocity.

The guiro component is able to use this algorithm by detecting swipes across its ridges as sequential impulses. The swipes generate considerably less force than taps on the pads, thus the threshold to trigger impulses was lowered for its sensor configuration.

3.2.3 Sound synthesis

Using the Slapbox's sensor input from each surface, a simple mapping was designed to trigger percussion sounds using audio sampling. Using the gesture extraction algorithms above, the position relative to a pad's center and velocity value can be extracted from each tap on any of the four full sensing pads and mapped to synthesis parameters.

When tapped, the pads trigger an audio sample of either a kick, snare, hi-hat, or clap. Two similar samples of each pad's respective sound type are played at the same time, with the tap position relative to the pad's center determining a crossfade value between the two samples. The smaller back pads don't detect this parameter, therefore only one sample is played when triggered. The velocity of the tap controls the overall gain of the sample(s), imitating the acoustic property of strike force directly relating to volume. This configuration provides dynamic percussion timbres and imitates the sonic changes when striking an acoustic drum head in different positions.

The guiro component triggers a pitched audio sample when any of its 11 ridges are touched. The playback speed for each ridge's triggers range from the sample's original speed to twice as fast, distributing pitches linearly across an octave.

Two mappings for the buttons on the top panel were explored over the course of the instrument's creation. The first applies a delay effect to the overall audio. When a modulating button is pressed on the top panel, the corresponding side panel near the button acts as a delay effect modulator instead of a drum pad. The position where pressure is applied to the side panel modulates the delay rate, changing to 200 ms when pressing in the center and defaulting to 400 ms when no pressure is applied. The delay effect is set to 50% feedback and is applied to all output audio as long as the button is held down.

The second mapping has the buttons control the playback speed of the Slapbox's audio. The left button slows the playback speed of audio samples to half of the original, and the right button doubles the speed. This speed change is perceived as a change in pitch for the Slapbox's percussive sounds, expanding the pitch range to 3 octaves for each sound. In place of the delay functionality in the first mapping, this mapping retriggers audio samples at a fixed rate while pressure is applied

to a pad after a strike.

Synthesis was implemented in the C++ programming language, creating a simple granular synthesis engine to achieve polyphonic sampling. Granular synthesis generates small bursts of overlapping audio at fast rates (grains), allowing us to use this engine for polyphonic sampling by generating one grain per audio clip. In the case of the full pads with position estimation, this results in two grains per hit, each with scaled amplitudes according to the position crossfade value. The latency between a tap and its audio output was able to be reduced noticeably by reducing the audio block size to 32 samples per block.

Graphical user interface

Aside from the visual feedback provided by the LEDs, a graphical user interface (GUI) was created to visualize pad strike position and intensity in real time using Bela's GUI framework. Though initially implemented as a debugging tool for visualizing gestural signals from sensors (see Section 3.2.2), it can also be used during a performance to further confirm the registration of interactions with the device. The interface presents a flattened version of the box's components that reflects their centroids, pressure and impulse status. As seen in Figure 3.6, the line thickness of each panel's outline represents pressure intensity, and the centroid is inscribed in each pad when pressure is applied. Sensor normalization can be triggered with this interface to resolve sensor drift that may occur over time, though it is also performed automatically when the instrument is turned on to avoid the need to use the GUI for every performance.

3.3 Evaluations

Informal user evaluations were completed with two professional percussionists throughout the project to gauge the Slapbox's fulfillment of its design goals and identify opportunities for improvement.

The two percussionists were told the Slapbox was a digital musical instrument for percussion performance and given unlimited time to play the instrument, typically improvising for about 30

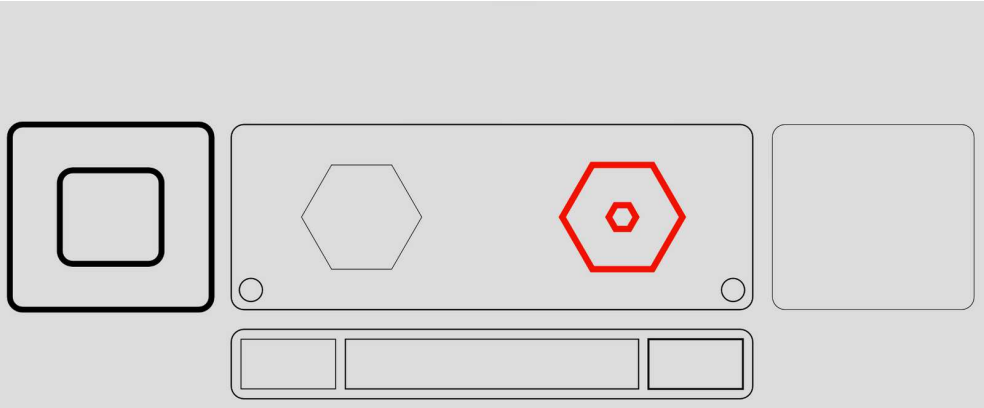

Fig. 3.6 Slapbox GUI showing a flattened version of the instrument's interaction surfaces.

minutes each. They were asked to freely explore the instrument while providing feedback about its responsiveness and playability from their own musical perspective. The instrument was used both on a table and on the user's lap to explore different playing techniques.

3.3.1 Performer backgrounds

Both participants are active performers currently pursuing a doctoral degree in percussion, one of whom had already incorporated digital percussion instruments and effects into their performances in the past. While both percussionists evaluated the instrument before the back panel was iterated upon, only the latter percussionist was able to return for subsequent sessions.

3.3.2 Instrument impressions

Original Slapbox design

Both percussionists engaged with the Slapbox after only a few minutes of experimentation, achieving complex interactions in the form of fast-tapping passages involving complex rhythms using several surfaces on the device.

Both performers noted that the LEDs on the tap panel were helpful indicators to confirm strikes on the box, and the GUI was able to help the participants better understand the functions of the

drum pads. The cork surface was liked by both participants, though one of them had concerns about its durability due to its thickness. Both participants also regarded the instrument's response time as immediate, which was one of the main goals of the design.

The delay modulation behavior using the buttons was well received, and one participant was able to create complex rhythms by weaving the delayed audio together from hits. One percussionist suggested toggling the button effects instead of requiring them to be held down, as it took away much of the mobility in one hand when holding a button.

Both performers noticed a high number of false positive detections resulting from the back panel's guiro and side pad components. The guiro's impulse threshold was set low to detect light brushes, yet produced many false triggers when adjacent sides were struck, leaking signals to the guiro's sensors.

Iterated Slapbox design

After the first improvisation sessions, the Slapbox was iterated upon as discussed in Sections 3.2.1 and 3.2.3, creating an improved back panel and modifying the button mappings. Rubber pads were also added to the instrument's bottom panel by request of one of the performers to prevent the Slapbox from slipping when struck on a table. Two additional improvisation sessions were conducted with the second percussionist to evaluate how the changes affected how the instrument is perceived and used.

Interestingly, the playing technique of the second percussionist changed when the retrigger behavior was added to the instrument. Instead of using fast strikes as they had in the first version, the performer preferred to hold one or more surfaces to retrigger samples while playing with the octave-modulating buttons to achieve different variations of the sounds. Slightly changing the instrument's mappings resulted in the user re-interpreting, or co-adapting (Mackay, 2000), their playing techniques.

The false positive impulse detections that were evident during the first improvisation sessions were substantially reduced with the new back panel implementation. The percussionist noted, "I

like how it responds. Last time there was a bit more false triggering. In this [version], I feel a lot more in control of the instrument". They also enjoyed the retriggering behavior and playing the pitched guiro ridges "like a marimba", to complement the atonal percussion sounds from the other surfaces. An improvisation performance video[6] was also created by the percussionist after taking the Slapbox home for several weeks to practice with it, praising its ability to create complex rhythms with ease.

3.4 Summary

This chapter presents the Slapbox, the redesign of a percussion DMI built with the goal of being a responsive, reliable and replicable instrument. The original instrument is first introduced, proposing challenges to its maturity to be addressed in the redesign. Several rounds of design iterations are conducted to work towards these goals, regularly evaluating the instrument with percussionists to gauge the instrument's performance and to receive feedback for further iterations.

[6] https://www.youtube.com/watch?v=u8_jG9uUoYQ

Chapter 4

Mapping tools case study: libmapper

This chapter discusses recent developments to libmapper, a distributed signal mapping framework, with the goal of improving its usability by artists. Development principles are implemented to improve the framework's distribution and lower its barriers for entry and operation, leading to a more maintainable and stable set of tools. Evaluations to determine the impact of the changes on the framework's maturity are conducted in the form of artistic projects with two artists of varying technical skill levels.

4.1 Introducing libmapper

The introduction of libmapper and its challenges for maturity as well as the discussion of the installation project (Section 4.3.2) have been adapted from Boettcher et al. (2023).

4.1.1 Framework description

As discussed in Section 1.3, libmapper is a distributed signal mapping framework that enables complex mappings between signals over a network. Devices such as control surfaces and synthesizers implement libmapper's library using one of its language bindings[1] and create their signals on the

[1] http://libmapper.org/ecosystem/language_bindings.html

network. By using any of libmapper's application bridges[2], artists can connect libmapper signals to parameters in their existing artistic programs. Mappings between signals can finally be made and managed using any of the framework's several session managers and user interfaces[3]. Various utility programs have also been developed to handle the recording, looping, and visualization of mappings (Frisson et al., 2021).

4.1.2 Challenges for maturity

The libmapper project has shown a clear potential for use in mapping research in recent years, motivating many researchers to make contributions to the library, its mapping interfaces, language bindings and application bridges. The libmapper library has been embedded into a number of gestural devices as well including TorqueTuner (Kirkegaard et al., 2020), the T-Stick (Kirkegaard et al., 2020) and Probatio (Calegario, Wanderley, et al., 2020). While these contributions bring valuable new features and research attention to the framework, its user base is largely confined to developers with highly technical experience.

Aside from the difficulty of maintaining such a vast array of tools with research funds, I speculate that the technical requirements to install and operate the tools themselves are primarily to blame for this characteristic. Firstly, distributed mapping tools are middleware, requiring input and output devices to be set up before they can be used. The fact that the libmapper project aims at supporting users of many programming languages, environments, applications and hardware platforms creates challenges for software distribution, especially since many are "moving targets"—language bindings for Max that were stable and performant in 2012 will not run under newer versions of Max[4], and web applications (such as the webmapper session manager seen in Figure 4.1) are sometimes broken by new web browser releases.

Additionally, the popularization of new artistic software will require the creation of corresponding libmapper bridges. Some discussion has taken place about resolving many of these issues by

[2] http://libmapperorg/ecosystem/protocol_bridges.html
[3] http://libmapperorg/ecosystem/user_interfaces.html
[4] https://cycling74.com/products/max

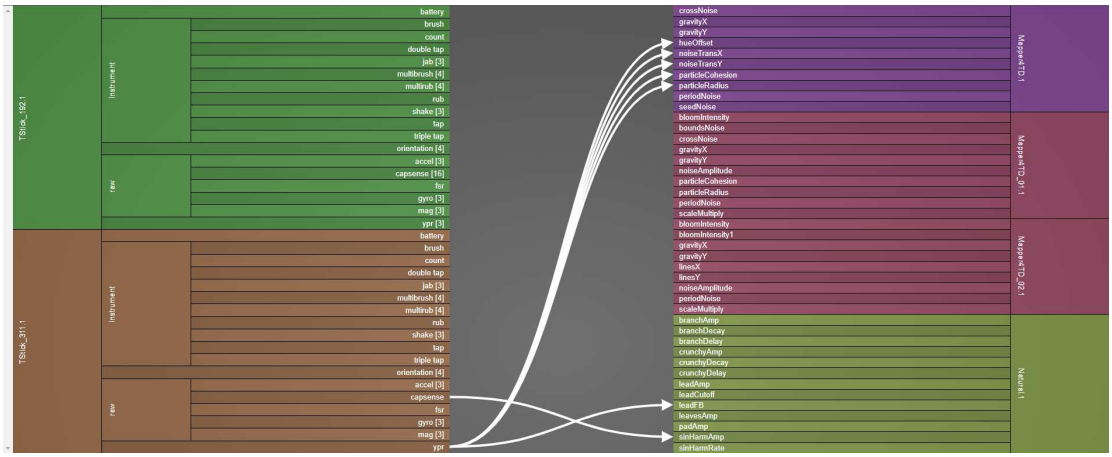

Fig. 4.1 Webmapper's List View during the audiovisual installation discussed in Section 4.3.2, showing gestural signals from T-Sticks on the left connected to modulation signals in TouchDesigner and SuperCollider on the right.

representing libmapper maps with WebAssembly, allowing users to write map expressions in whatever language they are most familiar with[5], though this would require a large reorganization of the framework.

Finally, the prioritization of new features over stability may also be partly responsible for these consequences. Alas, it can prove difficult to justify important, yet mundane, engineering work in a research context. While the framework boasts many benefits when compared to competing mapping approaches, its usability barriers prevent the user feedback needed to fully determine its feasibility amongst its peers. In order to open the doors for artistic uses of the framework and to answer research questions regarding mapping design, the focus of development is shifted to overcoming obstacles that prevent its common use among artists.

4.2 Addressing barriers for entry and operation

I have chosen to address three main issues that plague the usability of libmapper for the end user. The first concerns improving the ease of installation and setup for libmapper and its bridges. Next, I present a newly created Python module that handles many of the challenges associated

[5] https://groups.google.com/g/dot_mapper/c/foPkuO7NpbA/m/xX3qPj9mBQAJ

with designing a portable and stable distributed mapping session with libmapper. Finally, several improvements are introduced to webmapper with the intent of increasing the readability of signals and their metadata for the user. These three developments, detailed in the coming sections, aim to improve the accessibility and usability of libmapper, removing technical barriers to support the pursuit of artistic endeavors.

4.2.1 Distribution and documentation

The many tools that make up the libmapper framework make its distribution an especially difficult task. I have chosen to focus on supporting continuous integration (CI) of the libmapper library and webmapper with automated building and distribution, as well as improving the documentation and organization of libmapper's application bridges whenever possible.

CI and package managers

CI is an important engineering principle that encourages automated building and releasing of software. Often left out of research projects, CI principles reduce debugging time using automated tests and support stability by encouraging frequent releases (Hilton et al., 2016). Using CI for libmapper and webmapper, users can be presented with a precompiled package that can be installed through their chosen package manager and platform. Github Actions[6] has been utilized for these purposes, allowing automation of the building and testing of libmapper and its language bindings for all platforms when updates are pushed.

Libmapper itself functions as a dynamically-linked (or shared) library in which programs access its functions at runtime. While Linux and MacOS contain dedicated directories in which these types of libraries are installed, Windows leaves it up to the user, making general installation difficult. For this reason, I did not prioritize the distribution of the libmapper binaries as developers will often want to build shared libraries directly from source. However, I decided to align with Ubuntu's library system and created a personal package archive for libmapper[7] that allows

[6] https://github.com/features/actions
[7] https://launchpad.net/~libmapper/+archive/ubuntu/libmapper/

installation using Ubuntu's package manager *apt*.

Libmapper's Python bindings are also now built with CI, and are able to be distributed using pip, Python's package manager. By simply running `pip install libmapper`[8], the bindings for the user's platform are installed for their Python environment and can be easily updated at any time. As webmapper utilizes the Python bindings in its design, users no longer need to build any software from source to operate the application. The creation of standalone webmapper executables for all platforms has been enabled using PyInstaller[9], providing a method of using the application without entering the terminal. A Windows installer has also been created for the standalone version of webmapper, integrating it directly alongside the user's other applications.

Previously restricted to MacOS, I have enabled the use of libmapper's Max, Pure Data[10] and SuperCollider[11] application bridges on Windows as well using CMake[12]. A secondary signal creation option for the SuperCollider bridge was also created to provide more flexibility for users. Now, in addition to using signal generators provided by the plugin, users are now able to route signals using Busses[13], offering the possibility to create signals outside of user-defined synthesizers. For the Max bridge, I have packaged the plugins and submitted them to the Max package manager[14]. If the package is accepted it will become available for one-click installation for all platforms within Max, avoiding all manual compilation or installation. The packages are created automatically with Github Actions, making releases easier for developers as well.

These improvements to libmapper's distribution enable artists to grab the most recent releases easily without having to manually build the tools from source in order to use them. Additionally, our release process is much easier and quicker using the virtual build environments provided by Github Actions.

[8] https://pypi.org/project/libmapper/
[9] https://pyinstaller.org/en/stable/
[10] https://puredata.info/
[11] https://supercollider.github.io/
[12] https://cmake.org/
[13] https://doc.sccode.org/Classes/Bus.html
[14] https://docs.cycling74.com/max8/vignettes/package_manager

Documentation and continuity

Another important step taken after these changes were made is to update the libmapper website[15] with the most recent links and documentation for installation and use. A number of recent related projects and publications were also added to the website to showcase the potential of the framework. Usage instructions and build scripts have been updated for many of libmapper's bridges to make them easy to install and use for any platform.

Though the changes presented here may seem obvious, it is important to realize why it has taken so long for them to be implemented to avoid this barrier in related research projects. For one, the rise of CI tools such as Github Actions has only recently opened up the opportunity to automate deployment using virtual environments. Aside from availability, this type of work is not easily justified as research, pushing funding towards feature additions rather than the maintenance and development required to make a project available to a wider audience. I argue that this CI work should be a compulsory part of user-facing research projects in order to increase the feasibility of the projects for artists.

4.2.2 Mapping session management

Webmapper (Wang et al., 2019), libmapper's most-used graphical interface for mapping design, contains basic features for loading and saving mapping sessions using Javascript Object Notation (JSON) structures. In order to address more complex session management needs and improve the portability of sessions, I have created a Python module for mapping session management that can be used from the command line or imported as a library into other Python scripts.

I call the module *mappersession*, and have made it installable through pip similarly to libmapper[16] to align with the CI principles proposed in this paper. Using a Python module permits session management to operate from any Python program or a command line, and work is in progress to extend the module for use in C/C++ programs. Running the session manager from

[15] http://libmapper.org
[16] https://pypi.org/project/mappersession/

the command line provides a headless mode of session management suited well for automating the setup of artistic works. I have also written detailed installation and usage instructions for the module in its Github repository[17] and public package to align with documentation standards for Python packages.

mappersession features

The mappersession module maintains backward compatibility with webmapper's JSON structure, allowing the loading of previously saved mappings with older versions of webmapper. To improve the portability of session files, a versioned JSON schema has been designed for storing information about maps, user interface properties and signal values a user chooses to be initialized once the session is loaded.

Aside from simple loading and saving, mappersession also supports *persistent sessions* to handle devices disconnecting and reconnecting. In a persistent session, the manager monitors the libmapper signal graph and waits until all signals in each managed map are present on the network before loading the map, and reloads the map as necessary as its signals reappear. In a distributed framework like libmapper, this is an important feature for performances with many devices to keep mappings active throughout the whole session.

Additionally, users are able to load any number of sessions at once, changing the active session with a libmapper signal representing the session index. By mapping another signal to the index signal, users can explore the effects of cycling through groups of mappings in real time.

4.2.3 Improving signal readability

Aside from the distribution changes from Section 4.2.1, I also focused on the readability and usability of the webmapper graphical session manager. Refining these rough edges should make the program (and the framework) more approachable for artists designing mappings. Several existing bugs for Windows users have been fixed, including more readable network interfaces and

[17] https://github.com/libmapper/mappersession

improvements to network interface selection and clean exiting of the program.

When mapping between signals that do not have their own graphical visualizers, it can sometimes be difficult for users to understand the behavior of the source signals or to tune map processing expressions. For this reason, a simple signal value plotter has been added to webmapper (Figure 4.2) that can be opened for any signal on the network. A more sophisticated standalone signal plotter that supports multiple signals, vectors and signal instances was also added to the libmapper utilities[18]. This feature has proven useful in the design of devices as well, using visualized sensor values to adjust signal ranges in the firmware of the device.

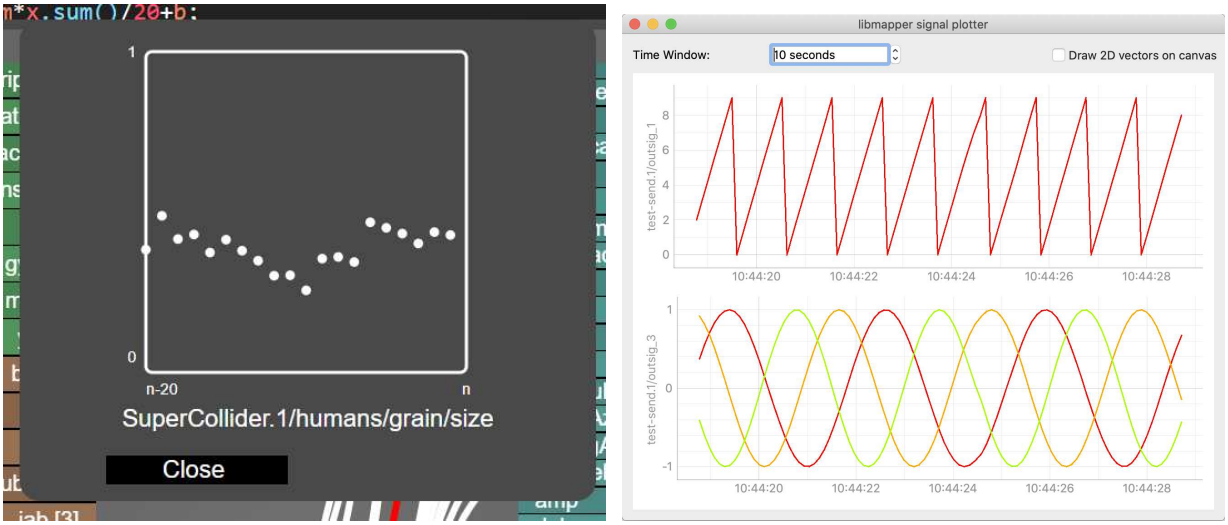

Fig. 4.2 Left: the real time signal value plotter in webmapper, with the x-axis as time and the y-axis as the signal's range; right: a standalone signal plotter written in Python with the same axes as the webmapper plotter.

I have also simplified the signal displays in List and Grid View by moving less-relevant metadata fields to a tooltip rather than showing them in every box. Before this change, each signal displayed its length, range, data type and unit all within its box. While many of these fields are useful when fine-tuning mappings, displaying them all in each signal box can hinder the readability of larger lists of signals. Instead, a metadata tooltip was implemented that appears when hovering over a signal (Figure 4.3), providing scalability for new fields and reducing visual clutter.

[18] http://libmapper.org/ecosystem/utilities

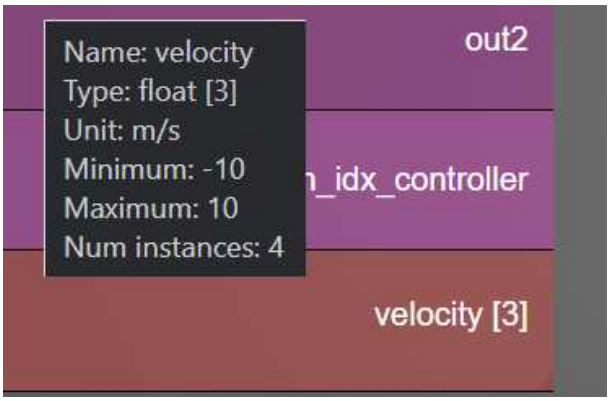

Fig. 4.3 An example of signal metadata displayed as a tooltip to increase signal readability. The signal's name and length is displayed at all times, with its other properties appearing in a tooltip when the mouse hovers over the component.

4.2.4 Mapper4Live: a libmapper to Ableton Live bridge

Ableton Live[19] is a popular commercial music production and performance program known as a digital audio workstation (DAW). Uniquely, it presents a number of developer-friendly tools for creating devices that can interact with the production session, enabling the creation of a libmapper bridge called *Mapper4Live*. Details regarding Mapper4Live's design and implementation have been adapted from Boettcher, Malloch, et al. (2022), where the plugin was first presented.

Ableton's Live Object Model

Ableton Live uniquely presents its internal structure to developers in the form of its Live Object Model (LOM). The LOM is hierarchically structured to organize tracks, software devices and audio parameters in the production session and can be accessed using Max objects. State variables such as the currently selected track or audio parameter can be accessed via the LOM to track the user's interactions with the program. Parameters from synthesis and effect plugins are contained in this hierarchy as well, giving Max for Live developers the ability to control other devices in the session.

These features provide a number of useful tools for mapping frameworks. In the case of libmapper, this results in the ability to view and control the parameter spaces of all audio devices

[19] https://www.ableton.com/en/live/

in the production session, giving mapping designers a multitude of new synthesis and audio effect signals that can be connected to gestural controllers.

Bringing communities together

A partnership between Ableton and Cycling 74, the company that maintains and develops Max, embeds a version of Max into Ableton Live called Max for Live[20] that lets developers create their own software devices in Ableton Live using the Max framework. Functionally the same as software plugins hosted by Ableton Live, Max for Live devices can produce and process audio as well as MIDI. These hackable devices also have access to the LOM, giving developers access to other plugins and parameters in the production session. This intended extensibility of the Ableton Live platform provides a natural entry point to connect with libmapper, and thus Mapper4Live was created as a Max for Live device.

Functionality in other DAWs

Mapper4Live is built as a Max for Live device, meaning that it is only compatible with Ableton Live and would need a different implementation to work in other DAWs. Max for Live is unique in that it exposes the session parameters, and most popular DAWs do not offer similar tools to developers. However, a similar approach can be done with other DAWs as well if the proper interfaces are exposed. For example, researchers at the University of Bordeaux are exploring integrating libmapper into the structure of Ossia Score (Celerier et al., 2015), an open-source DAW. This addition to Score would give similar functionality to Mapper4Live, letting users expose libmapper signals from the production session.

Prerequisite work

Ableton Live, as well as Max for Live runs on Windows and MacOS, while libmapper was originally built and extensively tested under Unix environments (MacOS and Linux), with Max external

[20] https://www.ableton.com/en/live/max-for-live/

objects available for MacOS. It would be ideal for the Mapper4Live interface to operate on both platforms that Ableton and Max support, and in this section I present updates to the existing components necessary: the main libmapper library as well as the Max/MSP external objects[21].

While in theory it had been possible to build libmapper on Windows using the MinGW toolchain[22], any applications compiled this way would not work with Windows-based applications developed using the Visual Studio compiler and run-time. This meant for example that the Max external objects, built with Windows via the Max software development kit (SDK) in Visual Studio, would not be supported. As such, changes were made to libmapper and subsequently the Max and Pure Data externals. The most notable changes include the removal of variable length array definitions[23], which are not supported by the Visual Studio C compiler. In addition to enabling the development of Mapper4Live, these updates to libmapper provided better compatibility of the library including support for native Windows applications, as well as Max external objects in Windows. With these changes, it was possible to embed libmapper, via the Max external object, into a Max for Live device, and provide the fundamental interfaces to implement the Mapper4Live plugin.

Plugin design

The publicly available LFO (low frequency oscillator) Max for Live device[24] was used as a reference for retrieving information from the LOM because of its parameter mapping functionality. Max for Live devices are inherently editable, allowing the parameter mapping subpatches in the LFO device to be copied and tweaked for the new plugin. The *live.path* Max object can be used to detect changes in LOM variables, and is used by Mapper4Live to listen to changes in the currently selected parameter when adding new signals. Once a new parameter is selected to be added to Mapper4Live, the *live.object* Max object retrieves information about the parameter including its name, parent device, value range and ID within the session. The parameter's name, parent and

[21] https://github.com/malloch/mapper-max-pd
[22] https://www.mingw-w64.org/
[23] https://github.com/libmapper/mapper-max-pd/pull/2
[24] https://www.ableton.com/en/packs/max-live-essentials

ID are formed into a unique hierarchical address for the libmapper network, while the range allows libmapper to automatically normalize incoming values for the signal. Finally, the device creates a mappable libmapper signal for the parameter on the network.

Fig. 4.4 The Mapper4Live plugin interface. Parameters from the Claverb instrument in Ableton Live are exposed on the libmapper network where gestural controllers can modulate their values in real time.

The plugin operates by users first clicking an open Map button, and then clicking on an Ableton Live parameter in the session that they wish to connect with libmapper. Once created, the signals will appear on the libmapper network under a *mapper.x* device, x being the instance number of the object. This allows users to create multiple instances to separate signals between tracks if intended. Clicking a Map button's corresponding "X" button to its right will remove the signal from the network, deleting any connections containing the signal as well.

Although Mapper4Live could exist as an audio effect device, it was created as a MIDI effect device in order to always place it at the beginning of the chain for visibility (seen in Figure 4.4). The device can be placed on any track without any functional changes and can create signals from any other track's parameters. Once signals are created using Mapper4Live, webmapper can be opened via the "edit mappings" button to manage connections. Users can also connect mappings on the network with libmapper's command line functions, but webmapper provides much more user-friendly controls for the connections.

4.2.5 Mapper4TD: a libmapper to TouchDesigner bridge

A new libmapper bridge has also been recently created for the popular multimedia art program TouchDesigner[25], opening up new opportunities for real-time audiovisual mapping for an established community. Seen in Figure 4.5, Mapper4TD[26] supports both source and destination signal types, letting users send TouchDesigner's many source signals out to other devices as well as modulate TouchDesigner parameters with external devices.

Fig. 4.5 Internals of the Mapper4TD container, showing its inline setup and usage instructions at the top, components for source signals on the left (inSources), destination signals on the right (dstSignalNames) and a libmapper device written in Python that manages signal networking in the bottom left (MapperDevice).

[25] https://derivative.ca/
[26] https://github.com/libmapper/Mapper4TD

Plugin implementation

Plugins in TouchDesigner are called *operators*, and are able to connect signals to each other with drag-and-drop wires. TouchDesigner is bundled with its own Python environment, providing abstract structures for developers to use when creating a plugin. I was able to use this environment along with libmapper's Python bindings to create a libmapper device and control its signals from within a TouchDesigner operator. One of TouchDesigner's LFO operators was then connected to trigger signal updates at a fixed rate and push them to the network. Finally, Mapper4TD was packaged into a portable Container[27] along with detailed setup and usage instructions. As the plugin was designed entirely with Python and TouchDesigner's internal operators, it is cross-platform compatible and requires no compilation.

4.3 Artistic evaluations

In order to determine the impact of the applied development principles on libmapper's maturity, I have organized two evaluations. The first consists of experimentation sessions with Mapper4Live and Probatio (discussed in Section 2.2.1), where the capabilities of Mapper4Live are tested to discover the workflow and benefits of using a complex mapping framework in a music production program. Next, libmapper's barriers to entry and operation are evaluated with the design of an interactive audiovisual installation by professional sound and visual artists.

4.3.1 Mapper4Live experimentation

Mapper4Live opens many opportunities for exploring complex mappings with professional sound synthesis and effects software. Probatio was chosen as an input device for the explorations due to its wide range of interaction types (e.g., joysticks, knobs, pistons) and compatibility with libmapper. In the first experimentation session (seen in Figure 4.6), an artist improvised using a piano MIDI controller while its synthesis and effect parameters were modulated by Probatio modules. In the

[27] https://derivative.ca/UserGuide/Container_COMP

second session, MIDI was automatically generated using a machine learning model[28], leaving only one performer to modulate the incoming notes.

Fig. 4.6 A jam session using Probatio signals mapped to Ableton Live parameters using Mapper4Live.

Expressivity affordances

Various mappings were tested between the Probatio modules and Ableton parameters in each session to determine what (if any) benefits arose in a music production environment when using a complex mapping framework and novel input devices. Libmapper's expression syntax[29] was used to fine-tune relationships between signals and create new representations. For example, the joystick's X and Y position signals were used to calculate the overall movement speed for some mappings, forcing the performer to supply continuous energy to the system for modulation to occur. Typical mappings in music production programs are limited to simple one-to-one mappings, therefore this affordance of intimate control over mappings with Mapper4Live provides music producers with an improved set of tools for expressing their intents. It is important to mention that modulations from

[28] https://magenta.tensorflow.org/music-transformer
[29] https://github.com/libmapper/libmapper/blob/main/doc/expression_syntax.md

Mapper4Live can also be recorded into the audio timeline as automation data, allowing artists to record portions of gestural data in real time onto the track and even edit each data point at a later time, integrating directly into Ableton's modulation workflow.

4.3.2 An interactive audiovisual installation

The developments described in this chapter are intended to increase the usefulness and desirability of libmapper as a mapping option for artists with little to no development experience. To evaluate this claim, I have organized the creation of an interactive audiovisual installation project by commissioning two artist collaborators with varying levels of technical experience. The artists were able to make their own design choices to create two distinct interactive sessions over the course of one month, but were constrained to using libmapper for their signals and mappings. This project provided the opportunity to evaluate many of the changes to libmapper introduced in this chapter including ease of setup and operation, session management and the mapping design process for artists. After describing the installation design process and demonstration, feedback from the artists was assessed to determine the impact of the changes on the usability of the mapping framework and the effectiveness of the development principles imposed.

Artist profiles

The sound artist is a professional software engineer experienced with SuperCollider for live coding performances and freelance works, using the MacOS platform for their development. The visual artist, with rudimentary Python scripting experience, is proficient with TouchDesigner on their Windows machine and has created interactive patches in the past using the application's integrated LeapMotion and MIDI modules. While neither artist has used a distributed mapping framework like libmapper before, both have experience with designing basic mappings with MIDI and OSC. The diversity in the artists' development experience and platform preference presents a unique opportunity to examine the ease of setup and operation of libmapper throughout the design of the installation.

Devices and programs

The installation was designed using a combination of tools developed at the Société des Arts Technologiques (SAT) and the Input Devices and Music Interaction Laboratory (IDMIL). Three T-Stick DMIs (Malloch & Wanderley, 2007) were used as input devices, each providing 18 gestural signals made available on the libmapper network and through OSC. The sound artist used SuperCollider for synthesis, adopting the SAT's open-source SATIE plugin (Bouillot et al., 2017) for spatialization. The visual artist created projection-mapped visuals with TouchDesigner, testing the program's new libmapper bridge. As each of these systems is compatible with libmapper, the T-Sticks are able to control the audio and visual signals in each program in real time.

The design process

The artists began by choosing a theme to connect the two sessions, deciding upon representing human interactions with nature and technology. The artists then set out to design audio and visual patches for nature and technology, each with their own set of mappable signals. In order to emphasize experimentation in the mapping design process, the artists separately chose which signals from their programs to expose on the network and left mapping design for when the patches were nearly complete.

Patches for TouchDesigner and SuperCollider were designed by the artists using generic output models for audio and video, utilizing SATIE's abstracted models for sound spatialization. The use of generic output models and distributed mappings makes the project portable to many types of systems with different projection mapping and spatialization requirements.

Mapping design for the project occurred during several experimentation sessions throughout the process with myself and the artists, searching for mappings that were meaningful to the piece. The ability of both artists to understand the source signals from the T-Sticks and the destinations in the audiovisual patches was an important requirement for mapping ideation, leading to the creation of tables with detailed signal descriptions and modulation examples. By virtue of

libmapper's distributed approach, mapping ideas could be created and tested easily without the need to change any of the devices or patches.

The artists aimed for the interactions to mirror the theme of each session by using continuous, flowing media and movements for nature and discrete, abrupt ones for technology. The final nature-themed session contained 19 maps and the technology-themed session contained 27 maps, with control divided between three T-Sticks. The T-Stick's *Shake*, *squeeze* and *orientation* signals were used with one T-Stick each, keeping the devices consistent in their interaction types (Figure 4.7). All patches and mappings for each session have been made open source as well to encourage further contributions or reuse on other systems[30].

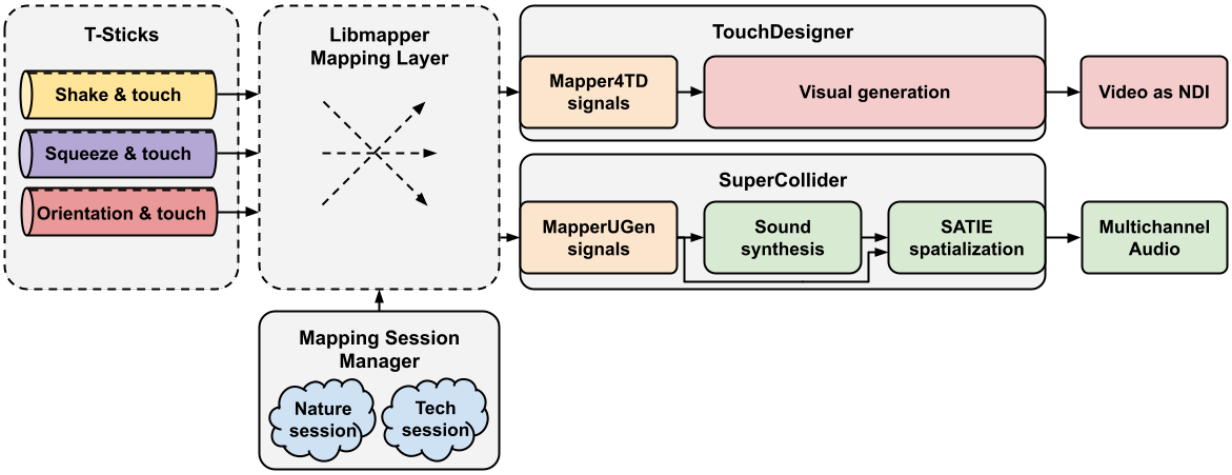

Fig. 4.7 Overview of the systems and interactions used in the interactive installation project.

Satosphére demo session

As the final stage of the artistic explorations proposed in this project, the artists were invited to set a demonstration of their created immersive space in the Satosphére. The Satosphére, created in 2011 and located in Montréal, is a full dome with a diameter of 18 meters, and is a permanent immersive modular theatre dedicated to artistic creation and events[31]. This space is

[30] https://github.com/IDMIL/Human-Nature-Installation
[31] https://sat.qc.ca/en/satosphere

capable of 360-degree video projection and a complex speaker setting using 157 speakers grouped to form 31 virtual speakers organized around the dome. These characteristics make the Satosphére an excellent candidate for testing the mapping capabilities of libmapper, webmapper, the bridges developed during this project, and SATIE's ability to adapt to different venues by invoking various spatializer presets.

Moreover, the Satosphére's equipment is configured to provide tools for artists to connect their devices. The connections using audio and video on the Satosphére usually employ Network Device Interface (NDI) as the standard video transmission protocol and MADI-compatible devices[32] for carrying digital audio. This setup allowed the artists to work individually in generating artistic content without concern about how the devices will be connected at the venue.

The same workflow was employed for the data streaming. In that sense, using a decentralized signal mapping framework using libmapper allowed artists to individually work with their content and define which control parameters would be exposed to be subsequently mapped during the demo session. These control parameters were available as libmapper signals, created using the bindings and bridges described in Section 4.2.1.

Simultaneously, I focused on creating mapping sessions based on the libmapper signals (control parameters) provided by the artists. This process allowed modular prototyping while streamlining the setup for the event. Each artist was able to simply connect with the available network to automatically expose their libmapper signals.

Unfortunately, multiple technical issues were encountered during setup. These issues were not caused by the use of libmapper, but resulted from networking and connectivity limitations of some devices. Describing some of the issues encountered might be useful as a description of commonly found obstacles during interactive demos, performances and installations. The strategies used to overcome these problems may be relevant and applicable to future projects.

First, a large number of missing User Datagram Protocol (UDP) packets were noticed when more than one T-Stick was connected to the network. This caused the T-Sticks to intermittently

[32] https://www.rme-audio.de/rme-madi-technology.html

disconnect and reconnect from the network, blocking the mapping session from being fully active. Upon troubleshooting, the issue was attributed to network problems regarding the particular model of microcontroller in the T-Stick. Working with distributed mapping allowed me to program a quick on-site fix in the form of an OSC-to-libmapper forwarding program that discovers OSC signals, creates corresponding libmapper signals on the network and forwards the signal values through to libmapper. Modifying the point in the network where the T-Stick libmapper signals were created, I could overcome the issue, albeit by adding a bit of latency to the pipeline. Nevertheless, the sessions were finally able to connect all of the mappings on site the day it was developed.

Another hurdle appeared as I realized that the Satosphére requires NDI over Ethernet for video, while libmapper requires Wi-Fi for T-Stick connections. After problems arose when attempting to use both network interfaces at once, a Magewell HDMI-to-NDI converter device[33] was instead utilized, avoiding the need to use the Ethernet interface on the TouchDesigner machine.

Overall, the setup time and troubleshooting related to mappings were straightforward in comparison with similar events organized at the Satosphére, according to the venue technicians. More importantly, assembling the individual artists' work into a cohesive installation posed little challenge as each libmapper device could be discovered over the network, and the mapping session manager could automatically restore all previously saved connections.

Feedback from artists

After the conclusion of the installation project, I interviewed the artists to gather information about their experience using libmapper. I explored the effect of technical experience on the ease of setup and use of the framework as well as the resulting feasibility and desire to use it again in the future.

The visual and audio artists claimed that it was simple and quick for them to install libmapper, webmapper and the bridge for their respective programs and platforms. This alone speaks to an

[33] https://www.magewell.com/products/pro-convert-hdmi-tx

increase in usability for artists across platforms and technical levels as a result of the distribution and CI principles implemented. The audio artist had some initial trouble integrating MapperUGen into their preferred SuperCollider workflow as their method of sequencing continuously created and destroyed the libmapper signals, but was able to find an alternative using the new signal Bus feature discussed in Section 4.2.1. The visual artist was able to use their basic Python skills to install libmapper and set up Mapper4TD in TouchDesigner without issue by following the instructions in the Mapper4TD container. Each artist went about choosing signals to expose on the network by experimentation, often testing new audiovisual signals with various T-Stick signals to get a sense of the effect on the patch and to find desirable connections for the session.

When asked whether they would be willing to use libmapper in the future, both agreed that it would be a great fit for sufficiently complex artistic projects like this one and felt comfortable using the framework. Once they had the concept of distributed signals and mappings "figured out", they enjoyed the visual style of designing mappings with webmapper and recognized the benefits of libmapper compared to MIDI and OSC. The positive feedback regarding libmapper's ease of setup and use for different platforms, programs and technical levels points towards an increase in usability and accessibility for the framework due to the developments introduced in this chapter.

In addition to the informal interviews, I utilized the System Usability Scale (SUS) survey introduced in Section 2.2.1. After receiving responses from each artist, the individual SUS scores were 72.5 and 40, producing a mean value of 56.25. One observation from the results is that the artist working as a programmer scored the framework much higher than the non-developer artist. This indicates that the framework requires additional work to become easily usable by artists of all technical levels. As this is the first instance of a SUS evaluation for libmapper, its score may also serve as a baseline for future work toward the same goals.

4.4 Summary

This chapter introduces libmapper, a distributed signal mapping framework, and reviews several challenges to its maturity including documentation, compatibility, automated distribution and

mapping session management. Developments to address these barriers are discussed, evaluating their impact by organizing the design of an interactive audiovisual installation piece by artists. Positive feedback regarding the framework's ease of setup and operation suggests the effectiveness of the development principles imposed.

Chapter 5

Conclusions and future work

I have presented two case studies aimed at developing the maturity of DMIs and mapping tools using design and development principles introduced in literature. In the first study, the redesign of a standalone percussive DMI is discussed, using continual iteration and evaluation loops to improve the instrument's reliability, stability and usability by artists. The second study reviewed recent developments to libmapper to support its availability, compatibility and usability.

This chapter summarizes the impact of my contributions to each project's maturity and suggests future work toward the same goal.

5.1 Impact of the applied practices

Several design and development principles supporting NIME maturity have been introduced and applied to the case studies in this thesis. First, the Tapbox DMI was evaluated to set the stage for its redesign: the Slapbox. Working from an existing instrument allowed me to reuse much of the enclosure and electronics from the original design. Focusing on prototyping the new interaction methods, I made quick design decisions to evaluate the capabilities of sensors as well as synthesis methods to map to the interactions. Throughout the instrument's design, its build instructions and software were continually documented to increase its replicability for future designers. Next, multiple iteration and evaluation loops were performed with professional percussionists to identify

shortcomings in performance and usability. These practices formed the Slapbox into a replicable, engaging instrument that artists are able to pick up and play with no help from the original designer.

The second case study involved developing the maturity of libmapper, a distributed signal mapping framework. The framework's large collection of tools born from research has created several barriers to its entry and operation by artists and researchers with limited development experience. The first barrier addressed was the distribution of libmapper and its related tools. CI practices were used to automate the library's releases for all platforms, eliminating the need for users to manually compile and install the program.

Similarly, many of libmapper's application bridges were made compatible with all platforms, making sure to document the new repository and release links on libmapper's website. Two new cross-platform application bridges were also presented, providing new contexts for the framework's use and evaluation in established artistic communities. An interactive audiovisual installation project was organized with two artists to evaluate the impact of many of these developments, resulting in positive feedback regarding libmapper's usability.

Feedback from the case studies has reinforced the effectiveness of the above practices to support usability, availability, replicability and communities for both DMIs and mapping tools.

5.1.1 Applications to other projects

The aforementioned development and design principles can be applied to other projects as well by using the design diagram presented in Figure 2.1. By following this flow of design priorities, the project can continually improve its performance while supporting stability and replicability. While most research projects are created with limited timelines, these principles should be prioritized to encourage long-term use by researchers and artists.

The software development principles discussed in Section 4.2.1 are especially important for research tools, which often lack the necessary distribution and documentation to make the project extensible for developers and accessible to users. Good CI practices to support testing and dis-

tribution are vital for research software that functions as middleware such as libmapper, where multiple tools must all be accessible and operable by a user at once.

5.2 Future work

5.2.1 Slapbox improvements

Based on suggestions from the evaluations and design goals, there are many opportunities to continue development on the Slapbox to increase its reliability. The Slapbox's Velostat sensors perform well to detect both continuous and sudden interactions, yet are unable to provide accurate velocity estimations for strikes. Although the accuracy of detecting strikes has increased notably since the original design, sensor fusion with PVDF film[1] or other transducers might improve velocity estimations without hindering strike detection.

Loading custom samples into the Slapbox was a popular request during the user evaluations, and could be accomplished using the GUI in a future iteration. Related to this, additional research could also explore new mappings between the device's control space and sound synthesis. For example, percussion physical modeling synthesis could be used to imitate acoustic responses on a drum head using the position value from pads. The Slapbox's interaction surfaces can also be used for non-percussive or harmonic mappings to explore its use in other musical contexts.

Another development would be to utilize libmapper to open opportunities for user-defined mappings. This change would require the declaration of the Slapbox's signal space which would include sensor values and high-level gesture data as well as its synthesis parameters. Table 3.2 illustrates one method of exposing the Slapbox's signal space with libmapper. External synthesis engines compatible with libmapper could be used if implemented as well, treating the Slapbox as a control surface. This would provide opportunities for further evaluations of how the Slapbox is used with new types of mappings and synthesis.

Aside from these features, there is still work to be done to support the instrument's replicability.

[1] https://piezopvdf.com/piezo-pvdf-film/

The Slapbox's construction process and software organization is well-documented, though only one copy currently exists. Building additional Slapboxes would enable new designers and artists to learn about and use the device, and may uncover challenges to its replicability that I was not aware of.

5.2.2 Libmapper improvements

Though the developments presented in Chapter 4 are a step in the right direction to support users, there is still more work to be done to improve the distribution and usability of libmapper.

Session management

Mappersession, the mapping session manager presented in Section 4.2.2, will be expanded upon as well by creating a shared object library similar to libmapper to allow for use in both Python and C programs. Though this will require another large push for distribution and CI with Github Actions to support usability, it is important to have a session manager that can be integrated into applications with any language.

The session manager currently makes some unsafe assumptions about the state of the network, which may lead to conflicts when multiple sessions are active. Future versions will let users choose more rigorous methods of map staging instead of persistently reconnecting any disconnected maps, such as one-time staging to initialize a session as devices appear. Webmapper's integration with mappersession will also be developed further to utilize mappersession's ability to save and restore graphical data in sessions. This feature is useful for visualization methods that allow users to organize the positions of connections in a session, restoring the state when the session is loaded.

Distribution

Next, all of libmapper's bridges should be built using Github Actions for CI and distributed to each community's package manager or repository in order to increase visibility, get additional feedback about the framework and grow a user community. Many of the bridges also lack support

for creating vectored and instanced signals. These features unlock many new opportunities for mappings, yet have remained largely untested by users due to the lack of support from bridges.

Additional usability trials

The interactive audiovisual installation project discussed in Section 4.3.2 produced interesting new use cases for libmapper and allowed the testing of many new developments to the framework, yet fell short of providing conclusive usability results. The small sample size of artists evaluating the framework was not enough to definitively evaluate the impact of the developments on the framework's usability for artists of all technical levels. Additional evaluations should be conducted with a larger group of artists to determine the state of usability and discover a path forward to support its maturity.

Bibliography

Agusa, K. (2004). Software Engineering Evolution. *Proceedings of the International Workshop on Principles of Software Evolution*, 3–8.

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *Journal of Human–Computer Interaction*, *24*(6), 574–594.

Boettcher, B., Malloch, J., Wang, J., & Wanderley, M. M. (2022). Mapper4Live: Using Control Structures to Embed Complex Mapping Tools into Ableton Live. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Boettcher, B., Meneses, E., Frisson, C., Wanderley, M. M., & Malloch, J. (2023). Addressing Barriers for Entry and Operation of a Distributed Signal Mapping Framework. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Boettcher, B., Sullivan, J., & Wanderley, M. M. (2022). Slapbox: Redesign of a Digital Musical Instrument Towards Reliable Long-Term Practice. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Bouillot, N., Settel, Z., & Seta, M. (2017). Satie: A Live and Scalable 3d Audio Scene Rendering Environment for Large Multi-Channel Loudspeaker Configurations. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 404–409.

Bowers, J., & Archer, P. (2005). Not Hyper, Not Meta, Not Cyber but Infra-Instruments. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 5–10.

Brooke, J., et al. (1996). Sus-a Quick and Dirty Usability Scale. *Usability Evaluation in Industry*, *189*(194), 4–7.

Calegario, F., Tragtenberg, J., Frisson, C., Meneses, E., Malloch, J., Cusson, V., & Wanderley, M. (2021). Documentation and Replicability in the NIME Community. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Calegario, F., Tragtenberg, J., Wang, J., Franco, I., Meneses, E., & Wanderley, M. M. (2020). Open Source DMIs: Towards a Replication Certification for Online Shared Projects of Digital Musical Instruments. *Proceedings of the International HCI Conference*, 84–97.

Calegario, F., Wanderley, M., Tragtenberg, J., Wang, J., Sullivan, J., Meneses, E., Franco, I., Kirkegaard, M., Bredholt, M., & Rohs, J. (2020). Probatio 1.0: Collaborative Development of a Toolkit for Functional DMI Prototypes. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 21–25.

Calegario, F., Wanderley, M. M., Huot, S., Cabral, G., & Ramalho, G. (2017). A Method and Toolkit for Digital Musical Instruments: Generating Ideas and Prototypes. *IEEE MultiMedia*, *24*(1), 63–71.

Caramiaux, B., Françoise, J., Schnell, N., & Bevilacqua, F. (2014). Mapping Through Listening. *Computer Music Journal*, *38*(3), 34–48.

Celerier, J.-M., Baltazar, P., Bossut, C., Vuaille, N., Couturier, J.-M., & Desainte-Catherine, M. (2015). Ossia: Towards a Unified Interface for Scoring Time and Interaction. *Proceedings of the International Conference on Technologies for Music Notation and Representation*.

Chiprianov, V., Kermarrec, Y., & Rouvrais, S. (2011). On the Extensibility of Plug-ins. *Proceedings of the International Conference on Software Engineering Advances*, 557–562.

Cook, P. R. (2009). Re-Designing Principles for Computer Music Controllers: A Case Study of SqueezeVox Maggie. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 218–221.

Dannenberg, R. B., & Chi, Z. (2016). O2: Rethinking Open Sound Control. *Proceedings of the International Computer Music Conference*, 493–496.

Frisson, C., Bredholt, M., Malloch, J., & Wanderley, M. M. (2021). Maplooper: Live-Looping of Distributed Gesture-to-Sound Mappings. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Fukuda, T., Meneses, E., West, T., & Wanderley, M. M. (2021). The T-Stick Music Creation Project: An Approach to Building a Creative Community Around a DMI. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Goudard, V. (2019). Ephemeral Instruments. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 349–354.

Hannula, M. (2008). Catch Me if You Can-Chances and Challenges of Artistic Research. *Arts and Research: A Journal of Ideas, Contexts and Methods*, *2*(2).

Harrold, M. J. (2000). Testing: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*, 61–72.

Heron, M., Hanson, V. L., & Ricketts, I. (2013). Open Source and Accessibility: Advantages and Limitations. *Journal of Interaction Science*, *1*(1), 1–10.

Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016). Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. *Proceedings of the International Conference on Automated Software Engineering*, 426–437.

Hunt, A., & Wanderley, M. M. (2002). Mapping Performer Parameters to Synthesis Engines. *Organised Sound*, *7*(2), 97–108.

Hunt, A., Wanderley, M. M., & Paradis, M. (2003). The Importance of Parameter Mapping in Electronic Instrument Design. *Journal of New Music Research*, *32*(4), 429–440.

Huott, R. (2002). An Interface for Precise Musical Control. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 1–5.

Ergonomics of Human-System Interaction — Part 210: Human-Centred Design for Interactive Systems (Standard). (2019). International Organization for Standardization. Geneva, CH.

Jathal, K., & Park, T.-H. (2016). The HandSolo: A Hand Drum Controller for Natural Rhythm Entry and Production. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 218–223.

Jordà, S. (2004). Instruments and Players: Some Thoughts on Digital Lutherie. *Journal of New Music Research*, *33*(3), 321–341.

Kapur, A., Davidson, P., Cook, P. R., Driessen, P. F., & Schloss, W. A. (2004). Digitizing North Indian Performance. *Proceedings of the International Computer Music Conference*.

Kirkegaard, M., Bredholt, M., Frisson, C., & Wanderley, M. (2020). TorqueTuner: A Self Contained Module for Designing Rotary Haptic Force Feedback for Digital Musical Instruments. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 273–278.

Koehly, R., Curtil, D., & Wanderley, M. M. (2006). Paper FSRs and Latex/Fabric Traction Sensors: Methods for the Development of Home-Made Touch Sensors. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 230–233.

Lazarus, N., & Bedair, S. S. (2020). Creating 3D Printed Sensor Systems with Conductive Composites. *Smart Materials and Structures*, *30*(1), 015020.

Lepri, G., & McPherson, A. (2019). Making Up Instruments: Design Fiction for Value Discovery in Communities of Musical Practice. *Proceedings of the International Conference on Designing Interactive Systems*, 113–126.

Mackay, W. (2000). Responding to Cognitive Overload: Co-Adaptation Between Users and Technology. *Intellectica*, *30*(1), 177–193.

Malloch, J., Sinclair, S., & Wanderley, M. M. (2007). A Network-Based Framework for Collaborative Development and Performance of Digital Musical Instruments. *International Symposium on Computer Music Modeling and Retrieval*, 401–425.

Malloch, J., Sinclair, S., & Wanderley, M. M. (2013). Libmapper: (A Library for Connecting Things). *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, 3087–3090.

Malloch, J., & Wanderley, M. M. (2007). The T-Stick: From Musical Interface to Musical Instrument. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 66–70.

Marquez-Borbon, A., & Martinez Avila, J. P. (2018). The Problem of DMI Adoption and Longevity: Envisioning a NIME Performance Pedagogy. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 190–195.

McFee, B., Kim, J. W., Cartwright, M., Salamon, J., Bittner, R. M., & Bello, J. P. (2018). Open-Source Practices for Music Signal Processing Research: Recommendations for Transparent, Sustainable, and Reproducible Audio Research. *IEEE Signal Processing Magazine*, *36*(1), 128–137.

McPherson, A., & Kim, Y. (2012). The Problem of the Second Performer: Building a Community Around an Augmented Piano. *Computer Music Journal*, *36*(4), 10–27.

McPherson, A., Morreale, F., & Harrison, J. (2019). Musical Instruments for Novices: Comparing NIME, HCI and Crowdfunding Approaches [Publisher: Springer]. *New Directions in Music and Human-Computer Interaction*, 179–212.

Medeiros, R., Calegario, F., Cabral, G., & Ramalho, G. (2014). Challenges in Designing New Interfaces for Musical Expression. *Proceedings of the International Conference on Design, User Experience, and Usability*, 643–652.

Moore, F. R. (1988). The Dysfunctions of MIDI. *Computer Music Journal*, *12*(1), 19–28.

Morreale, F., & McPherson, A. P. (2017). Design for Longevity: Ongoing Use of Instruments from NIME 2010-14. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 192–197.

Morreale, F., Moro, G., Chamberlain, A., Benford, S., & McPherson, A. P. (2017). Building a Maker Community Around an Open Hardware Platform. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 6948–6959.

O'Modhrain, S. (2011). A Framework for the Evaluation of Digital Musical Instruments. *Computer Music Journal*, *35*(1), 28–42.

Preece, J., Sharp, H., & Rogers, Y. (2015). *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons.

Sokolovskis, J., & McPherson, A. P. (2014). Optical Measurement of Acoustic Drum Strike Locations. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 70–73.

Staudt, P. (2016). *Development of a Digital Musical Instrument with Embedded Sound Synthesis* [Master's thesis, Fakultät I Institut für Sprache und Kommunikation, Technische Universität Berlin]. Germany.

Sullivan, J. (2015). Noisebox: Design and Prototype of a New Digital Musical Instrument. *Proceedings of the International Computer Music Conference*, 266–269.

Sullivan, J. (2021). *Built for Performance: Designing Digital Musical Instruments for Professional Use* [Doctoral dissertation, McGill University].

Sullivan, J., Vanasse, J., Guastavino, C., & Wanderley, M. M. (2020). Reinventing the Noisebox: Designing Embedded Instruments for Active Musicians. *Proceedings of the International Conference on New Interfaces for Musical Expression*.

Sullivan, J., & Wanderley, M. M. (2018). Stability, Reliability, Compatibility: Reviewing 40 Years of DMI Design. *Proceedings of the Sound and Music Computing Conference*, 319–326.

Tindale, A. R., Kapur, A., Tzanetakis, G., Driessen, P., & Schloss, A. (2005). A Comparison of Sensor Strategies for Capturing Percussive Gestures. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 200–203.

Tom, A. J., Venkatesan, H. J., Franco, I., & Wanderley, M. (2019). Rebuilding and Reinterpreting a Digital Musical Instrument - The Sponge. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 37–42.

Wanderley, M. M. (2001). Gestural Control of Music. *Proceedings of the International Workshop on Human Supervision and Control in Engineering and Music*, 632–644.

Wanderley, M. M., & Depalle, P. (2004). Gestural Control of Sound Synthesis. *Proceedings of the IEEE*, *92*(4), 632–644.

Wang, J., Malloch, J., Sinclair, S., Wilansky, J., Krajeski, A., & Wanderley, M. M. (2019). Webmapper: A Tool for Visualizing and Manipulating Mappings in Digital Musical Instruments. *Proceedings of the International Symposium on Computer Music Multidisciplinary Research*, 823.

Wessel, D., & Wright, M. (2002). Problems and Prospects for Intimate Musical Control of Computers. *Computer Music Journal*, *26*(3), 11–22.

West, T., Caramiaux, B., & Wanderley, M. (2020). Making Mappings: Examining the Design Process. *Proceedings of the International Conference on New Interfaces for Musical Expression*.