UNIVERSITY
OF OSLO

Master’s thesis

Exploring multi-touch instruments with MPE and MIDI 2.0

Kristian Wentzel

Music, Communication and Technology
30 ECTS study points

Department of Musicology
Faculty of Humanities

Autumn 2023

Kristian Wentzel

Exploring multi-touch instruments with MPE and MIDI 2.0

Supervisor:
Alexander Refsum Jensenius

Abstract

This thesis explores multi-touch instruments with MIDI Polyphonic Expression (MPE) and MIDI 2.0 from a performer's perspective. It discusses mapping strategies between a multi-touch controller and a synthesizer sound engine and provides insights into which synthesizer parameters are more suitable to control. It discusses an iterative prototyping approach to designing an expressive multi-touch instrument with MPE using a multi-dimensional controller connected to a synthesizer sound engine designed in Pure Data. Finally, it looks into the new MIDI 2.0 protocol, providing reflections on how this can benefit expressive control of an instrument in music performances compared to using MPE.

Acknowledgements

First and foremost, I would like to thank my supervisor, Alexander Refsum Jensenius, for his unparalleled expertise, positivity, and ability to inspire and keep my spirits up through this thesis. Thank you to all the great teachers and lecturers at the Music, Communication and Technology master's program led by Stefano Fasciani. The different courses provided have been a truly joyful, content-rich, and sometimes exhausting journey. Thank you to my fellow students in the program for all the good times — and the networked music performances.

I would also like to thank my family for bringing me up in a home full of music and for encouraging me when I decided to pursue a career in this profession.

Last, I will give my most heartfelt thank you to the ones at home. We were three when I started this journey, and now we are four. I could not have completed this without the most supportive, loving, caring, and amazing girlfriend at home, Sigrun. Thank you, Elvin and Ellinor, for letting Dad go to school.

Contents

Abstract . . . i
Acknowledgements . . . iii
1 Introduction . . . 1
1.1 Research Questions . . . 2
1.2 Definitions . . . 3
1.3 Scope and limitations . . . 4
1.4 Contributions . . . 4
1.5 Outline . . . 5
2 Theoretical Overview . . . 7
2.1 Expressive Multi-Touch Instruments . . . 7
2.1.1 TouchKeys . . . 8
2.2 Digital Sound synthesis . . . 10
2.3 Pure Data (Pd) . . . 11
2.4 The MIDI Protocol . . . 11
2.4.1 MIDI 1.0 . . . 12
2.4.2 MPE . . . 14
2.4.3 MIDI 2.0 . . . 14
2.5 Alternatives to MIDI . . . 16
2.5.1 ZIPI . . . 16
2.5.2 Open Sound Control (OSC) . . . 17
2.6 Mapping Strategies . . . 18
2.7 Summary . . . 21
3 Methodology . . . 23
3.1 Literature Review . . . 23
3.2 Iterative Design and Prototyping . . . 23
3.3 Programming . . . 24
3.4 Performance Research . . . 25
3.4.1 Introspection and Self-Reflection . . . 25
3.5 Practice Research . . . 25
3.6 Summary . . . 26
4 Mapping . . . 27
4.1 Experiences From Designing the P-6 . . . 27
4.1.1 The Relevancy of Input Sources . . . 28
4.2 Exploration of Destination Parameters . . . 29
4.2.1 The Relevancy of Resolution . . . 30
4.3 Mapping a Composition . . . 31
4.4 Summary . . . 31

5 Instrument Design . . . 33
- 5.1 Pre-production stage . . . 33
- 5.2 Designing the Synthesizer . . . 35
- 5.3 The Iterative Design Process . . . 37
  - 5.3.1 Prototype v1 . . . 37
  - 5.3.2 Prototype v2 . . . 38
  - 5.3.3 Prototype v3 . . . 39
- 5.4 The Instrument: P-6. . . . 40
- 5.5 Final Evaluation and Reflection . . . 41
- 5.6 Summary. . . . 41

6 Discussion . . . 43
- 6.1 Summary. . . . 43
- 6.2 Towards MIDI 2.0. . . . 44
  - 6.2.1 Make MIDI Great Again . . . 44
  - 6.2.2 What Will Become of MIDI 2.0? . . . 45
- 6.3 Answer to the Main Research Question . . . 45
  - 6.3.1 Answer to RQ1: Suitable parameters . . . 46
  - 6.3.2 Answer to RQ2: Design approaches . . . 46
  - 6.3.3 Answer to RQ3: Expressive control. . . . 46
- 6.4 Future Work . . . 46

A More on MIDI 2.0 . . . 49
- A.1 More on the MIDI 2.0 Specification . . . 49
- A.2 The Current Status of MIDI 2.0 . . . 53
  - A.2.1 Digital Audio Workstations (DAWs) . . . 54
  - A.2.2 Programming languages . . . 54
  - A.2.3 Operative Systems (OSs) . . . 55
  - A.2.4 Hardware . . . 55
  - A.2.5 Prototyping Tools . . . 55

B More on ZIPI . . . 57

C More on OSC . . . 61

D More on Sound Synthesis . . . 65
- D.1 Digital Sound Synthesis Techniques . . . 65
- D.2 P-6 Synthesizer Modules . . . 66

E Design & Prototyping Diary. . . . 69
- E.1 Oct/Nov 2023 - Initial design phase . . . 69
- E.2 11/12/2023 . . . 69
- E.3 12/12/2023 . . . 70
  - E.3.1 Iteration v0.1 – Oscillators . . . 70
- E.4 13/12/2023 . . . 71
  - E.4.1 Iteration v0.2 – Basic Single Voice . . . 71
  - E.4.2 Iteration v0.3 – Single Voice . . . 72
- E.5 14/12/2023 . . . 72
  - E.5.1 Iteration v0.4 – MPE . . . 73
- E.6 19/12/2023 . . . 73
  - E.6.1 Iteration v0.4.1 – MPE (+ sync) . . . 73
- E.7 20/12/2023 . . . 74
  - E.7.1 Iteration v0.5 – Polyphony (global modulation) . . . 74
- E.8 21/12/2023 . . . 75
  - E.8.1 Iteration v1 – It’s alive! . . . 75

E.9 04/01/2024 75
E.10 08/01/2024 76
E.11 10/01/2024 77
E.12 11/01/2024 77
E.12.1 Iteration v2 – Modulation Matrix + HPF 78
E.13 16/01/2024 79
E.14 18/01/2024 79
E.15 22/01/2024 79
E.16 23/01/2024 80
E.17 24/01/2024 80
E.18 25/01/2024 80
E.18.1 Iteration v3 – The Knob 81
Acronyms 83
Bibliography 87

Chapter 1

Introduction

Since the advent of novel, portable keyboard-based instruments (synthesizers, samplers, electroacoustic instruments, etc.), musicians have embraced these piano-like instruments and explored their new affordances. My motivation for this thesis lies in exploring new ways of performing music as an improvising keyboardist, looking more closely into the possibilities and challenges of utilizing expressive multi-touch control. With MIDI 2.0 well on its way but not yet implemented in instruments available to the public, the design and exploration of an expressive multi-touch instrument using MIDI Polyphonic Expression (MPE) seems relevant and with transfer value to future MIDI 2.0 instruments. MPE can, in a way, be seen as MIDI 1.5 in being a bridge between MIDI 1.0 and MIDI 2.0, as it enables greater individual control by dedicating one channel per voice when playing polyphonic.

As a keyboardist, one must train to have individual control of both hands, each with five fingers. With big keyboard rigs, one must also keep track of several different instruments and possibly play multiple at a time. The instruments could differ significantly according to their type (organ, electric piano, etc.) and have a vast repertoire of possible synthesis models and associated controllable parameters. It requires a certain *cognitive load* to keep track of all the instruments and settings, each having their own degrees of freedom (DoF), cumulatively adding to the total dimensionality needed to control. In [25, p. 191], Jensenius defines degrees of freedom as the number of independent movement variables within a mechanical system. He continues by considering the challenges of using a complex system since there may be a correlation between a higher cognitive load and more degrees of freedom, as discussed in [46] about *element interactivity*. All this is in addition to the complexity of handling the musical structure and form as well as communication with other musicians during a performance.

Today, with all the available digital hardware and software instruments, one can perform with only one master keyboard instrument capable of emulating (or controlling several emulations of) a range of classic keyboard instruments. This introduces the concept of *mapping* to make connections between the available controllers and the desired parameters of the sound engine. To make a meaningful connection, this calls for a good mapping strategy [24]. The introduction of the MIDI 1.0 protocol in 1983 was groundbreaking because it provided a means of easily connecting different musical instruments. Using a master keyboard shifts the cognitive load of having control of several instruments, each with its individual sounds, over to keeping track of all the different sounds in one (or a few) keyboard instrument(s). As the search for and programming of these sounds is often done in pre-production stages — by saving instrument patches to scroll through when performing — this could free up cognitive bandwidth so the musician can focus more on the music performance. Since the originals of the emulated keyboard instruments usually have some degree of controllability through their interface (drawbars, switches, and a knob on the Hammond B3 organ, several knobs and switches on a Minimoog Model D synthesizer, etc.), we get a new challenge in being able to control similar parameters with our modern master keyboard interface. Because of these instruments' vastly different interfaces and affordances, we can not possibly aim to control everything with a generic all-round master keyboard.

New technologies introduce novel *affordances*; the interaction possibilities offered to the user [25]. Some of today's digital instruments focus on expressive multi-touch control, such as the Haken Continuum, Linnstrument by Roger Linn Design, ROLI Seaboard, Osmose by Expressive E, and Andrew McPherson's TouchKeys. The design of the latter three instruments closely resembles traditional keyboard instruments and enables expressive multi-touch control per key over a familiar keyboard layout. The ROLI Seaboard has a continuous surface with elevations for the black keys, while the Osmose and TouchKeys look and feel more like a traditional keyboard. In this thesis, I make use of one of these keyboard-like multi-touch instruments, the TouchKeys, to explore mapping strategies for performing expressive performances using the new MPE specification of the MIDI protocol.

As different expressive multi-touch instruments could have various degrees of additional control change dimensionalities (sliders, knobs, etc.), I have limited my project to exploring mappings using only the multi-touch control of the keyboard. The multi-touch control surface is common to all the above-mentioned controllers, although they appear in different forms. To further limit this project's scope, I have used a polyphonic virtual-analog synthesizer as the starting point for my sound engine, thus not aiming to control every possible type of keyboard instrument. This synthesizer has been built in Pure Data (Pd), taking inspiration from the Sequential Prophet-6. This sound engine was the starting point for an iterative process of designing, developing, and exploring mapping strategies between controller and sound engine. Using an open-source software synthesizer for this work makes it possible to expand the sound engine with new functions and parameters to control along the process if desired.

1.1 Research Questions

The overarching goal of this project is to find generally applicable mapping strategies for using expressive multi-touch instruments for expressive control in performing with different keyboard instruments. These mapping strategies can then be applied in various use cases and built upon in the future, e.g., with the new MIDI 2.0 specification. In this research, I look into which parameters are more suitable for multi-touch control and, hence, allow for more expressive playing.

It was a conscious choice to use open-source hardware and software (TouchKeys and Pure Data) for this project. This makes it possible for others to recreate and continue this line of research. It positions the designer's creativity as the principal limit of the mapping strategy exploration instead of hitting limitations in trying out ideas using closed hardware and proprietary technology.

My main research question is: **How can we design and explore an expressive multi-touch instrument using MIDI Polyphonic Expression (MPE) and MIDI 2.0?** This can be broken down into three operational sub-research questions:

RQ1: Which parameters in a sound engine feel more suitable for being controlled by an expressive multi-touch instrument?

RQ2: How can we design an instrument using an expressive multi-touch controller to interact with a synthesizer using MIDI Polyphonic Expression (MPE) for expressive performances?

RQ3: How can MIDI 2.0 benefit a performer's expressive control of an instrument in music performances compared to using MIDI 1.0 with MPE?

The following chapters aim to answer these proposed questions.

1.2 Definitions

Before moving on, I will define some key terms in the thesis. Several of these definitions are inspired by Jensenius' action–sound theory [25]. By *instrument*, I mean a musical instrument in which there is an interface you can interact with to control a sound-producing mechanism that generates sound. The sound could be generated acoustically as with the strings of a guitar, electrically as with the voltage produced by the Voltage-Controlled Oscillator (VCO) of an analog synthesizer, or as a stream of binary data outputted by the engine in a digital software instrument.

The *control interface* and the *sound-engine* parts of an instrument can be two different devices that are connected through the *mapping* of their parameters. I see digital instruments' control interface, mapping strategy, and sound engine as equally important parts of the instrument design. Although I acknowledge that getting audible feedback when playing — actual sound — could be seen as an essential trait of an instrument, I also recognize that not every electronic and digital instrument nowadays has built-in speakers. I consider an external speaker system as a separate entity needed to fulfill the experience of playing on such an instrument that lacks a sound-producing part of its own.

With *multi-touch*, I mean a set of affordances of an instrument that enables individual control of notes for expressive performances. This has been an essential aspect of traditional acoustic instruments, such as the violin, where you can individually control the pitch and vibrato on multiple strings using various fingers. Concerning this thesis, multi-touch will be the sensors and controls of an instrument or its control interface.

In this thesis, I will use the term *synthesizer* to address a specific kind of instrument. Usually, I think of a synthesizer as an analog or digital instrument that generates its sounds through a sound engine, usually through one or several oscillators. (This will exclude an instrument using samples of pre-recorded sound for sound generation, which I instead would call a *sampler* instrument). In this thesis, I will refer to a (virtual) analog subtractive synthesizer instrument when using the term synthesizer. Subtractive synthesis is a type of synthesis where tones from oscillators, often rich in overtones, get sculpted by subtracting frequencies and amplitude by applying, e.g., low-pass filters for subtracting high frequencies or an envelope-controlled Voltage-Controlled Amplifier (VCA) to shape the amplitude over time. These synthesizers can be monophonic or polyphonic. By virtual analog, I mean such an instrument recreated digitally in a hardware or software synthesizer. Many synthesizers are designed as piano-like hardware instruments without speakers or as software instruments. Still, they can also be 'tabletop' modules without a keyboard or modular synthesizer units, as in a Eurorack system. A few synthesizers with internal speakers exist, as in the ARP 2600 from 1971.

An *expressive performance*, or to perform expressively, requires, in the way I define it, an instrument capable of capturing the performer's expressive gestures. This would require an instrument with more sensing capabilities than that of a traditional piano-like instrument, which in its simplest form is only capable of capturing key presses 'on' and 'off' on the different keys of the keyboard, as well as the speed (velocity) of which those keys was pressed and released. With an actual piano, and traditionally on piano-like keyboard instruments, there is no way to shape the individual tones after the key is pressed. To simplify this, the performer is limited to deciding how loud each note will be. Some piano and piano-like instruments might have a few or several sound-modulating capabilities, like a sustain pedal, a modulation wheel, or knobs mapped to a synthesizer's control change parameters. But these have traditionally been affecting *every* note played without the possibility to modulate individual notes. Where unique shaping of the individual tones is taken for granted in many other instrument groups, like guitars, this was lacking in most keyboard instruments until multi-touch keyboard instruments started to appear.

1.3 Scope and limitations

This master's thesis is limited to 30 ECTS study points, equivalent to one semester's work. There are also some limitations in that the MIDI 2.0 standard is just in the start phase of being rolled out, as will be explained further in Section 6.2.

The interdisciplinarity of this thesis combines my artistic background as a musician and performer with the technical-oriented process of iteratively designing an instrument through programming with Pure Data. This also involves designing and experimenting with mapping strategies for meaningful relations between the expressive multi-touch controller and the synthesizer engine through the use of MPE.

There was a lot of Jazz, Soul, and Blues music in our home when I grew up, which sparked my interest in improvised and groove-based music. As a performer, I have played in Soul, Jazz, and Pop-oriented bands, working as an all-round freelance musician and session musician in recording studios, exposed to various genres and challenges.

Over the years, I gradually expanded my instrument park, starting with a sample-based stage piano, just using the sounds it was shipped with. My first expansion was the electroacoustic Rhodes piano with amplifier and effect pedals, which afforded a lot of new sounds and expressive opportunities. The next addition to my setup was a Moog Little Phatty analog monophonic subtractive synthesizer. As much as it sparked a great deal of excitement and curiosity in learning about the technicalities of sound synthesis, this instrument stood on my shelf for several years before I took it on the road to perform with it.

A few years later, it became more and more often a demand to include synthesizers in my setup as a freelance musician, and I invested in several other synthesizers over the coming years: analog and digital, monophonic and polyphonic, hardware and software. In the meantime, I had also invested in a drawbar organ instrument and a rotary speaker. Hammond B3, a drawbar organ instrument intended initially to substitute pipe organs in houses of worship, has been a central instrument to much of the music I grew up with. This instrument consists of 96 tone wheels, each providing a sinusoidal tone before they get distorted and shaped through the instrument's circuits and the often accompanying rotary speaker. I often think of this as a kind of electroacoustic additive synthesizer [44, p. 134] because of its design, in which the nine drawbars are used to add different harmonics and sub-harmonics provided by the tonewheels.

By owning a drawbar organ and getting a deeper understanding of how this instrument is designed, I could utilize this knowledge expressively to shape new sounds instead of being limited to organ presets on a stage piano without its idiomatic modulation possibilities. When I grew accustomed to performing with synthesizers — a completely different field of keyboard instruments compared to a stage piano with preset sounds to scroll through — I appreciated all the new affordances in sound design and expressibility. This presented me with a whole new dimension of shaping my sound through low-pass filters and modulation of the sound as I performed live, which was unprecedented. I see this as a continuation of the affordances and fascination with which the Rhodes electric piano with effect pedals gave me some years earlier.

Today, the expressive multi-touch controllers have opened yet another door with more dimensions to explore in expressivity. The MCT master's program has been a fine fusion of my two big interests in music and technology.

1.4 Contributions

Below is a list of contributions provided by the work on this thesis.

- A set of sound engine parameters suitable for being controlled by an expressive multi-touch instrument. This collection of parameters could be a starting point when designing instruments, prototypes, or instrument patches.

- A Pure Data synthesizer prototype as an open-source project enabling collaborative continuation of the design.
- Insights into designing and utilizing expressive multi-touch instruments for expressive performances using MPE, including mapping strategies.
- A summary of the historical background of protocols used in digital music instruments preceding MIDI 2.0, including MIDI 1.0, ZIPI, OSC, and MPE, to see better the context on how MIDI 2.0 continues this line.
- A snapshot of the current state of the MIDI 2.0 specification, and a look into how this technology could affect musicians in the future.

1.5 Outline

This thesis begins with a literature review in Chapter 2, where I go through relevant concepts, theories, and definitions that will be used in the thesis. Chapter 3 introduces and reflects on the methods used in the project. Chapter 4 focuses on Research Question 1, concerning over-arching concepts of mapping parameters from an expressive multi-touch instrument to the parameters of a sound engine, explaining the process of researching mapping strategies. Chapter 5 describes and explains the process of designing an expressive multi-touch instrument, focusing on Research Question 2. The concluding Chapter 6 summarises the thesis and discusses the relatively new MIDI 2.0 specification concerning Research Question 3, with its benefits and potential impact. Finally, I wrap up the discussion and try to answer my research questions before proposing future work.

Appendices provide additional information on MIDI 2.0 (Appendix A), ZIPI (Appendix B), OSC (Appendix C), Synthesizers (Appendix D), and a diary documenting the design process (Appendix E).

Code and supplementary material can be found in the accompanying OSF repository[1].

[1] https://doi.org/10.17605/OSF.IO/JFMCV

Chapter 2

Theoretical Overview

In this chapter, I will lay out the background for my thesis by presenting a theoretical overview of the fields and technologies on which I build my work within this project. The topics covered are *expressive multi-touch instruments* in general before looking more closely into the *TouchKeys* specifically. Next comes a glance at *digital sound synthesis* and which techniques I have used in my design before introducing the programming language I have utilized for prototyping my synthesizer design, Pure Data. Some extra attention is then paid to the MIDI protocol with its subsets MIDI 1.0, MIDI Polyphonic Expression (MPE), and MIDI 2.0, as these topics are of great relevance to this thesis. This is followed by a few alternatives to MIDI, namely ZIPI and Open Sound Control (OSC), before a discussion about *mapping strategies* and a short concluding section to this chapter.

2.1 Expressive Multi-Touch Instruments

In the introduction, I mentioned the terms *expressive performance*, *multi-touch*, and *instrument*, which cover the components of **expressive multi-touch instruments**. This group of instruments could also be called *multi-dimensional* since they often consist of a separate multi-touch controller connected to an external instrument or synthesizer. Wanderley et al. [47] describe digital music instruments (DMI) as an instrument with a separate gestural interface or controller unit. A multi-touch instrument affords individual control of tones and often provides continuous-like control. I use the wording continuous-*like* since the sensor output of all digital interfaces eventually gets sampled down to a given amount of discrete steps according to the bit resolution used. However, the controller's physical surface and 'feel' (its sensory feedback) might aid in giving an impression of continuous control through multiple degrees of freedom (DoF). Typically, you can slide, press, and shape tones three-dimensionally over a multi-touch control interface.

If the controller is piano-like, the X-axis *could* typically resemble the pitch-axis on a piano, with the Y-axis being analogous to moving 'in' and 'out' of the length of a key. Note that the control surface won't necessarily have to resemble a piano-like instrument at all. The Z-axis could often be determined by the amount of pressure applied to, or the surface area of fingers touching, the controller. With its proceedings, the International Conference on New Interfaces for Musical Expression (NIME) has introduced a range of novel expressive multi-touch instruments and controllers through its years, of which a few are covered in [26], of which one being the TouchKeys that I soon will write about. In his book [25], Jensenius discusses several 'multi-dimensional controllers.' The controllers discussed all use MPE for communication with the external instrument. A more extensive survey on these multi-dimensional controllers can be found in [11], where Bye refers to them as 'MPE Controller.'

The controllers covered by Bye are:

- ROLI Seaboard Grand
- Madrona Labs Soundplane
- Roger Linn Linnstrument 128
- Haken Continuum
- TouchKeys

Of these, the ROLI Seabord Grand, Haken Continuum, and TouchKeys are more or less all piano-like. In contrast, the Madrona Labs Soundplane and Roger Linn Linnstrument 128 are guitar-like, with a control surface resembling a fretboard. All these multi-touch controllers can be mapped to a synthesizer or another sound engine in various ways, as we will soon see in 2.6, connected to sound-producing speakers for expressive performances. These components, with their connections, make up an expressive multi-touch instrument.

At the start of 2024, it seems that such instruments are rising in popularity as more and more instruments get released to the public. ROLI continues to expand its modular and portable BLOCK series of MPE controllers. I used the *Lightpad Buzz Bluck* from this series for prototyping. Expressive E has also released its Osmose standalone expressive synthesizer. These additions to the growing range of expressive multi-touch instruments all support MPE, and not MIDI 2.0, even though the specification has existed for a few years. It will be interesting to see if some or all of these instruments get upgraded with MIDI 2.0 support in the future, through a firmware update or otherwise, like the TouchKeys became upgraded to support MPE through its accompanying software some time after that specification was released. The following subsection will discuss the controller mainly used in my experiments, the TouchKeys.

2.1.1 TouchKeys

TouchKeys is a capacitive multi-touch sensing controller by Andrew McPherson, introduced in 2012 [34]. An unnamed early iteration of the interface was published in 2011 [37], which provides more insight into the technical aspect of the design[1]. An interesting difference with this controller, compared to the others covered in the previous section, is that this controller is designed to be an augmentation of keyboards. It consists of a set of multi-touch sensors, one for each key, to be applied to a keyboard. These augmented keys capture finger positions on their surface through capacitive touch sensors. The TouchKeys is designed to combine this sensor data with aftertouch Control Change, and MIDI note onsets and offsets from its host keyboard (if mounted to an electric keyboard compatible with MIDI).

Internally, the sensor data from TouchKeys get digitized into OSC packets, which you, through the TouchKeys mapping software, can map to a synthesizer or sound engine using OSC and MIDI, including MPE. TouchKeys can be employed on an acoustic piano or a keyboard without MIDI support by setting its state into 'standalone' mode. This makes the touch sensors trigger notes as well, rather than using the host keyboards MIDI note-on data, which reduces the degrees of freedom since there will be no distinction between gestures aimed at simply controlling or shaping an already existing note without it triggering a new note.

I used the TouchKeys in 'MPE' mode for my instrument. The current manual for the TouchKeys software [36], version 0.2 from September 2015, seems not to be up-to-date as there is no mention of the MPE mode. One consideration I had was that by using MPE with the MIDI protocol, it could be more accessible to expand my instrument design with a transition to MIDI 2.0 in the future.

[1]The reader can see a hint of the controller's upcoming name in the OSC addresses listed in Table 1, starting with '/touchkeys/...'.

TouchKeys was an attractive option because of its accompanying open-source software and customizability versus, e.g., the ROLI Seaboard Grand, a closed proprietary system. One could say that from a consumer perspective, the Seaboard Grand could be more relevant to many as it is shipped with internal sounds and can be used stand-alone by plugging audio cables straight into it. The TouchKeys is only a controller interface and is old by now. McPherson even announced the discontinuation of its production early in 2023 through a post in the TouchKeys Google Group [35]. However, I don't see this as a drawback. With all its degrees of freedom and mapping possibilities, I think using TouchKeys has a high transfer value. I will argue that closed proprietary or complete systems pre-mapped to internal sounds would be more problematic to exploit, so to speak, for custom mappings to a synthesizer when designing an expressive multi-touch instrument. The transfer value also holds for designing the synthesizer sound engine in Pure Data. A goal in this thesis has been that the prototype design presented in chapter 5 could be of value for future work on designing expressive multi-touch instruments with MIDI 2.0.

Concerning the technical capacities of the TouchKeys capacitive sensors, we learn in [34, 37] that the sensors Integrated Circuit (IC) of each key have a possible scan rate of about 250 Hz. However, running it at 125 Hz provided a more reliable result. This translates to a sensor reading approximately every 8 ms, which is acceptable according to Wessel and Wright's upper limit of 10 ms latency in a system [50]. This, nevertheless, has room for improvement in future designs since there are other latency-inducing elements in a digital instrument, in which the sensor reading scanning is among the first in this path of signals from sound-producing action to perceived sound. The data from each sensor are digitalized through a 10-bit ADC, which is higher than the 7-bit resolution of MPE in MIDI 1.0, although less than the 32-bit Control Change (CC) messages of MIDI 2.0. If OSC is used as the protocol for communication with the synthesizer, or 14-bit MIDI CC, one could leverage the full resolution provided by the sensors. The capacities of its sensors tell us that the TouchKeys would have a performance boost in bit-resolution if upgraded to support MIDI 2.0, compared to MPE. But it doesn't seem future-proof as a relevant multi-touch controller concerning fully exploiting the 2.0 specification without overhauling its hardware design.

In his commentary on his article for the "A Nime Reader" publication from 2017 [34, pp. 428-430], McPherson talks about the role of keyboard instruments in the NIME community. On the one hand, he thinks of it as a mental obstacle, almost like a hang-up of many designers of new interfaces for musical expression. However, one should not neglect the importance and power of familiarity. For many, the keyboard has a well-known form factor that could enable an instantaneous dive into actual musicking, even for non-keyboardists. A completely novel instrument design, which might or might not be intuitive in how to approach, could very well be exciting to try and play around with. But in the long run, to truly master and be a virtuoso on an instrument, one must put in a lot of dedication, time, and effort. McPherson thinks that the relative success of TouchKeys stems from the possibilities of leveraging one's existing keyboard technique when performing with it. He mentions the lack of an identifiable 'sound' since it is a controller without a sound engine.

It is an interesting discussion about how closely related one can be with a controller. I believe that musicians can achieve a deep sense of control intimacy with a controller and develop a close relation to it even without it having its own 'sound.' However, this would involve a high dependency on the mapping strategies and the sound engine, as the controller is a gestural interface. Is it then the repertoire of mapping strategies applied, the sound engine design, or maybe even the performers playing technique that gives the controller its perceived 'sound' and identity? I do not have a definite answer, but I would argue that its 'sound' is inseparably tied to the performer mapping it to a sound engine in a complete instrument design. But not without regard to the performer's playing style in shaping this 'sound.' This will give each TouchKeys its distinct 'sound' in the hands of dedicated performers willing to make it *their* instrument and the willingness to practice and master it. As McPherson says at the end of his afterwords: "...the most interesting research outcomes are the things we can learn about the players themselves and the way they engage with technology” [34, p. 430].

Dahlstedt, in his expert commentary from the same republication [34, pp. 430-432], has an exciting view on the TouchKeys. By having multi-touch sensors with high resolution on each key, the controller shifts a traditional focus on piano performances from being polyphonic with the use of up to all of the ten fingers to focus on fewer notes, maybe even just monophonic passages. Since the tonal control of a note on an acoustic piano by and far stops when the hammer strikes the string, the focus has been on structural complexity and the relationships between harmonies and notes, both temporal and dynamic, as opposed to instruments with continuous control where the tone can be shaped until the sound has faded out. With continuous and high-dimensional control, the instrument allows the performer to dive into individual notes and be more expressive on a micro-level, which is precisely what TouchKeys provides. Dahlstedt sees this as an inwards expansion of the keyboard paradigm while at the same time keeping its traditional interface, not hindering the use of conventional piano techniques. In the discussion about supreme control over monophonic sound versus reduced individual control in polyphonic instruments, Dahlstedt has an exciting remark touching on the issue of degrees of freedom and cognitive load: “There is a kind of inverse correlation between polyphony of sound and intimacy of control in musical instruments, distributing the cognitive load of the player over available variables.” This will be of concern regarding mapping strategies in the instrument design.

2.2 Digital Sound synthesis

In [44, Chapter 2], Roads traces back the first experiments on digital synthesis to 1957. This was performed by Max V. Mathews and his colleagues at Bell Telephone Laboratories in New Jersey. The history of synthesizers itself starts even earlier. Roads mentions the first concert where an audience could experience a massive sound synthesizer. In the autumn of 1906 in New York, the Telharmonium of Thaddeus Cahill poured its sinusoidal waves over around 900 listeners. This section will not go deep into details about digital audio or digital signal processing (DSP). Appendix D provides a survey of different digital synthesizer techniques.

My instrument design, which will be discussed more in-depth in Chapter 5, is based on *subtractive synthesis* (described in Section D.1). The synthesizer is *polyphonic*, capable of simultaneously producing more than one note, making it a ’virtual analog subtractive polyphonic synthesizer.’ The inspiration for my design was the subtractive synthesizer *Sequential Prophet-6*.

Modular synthesis is another relevant field of sound synthesis, which could be seen as more of a form factor than a technique. With this synthesizer concept, you have a system consisting of individual modules patched together through cables that speak to each other through audio and Control Voltage (CV) signals. Early modular synthesizer systems introduced modules like the Voltage-Controlled Oscillator (VCO), Voltage-Controlled Amplifier (VCA), Low-Frequency Oscillator (LFO), Low-Pass Filter (LPF), Low Pass Gate (LPG), and Attack, Decay, Sustain, and Release (ADSR) envelope generators that are fundamental building blocks relevant in digital sound synthesis.

The ’voltage controlled’ terminology is still used today, even in the digital domain. Don Buchla and Robert Moog made modular synthesizer systems in the 1960s before designing more compact and mobile synthesizers where the components, or modules, were pre-patched and integrated inside a closed casing. Modular synthesis has had a renaissance for the last couple of decades since Doepfer Musikelektronik introduced the Eurorack format in the 1990s. For more on modular synthesis, see [45].

To think *modularly* has been a great help to me when designing my instrument as it makes it easier to break the synthesizer sound engine down into modules and implement one component at a time. As we will see in the next section, the programming environment I chose to design and implement my synthesizer could resemble a modular system.

2.3 Pure Data (Pd)

As the tool for designing and implementing my synthesizer sound engine, I have used Pure Data (Pd)[2]. This is an open-source graphical programming environment for multimedia applications. It was introduced in 1996 by Miller Puckette [41] and further elaborated on in [40]. Pure Data can be seen as a continuation of the *Patcher*, which in [42] is introduced as "the most interesting window type in MAX." The MAX system itself was first included in the 4X real-time system developed at IRCAM [13]. The Patcher evolved into what we today know as Max/MSP (Max Signal Processing). While Max/MSP is commercial software, now owned by Cycling '74, Pd remains open-source. Both are examples of *dataflow* programming languages.

In the Graphical user interface (GUI) of Pd, you are presented with a main window with a terminal printout. When you start a new file, called a 'patch,' an empty white canvas is created where you can connect objects through virtual patch cables. A canvas can have any number of sub-canvases, also named sub-patches. By *abstracting* parts of the program into separate (sub-patch) files, you can make it more modular and reuse components easily. This could benefit clarity by making 'clean code' and thinking of the instrument design like a modular synthesizer.

There are two object types in Pd, 'control' and 'tilde,' in which the control objects compute their functions sporadically due to triggering events. The tilde objects perform audio computations and are recognized by the accompanying tilde sign '∼,' which always follows the first object-specifying atom. As an example, typing "osc∼ 440" into an object box creates a cosine wave oscillator object in which the creation argument '440' sets its frequency to 440 Hz when created. Objects and boxes in Pd equipped with inlets and outlets are connected through connecting lines of either the 'control' or 'signal' type. For control object arguments, timing in Pd appears in milliseconds.

What makes Pd suited to my purpose is its ability to handle operations in 'real-time,' unlike other text-based languages where you write the lines of code first and then run the code later. Roads [44, p. 111] argues that 'graphical editors' is one of six proposed categories of *musician's interfaces* for providing computers and synthesizers with synthesis data, where 'languages' is another. Although both Max and Pd have been influential by virtue as graphical programming environments, Puckette reveals in a podcast episode that his main concern when first developing Max was to make a computer music system being instrumental in running and reacting to audio in 'real-time' [43].

2.4 The MIDI Protocol

To better understand what Musical Instrument Digital Interface (MIDI) is, we will examine briefly its history, reception, and how it impacted the music industry. MIDI was first demonstrated to the public at the Winter National Association of Music Merchants (NAMM) Show in 1983[3], and its original specification was published by the International MIDI Association (IMA) in August of the same year [1]. This hardware and software specification encompasses a protocol for communication, a digital interface, electric connectors, and a file format and was intended for communication and synchronization between electronic music instruments and audio equipment.

The MIDI protocol was conceived after a few leading instrument manufacturer companies gathered for a universal communication solution between their devices. It was a growing wish and need among them to have a way of communicating between musical instruments. In the analog synthesizers of the previous decades, Control Voltage (CV) emerged as a way of transmitting control signals internally and between musical instruments. However, manufacturers had

[2] https://puredata.info/

[3] According to [30], the specification was revealed a few months earlier in an article by Robert Moog of Moog Music in the October 1982 edition of the *Keyboard* magazine.

different ways of implementing this, which made it hard to synchronize between devices. These differences resulted in fragmented proprietary solutions among the companies, making the instruments unable to communicate appropriately with each other.

In [22, pp. 5–7], a part of the process of standardizing this communication scheme is documented, mentioning work on creating the Universal Synthesizer Interface (USI) digital electronic instrument protocol by Dave Smith and Chet Wood of Sequential Circuits. USI was proposed to the Audio Engineering Society in the fall of 1981 and can be seen as a predecessor to what ended up as MIDI.

Several major musical instrument companies (Roland, Yamaha, Korg, Kawai, and Sequential Circuits) worked together to modify USI and renamed it Musical Instrument Digital Interface. An association between instrument companies involved in this new specification was officially formed in 1984 called the MIDI Manufacturers Association (MMA). They released the "MIDI 1.0 Detailed Specification" [3] the following year, which is still relevant today. The MIDI Manufacturers Association and the MIDI Committee of the Japanese Association of Musical Electronics Industry (AMEI) have worked together on developing and releasing the official MIDI standards since AMEI got formed in 1996.

2.4.1 MIDI 1.0

The official MIDI 1.0 specification has gone through several revisions and has been enriched by several addenda over its 40 years. All documents related to the specification are available from The MIDI Association (TMA), a non-profit organization launched by MMA in 2016 to support the global MIDI community[4].

As of this writing, there are four documents dealing with the *MIDI 1.0 Core Specification*, which include the aforementioned "MIDI 1.0 Detailed Specification" currently at version 4.2.1 as of a revision from February of 1996. This revision incorporated several additions to the core specification up until that time. The three other documents specify MIDI Time Code, MIDI Machine Control, and MIDI Show Control. There are also 20 addenda documents, which together form a collection of changes and additions made to the specification after the 1996 core revision.

MPE has its own specification document dealing with this significant expansion of MIDI 1.0. Lastly, of the MIDI 1.0 specification bundle, there are nine documents dealing with the General MIDI (GM) Specifications, consisting of GM 1, GM 2, and GM Lite. GM was introduced to make an even more cohesive experience when working with MIDI across devices. Notably, devices with GM share the same sound sets, meaning you can work on one device and play it back on another GM device and get the same auditory result. It is widely used in computer sound cards to play back .mid-files of the Standard MIDI File format.

Critique of MIDI 1.0

The MIDI 1.0 specification has been embraced and critiqued for its simplicity [29, 38]. This relative simplicity has enabled the specification to grow while hindering musicians who wanted to push the limits of expressive performances. Part of its popularity has to be credited to the fact that it is relatively simple and cheap to implement, which has helped the technology to gain a foothold in the music industry and has since been implemented into nearly every keyboard since 1983. This is in addition to its presence on various other musical gear, including sound cards, mixing consoles, and beyond.

Of the many critiques over the last 40 years, [29] was an early raised voice addressing several of its shortcomings. The main critique concerns limitations in bandwidth, time resolution, access to synthesizer parameters, and that it is not bidirectional. It also mentions how the protocol

[4]The documents are free to download if you register as an *individual member* at TMAs webpage https://midi.org/specifications

favors keyboard instruments and is *event-based* instead of *sample-based.* The latter means that MIDI sends information about human gestures through values of touched keys, sliders, knobs, etc., as opposed to audio samples. The data is transmitted serially with a data rate of 31.25 kilobaud (kBd) as ten-bit bytes, where the first (start bit) and last (stop bit) are stripped away by the receiver. This means that only 80% of the bandwidth is used for the actual data bytes. Regarding the lack of two-way communication, the author misses the opportunity for requests and responses with connected devices, which is one of many new features in MIDI 2.0.

The article provides a thorough review of the code specification. Of all the raised concerns, only a short sentence is offered to the low data resolution: "This restricts data to occupy the low-order 7 bits of each data byte." Next comes an interesting and bold claim that most MIDI computer systems, even MIDI itself, are meant for pop musicians. This is substantiated by that the systems used were mostly proprietary, with a silent cry by the author for more development environments aimed at computer music research. I think this is not a big concern today, with all available external MIDI libraries for programming languages and implementation in open-source and embedded systems.

Further in the article is a brief recognition of the positive sides of MIDI: to easily connect different devices of different manufacturers. It points out that most of the shortcomings are a result of people trying to do things with MIDI, which the protocol was not designed for.

Another influential critique is [38] from 1988, with the unadorned title "The Dysfunctions of MIDI." It is presented as a critique from a musical point of view, with a focus on performance capture, representation of musical control processes, and synthesizer control. The article starts with aesthetic motivations, which touch on the importance of expressiveness in music and the need for performance control of instruments in real-time to channel the performer's expressivity. It takes a moment to recognize that MIDI is great before proceeding with its dysfunctions. Through an analogy with the transmission of speech through four levels of synthesis quality, the author makes a point about how little available bandwidth MIDI provides for faithfully communicating expressiveness and affection. Next is a concern regarding how important control intimacy is when performing on an instrument, which relates to what Jensenius discusses regarding *spatiotemporality* [25].

The article mentions the delay introduced when transmitting multiple bytes of MIDI data serially, where one event typically takes around one millisecond. Depending on the circumstances, a little delay or latency up to 30 ms could be acceptable in some musical situations. However, in some musical situations, millisecond delays matter. There is also an uncertainty in how long delay there will be before a musicking gesture is transmitted to the synthesizer. The term *temporal smearing* is introduced for small delays of attack times, reducing the performer's control intimacy. Following is the problem with MIDIs event orientation, unsuitable for expressing continuous variations as it favors events such as keyboard presses. As the serial stream of continuous MIDI events gets shared between the other events, 'jitter' inevitably gets introduced. Of these matters, the author recognizes three dysfunctions of MIDI: *temporal smearing*, *temporal uncertainty*, and *sample jitter.*

Regarding capturing musical gestures, the article exemplifies how much bandwidth is needed for transferring certain musical gestures. The available MIDI bandwidth seems just enough in many conditions. Temporal smearing will happen on occasions like playing big chords with both hands on a keyboard, though, as the individual notes are transferred serially and not instantaneously. Synthesizers are among the instruments unsuited for the MIDI protocol because of limited bandwidth and since the events and data transferred must be converted into values appropriate to the synthesizer's parameters. This conversion and knowledge of the signal flow are big parts of designing digital instruments, as I have experienced during the work on my instrument. The author claims the MIDI transmission rate to be one to three orders of magnitude lower than what is needed for achieving perfect control intimacy in a real-time performance of an expressive synthesizer [38, p. 26].

To address the issues with what the author describes as the *sluggishness* of MIDI, three ways are proposed for dealing with this, each introducing different reductions of control intimacy. First is 'clipping,' where the transmission happens in real-time, and to maintain this, the information gets omitted if the transmission rate is exceeded. Secondly is 'triggering,' where complex events are pre-programmed and simply get triggered. I think of ADSR envelope generators as an example of this. The downside of pre-programming is inflexibility in expressivity, as it excludes complete and spontaneous control of the synthesizer. The third is 'smearing,' where the transmission of generated events gets *clogged up* until everything is transmitted. This results in the latter events not being realized in real-time.

2.4.2 MPE

MIDI Polyphonic Expression (MPE) was officially adopted by the MIDI Manufacturers Association (MMA) in January 2018 [4]. The work on the specification can be traced back to the Winter NAMM in 2015, when Roland Lamb, founder and CEO of ROLI, and other makers of 'multi-dimensional' controllers met up to discuss the idea which would end up as what we know as MPE[5].

Several software and hardware designers were responsible for turning the blueprint into an official specification, building on MIDI 1.0. MPE became a significant addition to MIDI and solved a lot of shortcomings MIDI had regarding expressive gestural control when performing. Previously, using the pitch-bend, modulation wheel, aftertouch, and other Control Change (CC) messages was channel-wide, which would affect all notes played. MPE enables individual control of notes, e.g., of controlling pitch and articulation per note. Each note is assigned to a designated channel, rotating through a block of 'per-note channels' to allow this 'per-note control.' A dedicated 'common' channel is used for messages that apply to all notes. This means that you could have a maximum of 15 notes of polyphony with MPE over MIDI 1.0, as the specification has a maximum of 16 channels.

The MPE specification allowed for much more refined control with compatible instruments for more expressive performances and has by some been coined *MIDI 1.5*. The way MPE deals with individual control of notes by assigning them to separate channels resembles the fourth *Receiver's Mode* in MIDI 1.0 (Omni Off/Mono) [3, p. 22], in which an instrument set to this mode will respond to only one monophonic voice or note per channel. This mode was used in MIDI guitar systems with one voice per string transmitted over six channels [22, p. 20]. The MPE specification [6] is currently in version 1.1, released on April 14th, 2022. In MIDI 2.0, the concepts of MPE are expanded even further. 'MPE' is one of the available profiles in the new 'Profile Configuration,' which is one of three areas of MIDI 2.0s new MIDI Capability Inquiry (MIDI-CI) protocol.

2.4.3 MIDI 2.0

Already in 1986, there was a prediction about MIDI 2.0 and its presumed backward compatibility in the December edition of the *Music Educators Journal* [18] which ended up being astonishingly correct:

> "There have been some complaints with current MIDI specifications, especially concerning the speed of data flow, and it is probable that an upgrading of the specs is in the works. In theory, any revisions would be a 'superset' of the current MIDI standard—that is, MIDI 2.0 instruments would continue to function with MIDI 1.0 instruments, but the old instruments would not have some of the features of the new ones."

[5]More on MPE, including lists of compatible software and hardware, can be found on the ROLI webpage: https://roli.com/mpe (accessed November 15th, 2023)

The information provided in this section has been collected and summarized from a few different sources, including documents from the official MIDI 2.0 specification [5, 7], a book chapter about “The MIDI 2.0 Spec” from [22, pp. 45-57], and a YouTube video from The MIDI Association [2]. There is not much literature on this new protocol yet. Other relevant work includes a master thesis [19] which approaches MIDI 2.0 from an engineer and computer scientist’s perspective.

When MIDI 2.0 was announced at the Winter NAMM Show in 2020, maintaining backward compatibility was an essential feature of the new specification. This includes full support of both protocols, as MIDI 2.0 seeks to enhance upon MIDI 1.0 features as much as possible. Bidirectional communication is another important new feature, meaning that MIDI 2.0 instruments, for example, now can know if a message got correctly received and the capabilities of the other device(s). The MIDI Association has named these foundational new features “The Three Bs”:

1. Bidirectional communication
2. Backwards compatibility
3. Both protocols

While MIDI 1.0 only offered a 7-bit range of values (128 discrete steps), MIDI 2.0 provides a much higher range with 16-bit resolution (65,536 discrete steps) on velocity values and a 32-bit resolution (4,294,967,296 discrete steps) on Control Change (CC) values. Note that there is a way of combining two CC messages into one 14-bit (16,384 discrete steps) message stream within the MIDI 1.0 specification by combining a ’coarse’ CC message with its corresponding ’fine’ CC message.

The ’MIDI 1.0 Pitch Bend’ message is also in 14-bit resolution, consisting of a status byte followed by two data bytes. The new way of sending MIDI messages is through the format named Universal MIDI Packet (UMP). This format is transport medium agnostic (as long as the technical requirements are met). It can be sent through Ethernet, Thunderbolt, Wi-Fi, and whatever medium the future brings.

As a part of future-proofing the new specification, plenty of room has been reserved for upcoming message types and controller types not yet envisioned. Other improvements introduced in MIDI 2.0 are tighter timing through ’Jitter Reduction (JR) Timestamps’ and the 16 channels expanding into 256 by providing 16 groups of 16 channels. These 16 groups operate separately from each other, each with their own sets of system messages. Some groups can use the MIDI 1.0 protocol, while others utilize the MIDI 2.0 protocol, as the protocol is chosen independently per group.

With MIDI 2.0, there is also built-in support for **per-note events**, which enhances upon the features introduced with MPE, meaning better compatibility with expressive multi-touch instruments. The *MIDI 2.0 Core Specification* consists of six documents[6]:

1. MIDI 2.0 Specification Overview
2. MIDI Capability Inquiry (MIDI-CI)
3. Common Rules for MIDI-CI Profiles
4. Common Rules for MIDI-CI Property Exchange
5. Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol

[6]The collection of MIDI 2.0 Core Specifications are available at The MIDI Association webpage: https://www.midi.org/specifications/midi-2-0-specifications/midi2-core

6. MIDI Clip File Specification (SMF format for UMP)

The fifth entry in this collection [7] specifies the UMP Format and the MIDI 2.0 Protocol, which is an extension of the MIDI 1.0 Protocol. The latest version of this document is currently 1.1.2, published November 10th, 2023. For a more in-depth look into the MIDI 2.0 protocol, see Appendix A.

2.5 Alternatives to MIDI

After MIDI entered the scene, there have been attempts to make better protocols for musical purposes that could afford greater expressivity. These technologies seem to have tried to continue the road MIDI paved in communicating digitally between music devices while improving upon its various shortcomings. Some of these technologies, if not all, can be seen as results of the critique about MIDI, as its different deficiencies (discussed in Section 2.4.1) have been addressed and attempted to be fixed. However, some of these alternatives are already relics of the past, not managing to get a foothold in the industry. At the same time, MIDI has continued to grow and thrive as the leading industry standard despite its flaws and shortages.

Of the protocols forgotten by the vast majority, we have ZIPI, which will be mentioned below as an exciting part of the critique and discourse about MIDI. A relevant alternative to MIDI, which became widespread and is extensively used today, is Open Sound Control (OSC).

2.5.1 ZIPI

One notable attempt to improve the shortcomings of MIDI is Zeta Instrument Processor Interface (ZIPI) with its Music Parameter Description Language (MPDL). This work was initiated by researchers at UC Berkeley's Center for New Music and Audio Technologies (CNMAT) and Zeta Instruments. Its documentation from the 12th of January, 1994 can be found in [57], a collection of four documents: "ZIPI Music Parameter Description Language," "ZIPI Network Summary," "ZIPI Examples," and "MIDI/ZIPI Comparison." These are all tagged 'version 1.0' and seem to be early versions of what became published as a set of articles under a section named "The ZIPI Music Interface Language" in the *Computer Music Journal* volume 18, number 4, published winter of 1994. In addition to the articles [32, 33, 51, 53] building on the four documents from the manual, we also find "ZIPI: Origins and Motivations" [31], and "Answers to Frequently Asked Questions about ZIPI" [52]. I will look briefly into ZIPIs origins before comparing it with MIDI.

In speaking about the origins and motivations for ZIPI, [31] mention the focus on keyboard-like controllers and instruments. Keyboards were already fitting interfaces for controlling analog synthesizers and became even more suited for interacting with the emergence of sampler and synthesizer instruments that supported MIDI. This is because of the implications and constraints of the MIDI protocol itself with embedding information about *pitch* (encoded as a semitone note number), *loudness* (velocity), and *timing* in its 'note-on' messages. This format favors percussive devices and keyboard interfaces, making it less appealing to instrumentalists from other instrument groups, such as wind, string, and brass players. A notable exception is the electronic wind instrument (EWI) by Nyle Steiner, a MIDI controller in a shape reminiscent of a saxophone. The article addresses issues of MIDI in combination with instruments where the pitch, loudness, and timing are decoupled from their inherent continuous nature, such as in a violin.

The MIDI protocol is not suited to carrying expressive gestural information from instruments with continuous control, which was one of the motivations for starting the work on ZIPI. Already in 1989, Zeta Music and CNMAT began collaborating on the first concepts of the upcoming musical data language of ZIPI. A few years later, ZIPI was presented to the public at the Winter NAMM Show in January of 1993, 10 years after MIDI got announced at the same place.

How does ZIPI compare to MIDI, and in which way does it improve upon performing expressive gestural control of synthesizers? A list of nine areas where ZIPI allegedly is superior over MIDI provided in [51] makes me wonder why it didn't catch on with musicians and the music industry. A review of these nine areas and more on ZIPI is found in Appendix B. I can only guess that the work of a relatively unknown instrument manufacturer, Zeta Instruments, in co-operation with UC Berkeley's Center for New Music and Audio Technologies, although being experts and visionaries in the field, could not compete with the instrument manufacturer behemoths of the music industry united in the MIDI Manufacturers Association (MMA).

2.5.2 Open Sound Control (OSC)

Open Sound Control (OSC) is a protocol for communicating between musical instruments, computers, and multimedia devices in general introduced in 1997 [55], authored by Matthew Wright and Adrian Freed. Together, they developed the protocol at UC Berkeley's CNMAT, from where ZIPI originated a few years earlier and in which Matthew Wright was also heavily involved. This implies a connection and sort of continuation between the two technologies.

The official specification of OSC v1.0 [54] was released in 2002. There was an official OSC v1.1 specification in the making, which didn't get published, and the NIME article [14] stands as a testament to the version 1.1 upgrade. OSC has been extensively used in academic environments by researchers and the NIME community as it is suitable for designing and prototyping instruments and controllers and also by music technology hobbyists for making DIY instruments. It has been included in the programming languages Pure Data and Max/MSP, tools used by many in this community. OSC is being supported by prominent actors such as Apple, with their integration of OSC in their iOS devices and macOS.

Some strengths of OSC are its flexibility and ease of use, as you f.ex. can send accelerometer and gyroscope data carried as OSC messages from your smartphone to your computer, instrument, or musical device. There are OSC libraries for programming languages such as Python, and OSC messages can carry MIDI data. This could answer many of the protocols' big successes. TouchKeys is using OSC internally, as the serial data from the controllers on each key get unpacked into OSC messages. This OSC data can then be mapped dynamically to MIDI messages through the TouchKeys GUI [34].

There was an article in the Proceedings of the International Conference on New Interfaces for Musical Expression in 2003 named "OpenSound Control: State of the Art 2003" [56] taking the temperature of the then six years old protocol. This article was presented as a user-friendly overview of OSC to compliment the more technical official specification. Max/MSP gets mentioned as the first programming environment that implemented OSC through the use of Max 'externals.' The way to incorporate OSC in Pure Data patches at the time was by using third-party objects. Now we have the 'oscformat' and 'oscparse' objects included in Pure Data.

As we can see through these articles and in the specification, OSC has an entirely different approach to being a protocol designed for communicating between musical devices. Although both formats have gotten a foothold in the music industry and music community, MIDI has been implemented far more widely by commercial instrument manufacturers in their devices and instruments. Both have surpassed the boundaries of only being used in music and sound applications. The simplicity and flexibility of OSC still make it as relevant and useful as ever.

In the [56, pp. 142-143] version of the article published in [26], there are some interesting reflections in the expert commentary by Dannenberg. He speaks about how OSC simplified standards that existed at the time, but also how it is *too* simple. Because of, or albeit of, its simplicity, it has undeniably been popular. Dannenberg continues with a postulation that the inclusion of OSC 'objects' into both Max/MSP and Pd could very well be the reason why it caught on to the music community. This eases inter-process communication, as you could set up a connection between, e.g., Pd and a Python program using a OSC library running machine learning (ML) based image recognition of hand gestures from the computer camera with the use of OSC. That TouchKeys uses OSC internally for sending its sensor data to the computer software is an excellent example of the protocol's interoperability.

2.6 Mapping Strategies

Mapping is a vast field, too big to cover fully in this thesis. This section will present and review some seminal and relevant literature on the subject. [24] starts by stating that the human control device is inseparable from the sound-generating device in acoustic instruments. With electric instruments, there is a separation between the input device and the sound synthesis device, and we need to make the connections between them by mapping the input device's parameters to the synthesis device's relevant parameters. In [25], Jensenius provides a chapter about "Spatiotemporality," which discusses this separation between the performer controlling the input device and the output of the instrument, both concerning the physical distance and the *latency*, distance in time. In [25], there is also a chapter about "Action-Sound Mappings," dealing with the issue and importance of mapping and reviewing relevant literature on the field.

Continuing with Hunt et al. [24], there is an observation of a tendency in instrument design that a single parameter from the input device is mapped to a single parameter on the sound synthesis device. This basic mapping strategy is called *one-to-one* mapping. Other basic mapping strategies are *one-to-many*, where one parameter on the input device controls several parameters on the sound synthesis device, and *many-to-one*, in which several dimensionalities of the input controller are combined to control one parameter of the sound synthesis device. Combinations of these three strategies result in a *many-to-many* strategy. The relationships between parameters could also be non-linear in their complex relationships. The authors have observed that it is favorable to employ complex mapping strategies to engage the users and maximize their performance. This was increasingly true for complicated use cases, while the most straightforward test favored basic one-to-one mappings.

Details about their experiment are found in [23]. These findings suggest to me that if your goal is to control the amplitude of an instrument or similar trivial tasks, a single slider or knob is preferred, while when designing a more complex expressive multi-touch instrument, it requires a more complex mapping strategy. Hunt et al. also found that the scores improved over time when using complex mapping strategies, which resonates with me, namely that performers need to practice and invest time in their instrument to master it. This also reminds me of the quote about having a "low entry fee with no ceiling on virtuosity" concerning instrument design [50]. Further, [24] define mapping as "the liaison or correspondence between *control parameters* (derived from performer actions) and *sound synthesis parameters*," a definition I can abide by. 'Performer actions' could also be referred to as 'performer gestures' and 'musicking gestures' in other literature. Hunt et al. continue by presenting two main *points of view* regarding mapping:

1. Mapping is a specific feature of a composition

2. Mapping is an integral part of the instrument

The authors conform to the second point of view. So do I, but I also recognize the presence of the first point of view, which I have personally observed arising in inspired moments of instrument design. I will discuss this later in Section 4.3. Two main directions in *types of mapping* are also presented:

1. The use of generative mechanisms to perform mapping

2. The use of explicit mapping strategies

In my instrument design, it is the direction of explicitly defining the mapping relationships which is relevant. The three basic mapping strategies and the many-to-many combination are examples of explicit mapping strategies. In their review of mapping literature, there is a passage about how the expressivity of an instrument is highly dependent on the mapping strategies used. This was an important consideration for my instrument design, as its expressivity could vary considerably when employing different mapping strategies. I found it hard sometimes, though, to make definitive boundaries between the synthesizer design and mapping stages as the iterative process of designing the instrument became more of a fluent circular process when working with the Pd patch containing both the sound engine and the mapping between it and the output of the TouchKeys or Lightpad software (which in itself also is used for mapping). When a synthesizer design iteration was ready, and I mapped its parameters to the controller, this process could spark new ideas for parameters to implement and control in the subsequent design iteration. And vice versa, adding features to the synthesizer could give clear ideas on how to map these to the controller. All of this was an explorative process with sometimes explicit assumptions in advance on how to make the connections in combination with systematic testing of mapping strategies.

At the end of [24], the authors contribute to what they coin "a general model of mapping for expert interaction." This model is based on separating the mapping layer into two independent layers, with an added abstract parameter layer in between. This mapping architecture applies one mapping layer between the input device and the abstract parameter layer. The other layer concerns the mapping between the intermediate abstract parameters and the synthesizer sound engine. This concept was first presented in [48], with the work on the ESCHER system. Using an abstract layer also affords the conceptualizing of high-level parameters with more complex mapping than direct connections between sensors and synthesizer parameters. As an example, one could have *brightness* as an abstract parameter in between the Y-axis of the keys' multi-touch sensors output, increasing the cutoff frequency of a lowpass filter while also crossfading the oscillator to a more harmonic-rich waveform. Another abstract parameter named *space* could be affecting the release stage of the amplitude envelope and increasing the mix of a reverb effect towards being more *wet*, activated gradually if notes are held longer than an arbitrary threshold. This abstract high-level set of intermediary parameters could resemble the macro controls found in some synthesizers, like *Massive* from Native Instruments. The concept also reminds me of having an application programming interface (API) between a controller device and a synthesizer sound engine, enabling modularity in changing controller and synthesizer devices, which is a way of designing an expressive multi-touch instrument appealing to me. With this structure, I could continue the instrument design in the future using a MIDI 2.0 supported multi-touch controller.

More on gestural control of sound synthesis can be found in [47]. This article defines the aforementioned digital music instruments (DMI) with separate input devices and sound engines. It also places the interaction between gestural input devices controlling computer-generated synthesizers as a specialized branch of Human-Computer Interaction (HCI). There is an exciting field of research regarding HCI in connection with music, although it is out of scope for this thesis to pursue it further. More on the subject can be found in [20, 21]. One aspect of designing digital music instruments, which is also essential to consider, is the feedback of the instrument to the performer. [47] introduce two ways to approach feedback from an instrument: one focuses on the possible functions of a gesture, and the other concentrates on its physical properties. With the *functional* approach, which I am concerned with in my case, the feedback is either *primary* or *secondary*. Secondary feedback is the actual sound output produced by an instrument, whereas primary feedback concerns the visual, auditory, and tactile-kinesthetic (*tactual*) feedback of the interaction with the instrument. This could be the clicking of keys from physically pressing them down or LED lights indicating the value of a knob. In my case, the primary feedback of the TouchKeys is in many ways similar to other piano-like interfaces or even acoustic pianos, except that there is a slight tactual difference in the keys because of their mounted sensors and visual difference because of the LED lights and screen provided by the host keyboard. There are also spatiotemporal differences in the secondary feedback of my instrument design, especially when comparing it to an acoustic piano. Other multi-touch controllers provide different primary feedback, especially concerning the tactual 'feel,' discussed in the survey of other controllers found in [11, 25].

Baalman has another exciting view on mapping as she thinks of the whole unseen process, from physical gesture to an audible output, as the mapping of an instrument [9]. This is a more all-encompassing view on the subject inspired by an essay from deLahunta [12]. The author provides an interesting discussion regarding mapping and reflections around an instrument's design process in this article, which also interests me regarding my iterative design methodology discussed in the upcoming Section 3.2 about iterative design and prototyping. As the title of [9] reveals, "Interplay Between Composition, Instrument Design and Performance," a central point in the article revolves around the blurring of boundaries between these three subjects. Since code and mappings could be changed during performance in many modern instruments, this breaks up traditional boundaries between instrument design and performance. The choices made while designing the instrument could also blur the view of whether it is to be looked upon as a composition. This boundary could, e.g., be clouded by a pre-coded sequence of notes to be triggered by the performer. This makes designing the instrument, practicing on it, performing with it, and composing for or through it a fluid endeavor that could span and evolve over a long time frame, even years.

Regarding Baalman's view on mapping, she discerns five steps between the performer's gestures and the output media providing feedback: *gesture*, *sensors*, *electronic circuits*, *computational model*, and *output parameters*. Here, the output media provides feedback to the performer's perception, guiding their decision-making in adjusting their gestures acting upon the sensors or even the instrument's code during rehearsal or performance. Figure 2.1 illustrates the relationship between these steps. This could be seen as an iterative process related to the iterative instrument design method, especially during practice sessions on the instrument.

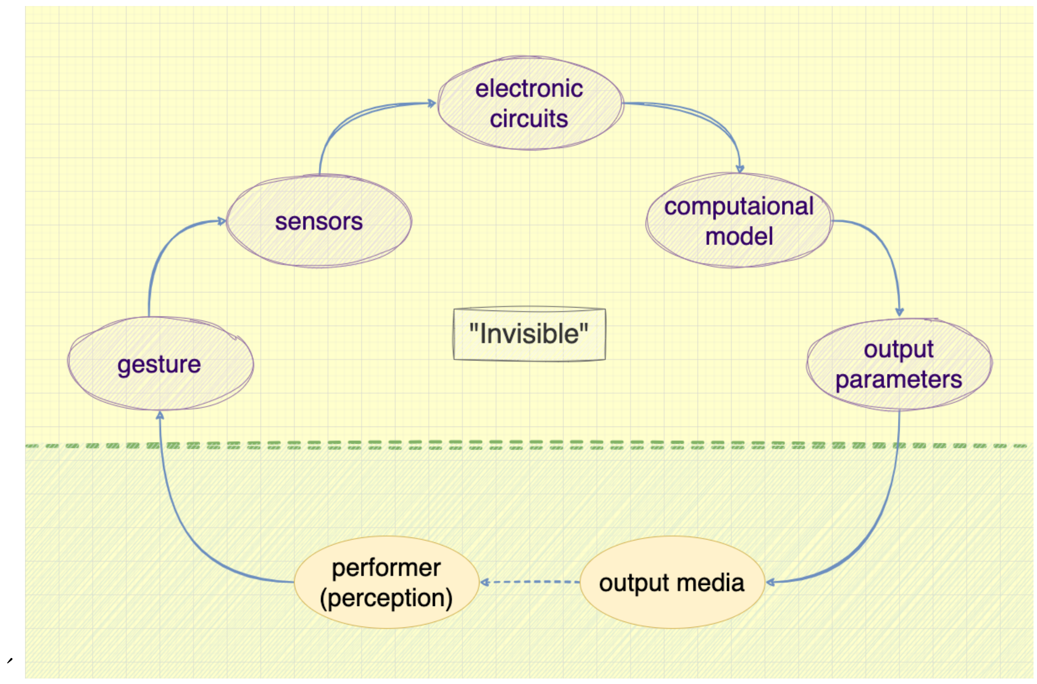


Figure 2.1: An illustration of the steps in a mapping process from Baalman [9], inspired by deLahunta [12], where mapping is the "invisible" part of an instrument.

Nilsson [39] is also discussed in [9], about his division of *design time* and *play time* when designing digital music instruments. These terms are used in the iterative design process, as the designer often alternates between these states during the instrument design. With this approach, experiences and reflections during playtime will guide the performing designer in changing and evolving the instrument further during design time. Multiple iterations of this process are applied until a satisfactory result or a deadline is reached. This approach requires the designer to be a performer, and Baalman's article is founded on her artistic practice with her piece "Wezen—Gewording." I will adopt the terms 'design time' and 'play time' when discussing my instrument design later in this text.

A colloquial paper by the same author [10] also addresses the subject of mapping as she seeks to find a method to study the mapping question. Her focus on artistic practice is evident here, too, as the proposed method involves interviews with the *artists* who designed the instruments in combination with a profound analysis of the builds of the instrument design. I took note of this method during my design process, using self-reflection instead of interviewing myself. Baalman also draws a line to HCI, arguing that an artist's definition of a good interface may stray far from what is generally considered a good interface in this discipline.

Baalman [10] also touches upon how art projects and instruments differ, but the tools used are often similar. Standard methods and techniques exist for processing, translating, and mapping gesture data. The author discusses how these standards can be transferred to newer technologies and use cases, as the methods and techniques applied could remain the same. A vocabulary for this knowledge is needed for artists to discuss and transfer their knowledge to each other across different software and devices. I think there is a lot of tacit knowledge among artists regarding their technological skills, which a common vocabulary, as proposed by Baalman, could help put words on.

The paper also stresses the need for a deep understanding of the instrument design's physical build and signal path, including if and how the signals are translated, aggregated, routed, compressed, and processed from sensors to the output medium. The author concludes with a remark on the interdisciplinary aspect of the art of mapping, recognizing the designer's requirement to possess artistic *and* technical skills. [10] is part of the author's work on [8], which is a book providing deeper insights into the subject of mapping and designing interactive technologies from an artist and engineer point of view.

2.7 Summary

This chapter has provided a theoretical overview and literature review of relevant topics, including expressive multi-touch instruments, digital sound synthesis, Pure Data, the MIDI protocols, critiques and alternatives to MIDI, and mapping strategies. Extra reading on some of the subjects, offering more technical insights, are provided in appendices A, B, C, and D.

Chapter 3

Methodology

When working on this thesis, I have used different methods to design an expressive multi-touch instrument and answer my research questions. The methods used include an extensive literature review of relevant synthesis methods and communication platforms and an iterative prototyping process, including designing, developing, and testing a multi-touch instrument. This chapter provides a methodological reflection of my interdisciplinary exploratory process.

3.1 Literature Review

Doing literature reviews has been an essential part of this thesis, in the form of reading, reflecting upon, and reviewing book chapters, articles, protocol specifications, and YouTube videos. It has taken a considerable amount of time to collect the various threads to get a grasp and an overview of the still relatively new MIDI 2.0 Protocol, as there is not much literature on the subject yet. This included careful reading of the protocol and watching YouTube videos of announcements and talks about MIDI 2.0, as the The MIDI Association has been releasing content regularly on their YouTube channel[1] since the updated version got announced in 2020. I have also sourced articles and book chapters found relevant to this thesis and the process of designing an expressive multi-touch instrument, as well as to present the history of MIDI, including some of the critique it has received and endured, and alternative protocols for comparison. Multiple disciplines and fields of knowledge are involved, each with a vast corpus of literature. Since covering all literature of interest is out of scope, I have provided a selection I found the most relevant.

3.2 Iterative Design and Prototyping

My approach to developing a multi-touch instrument is an *iterative design and prototyping* method. It combines two research methods and processes: ’prototyping’ and ’iterative design.’ This combination has given me the benefits of both approaches by applying iterative design principles to prototyping the instrument. It has enabled me to evaluate and provide myself with feedback to design prototypes quickly between iterations.

Principles from [17] have been used when applying an iterative design methodology in the design process of my instrument. In this article from 1985, Gould and Lewis present three recommended principles of design:

1. Early Focus on Users and Tasks
2. Empirical Measurement
3. Iterative Design

[1] https://www.youtube.com/@TheMIDIAssociation

The first two principles revolve around including users in the plural, while I only have myself as the sole user in the design process presented in this thesis. *Early focus on users and tasks* is about understanding through studying the intended users and the work expected to be accomplished. To make a quick thought experiment with this in mind: my instrument design is intended for musicians, who presumably are keyboard players that want to express themselves on a synthesizer instrument. They have felt the limitations of MIDI 1.0 and traditional synthesizers and desire to explore and perform an instrument with more expressive affordances. The work to be accomplished is to practice on and perform with this instrument, hopefully over time, to arrive at a certain level that could bring the performer and the instrument to the stage with an audience or end up being recorded and released to the public.

The second principle, *empirical measurement*, includes the users early in the process, where the prototypes are used for real work, and both the performance and user reactions are analyzed. This has partly been done in my research, with only me as the user, and without the structured observation and analysis process proposed in [17]. The principle I have adopted from the article, *iterative design*, is about resolving the problems found through user testing. To fix these problems, the authors claim that the design must be iterative. There is a need for a repeating cycle of design, measuring of tests, and redesign as much as is necessary. In my case, the timeframe has been a limiting factor in the amount of iterative design cycles. My design time phases have resulted in prototypes, which have been tested and measured during the following play time phases. These paved the way for improving the instrument when returning to the drawing board for redesigning.

By prototyping through an iterative design process, I have been able to make quick mock-ups of an instrument. These prototypes still had all the elements of an instrument to be playable, including a sound engine and mappings between the multi-touch controller used to play the instrument. The controller mainly utilized, the TouchKeys, is a final product in itself. The synthesizer is connected to external speakers to enable sound from the instrument. In this regard, I recognize the instrument prototype to encompass the sound engine, the synthesizer designed in Pd, including its mapping to the multi-touch controller and the speakers providing sound.

3.3 Programming

To design the synthesizer sound engine used in my expressive multi-touch instrument, I have been programming in Pure Data. This visual programming language suits music applications, which I have already covered in 2.3. My programming background has been limited to some knowledge of Visual Basic in my teens, a course in Max/MSP, and exposure to Python and Pd during my master's studies.

Since Pd is structured so that objects get connected through virtual patch cables, this process is also referred to as *patching*, and a Pd file (.pd) can be called a Pd patch. Programming and patching sometimes get used interchangeably, especially when working with keyboard instruments. Patching is employed as a term for the act of connecting different modules of a modular synthesizer through audio cables.

To program a synthesizer is another way of saying that you are setting it up for a specific purpose, essentially doing sound design, which can be stored as patches on digital instruments (or digitally controlled analog instruments). A patch, or patches, is, in this case, used as a term for the preset sounds (presets) found on many keyboard instruments and synthesizers. These often consist of a mix of factory presets and user presets. In this work, I do both by using Pd and the TouchKeys. I am programming the synthesizer in Pd, but the different iterations of my design could also be seen as individual instrument patches. To do the programming, I am patching together programming language objects. Also, in the TouchKeys GUI, I am saving presets of the mappings I have programmed for controlling my synthesizer.

3.4 Performance Research

Although not at the center of the project, I have also applied some methods from performance research in this project. In and after the play time [39] phases of my iterative design process, I have been reflecting on the performances regarding how the instrument has responded to my gestures and the experience of performing with it. The play time consisted of improvised performances and systematic testing of features. I have taken advantage of my background as a musician who is used to improvising solos on synthesizers during performances to evaluate the instrument design and how well its mapping of the multi-touch controller affords an expressive performance. This was done through introspection and self-reflection and by listening to and evaluating recordings of the play time improvisations.

3.4.1 Introspection and Self-Reflection

Introspection and self-reflection are closely related and sometimes used interchangeably to describe the same procedures. Using introspection to look more closely into how I have responded to the instrument during play time has provided valuable insights into how the instrument has felt to perform. Although these are only my subjective thoughts and feelings, they have given me a better understanding of what is working and what is lacking in the current iteration of the instrument design. Along with self-reflection, this has provided me with input on bringing the instrument design further in the following design time phase for its next iteration. Self-reflection is the act of self-observation of your thoughts, desires, and feelings, in which written forms of self-reflection can be performed by making use of a learning diary, learning protocol, or portfolio, as stated in [16].

I have kept a research diary throughout my design process and commented in the Pd code explaining what is happening. The comments in the Pd code have also served as valuable documentation of the patch for me to remember what, why, and how I have implemented certain parts of the instrument. However, it is not pretending to be a complete instrument documentation since I have been prototyping and have not finalized the design. With some implied knowledge of sound synthesis, the comments should, by and large, be able to explain the patches to other users interested in looking into my instrument design. Changes in the instrument design have regularly been uploaded to a GitHub repository[2].

The self-reflection method has been significant during the iterative design process to document and structure my reflections and keep track of how the prototype performed. This became a transparent way for me to note what did and did not work and a place to write down emerging ideas to implement in the following prototype incarnation. By evaluating and reflecting on audio recordings of the play time improvisations, I have also pinpointed flaws in the design to be improved in the upcoming (re)design cycle.

3.5 Practice Research

As this thesis, to a substantial degree, resolves around *making* (an expressive multi-touch instrument) and *doing* (explorations and performances with it), it is very much founded on practice-based research. I have had to use my creative side in the design process of the instrument during design time and when improvising with it during play time. This has been an exploratory journey. This work could also be seen as practice-led research by providing knowledge on how this can be applied in other use cases. Specifically, I hope my process of designing the instrument prototypes can be continued and expanded with MIDI 2.0-supported multi-touch controllers and programming languages in the future. I have used a *practice research* methodology by including practice-based and practice-led research.

[2]https://github.com/wnetzel/all_in_the_multitouch

3.6 Summary

In this chapter, I have reflected upon the methods applied in this thesis, which include literature review, iterative design, prototyping, programming, performance research, introspection, self-reflection, and practice research. Other methods could have been used, and different routes could have been taken, but the scope of this thesis has provided limitations.

I quickly realized that developing a performance-ready design was out of the project's scope. I considered setting up a formal user study but acknowledged that the prototyping would take too long to have something ready for other users to try. Instead, I have relied on my experience as a performer by using my subjective feedback through introspection and self-reflection. The advantage of this has been the possibility of rapidly making changes and trying out new ideas during design time. It is safe to presume that input from other users with different backgrounds, needs, and other aesthetic choices in sound design and playing styles would have contributed to a more robust instrument prototype.

There has not been any formal evaluation of the instrument design. The practice research methodology I have used is related to *applied research* in the parts of my work when I am solving the practical problem of designing and exploring an expressive multi-touch instrument. But there have also been parts of this research, mainly about MIDI 2.0, which has also been about gaining knowledge. Taking all into consideration, I hope that the methods used have led to more insight into how to design and explore an expressive multi-touch instrument from a performer's perspective and that it could serve as a foundation for future work on the subject using MIDI 2.0.

Chapter 4

Mapping

This chapter tackles the first research question, dealing with which sound parameters are more suitable for being controlled by an expressive multi-touch instrument. This first sub-question aims to answer an overarching question regarding mapping strategies from a performer perspective. Since no other informants or user studies are involved, I found it suitable to formulate it with a 'feel.' Jensenius describes the *feel* of an instrument in [25, pp. 177-183] as a combination of its 'look,' 'sound,' and 'touch.' The following is my reflective and subjective answer to the question from a performer's point of view. After all, music is about expressing feelings, and each performer has their musical language and ways to express themselves. With that said, I hope to present something relevant as a foundation to build upon for other performers customizing and performing on their expressive multi-touch instruments.

4.1 Experiences From Designing the P-6

The instrument design of the P-6 will be discussed in Chapter 5. Still, I found some experiences made while exploring the instrument to be more suitable in this discussion about mapping. Synthesizer parameters have different degrees of feedback when being modulated; some are more distinct with instantaneous audible changes, while others are more subtle. I recognized four distinctive parameter groups when designing my instrument: parameters affecting *pitch*, *timbre*, *amplitude*, and *temporality*. I would say that pitch modulation is the most instantaneously recognizable of these. Next, depending on the application, changing the amplitude and timbre of the sound could range from distinct to more subtle. Every modulation's effect also relies on the set depth of the modulation and the physical gesture itself. Finally, temporal changes could be the hardest to spot. Extra attention has to be given to this category, as not all mappings would yield results. For example, there is no use in modulating the attack stage of an amplitude envelope with the note-off velocity, as this stage of the envelope is finished by that time. With my instrument, I find it rewarding to modulate the attack time with note-on velocity and the release time with note-off velocity, an approach I cannot recall having experienced before. A modulation matrix is excellent for exploring new mapping ideas.

I made some reflections while implementing the modulation matrix of the second iteration of my instrument prototype, consisting of 20 synthesizer parameter destinations. Not every knob of the GUI ended up being selectable, as I chose to leave out the frequency control of the oscillators and the different envelope amount knobs. At the time, this felt like a logical selection. But in retrospect, I think that modulating the *Fine* tune control of oscillator 2, for example, could have provided some exciting new possibilities. I thought the limitations of a 2-dimensional modulation matrix with set boundaries would be noticeable. Still, in reality, this, along with the knob control panel, provide much more possibilities than I could fully explore for the scope of this thesis. From a performer's point of view, I think this is a good design choice. Figure 4.1 shows a sub-patch containing the internal routing of one row of the modulation matrix. However, there could be virtually endless possibilities with a more complex many-to-many mapping strategy providing intricate and non-linear expressive control of the instrument, which the knob in the third iteration is a proof-of-concept of.

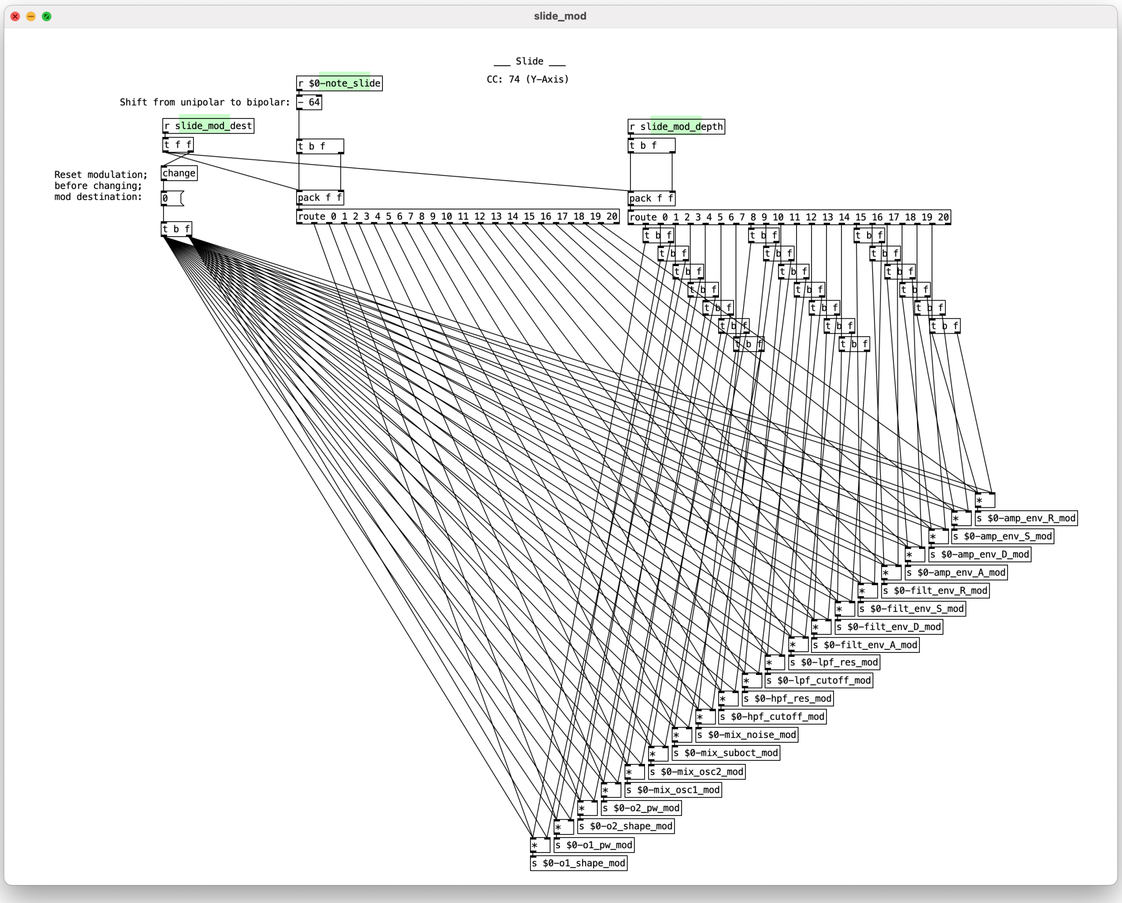


Figure 4.1: One row of the modulation matrix's internal routing, connected to all the 20 possible destination parameters after the first unconnected 'off-state.'

When mapping between an input controller and the synthesizer sound engine, it is crucial to consider the data transferred through these connections. Signal flow can soon become complicated, with signals from one source modulated by signals from other sources, each demanding scaling according to their origin and destination. The signal could, e.g., be of a 7-bit MIDI 1.0 origin, scaled to a range of 0–1, before arriving at a synthesizer parameter controlling the cutoff frequency of a filter operating between 50–20000 (Hz). There is also the question of the polarity of the modulating signal: Should it add to the destination parameter, subtract from it, or should it be a bipolar signal? Complete knowledge of the signal flow in an instrument design is also stressed in [8, 10].

4.1.1 The Relevancy of Input Sources

As I mentioned when discussing the modulation of temporal parameters, the input source is important. This is not only true regarding what 'feels' right by which gesture is chosen to modulate which parameter but also related to the stages of a voice's lifetime. To exemplify this, I think of an envelope with three stages: Attack, Sustain, and Release (ASR). The attack and release stages appear instantaneous when first striking and later lifting the finger from the multi-touch controller. Still, they are affected by the slight velocity variations of those stages.

As with the temporal synthesizer parameters, attention should be paid when using these control dimensions, especially in combination. Everything in between is the sustain(ed) stage, where one can slide, glide, press, and shape the tone according to the multi-touch controllers. Like an ADSR envelope, the sustain stage represents the amount of modulation before the voice is released. In this context, every dimension of the multi-touch control during the sustain stage has its level dynamically set through gestures. Next is an overview of the input parameters I have used throughout my instrument design and exploration.

ROLI Lightblock Control

ROLI uses the term *5D Touch technology* to describe the capabilities of their multi-touch MPE controllers. When looking into the technology behind the 5-dimensional control, it appears to be a fancy name for clever usage of well-known MIDI technology. Below is a description of which type of data is transmitted by each dimension:

Strike: Note-on velocity.

Press: Channel pressure.

Slide: Y-axis motion through CC #74.

Glide: X-axis motion through pitch-bend.

Lift: Note-off velocity.

I think of these dimensions as adding five extra degrees of freedom to the instrument. It feels intuitive and pleasing to perform expressively with this multi-touch control design. The Lightblock has a limited range because of its small size, but their range of Seaboard controllers provides more extensive control surfaces with *keywaves*.

TouchKeys Control

TouchKeys can recognize various gestures and affords various control possibilities through different mapping types. Each of these is not as easily categorizable as an extra degree of freedom as the ROLI dimensions. There are seven main mapping types: *Control*, *Vibrato*, *Pitch Bend*, *Split Key*, *Multi-Finger Trigger*, *Onset Angle* (experimental feature), and *Release Angle*. I found Control, Vibrato, and Pitch Bend the most relevant to my instrument design. Control is further divided into five input parameters: *X position*, *Y position*, *Contact Area*, *2-Finger Mean*, and *2-Finger Distance*. My second prototype iteration used every parameter except the 2-Finger Mean. Vibrato can trigger an LFO through side-to-side motion, which modulates the pitch-bend or a CC destination depending on the mapping. Finally, Pitch Bend is used for pitching up or down through the y-axis. I think the ROLI glide approach feels much more natural: to control the pitch sideways along the same axis as with a traditional keyboard. But this makes sense since the TouchKeys keys get depressed, making x-axis motions across keys inconvenient. More on TouchKeys mapping possibilities in [36].

4.2 Exploration of Destination Parameters

With experience from designing the instrument and more intuitive exploration of destination parameters, I have supplemented this with a more systematic approach to get a complete picture of destination parameters and which feels more suitable to be controlled. Using the modulation matrix of the second iteration, exploring the available synthesizer parameters with each input source was uncomplicated. The initialization preset for the second iteration of my instrument sums up which parameters felt best to me to control with this instrument design.

’Slide’ controls the ’Oscillator 1’ pulse width, where its waveshape is set between a sawtooth and pulse. When using the TouchKeys, timbre changes and pitch-bends through the y-axis can be controlled simultaneously. ’Glide’ controls the ’Oscillator 2’ shape with the TouchKeys. (When using the ROLI Lightblock, ’Glide’ is hard-coded to pitch-bend instead of sending CC messages affecting this row.) ’Strike’ controls the low-pass filter resonance, boosting the sound if struck harder. ’Press’ controls the low-pass filter cutoff frequency, which *opens* the sound by making it brighter when pressed. This enables excellent expressive control of each voice individually. ’Lift’ controls the amplifier envelope release time, causing the voice to last longer when released quickly. This is easier to perform when pressing the note first, which yields exciting combinations of temporal and timbre variations to the voice. ’Area’ controls the mixer’s sub-octave level, making the voice sound *fatter* when applying more of the finger to the note. ’Distance’ is not connected to any destination when the patch opens, as I thought to leave it to facilitate explorations by the user. One option is to *pinch* the mixer’s white noise level with this.

While working with the input sources, I found the ’Area’ source somewhat related to ’Press,’ as it can sometimes be hard to control them individually without affecting the other. This is also true for different input sources; for example, pressing could slightly slide or glide a note. Together, this makes an intricate web of expressive multi-touch control affordances, where the input sources are somewhat gesturally connected, independent of potential additional shared mapping connections.

4.2.1 The Relevancy of Resolution

While waiting for the 16-bit and 32-bit resolution of MIDI 2.0, I performed a comparison between using 7-bit and 14-bit CC values for controlling synthesizer parameters with a modified version of my first prototype iteration. It is hard to deny that 7 bits is insufficient for many applications. This is especially true for control of pitch and frequencies, which is why the MIDI 1.0 protocol combined two data bytes to provide 14-bit pitch-bend control from its infancy. Additionally, the specification allows for combining two MIDI CC values into 14-bit messages. To send 14-bit CC is easy with the TouchKeys software.

I planned to control the low-pass filter’s cutoff frequency for this experiment. To set it up, I turned the cutoff frequency down and turned off its envelope modulation. The ’slide’ (y-axis) control was then routed to this cutoff frequency, with full depth. Sliding a key would then modulate the parameter through its entire range (50–20000 Hz). I set the filter’s resonance to about three-quarters to emphasize the cutoff frequency. A high resonance can make the filter self-oscillate, providing a sinusoidal waveform at the cutoff frequency. Oscillator 1 was set between a sawtooth and pulse waveshape, with oscillator 2 being a sawtooth shape. These waveshapes have a lot of harmonics and information in the high-frequency range, which, together with the white noise, responds well to being passed through a resonant low-pass filter.

As I started with the usual 7-bit CC message, it was evident that this is a too low resolution for controlling such a parameter. I could hear quantified steps in the resonating pitch around the cutoff point as I gradually modulated the cutoff frequency to *open* the filter. When changing to 14-bit CC values, I could not discern any steps in the pitch, as the quantification had a high enough resolution to provide an illusion of continuous control. This significantly improved the control intimacy of the synthesizer and proved to me the relevancy of bit resolution in mapping. While not every synthesizer parameter is equally dependent on a high resolution, I think it is safe to say that the increased bit-depth of MIDI 2.0 will be welcomed.

4.3 Mapping a Composition

My approach to mapping in this instrument design uses explicit mapping strategies, where mapping is an integral part of the instrument [24]. Earlier, I designed an extended keyboard instrument for a project in a Motion Capture course. Here, I became aware of how mapping can also act as a specific feature of a composition. This was not a conscious choice from the start, but as the instrument patches took shape, I found that particular compositions evolved along with them. The framework for this extended instrument was that I used a Myo armband on my left hand, sending OSC data to a Pd patch, which converted these values to MIDI signals transmitted to an Empress Effects ZOIA; which is a digital modular synthesizer and audio effect processor in the shape of a guitar pedal. With my right hand, I played keyboards, with the audio output sent to the ZOIA.

I designed two utterly different instrument patches in the ZOIA for this setup, each with its distinct composition. The mapping was a central part of the design, and when the patches were finished, I practiced to finalize the compositions. It reached a point where playing something other than the associated composition on a patch felt a bit strange. The project was documented through a blog post [49]. A flowchart from this work is presented in Figure 4.2, illustrating one ZOIA patch's internal structure and mapping. As the mapping gradually transformed from being integral to the instrument to becoming a composition feature, this showed me that there are not necessarily set boundaries to these points of view.

4.4 Summary

I explored the different synthesizer parameters to control while working on the instrument design and in the play time between iterations. Sometimes, I had an intuitive idea of which parameter would feel more suitable to control. Other times, I stumbled upon new and inspiring ways of mapping gestures to synthesizer parameters. In this chapter, I have grouped synthesizer parameters into four categories and talked about how they differ: *pitch*, *timbre*, *amplitude*, and *temporality*. It is also essential to consider which input source is used to make meaningful connections. Which parameters in a sound engine feel more suitable for being controlled by an expressive multi-touch instrument depends on which 'control dimensions,' or input sources, are used on the controller to perform the control changes. The parameters and associated control dimensions constitute a *mapping relation*. Both are important when utilizing mapping strategies in instrument design.

The importance of having a high enough bit resolution is also mentioned, especially for destinations affecting frequencies and pitch, which I look forward to with MIDI 2.0. My approach to the mapping concept in my instrument is about mapping being an integral part of the instrument [24]. I have also looked into an example of mapping being a specific feature of a composition.

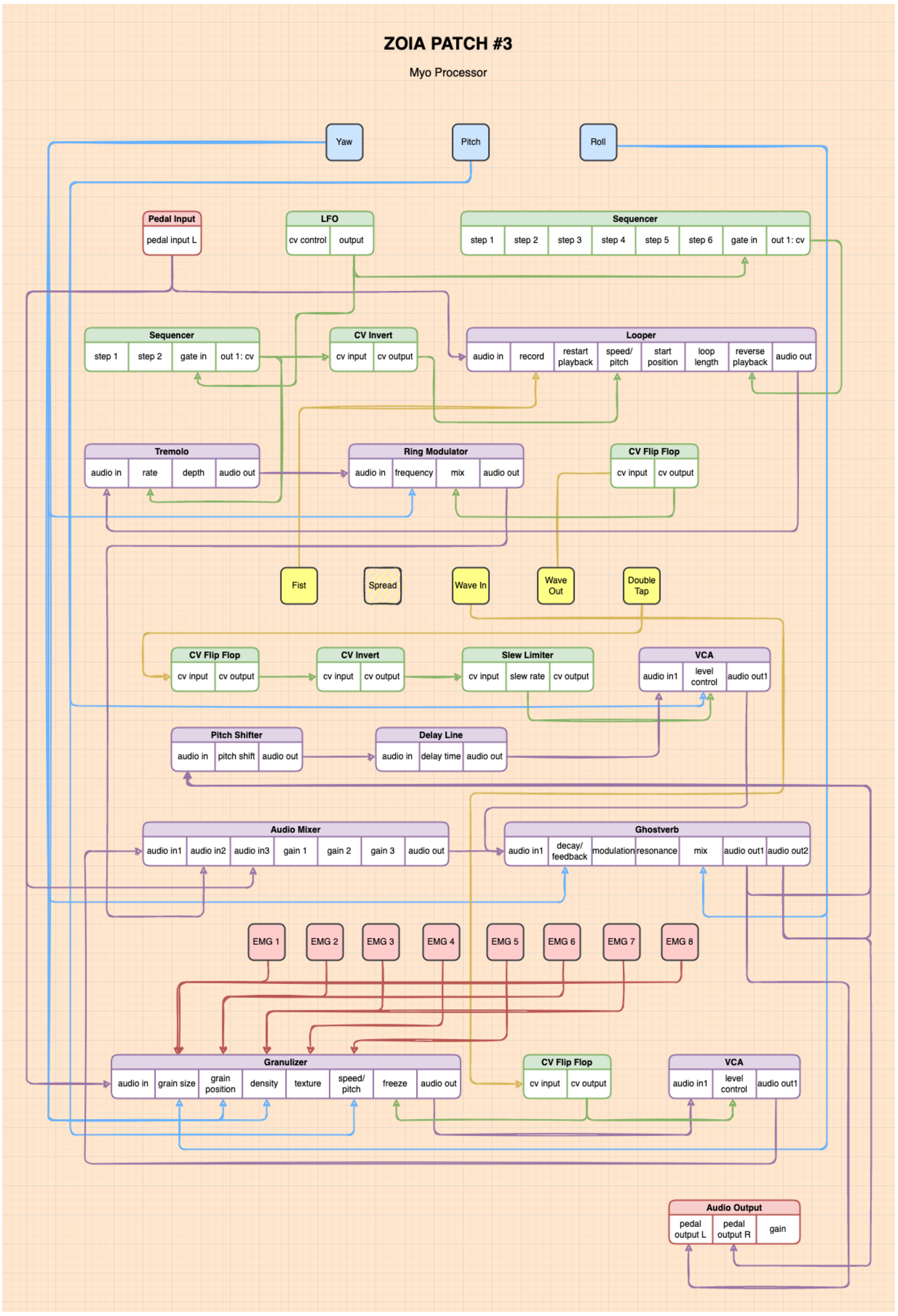


Figure 4.2: The internal structure and mapping of a ZOIA patch, used in an extended keyboard instrument design from [49].

Chapter 5

Instrument Design

This chapter presents and discusses the process of designing an expressive multi-touch instrument with MPE, the P-6. The chapter starts with a section about the pre-production stage of the design process. Next, it describes how the synthesizer sound engine was designed and implemented in Pure Data. Following is a section about the iterative design process, introducing the three prototype iterations of the instrument. The overall instrument design is then discussed before the chapter concludes with a final evaluation and reflection.

5.1 Pre-production stage

Before starting on the actual programming and implementation part of the design process, there was an initial phase of conceptualization I call the 'pre-production stage.' *Pre-production* is a term often used by performing musicians and artists about preparing a live set for upcoming concerts. This will usually include all the technicians and musicians involved in the production and is a period of rehearsals leading up to one or more dress rehearsals, preferably with an audience. When designing an instrument, much thought and preparation are required before getting your hands dirty. In the pre-production stage, I gathered a set of software and hardware tools suited for the task at hand before starting the design and implementation process to make an instrument prototype. The iterative design process will be described later in Section 5.3. The iterative process led to several *micro-iterations* before arriving at three distinct iterations of more complete instrument prototypes. This chapter does not aim to be a definitive guide on designing an expressive multi-touch instrument. Instead, it describes my process and personal reflections from a performer's point of view.

The inspiration for the sound engine of my instrument was the Sequential Prophet-6, a six-voice polyphonic analog synthesizer. This instrument is inspired by the Sequential Circuits' Prophet-5, first produced in 1978 and recently returned with its 'revision 4' in 2020. The Prophet-6 was upgraded with MPE compatibility through a firmware update, which sparked my interest in designing a multi-touch expressive instrument. Initially, I intended to design an instrument by making a more complete recreation of the Prophet-6 before adding new features. I knew that not every feature of the hardware synthesizer was relevant for software recreation, so I started by breaking down the instrument design into modules, as outlined on the instrument panel and in its operation manual [28]. The next step was to make a prioritized list. I identified which modules I thought were essential for constructing a basic playable synthesizer, continuing with which modules would enrich the instrument and which modules were not needed for my design. When recreating and implementing the essential modules and playing with the prototype, I soon strayed away from the initial idea of a more complete recreation. The core modules were recreated similarly to the original hardware. Still, the instrument design turned into something of its own before all applicable Prophet-6 modules were implemented, adding features not found in the original.

Before going deeper into details about the programming of the synthesizer, I will look into the hardware and software that I have used in the design process. The central controller for my expressive multi-touch instrument is the TouchKeys by Andrew McPherson, introduced in Section 2.1.1. For prototyping, when the synthesizer was ready to produce some basic sounds, I went and grabbed my small USB MIDI-keyboard, Akai professional LPK25, almost without thinking. The synthesizer prototype could only respond to basic MIDI 1.0 signals at this early stage. With its small form factor and portability, the USB-powered LPK25 was a perfect tool for rapid testing. Similarly, when the prototype was ready to migrate from MIDI to MPE, I thought of my ROLI Lightpad Buzz Block, an even smaller multi-touch controller. I have mentioned controllers by ROLI in Section 2.1, and their Lightpad is part of their modular Block system of MPE controllers. This is a small square controller with a LED-illuminated surface and a tactility akin to their Seaboard instruments, which became handy for the following design process. The portability allowed me to design and work on the instrument without needing to be in the Synthroom at the University of Oslo, where the TouchKeys resides.

In the context of modularity in instrument design, a downside is that different expressive multi-touch controllers might provide different input control possibilities and naming conventions of similar parameters. This made the first design iteration tailored to the more limited Lightpad. I created the second iteration incorporating many of the added control possibilities of the TouchKeys; however, the Lighpad can and is still being used. On the positive side, dealing with different MPE controllers made me more aware of the controller aspect of an instrument and how to leverage available possibilities tailored to the instrument design. It also helped enrich the foundation for my thoughts on mapping strategies, discussed in Chapter 4.

The software for designing my synthesizer sound engine was the 'vanilla' distribution of Pure Data (Pd), introduced in Section 2.3. I found this a suitable programming environment for developing and implementing my design, with the visual approach of patching objects together resembling the patching of synthesizer modules. Pd has objects that can handle MIDI 1.0 and OSC communication, but it lacks native support for MPE and MIDI 2.0. To be able to communicate between my synthesizer and an expressive multi-touch controller through the MPE protocol, the instrument depends on the 'pd_mpe' external[1]. In my Pd patch, I used a modified version of an abstraction for parsing the incoming MPE data included with this external library. This provided a convenient means of visualizing incoming values from the different voices of the controller, excellent for troubleshooting during the design process.

I chose to depend on another Pd external as well, the EL Locus Solus' Externals (ELSE)[2]. The ELSE has, among others, new GUI objects and a set of abstractions and objects suitable for synthesizer design, which makes the design process more manageable. Ultimately, I only used a few of these objects: A knob object was employed to have the GUI a bit more like the Prophet-6, plus some of the available oscillators and an ADSR envelope generator. The oscillators and envelope generator could be implemented using vanilla Pd objects, but not new GUI elements like the knob. However, the ELSE already has anti-aliased band-limited oscillators, which, along with the ADSR, were convenient to use in my synthesizer design.

Other necessary software was the TouchKeys software and the ROLI Dashboard software for controlling and mapping input parameters of the MPE controllers in use. Lastly, I made frequent pushes to a GitHub repository containing this project[3], including a development diary and the synthesizer design, through the GitHub Desktop software. There were a few compatibility issues with newer computers regarding the TouchKeys editor and the 'pd_mpe' external. The last released TouchKeys software from their website was compiled in August 2015 and did not open on my laptop. Luckily, I was handed a newer, unreleased build, which worked for me and on the Mac Mini in the University's Synthroom. The pd_mpe external, however, with the last

[1] https://github.com/DanielRudrich/pd_mpe, released under the GPL-3.0 license.
[2] https://github.com/porres/pd-else, released under the WTFPL license.
[3] https://github.com/wnetzel/all_in_the_multitouch

release from April of 2019, did run on my laptop with an Intel Core i7 chip but not on the newer Apple M1 chip on the Mac mini. This makes running my instrument prototype on newer computers a bit more complicated, but compiling compatible versions from the open-source files should be possible.

Computers were necessary for designing the synthesizer in Pd, of which I used my MacBook Pro laptop with macOS Big Sur (version 11.7.10) and the Mac mini with macOS Monterey (version 12.7.2) in the Synthroom. While designing, I used a few different sets of speakers to get sound out of the instrument. In the early stages of the process, I only needed to hear if there was any sound, so I used the internal speakers on my laptop. This continued for as long as the speakers were only utilized to hear crude changes in the generated sound, like distinguishing between different waveshapes, various sound sources, and other basic features. The laptop speakers have a limited and non-linear frequency response. Still, having the sound engine and speakers in the same device was convenient, with a connected small USB-powered controller. As soon as the synthesizer prototype got more sophisticated sound design capabilities, and especially after adding a low-pass filter and envelope generators for filter and amplitude control, I put on my AKG K240 Studio headphones. This improved the auditory feedback tremendously, with a more linear frequency response. The input controller and the output sound generator were now separated from the synthesizer with short cables.

Although sitting for hours with headphones on was tiring, this continued to provide a highly portable prototyping setup. When the second iteration of the instrument design was finished, I was ready to move on to proper studio monitors. I did not have the same need for portability but required excellent auditory feedback when exploring and playing the instrument. My design was completed using the Genelec 1032A monitor speakers in the Synthroom. The addition of monitor speakers and the TouchKeys controller to the instrument made it clear that I needed an external sound card in the setup. Until then, using headphones and the Lightpad controller, the computer's internal sound card was sufficient. However, the monitor speakers need two audio cables connected, so I added my Focusrite Scarlett 6i6 sound card to complete the instrument.

5.2 Designing the Synthesizer

The heart and soul of the instrument prototype is the synthesizer sound engine designed in Pd. After identifying the essential Prophet-6 modules to recreate as a starting point, implementing them in Pd began. A front panel patch window was set up to contain the different modules while the modules were made into separate abstractions. In the end, I had seven synthesizer modules, briefly described in Appendix D: Two oscillators, a mixer, high-pass and low-pass filters, and filter and amplitude envelopes. A master volume module was also added to provide a global volume control for the synthesizer's output. A simplified visualization of how the modules are interconnected inside the synthesizer is delivered in Figure 5.1.

In my experience, the modularization and iterative process was a good approach to instrument design. This broke the process into smaller steps, making maintaining an overview and a steady progression easier. The modular approach also resonates well with the synthesizer architecture in itself. In addition to starting with a few modules and later expanding, the design also evolved from containing basic features to more advanced ones. This sometimes demanded that the design and internal logic of the program change significantly. The first and maybe most significant step was to upgrade the synthesizer from monophonic to polyphonic. Initially, the monophonic synthesizer responded only to MIDI 1.0 signals. I planned on implementing a voice card abstraction to be cloned for each synthesizer voice, but this was incompatible with my initial module abstraction architecture. Each module abstraction file originally contained the GUI and the logic, which seemed to be a good design choice for a monophonic synthesizer. When you opened a module abstraction from the synthesizer front panel, it provided a clear view of the control parameters of the GUI with direct connections to the logic happening inside. With a

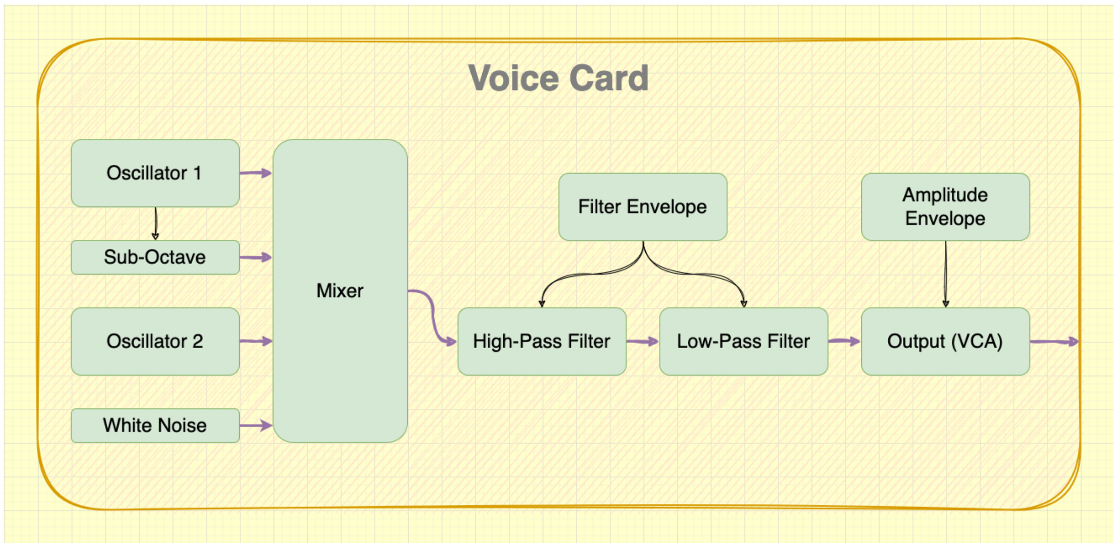


Figure 5.1: A simplified representation of how the modules are connected.

polyphonic synthesizer, however, the GUI and the logic of the modules had to be separated. The GUI should enable global control of synthesizer parameters for all voices, while the module's logic should reside inside the voice card for individual control of parameter modulation per voice. This resulted in the module abstractions being stripped to containing GUI elements and scaling the input signal. The logic of each module was moved and assembled to a new architecture inside the voice card abstraction.

The second transformation was when migrating from MIDI 1.0 to MPE control, which demanded some restructuring in how to treat input signals. This change also exposed an issue with the modulation logic. As the MIDI-keyboard used until then didn't have multi-touch control, the simple modulation offered was global. When MPE was implemented, the modulation of every voice also affected every other voice, which felt like a polyphonic synthesizer with cumulative monophonic modulation. This demanded a new way of acquiring controller data inside the voice card abstraction, where the modulation destinations also moved from the global GUI abstractions to the module logic inside the voice card abstraction. The result was per-voice modulation of the polyphonic synthesizer. Over the iterations, the synthesizer expanded to 20 modulation destinations. The input sources for modulation were also gradually extended to seven, with depth control for each, where the x-axis and y-axis input data were offset to provide bipolar modulation.

From the start, Pd Canvas objects were used to color the synthesizer modules to make them resemble a hardware synthesizer. This initially had an aesthetic purpose but gradually evolved to serve additional purposes. In addition to the pure aesthetics, colored areas provide visual sectioning of code in the patch to make it easier to navigate. Another vital use of colored canvases is to 'tag' send and return objects inside the sub-patches to see better the outgoing and incoming signals to and from other sub-patches. The send and return objects provide connections between objects without a visible patch cord between them, and the coloring of these objects makes the documentation of the code easier as well.

An effort was also made to future-proof the synthesizer. Instead of hardcoding the 0–127 MIDI 1.0 range into the synthesizer logic, the input data gets scaled internally according to a bit-resolution selector. This is currently initialized to the 7-bit values provided by MIDI 1.0 and MPE and can be changed to treat 16-bit and 32-bit values from the MIDI 2.0 protocol. It also easily enables using 14-bit values with MIDI 1.0 by combining two CC signals, supported by the TouchKeys controller. The bit selector worked as supposed when comparing 7-bit to 14-bit resolution for Section 4.2.1. 14-bit value handling is otherwise implemented in the synthesizer, with the 'Pitch Glide/Vibrato' control source currently received as 14-bit channel pitch-bend data hardcoded to a pitch-bend functionality. Handling data from the OSC protocol or other sources should also be trivial with the bit-selector.

5.3 The Iterative Design Process

The iterative design process was documented through a diary, logging the process and my reflections, which can be found in Appendix E. This document mainly concerns designing the synthesizer sound engine part of the instrument. Before arriving at the major design iterations, namely *Prototype v1, v2* and *v3*, I went through several steps, which I named 'micro-iterations.' The duration between designing and testing, or in other words *design time* and *play time*, was relatively short in the early stages of the instrument design compared to the later stages. There was a frequent need to test if a module feature appropriately worked before continuing with other features. In the start, the play time barely resembled any play time, as there was only a fundamental need to get audible feedback as expected. The instrument produced raw basic sounds, which didn't appeal to expressive playing.

The terms *design time* and *test time* are probably more suitable for these early stages to expand upon the concepts from [39]. But as the instrument grew, the frequency between designing and testing decreased, and the test time gradually transformed into actual play time. This transition was not instant, but I had more prolonged stretches of testing and playing as the instrument got more features to play around with. The mentioned micro-iterations, starting with *iteration v0.1*, is where this transition happened, each reflecting a milestone in the design, followed by this *test–play time*. The frequent swaps between design time and test time before arriving at the first micro-iteration could then be called 'nano-iterations.' The design time lasted longer when working on the final iterations, followed by more extensive play time stretches. Since the instrument was up and running, it was easy to grab the controller and test features when needed without breaking up the design time. Although the final prototypes are not production-ready, the design of these iterations feels more like an actual instrument.

To sum it up, there was a high frequency of short design time and test time nano-iterations at the start before arriving at the first design milestone. The following milestones were the results of micro-iterations with a decreasing frequency of design time and test–play time of gradually longer stretches. In the end, there were a few long stretches of design time and play time between the final iterations of the instrument.

5.3.1 Prototype v1

The instrument design in this iteration is an expressive multi-touch instrument with a six-voice polyphonic synthesizer made in Pd producing the sound. This iteration is still underdeveloped regarding the mapping strategy. It has a prototype of a modulation matrix with two input sources and seven synthesizer parameter destinations, with additional possibilities to modulate the filter and amplifier envelope amount with velocity. The glide (x-axis) of the multi-touch controller is hard-coded to perform pitch glides between notes. Regarding the discussion of degrees of freedom (DoF) and cognitive load of an instrument [25], I think of this instrument as inherently having two DoF with the sideways note selection motion and vertical velocity motion, where the control sources for modulation adds to this number. The added glide (x-axis), slide (y-axis), and press (z-axis) modulation sources add three additional DoF, making it five in total. Although the note selection and glide control share the same axis, I think these are separate actions. This is also true for the shared axis of velocity and press. Regarding the cognitive load, the limited modulation possibilities feel manageable. I used three of the six available voices when playing on it after it was finished. However, the small surface area of the Lightpad controller used with this iteration probably also affects the number of voices used.

I incorporated the synthesizer modules mentioned in 5.2 except for a High-Pass Filter. Every knob and button in the GUI has functionality that can be tweaked. The Pd patch opens with a preset of synthesizer parameter settings, which I found nice to use as a starting point when playing with this iteration. I utilized the ROLI Lightblock and headphones to complete this instrument prototype, but it could also be employed with speaker monitors and TouchKeys. However, the instrument does not yet use many possible multi-touch affordances of the TouchKeys. Except for a limited expressivity because of a basic mapping strategy, I was pleasantly surprised at how the instrument sounded. I think fondly of the analog Prophet-6 and the *warm* and rich sound quality it produces, which I assumed would be hard to emulate. Although more limited, this instrument prototype has its qualities. A view of the main GUI of the Pd synthesizer patch in *Prototype v1* can be seen in Figure 5.2.

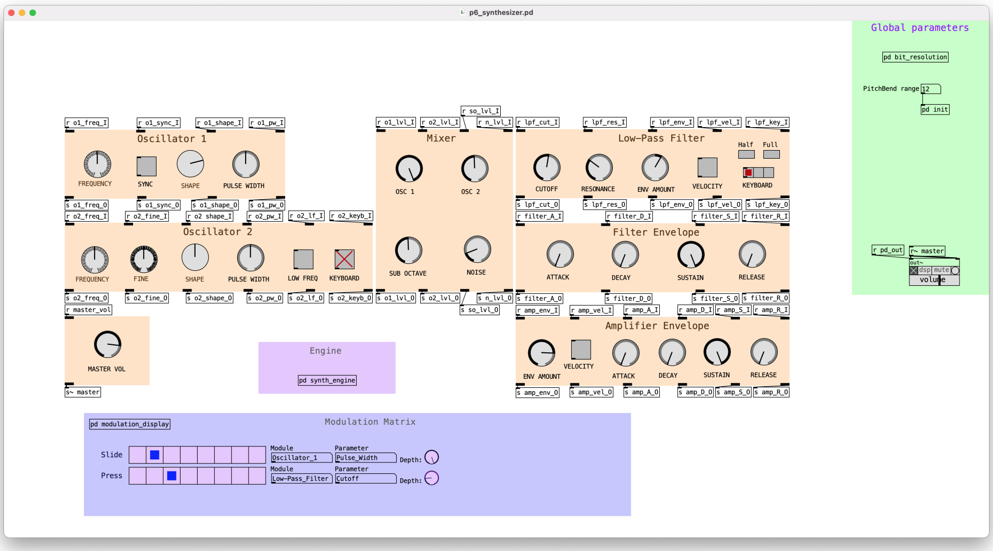
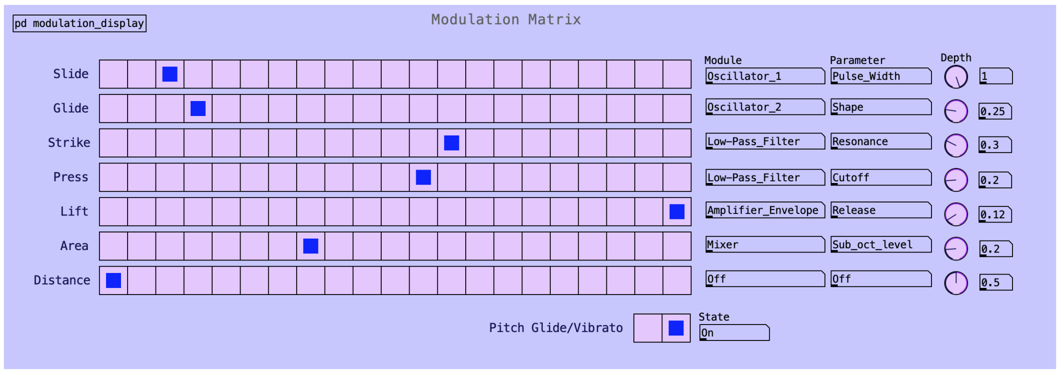


Figure 5.2: The front-panel GUI of the Prototype v1 iteration of my synthesizer design.

5.3.2 Prototype v2

The second iteration of the synthesizer design is an expanded and improved version of the first: A high-pass filter has been added, the modulation matrix has been boosted to seven input sources, an on-and-off toggle of the pitch glide functionality was added, and an extended selection of 20 synthesizer parameters to be modulated. The seven input sources expand this iteration's degrees of freedom to nine, with a perceptible addition to the cognitive load. However, because of the possibility to turn off the modulation routing for each source, the coinciding amount of DoF and cognitive load can effectively be adjusted by the user to suit their preferences. When first playing with it, I still revolved around using three voices but with a noticeably higher complexity and cognitive load. After a while, I expanded it to four voices with the right hand plus a bass note with the left hand.

With this iteration, I also added the intended monitor speakers and TouchKeys to the prototype, which expanded the sonic output and the expressive multi-touch control of the instrument design. Prototype v1 felt unfinished, but v2 performed more like a complete instrument. I enjoyed several play time sessions on this iteration, and the modulation matrix opened up many possibilities for expressive performance. I have also continued using the Lightpad to play with this iteration, which then omits three control dimensions. Figure 5.3 shows the added modulation matrix GUI of Prototype v2's synthesizer.



pd modulation_display

Modulation Matrix

	Module	Parameter	Depth
Slide	Oscillator_1	Pulse_Width	1
Glide	Oscillator_2	Shape	0.25
Strike	Low-Pass_Filter	Resonance	0.3
Press	Low-Pass_Filter	Cutoff	0.2
Lift	Amplifier_Envelope	Release	0.12
Area	Mixer	Sub_oct_level	0.2
Distance	Off	Off	0.5

Pitch Glide/Vibrato

State: On

Figure 5.3: A view of the modulation matrix added to the synthesizer design in Prototype v2.

5.3.3 Prototype v3

For the third iteration of the instrument, *Prototype v3* adds a proof-of-concept prototype of an abstract intermediate layer, providing a many-to-many mapping strategy. The Prophet-6 has a 'Poly Mod' module providing many-to-many mapping, which could have been implemented here. Still, as I had other visions for my design with the modulation matrix in the previous iterations, this module became out of scope to implement. Instead, I thought of an intermediate layer providing abstract parameters for mapping between the controller and synthesizer parameters, as discussed in [24, 48]. This design introduces a knob with an abstract name without further explanation. The knob controls the amount of modulation from a 'black box' of mappings hidden from the user, which also incorporates randomness. While only being a proof-of-concept prototype in this iteration, this approach will be interesting to pursue. Figure 5.4 shows a screenshot of Prototype v3 with "The Knob" added.

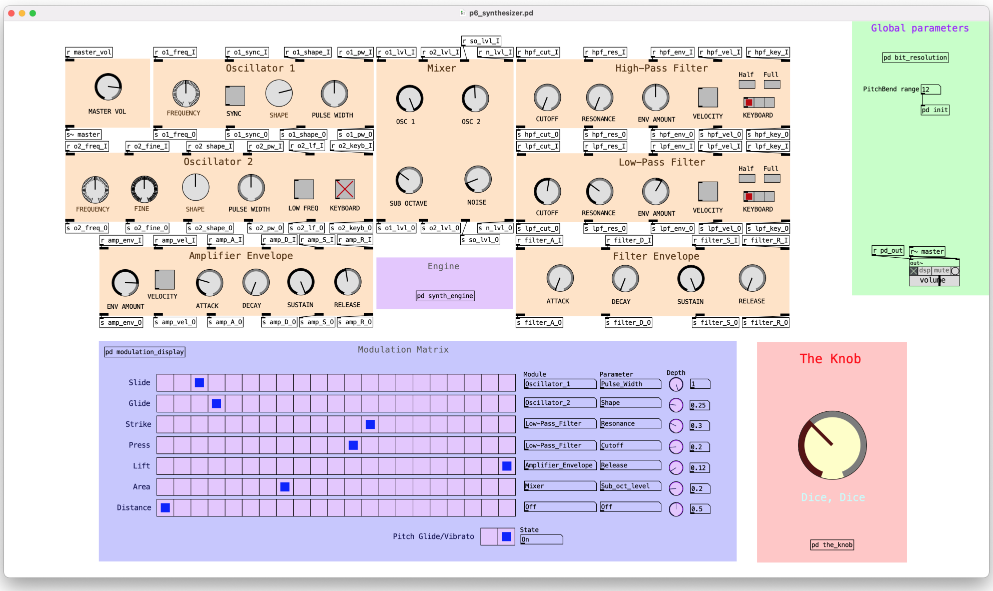


Figure 5.4: The main GUI of the synthesizer with "The Knob" controlling an intermediate abstract mapping layer in Prototype v3.

5.4 The Instrument: P-6

Although there are three prototypes of the instrument, I think the gist of it is present in all iterations, making them expansions or variations of an over-arching instrument design. I tried clarifying the instrument parameters through the main GUI, using labels borrowed from the Prophet-6. With some knowledge of analog synthesizers, the functionality should be mostly self-explanatory. Still, some mystique is kept to encourage instrument exploration through the modulation matrix. Each row of buttons for selecting modulation destinations has two displays showing which parameter the input source modulates. One indicates the module, and the other reveals the destination parameter. Where the modulation matrix provides one-to-one and many-to-one mapping strategies, the addition and further development of an abstract mapping layer affords more complex mapping. There are a few possibilities to do one-to-many mappings using the velocity modulation buttons with the strike (note-on velocity) input source. One could also utilize the synthesizer's slide control together with TouchKeys' pitch bend mapping, sharing the y-axis. This provides limited possibilities for many-to-many mapping.

The naming convention of the input sources, partly similar to the ROLI controllers, provides an idea of which gestures will be used for the modulation without explicitly telling it. For the curious user, more information is supplied through comments inside the sub-patches and the documentation. The value ranges of the different synthesizer parameters are not explicitly revealed but can be found by exploring the different sub-patches. Their ranges were set partly by comparing it to the Prophet-6 and partly to my taste. I discovered an exciting intersection between providing necessary information without oversharing when designing, as I wanted the instrument to be intuitive and still encourage exploration.

A visual representation of the instrument prototype with all its parts is provided in Figure 5.5, which shows how its components are connected.

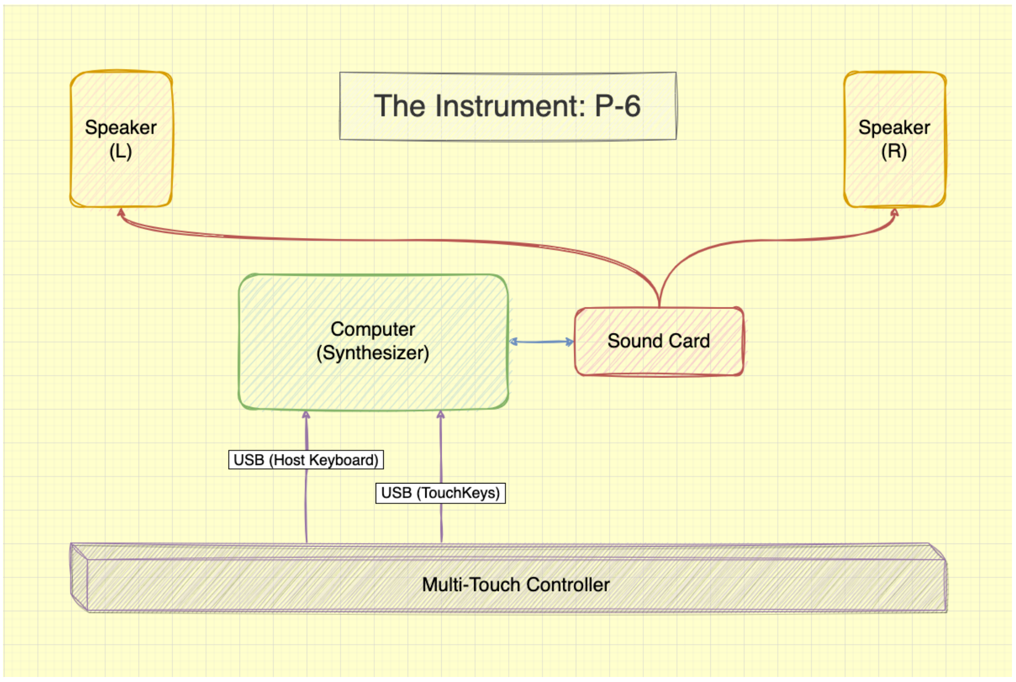


Figure 5.5: A simplified representation of the instrument design with all its components. When used with the ROLI Lightpad, only one USB cable is needed.

5.5 Final Evaluation and Reflection

I enjoyed playing on Prototype v2 the most. The intermediate abstract mapping layer has immense potential when it evolves from the current proof-of-concept prototype in v3. This is especially true for a developer–performer, as one could get inside the patch and expand it with code to tailor it to one's needs. Such an approach makes it more open-ended, where the instrument is not necessarily fully developed, which demands additional knowledge of Pd and synthesizer architecture. On the contrary, I think the modulation matrix is a better choice for the user–performer, even if it is limited to the possibilities provided. With this, the instrument design is completed, providing ample exploration space for the performer. Demonstrations of the instrument in action can be found in the associated OSF repository[4]. With future expansions, e.g., adding the 'Poly Mod' module, I hope the instrument design will be enjoyable and fulfilling for the user.

I sometimes get *in the zone* when designing or implementing features in the synthesizer. An intuitive idea of how to add or fix a feature might appear without necessarily considering why. If it doesn't work, I will analyze why and find a solution. If it functions, I will also find out why it did so to learn from it. This experience reminds me of being *in the zone* when performing and improvising. You don't necessarily think consciously about which scale or which rhythmical motifs to use all the time. An intuitive approach is made possible by all the hours of practice and experience in the craft. When this happens during the design time, I think this is because my knowledge of and experience with synthesizers are elevating my programming skills.

I have many ideas for modules and functionalities to expand the instrument for future iterations. E.g., to add modulation wheel control. This contradicts the design idea limitation of only using the multi-touch features of the controller. Still, it would have been nice to have a global modulation control option affecting every voice. Portamento control would also improve the instrument. Of other modules from the Prophet-6, audio effects and a distortion module would expand the sonic capabilities. The mentioned 'Poly Mod' would provide new complex mapping possibilities along with the modulation matrix. The 'Slop' module would make it appear to sound more vintage and *warm* by adding random imperfections and fluctuations to the individual voices, emulating the more unstable older synthesizers. A Low-Frequency Oscillator (LFO) module would also be one of the first features I would add to provide more modulation capabilities.

That said, we might have to think differently about synth design and manufacturing in the future, with the added possibilities of MIDI 2.0. LFOs and envelopes might be superfluous if we have such a full and crisp control of each note with new expressive multi-touch instruments. I look forward to seeing how the MIDI 2.0 protocol could soon enrich my instrument design and provide a more intimate and expressive multi-touch control.

5.6 Summary

This chapter has shown that three components are key to designing an expressive multi-touch instrument using MPE for expressive performances. First, we need a multi-touch input controller for capturing expressive multi-dimensional gestures. This controller needs to be connected and mapped to a sound engine. Second, we need a sound engine connected to a sound-producing device. I have exemplified this with the design of an instrument prototype where TouchKeys controls a synthesizer designed in Pure Data connected to a set of speakers. Third, a computer must run Pd with a sound card and speakers connected. Additionally, I have presented a modular approach to the synthesizer design and shown the modularity of the overall instrument design and how different parts of the instrument could be exchanged. I have also used the

[4]https://doi.org/10.17605/OSF.IO/JFMCV

ROLI Lighpad as a controller, various iterations of the synthesizer as the sound engine, and headphones as the sound-producing part. This design process resulted in the instrument *P-6*.

Chapter 6

Discussion

This concluding chapter starts by summarising my project, followed by a discussion about the coming of MIDI 2.0. Next, my answers to the main research question and the three sub-research questions are presented. The last section provides proposals for future work on the subject.

6.1 Summary

This thesis provides a theoretical overview with a literature review of topics relevant to my research questions about designing and exploring multi-touch instruments with MPE and MIDI 2.0. Three operational sub-research questions concern suitable parameters for mapping, approaches to instrument design, and expressive control with MIDI 2.0. The first topics are about expressive multi-touch instruments and the TouchKeys controller used in my instrument. Further, digital sound synthesis and Pure Data are covered, relevant to designing a synthesizer sound engine for my instrument. The MIDI protocol gets reviewed next, with subsections for MIDI 1.0, its early critique, MPE, and MIDI 2.0. MPE is currently used for communicating between the controller and synthesizer in my instrument. Next, a few alternatives to MIDI are presented, ZIPI, and Open Sound Control (OSC). Mapping strategies is the last topic to be covered, which is relevant for making meaningful connections between the controller and synthesizer of my instrument. When reviewing the topics, some parts were too technical to be a part of this story. As a result, these paragraphs were moved to Appendix A (MIDI 2.0), Appendix B (ZIPI), Appendix C (OSC), and Appendix D (sound synthesis), providing further knowledge.

In the methodology chapter, I have reflected upon the methods used in this work. Methods covered are literature review, iterative design and prototyping, programming, performance research, introspection and self-reflection, and practice research.

Next is a chapter discussing mapping, which provides a basis for answering RQ1. Here, experiences from designing the P-6 are presented and reflected upon. This chapter concerns the relevancy of input sources regarding synthesizer parameters to modulate, with a review of the ROLI Lightpad and TouchKeys used in my instrument design iterations. Following is an exploration of destination parameters, which looks into how I initialize the mapping when opening the second iteration of my synthesizer patch. The relevance of resolution in control values is also discussed. A discussion about a different approach to mapping, where mapping is a feature of composition, ends the chapter.

The chapter about instrument design provides the basis for answering RQ2. It starts with discussing what I have called the pre-production stage before covering the synthesizer's design process in Pure Data. Next is a breakdown of the iterative design process applied, with an introduction to the instrument's three main iterations: Prototype v1, v2, and v3. The instrument, P-6, is further described with all its components before ending the chapter with an evaluation and reflection on the instrument design.

6.2 Towards MIDI 2.0

This section will, together with the literature review from Section 2.4 and its subsections, the technical data and MIDI 2.0s current status from Appendix A, and experiences from exploring my MPE instrument, provide a basis for answering RQ3.

After designing and exploring my expressive multi-touch instrument P-6, I better see how beneficial technological advances can be for musical expression. Accustomed to acoustic pianos, electroacoustic keyboards, and analog synthesizers, the MIDI 1.0 protocol is limited in many applications. My concern has mainly been the 7-bit resolution and sometimes the increased spatiotemporality, decreasing the control intimacy with the instrument. MIDI is still unsurpassed for easily connecting instruments and musical devices of almost all sorts. There is not much literature on MIDI 2.0 yet, which I hope this thesis will contribute to. Of the available literature, I found [19] about "Applications and Implications of MIDI 2.0" worth mentioning. This thesis describes implementing parts of the protocol in Python.

6.2.1 Make MIDI Great Again

When reading the early critiques of MIDI 1.0 reviewed in Section 2.4.1, I got an impression that this once promising new protocol might have been a disappointment to many in the time after it got released. I even became aware of issues I do not recall having been consciously bothered by but probably have been exposed to, like the temporal smearing discussed in [38]. It seems like we have grown accustomed to and accepted these shortcomings for many years by now. We might be more indulgent with the protocol in 2024 than in the 1980s, especially with younger musicians growing into it without knowing other solutions or thinking it could have been otherwise. I find it interesting to see how little weight has been laid on the low resolution of MIDI 1.0 in the critiques I have read, which seems to me to have been one of the major concerns for years now.

By taking a brief step back to the article by Loy [29], we can see that MIDI 2.0 finally answers some of the early critiques of its predecessor by, for example, providing bidirectional communication. With no upper limit on the bandwidth in the specification itself, MIDI 2.0 should also be capable of capturing human performance for many more applications, together with the added message types for continuous per-note control.

MIDI Polyphonic Expression (MPE) has significantly contributed to the MIDI protocol in providing per-note control, enabling the design of new expressive multi-touch instruments. Although MPE was adopted by the MIDI Manufacturers Association in 2018, only two years before MIDI 2.0 got introduced in 2020, we are seeing a rise in popularity and availability of new MPE supported instruments and controllers today. We can not yet say the same for MIDI 2.0 instruments, although it has gone four years already. Ableton recently announced Live 12. This version provides more extensive MPE support, but to my big disappointment, there is no word about MIDI 2.0 in the release notes.

I think there are several reasons why MIDI 1.0 is still going strong. One is that it (generally) works. All the major music instrument manufacturers have embraced the protocol, and so have all the large software companies. Much of the early critique focused on what MIDI could have been, a general-purpose communication tool for all sorts of instruments. MIDI 1.0 failed to provide that, but it did support the exchange of control data for keyboard-based instruments. And that works well. MIDI 1.0 handles regular note-on/note-off messages from traditional keyboards as expected. That is sufficient for many people, yet things are changing with the advent of new controllers. This is also why MIDI 2.0 is necessary.

6.2.2 What Will Become of MIDI 2.0?

Observing how the MIDI 2.0 protocol rolls out in the upcoming months and years will be interesting. A boost of interest in this specification could result in more supported hardware and software instruments being produced, which further could spark the interest in making external libraries and support for the specification for more programming languages and DAWs. I think the appearance of multi-touch MIDI 2.0 controllers is long overdue. But there is no need for controllers if there are no MIDI 2.0 synthesizers or software to control. I hope to see a positive spiral effect if an increased availability of MIDI 2.0 products recruits more musicians to these products' technology.

This process probably depends on the publicity and implementation of MIDI 2.0 to gain momentum. As we are currently in the early stages of the MIDI 2.0 life cycle, there is still a bit of 'the hen or the egg' problem. How can musicians, producers, and developers use this technology if there are few supported products? And why would commercial companies use precious time and resources to develop such products if no one uses them? Fortunately, it seems that The MIDI Association is working on promoting and making this new specification visible to the public, both to individuals and corporations.

Remembering how big of an impact MPE made to the MIDI protocol, as a performer, I am excited to see what MIDI 2.0 will bring. We already have the specification available, providing material for making assumptions. It even means one could start designing their own MIDI 2.0 instrument today. While waiting to get my hands on such an instrument, and after looking into its specification, I have big expectations for how the future of expressivity will be with MIDI 2.0.

6.3 Answer to the Main Research Question

I started with the following main research question: *How can we design and explore an expressive multi-touch instrument using MIDI Polyphonic Expression (MPE) and MIDI 2.0?* Throughout the thesis, I have tried to demonstrate that it is possible to design an expressive multi-touch instrument using MPE by taking inspiration from an existing hardware synthesizer and creating a sound engine using Pure Data. To play the instrument, we can use one of the many expressive multi-touch controllers available today, like the TouchKeys. These parts can then communicate through the MPE extension to the MIDI specification. This affords multi-dimensional control of individual notes. The design and exploration process can be intertwined with an iterative prototyping approach, which could help make a more robust instrument. Different mapping strategies can also be explored for meaningful relations between the controller and the sound engine before making the last — but not least — exploration by practicing and performing with the instrument.

If we look into A.2, a technical answer to how we can design and explore an instrument using the MIDI 2.0 specification today could be to make a VST instrument in C++ by using Steinberg VST 3.7 SDK or an Audio Unit v3 instrument using Swift. One could also make an external for Pure Data and update my patch to use the upgraded bit-resolution and new affordances of MIDI 2.0. Then there is the issue of having a multi-touch controller send MIDI 2.0 packets to this patch, which could have been solved by making a new software that translates the native OSC output of the TouchKeys to MIDI 2.0. Either way, this gets too complicated and out of scope for my thesis.

My answer is to wait a little longer, as designing instruments with MIDI 2.0 will be considerably easier when supported hardware controllers and more external libraries for programming languages appear. I will upgrade and expand my instrument design with this protocol when an external MIDI 2.0 library for Pure Data appears. The answer to how to explore an expressive multi-touch instrument using MIDI 2.0 will ultimately be the same as for a MPE instrument, but with the added resolution and possibilities in expressivity that the new specification affords.

6.3.1 Answer to RQ1: Suitable parameters

Regarding RQ1, I conclude that parameters affecting pitch, timbre, and amplitude in a sound engine feel more suitable for being controlled by an expressive multi-touch instrument. This can, e.g., be the cutoff frequency of a low-pass filter, pulse width modulation, waveshape modulation, individual control of pitch per note, and white noise level, to name a few. Of these, parameters affecting frequencies and pitch are the most perceptible. Parameters controlling temporality are often not the most suitable for being controlled or even of any relevance. However, exceptions exist, e.g., controlling the release stage of an amplitude envelope to make a note longer when released. The controller dimension used for modulation is to be considered for meaningful connections. I recognize that parameters with a potentially more significant audible impact on the sound are generally the better choice, as these afford more expressive control — ranging from vast to subtle — when performing on a multi-touch instrument.

6.3.2 Answer to RQ2: Design approaches

As seen in the answer to the main research question, we can use an iterative prototyping approach to design a sound engine able to receive MPE messages in Pure Data, taking inspiration from existing hardware synthesizers. We can modularize the design and implement a few essential modules before expanding. Through different iterations, we can then add relevant features and functionality. As for operating this synth engine, we can choose one of many available expressive multi-touch controllers that support MPE. To make this instrument suitable for expressive performances, we apply mapping strategies to create the desired connections between the multi-touch control and the sound engine parameters. To complete the instrument, we connect it to speakers.

6.3.3 Answer to RQ3: Expressive control

The increase in resolution from 7-bit to 16-bit velocity and 32-bit CC messages and added per-note control possibilities in MIDI 2.0 are some consequential benefits of MIDI 2.0 that could significantly enhance expressive control of instruments. As we have seen, this upgrade is especially crucial for the control of parameters affecting frequencies and pitch. This can be a two-edged sword for the developer–performer, as the new Universal MIDI Packet (UMP) messages providing these possibilities also bring more complexity and challenges in designing instruments and mapping strategies for facilitating the new benefits. The user–performer can use their experience from exploring MPE instruments to better understand how to approach and embrace these new opportunities.

6.4 Future Work

After these first iterations of my instrument are presented, I will continue the work on my synthesizer design, adding modules and features to it. I will also aim to take it on the road to perform with it. It would be a pleasure to include others in the design process as well, and hopefully, the instrument's GitHub repository can grow and take on a life of its own. New mappings are still to be made, as I will pursue the path starting with the last iteration. As soon as there comes an external for implementing MIDI 2.0 with Pure Data, I will start upgrading the synthesizer. All the while, I will be searching for the first sign of a MIDI 2.0 multi-touch controller for the setup.

I hope this work could spark an interest in others, both for designing expressive instruments and the emerging protocol. As a performing musician and music educator, I am excited and optimistic to see how MIDI 2.0 will affect the music scene in the future.

Appendix A

More on MIDI 2.0

A more in-depth look at the MIDI 2.0 specification and its current status are provided in this appendix.

A.1 More on the MIDI 2.0 Specification

MIDI Capability Inquiry (MIDI-CI)

MIDI Capability Inquiry (MIDI-CI) is the name of the new protocol, which uses bidirectional communication to negotiate for extended capabilities beyond what is found in MIDI 1.0. This allows for easy configuration between instruments, devices, and software with MIDI-CI support and lets the *Sender* know the capabilities of the *Receiver*. MIDI-CI expands the MIDI functionality in three significant areas, which collectively are known as *"The Three Ps*:

1. Profile Configuration
2. Property Exchange
3. Protocol Negotiation

Profile Configuration allows for choosing and using among industry-standardized setup profiles suited for the types of instrument, controller, hardware, or software that are communicating in the MIDI-CI session. This allows for automatic and intuitive mappings of controllers to instruments or effects. E.g., if a Drawbar Organ software instrument is to be used, a hardware controller with a set of sliders, knobs, and buttons would automatically choose the Drawbar Organ profile with mappings to parameters like drawbars, Rotary Speaker speed, overdrive, percussion, etc., relevant for controlling such an instrument. MPE will continue to live on inside of MIDI 2.0 as it is to be included in the list of profiles. There are three different profile type categories, here listed with examples found in the MIDI 2.0 Core Specification document named *Common Rules for MIDI-CI Profiles* [5] and from their Winter NAMM 2021 Session about MIDI 2.0 Profiles Configuration and Property Exchange [2].

1. **Feature Profiles** or **Ancillary Profiles**
    - General MIDI
    - MIDI Show Control
    - MIDI Visual Control
    - Note Tuning
    - MPE
    - Sample Dump

- Mixed Data Types
- Orchestral Articulation
- Subtractive Synthesis Control
- Fluid Pitch & Tuning
- Perceptual Control

2. **Instrument Profiles**
   - Orchestral Strings
   - Piano
   - Drawbar Organ
   - Drums
   - Guitar
3. **Effect Profiles**
   - Reverb
   - Delay
   - Chorus
   - Overdrive
   - Rotary Speaker
   - Parametric Equalizer

Property Exchange enables the devices to describe what they can do and how to do it. This specification uses JavaScript Object Notation (JSON) wrapped in System Exclusive (SysEx) messages for getting, setting, and subscribing to device properties. This allows for DAWs and controllers to retrieve and display correct information regarding a connected instrument, like manufacturer and model, list of patches, controller mappings (as previously mentioned regarding Profile Configuration), synthesis parameters and their values, etc. This affords a more straightforward configuration of detailed parameters and is OS agnostic.

Protocol Negotiation allows for an agreement between devices on which protocol to use, MIDI 1.0 or MIDI 2.0, and also which features of that protocol are going to be used. This negotiation will only happen if a two-way MIDI-CI session is established, meaning that if a MIDI 2.0 device tries to engage in a dialogue with an incompatible MIDI 1.0 device (unable to respond), a "fall back" mechanism is triggered. Then, the MIDI 1.0 protocol will be applied to ensure full backward compatibility.

Universal MIDI Packet (UMP)

The new Universal MIDI Packet format will be used for the transfer of both MIDI 1.0 and MIDI 2.0 protocol messages. These packets consist of one to four 32-bit words, depending on the type of message (ranging from a 32-bit message in one 32-bit Universal MIDI Packet, up to a 128-bit message in one 128-bit Universal MIDI Packet). The change away from the byte stream used in the 5-pin DIN transport from MIDI 1.0 makes the traditional MIDI connector unsuitable for the new UMP format. This means that the MIDI 1.0 Protocol is currently, and could remain, the only protocol supported over the 5-pin DIN transport. While there are different types of UMP messages of various lengths and internal structures, the most significant 4 bits of each UMP message contain the **Message Type**. Table A.1 shows information about the current list of Message Types. The new Jitter Reduction Timestamps are a new optional feature that the

Universal MIDI Packet (UMP) Message Types		
MT	**Packet Size**	**Description**
0x0	32 bits	Utility Messages
0x1	32 bits	System Real Time and System Common Messages (except SysEx)
0x2	32 bits	MIDI 1.0 Channel Voice Messages
0x3	64 bits	Data Messages (including SysEx)
0x4	64 bits	MIDI 2.0 Channel Voice Messages
0x5	128 bits	Data Messages
0x6	32 bits	Reserved (for future definition by MMA/AMEI)
0x7	32 bits	
0x8	64 bits	
0x9	64 bits	
0xA	64 bits	
0xB	96 bits	
0xC	96 bits	
0xD	128 bits	Flex Data Messages
0xE	128 bits	Reserved (for future definition by MMA/AMEI)
0xF	128 bits	UMP Data Messages

Table A.1: List of Message Types in the *Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol* specification (version 1.1.2).

devices can agree to use when setting up a MIDI-CI session. This will prepend a JR Timestamp Message to every UMP, both for the MIDI 1.0 and MIDI 2.0 protocol.

As we've seen in Table A.1, MIDI 1.0 Channel Voice Messages can easily be sent in a Universal MIDI Packet by using Message Type '2'. I will exemplify this by constructing a Note-On message of a B♭ above middle C (MIDI Note #70) with an Attack Velocity value of 96 on Channel 7. With the old byte stream of MIDI 1.0, this would've consisted of a sequence of 3 (8-bit) bytes, as seen in Table A.2. A MIDI 1.0 message is structured so that it starts with a **Status Byte**, usually followed by one or two **Data Bytes**. The Most Significant Bit (MSB) (left-most bit) of each byte is reserved for identifying which of the two possible byte types it is. If the MSB is '1', we're dealing with a Status Byte, whereas a '0' means it's a Data Byte. This is the reason for the low resolution of 128 discrete steps in MIDI 1.0, as only 7 bits are left in each Data Byte for transmitting values. The remainder of the Data Byte is structured so the next three bits decide the **Message Type**, and the least significant nibble (the right-most four bits) leaves 16 possible values for choosing the channel. In a Note-On Message, the first Data Byte sets the note number, while the second Data Byte sets the attack velocity. Table A.2 shows the traditional MIDI 1.0 Note-On Message structure.

MIDI 1.0 Note-On Message																							
status & channel								**byte 2**								**byte 3**							
	Type			**Channel**					**Note #**								**Attack Velocity**						
1	0	0	1	0	1	1	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0	0	0

Table A.2: Example of the architecture of a 3-byte stream of a traditional MIDI 1.0 Note-On Message, sending Note # 70 with an Attack Velocity value of 96 on Channel 7.

If we were to construct the same Note-On Message example with an UMP using Message Type '2' (MIDI 1.0 Channel Voice Message) sent to Group 5, we use the same three bytes as in the MIDI 1.0 message seen in Table A.2 and prepend them with a byte so that the message gets carried in a single 32-bit packet. In the added first byte of this packet, the most significant nibble (left-most 4 bits) designates the Message Type, and the least significant nibble (right-most 4 bits) sets the Group. The UMP are constructed so that every message starts with a Message Type set by the first 4 bits. The UMP MIDI 1.0 Channel Voice Message is shown in Table A.3

MIDI 1.0 Channel Voice Message in UMP																															
MT=2				group				status & channel								byte 3								byte 4							
0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0	0	0

Table A.3: Example of the same MIDI 1.0 Note-On Message as in Table A.2 carried in Group 5 of an Universal MIDI Packet.

In a MIDI 2.0 Channel Voice Message, we have a 64-bit packet available, offering an expanded resolution affording much more refined gestural expressivity. The first 4 bits set the Message Type, and the next 4 bits set the Group. The following 8 bits form a Status byte containing a 4-bit opcode followed by a 4-bit Channel number. So far, this is similar to the MIDI 2.0 Channel Voice Message. After this, we get 16 bits of Index before having 32 bits of Data containing parameter or property value(s). This structure is visualized in Table A.4.

MIDI 2.0 Channel Voice Message																															
MT=4				group				status								index															
0	1	0	0																												
data																															

Table A.4: The structure of a MIDI 2.0 Channel Voice Message.

In a MIDI 2.0 Note On Message, the 16 index bits are split into an 8-bit Note Number and an 8-bit Attribute Type. Notably, the first of the eight bits used for Note Number is reserved, making the number of available notes the same as in MIDI 1.0 (a total of 128 possible note values). The 8-bit Attribute Type affords up to 256 different types to choose from. In a Note-On Message, the following 32-bit Data word gets divided into a 16-bit Velocity field (increasing the resolution from MIDI 1.0s mere 128 steps up to 65536 steps) and a 16-bit Attribute Data field. The structure of a MIDI 2.0 Note-One Message can be seen in Table A.5. Note that in MIDI 2.0, as opposed to in MIDI 1.0, an (Attack) Velocity value of 0 will not be interpreted as a Note-Off. It will be changed to a Velocity value of '1' instead. The Attribute Type and Data fields can be used for articulation, e.g., of a bow striking a stringed instrument or tuning details. Currently, there are four defined Attribute Types, whereas the rest of the values are reserved for future definitions:

1. **No Attribute Data**
2. **Manufacturer Specific**
   - The manufacturer decides the interpretation of the Attribute Data.
3. **Profile Specific**
   - The MIDI-CI Profile decides the interpretation of the Attribute Data.
   - If the device doesn't understand the active Profile, then the Attribute Data will be ignored.
4. **Pitch 7.9**
   - With this Attribute Type, the Note Number is treated as a Note Index without implying any scale or pitch. The 16-bit Attribute Data section is split into a 7-bit value representing a Semitone and a 9-bit value representing the fraction of a Semitone.

- The specification [7, p. 59] describes it further in the following words: *"Pitch is a Q7.9 fixed-point unsigned integer that specifies a pitch in* [100-Cent Unit] *HCUs."*
- This Attribute Type opens up for using microtonalities outside of the equal tempered tuning system and fine control over the pitch of each note when performing on an expressive multi-touch instrument.

MIDI 2.0 Note-On Message																															
MT=4				group				opcode				channel				r	note number							attribute type							
0	1	0	0					1	0	0	1					0															
velocity																attribute data															

Table A.5: The structure of a MIDI 2.0 Note-On Message.

I will not detail every message type in the MIDI 2.0 protocol. Still, I'll end this section by looking at other Message Types relevant to expressive multi-touch instruments. The next couple of message types seem to be an enhancement of some of the concepts introduced with the MPE Specification of MIDI 1.0, now an integral part of MIDI 2.0: **Registered Per-Note Controller** and **Assignable Per-Note Controller** Messages. These messages are accompanied by a MIDI 2.0 **Per-Note Management** Message to keep track of and enable independent control of Per-Note control of multiple notes sharing the same Note Number. There are 256 Registered and 256 Assignable Per-Note Controllers available (chosen by the 8-bit Index value of each message type), capable of sending 32 bits of data values. One of the more exciting options available here is Registered Per-Note Controller (RPNC) '3', named **Pitch 7.25**. As in the fourth Note-On Attribute Type, *Pitch 7.9*, this gives even greater control of the pitch of the individual notes when playing on an expressive multi-touch instrument. Where MPE have utilized 14-bit Pitch-Bend values (16384 discrete steps) for pitch-gliding between individual notes, with this RPNC you now have 32 bits distributed by having 7 bits representing Semitones (Pitch in HCUs), and 25 bits representing the fraction of a Semitone (fraction of a HCU). Note that you also have a dedicated MIDI 2.0 Per-Note Pitch Bend Message (as well as the channel-wide MIDI 2.0 Pitch-Bend Message) where the 32-bit data field is an unsigned bipolar value, where the range of the Per-Note Pitch Bend is dependent on the Sensitivity of Per-Note Pitch Bend. The expanded **MIDI 2.0 Control Change Message** has also been developed to carry 32-bit data values. However, it has the same 7-bit Index field as in MIDI 1.0 (128 different CCs types available). The last couple of Message Types I'll mention here is the MIDI 2.0 **Registered Controller (RPN)** and **Assignable Controller (NRPN)** Messages, which introduces 16384 Registered, as well as 16384 Assignable, Controllers. These are accessed through a structure of 128 Banks, each with 128 Controllers per Bank. MMA and AMEI define the specific functions of Registered Controllers, which translate and map directly to MIDI 1.0 Registered Parameter Numbers (RPN). The Assignable Controllers translate and map similarly to MIDI 1.0 Non-Registered Parameter Numbers (NRPN) but with all its available functions freely available to be chosen by the device manufacturer. Both these kinds of messages transmit a 32-bit data value.

A.2 The Current Status of MIDI 2.0

As of writing this thesis, the MIDI 2.0 specification is just starting to be implemented in hardware and software. Here is a non-exhaustive survey of the status of some popular DAWs, programming languages, and OSs, serving as a snapshot in time of the still relatively early days of MIDI 2.0 from when this thesis was written.

MIDI 2.0/MPE support in DAWs		
Name	Version	Supports
Ableton Live	11.3.20	MPE
Bespokesynth	1.2.1	MPE
Bitwig Studio	5.1.2	MPE
Cubase Pro	13.0.20	MPE
GarageBand (macOS)	10.4.10	None
GarageBand (iOS/iPadOS)	2.3.15	MPE
Logic Pro	$\geq$10.7.2	MIDI 2.0
Pro Tools	2023.12	None
Reaper	7.09	None

Table A.6: Support for MIDI 2.0 or MPE in select DAWs.

A.2.1 Digital Audio Workstations (DAWs)

Apple's Logic Pro X is still the only DAW supporting MIDI 2.0. The specification was implemented in version 10.7.2.

Although not yet supporting MIDI 2.0 in their own DAW — Cubase — Steinberg has provided support for making MIDI 2.0 compatible VST3 plug-ins in their VST SDK. This got added in the Steinberg VST 3.7 SDK.

MPE Support

Although not yet supporting MIDI 2.0, a few popular DAWs have official MPE support. I find it curious that Apple has added MPE support for GarageBand on their mobile devices (iPhones/iPads) but not on macOS[1]. Although Ableton has announced Live 12, there is no added MIDI 2.0 support yet, but the MPE support is expanding.

- Ableton Live
- Bitwig Studio
- Bespokesynth
- Cubase
- GarageBand (iOS/iPadOS only)

Support for MIDI 2.0 and MPE in select DAWs is shown in table A.6. Version number shows either from which version MIDI 2.0 got supported in the DAW or the latest version released as of when this was written if they're not yet supporting it.

A.2.2 Programming languages

Neither of the visual programming languages, Max/MSP/Jitter or Pure Data, have yet to get support for MIDI 2.0. MPE is supported natively in Max/MSP/Jitter, while Pure data can get support through using the *pd_mpe*[2] external.

[1]On the ROLI support pages, GarageBand for macOS is listed as MPE compatible. However, there is no reference to MPE in the GarageBand User Guide for macOS, while there is in the iOS and iPadOS User Guides. https://support.roli.com/support/solutions/articles/36000037202-compatible-synths-daws-and-instruments (accessed November 15th, 2023)

[2]https://github.com/DanielRudrich/pd_mpe (accessed October 24th, 2023)

Other general-purpose programming languages are starting to get support for MIDI 2.0 through external libraries, e.g. libremidi[3] and AM_MIDI2.0Lib[4] for C++, py-midi2[5] for Python, and MIDIKit[6] for Swift. The AM_MIDI2.0Lib C++ library is also meant to run on embedded devices, which could be useful in making Interactive Music Systems (IMS) utilizing MIDI 2.0.

A.2.3 Operative Systems (OSs)

Apple, Google, and Microsoft are all part of the MIDI Association, and a few Operative Systems have been supporting MIDI 2.0 for a while already.

Apple

Apple is supporting MIDI 2.0 on their personal computers, phones, and tablets. This support was introduced in macOS 11 (Big Sur), iOS 14, and iPadOS 14 respectively.

Google

Google made MIDI 2.0 accessible on mobile devices running on Android from version 13.

Microsoft

Microsoft hasn't yet implemented MIDI 2.0 into their OS — Windows. But since the MIDI 2.0 specification got updated in June 2023, they opened up their *Windows MIDI Services* GitHub repository[7] to the public as an open source project.

Linux

MIDI 2.0 got implemented in the Linux kernel version 6.5 through the Advanced Linux Sound Architecture (ALSA) drivers version 1.2.10.

A.2.4 Hardware

There are yet to be announced any MIDI 2.0 instruments, except the A-88 MkII by Roland. This is a controller, or master, keyboard without any internal sounds, which is marketed as having MIDI 2.0 compatibility.

A.2.5 Prototyping Tools

A few prototyping and development tools, both hardware and software, are made available for the MIDI Association members. These are currently available for corporate members only, not individual members.

Hardware

AmeNote has made a MIDI 2.0 prototyping tool called *ProtoZOA*[8]. Co-founder of AmeNote, Mike Kent, is also the Chairman of the MIDI 2.0 Working Group.

[3]https://github.com/jcelerier/libremidi (accessed November 7th, 2023)
[4]https://github.com/midi2-dev/AM_MIDI2.0Lib (accessed November 7th, 2023)
[5]https://github.com/jshjulian/py-midi2 (accessed October 24th, 2023)
[6]https://github.com/orchetect/MIDIKit (accessed November 7th, 2023)
[7]https://github.com/microsoft/midi (accessed October 24th, 2023)
[8]https://amenote.com/?page_id=74 (accessed October 16th, 2023)

Software

Yamaha Corporation has developed the MIDI Workbench software for testing and debugging of MIDI 2.0.

Appendix B

More on ZIPI

More information on ZIPI 2.0 is provided in this appendix.

To improve upon the familiar setup of Keyboard → MIDI → Sampler/Synthesizer module, Zeta Music and CNMAT worked on new elements for each component of this architecture [31, p. 49], which consisted of:

the Infinity Box: The first ZIPI controller, replacing the MIDI keyboard, which they describe as "a gesture-guided pitch-and-timbre-to-ZIPI converter".

The ZIPI network and its specification

Frisco: A synthesizer sound engine that can interpret messages from the Infinity box through the ZIPI network, described as: "An additive resynthesis engine with a control structure that will respond to ZIPI Music Parameter Description Language (MPDL)"

How does ZIPI compare to MIDI, and in which way does it improve upon performing expressive gestural control of synthesizers? [51] provides us with nine sections, each addressing different shortcomings with MIDI in which ZIPI aim to provide a better solution:

1. **Real Networks** MIDI has only one-way communication and needs one cable for each plug when connecting the '*IN*,' '*OUT*,' and '*THRU*' plugs to other devices. ZIPI is acting more like a computer network in that a ZIPI device has only one ZIPI plug, which could be connected to a hub enabling two-way communication between all connected devices.

2. **Bandwith and Efficiency** MIDI has a bandwidth of 31.25 kBaud used with only 80% efficiency, as it uses 10-bit bytes where a prepended 'start' bit and an appended 'stop' bit encapsulates the eight actual data bits. This works well for performing piano pieces over a MIDI keyboard but starts to lag when using several continuous controllers with high sampling rates. ZIPIs data rate is variable with no upper limit but a minimum of 250 kBaud. It will choose the fastest speed all connected devices can manage when multiple devices are connected. ZIPI is also reportedly using its bandwidth more efficiently [53].

3. **Flexibility in Message Addressing** Most MIDI messages address the whole channel, with the exception of *note-on*, *note-off*, and *polyphonic aftertouch*. ZIPI can send control and pitch messages to any note individually. MIDI cannot also play a whole chord or multiple notes simultaneously, which could sometimes result in temporal "smearing" of note onsets in a chord as discussed in [38]. With ZIPI, one can send messages to groups of notes in addition to single notes for instantaneous effect.

4. **Address Space** In MIDI, note addresses are inseparable from their pitch (7-bit MIDI Note Number), so even if an onset 'B♭' (MIDI Note #58) note is pitch-bended up a major sixth to a 'G' above (which is MIDI Note #67), it will require a note-off message addressed to the origin 'B♭' to stop. This could also result in unpredictable behavior if several notes of the same pitch are started, addressed with polyphonic aftertouch, and attempted to stop. ZIPI separates the pitch from the note address, where the address is just an arbitrary number. To control an onset note, you address it by its note number independently of its pitch.

5. **Distinguishing between Controller and Synthesizer Messages** You cannot directly set the pitch of a note in MIDI without circumventing this by sending a note-on message close in pitch and sending the position of the pitch wheel to bend it to the desired pitch. [51] argues that this describes a keyboardist's hand gestures more than being a pitch measurement. And the '*velocity*' term, which describes how fast a key has been pressed, is implied to mean *amplitude* or *loudness* with MIDI. ZIPI messages use 'pitch' and 'loudness' to describe the sound about to be produced, instead of 'key number' and 'velocity,' which describe the musician's produced gestures. ZIPI then has a separate second set of parameters for describing musicians' gestures.

6. **Controlling Drum Machines** Drum sounds with MIDI are usually mapped out with one fixed drum sound per key, which, because of the MIDI note address being the same as its pitch, limits the way individual drum sounds can be pitched and controlled. Say you want the provided snare drum to sound brighter a fifth above. With ZIPIs address space, it's easy to address the desired drum sound with separate information for pitch, pan, etc.

7. **Data Resolution** MIDI is mainly limited to 7-bit information because of the Most Significant Bit (MSB) of the byte being reserved as a status byte. ZIPI use any number of 8-bit data bytes for sending its messages with no per-byte overhead bits. And where MIDI uses a mere 4 bits for encoding channels, ZIPI uses 20-bit addresses. This is 1,048,576 possible addresses compared to MIDIs 16 channels. Even if MIDI potentially could have 2,048 individual notes using all of the 128 note numbers on all 16 channels, this is significantly less.

8. **High-Level Parameter Control** MIDI is limited and inflexible in its message architecture concerning high-level control of parameters. To turn down the main volume of a multi-timbral synthesizer, you have to send continuous CC volume data to each channel individually. In ZIPI, it is possible to address groups of instruments when sending control messages, not just an instrument individually, as well as sending messages to all groups at once. You can also use functions to modulate parameters, e.g., to make exponential changes to a parameter over time. This would've also had to be done by calculating and sending a stream of continuous CC messages with MIDI.

9. **Support for Pitch Trackers** Pitch trackers have been popular with, for example, MIDI guitar controllers using pitch trackers. With a MIDI bass guitar controller, this could present problems because you need one cycle of a signal to detect its pitch, and the MIDI doesn't support note onset without providing pitch information. This is exemplified in [51] with the lowest note of a five-string bass guitar, if tuned to the 'B' a semitone and three octaves below middle 'C.' This 'B' is 30.9 Hz, which use $\approx$32 ms for one cycle. To wait an excess of 30 ms before the note starts would've been too high of a latency easily detectable by the ear, three times as much as the accepted 10 ms limit [50]. However, the authors claim that the ear is forgiving with the content of the attack portion of a note onset. This means that even for a 30+ ms note onset,

one could've accepted noise or a wrong pitch while waiting for the correct frequency. With ZIPI, correcting the pitch after a note is articulated and adjusting the balance between its noise and pitch portions is easy.

Appendix C

More on OSC

More information on OSC is provided in this appendix.

OSC packets are sent to one or more destinations through an URL-style addressing scheme not limited by a pre-defined set of destination channels or CC destinations of fixed bit-lengths, as in MIDI 1.0. It is possible to use a pattern-matching syntax in the style of regular expressions [15] when specifying the destination(s) of messages. OSC provides high-resolution 64-bit time tags similar to the timestamps used in the Internet Network Time Protocol (NTP). Here, the first 32 bits represent the number of seconds since midnight on January 1st of 1900, and the last 32 bits represent the fraction of a second in units of around 200 picosecond (ps). There are two types of **OSC Packets**: An **OSC Message** and an **OSC Bundle**. By this design, OSC messages can be grouped in OSC bundles when multiple messages' effects should co-occur. Such bundles can be nested by containing other bundles. Some of the significant advantages of OSC over MIDI 1.0 are the open-ended addressing scheme, increased data resolution, and aspects of bidirectional communication. When pattern-matching is used in the destination address, every matching **OSC Address** will receive the message sent to the specified **OSC Address Pattern**. The sender of an OSC Packet is called the **OSC Client**, while the receiver is called the **OSC Server**. The **OSC Address Space** is built as a dynamic tree structure in which the **OSC Methods** (destinations) are the leaves at the end of the branches named **OSC Containers**, in which the Address Space is the total of all Containers and Methods. The structure and organizing of a OSC Address Space is freely customizable and is defined within each system. An example showing this addressing convention could be the following OSC address: *"/voice3/cutoff/frequency"*. An OSC Message is built up of the following three parts:

1. An **OSC Address Pattern** beginning with the '/' (trailing slash) character.
2. An **OSC Type Tag String** beginning with the ',' (comma) character.
3. zero or more **OSC Arguments**.

The original OSC v1.0 Specifitaion [54] include four required OSC data types, as well as 11 optional argument types plus '[' and ']' for indicating the beginning and the end of an array of data types. These types are referenced in an OSC Message In version 1.1 [14], several optional types were moved up to being required. The article also mentions several new optional types omitted in the text, referring to an update of the specification that never got published. Table C.1 shows the required OSC argument types introduced in v1.0 and 1.1, with their type tags and a description of each. The data types, as well as the packet itself, used in OSC, are transmitted as a multiple of 32 bits, which means that, e.g., the 's' and 'b' data types get zero-padded with 0-3 additional null characters and 0-3 additional zero bytes respectively to make the total number a multiple of 32. As stated in [55], OSC messages can also be requests for information, an efficient yet powerful bidirectional mechanism provided with this protocol. In a response from the **Responder** to the *Receiver*, a time tag includes information about when the message was processed. Below is a list of the available requests:

Required OSC Type Tags		
Type Tag	**Version**	**Description**
i	1.0	**Integer**: A 32-bit two's complement big-endian integer.
f	1.0	**Float**: A 32-bit big-endian IEEE 754 floating point number.
s	1.0	**String**: An ASCII string.
b	1.0	**Blob**: A byte array consisting of an int32 specifying the size, followed by that amount of 8-bit bytes of binary data.
T	1.1	**True**: Takes no arguments.
F	1.1	**False**: Takes no arguments.
N	1.1	**Null** (aka None, nil, etc.): Takes none arguments.
I	1.1	**Impulse** (aka "bang"): Used for triggering events. Takes no arguments. A renaming of 'Infinitum' from the v1.0 optional argument type.
t	1.1	**Timetag**: A 64-bit big-endian fixed-point OSC time tag in the NTP format.

Table C.1: List of required OSC Type Tags.

Exploring the Address Space: If a message's address ends with a '/' (trailing slash) character, the responder sends a list of all addresses found underneath the node in question.

Message Type Signatures: If a message's address ends with "[...]/type-signature", the responder sends a list of which types of arguments the destination is expecting.

Requests For Documentation: If a message's address ends with "[...]/documentation", the responder returns a string providing either the URL of a webpage with documentation of the OSC Method addressed, or the relevant documentation in the string itself.

Parameter Value Queries: If a message's address ends with "[...]/current-value", the responder returns the current value of the OSC Method in question.

Although [55] and [54] state that OSC is a protocol, the article [14] introducing version 1.1 counters by stating that its more fitting to see OSC as a *content format*. This view compares it to formats such as JSON, XML, and MusicXML. For clarity, I will refer to OSC as a protocol in this thesis.

OSC data can be carried across different transports and is not designed for one or the other. However, [55] reveals that bandwidth in the 10 megabit/sec or above is expected. This is at least 300 times faster than MIDI 1.0s bandwidth of 31.25 kilobit/sec. OSC data is expected to be delivered in packets, as MIDI 2.0s UMP, instead of the stream of bytes used in MIDI 1.0. The data encoded consist of 64-bit or 32-bit quantities.

In addition to additional type tags, version 1.1 adds a *path-traversing wildcard* operator '//' (double trailing slashes) from the XML Path Language (XPath), which makes the pattern-matching able to find OSC Methods across '/' boundaries of OSC Containers. For example, '//lfo-rate' would match with every OSC Method named 'lfo-rate' across the whole OSC Address Space at any depth. There was also an added framing mechanism for sending OSC messages over serial byte streams and through stream-oriented protocols, greatly expanding the available hardware transports and protocols to send OSC packets. A few OSC stream meta-data keys were also added.

[56] discusses the benefits of using OSC to organize real-time music software presented in a section of its own, which include polyphony management and using OSC between input data streams from gestural sensors or controllers (as the TouchKeys do) and signal processing or sound synthesis engines. Both subsections touch on relevant concepts for designing expressive multi-touch instruments when using the OSC protocol. The article ends with a small section

about ideas for the future of OSC, where the authors express a wish that OSC via TCP should be more supported and that the query system should be expanded and standardized by the community.

The NIME Proceedings article about version 1.1 from 2009 [14] provides new insight into the state of OSC six years later. The authors discuss how OSC often is referred to as a *standard*. However, it hasn't been ratified by any professional organization, nor have any resources been made available to carry OSC forward as such. There are some promising visions of the future of OSC and the road to version 2.0 through collaboration by the community, of which any concrete results unfortunately have yet to happen. At the end of the article, a new name is proposed, as the OSC protocol has grown out of only being used for sound applications: *Open Systems Control*.

Appendix D

More on Sound Synthesis

This appendix provides more information on digital sound synthesis techniques and the synthesizer modules present in my P-6 instrument design.

D.1 Digital Sound Synthesis Techniques

Here, I will briefly review some essential techniques in digital sound synthesis based on [44]:

Sampling Synthesis: Commonly just called *sampling*, this technique is in its purest form about using recorded audio samples and playing them back in desired pitches. This is widely used in stage pianos with patches of sampled real instruments and drum machines. Of the early samplers, we have the *Chamberline* and *Mellotrone*, which were keyboard instruments with tapes that played back when you pressed the keys.

Additive Synthesis: This technique combines simple waveforms, often generated by sinusoidal oscillators, into more complex waveforms. Additive synthesis is one of the oldest techniques, and its concepts are applied in centuries-old pipe organs. Of its appearance in electric instruments, we have the Telharmonium and the Hammond organs

Multiple Wavetable Synthesis: Roads [44] distinguish this from *fixed-waveform synthesis*, a way for digital oscillators to generate specific waveforms by scanning through a wavetable of the given waveform. In *wavetable crossfading*, two or more wavetables are played one after each other with crossfades in between. This synthesis technique has been marketed under different names by the big synthesizer manufacturers, such as *compound synthesis*, *vector synthesis*, and *Linear Arithmetic (L/A) synthesis*. *Wavestacking* is another type of multiple wavetable synthesis, similar to additive synthesis but using several more or less complex wavetables instead of just sine waves.

Granular Synthesis: This technique uses sound *grains* for building sound events. These grains could typically be in the 1–100 ms range. Here, we use tiny particles of sound instead of longer sound waves to synthesize sounds and build soundscapes. There is a wide range of synthesis techniques under the granular umbrella, where the use of sound grains is the common factor. short-time Fourier transform (STFT) also uses sound grains for the window function.

Subtractive Synthesis: Shaping sounds using *filters* is essential in analog subtractive synthesizers. Filters mainly *subtract*, or attenuate, frequencies from the sound spectrum, but some might also boost the area around the filter's cutoff frequency through increasing the *resonance*, or the *Q*, of the filter. When using rich sound sources such as a sawtooth waveform, you can use subtractive synthesis to sculpt the sound to various timbres. Envelope generators controlling an Voltage-Controlled Amplifier (VCA) and the filter, modulating the amplitude volume and filter cutoff frequency, are also influential in subtracting and shaping the sound temporally. Another temporal modulation source, the Low-Frequency Oscillator (LFO), could also be routed to the VCA or filter for tremolo or cycling filter-sweep effects. Iconic subtractive synthesizers include Moog's MiniMoog Model D and Sequential Circuits' Prophet-5.

Frequency Modulation (FM) Synthesis: There are many *modulation-based* synthesis methods, including *Ring Modulation (RM)*, *Amplitude Modulation (AM)*, *Phase Modulation (PM)*, and *Frequency Modulation (FM)*. Of these, FM is a digital synthesis technique that became popular with the public after the release of the Yamaha DX7. FM synthesis uses a signal (*the modulator*) to modulate another signal (*the carrier*). When a carrier oscillator gets modulated in frequency by a modulator oscillator in the audio range, this produces audible *sidebands* (modulation products), which get added to the carrier audio spectrum.

Physical Modeling Synthesis: This technique aims to emulate and model the acoustics of traditional instruments and physical objects. A reasonably simple, yet widely popular, example of Physical Modeling (PhM) is *Karplus-Strong* synthesis [27]. This method is capable of modeling plucked-string instruments, as well as timbres resembling drum instruments. The two most fundamental parts of PhM synthesis are the *exciter* and *resonator*. Here, the exciter acts like a trigger analogous to exciting a string with a bow or a drumskin with a stick, where the resonator affects the response of the instrument body. Other parameters often found in PhM are *decay* and *damping*.

Subtractive synthesis, used in my instrument, is associated with **East Coast** synthesis, a tradition Robert Moog pioneered in the 1960s. The Minimoog Model D is a classic example of subtractive synthesis in the East Coast tradition. The approach of **West Coast** synthesis, on the other hand, was pioneered by Don Buchla in the 1960s and could be considered to be more experimental with the use of complex oscillators, wave folding, and Low Pass Gate (LPG), among others. An icon of Buchla's design is the Buchla Music Easel.

D.2 P-6 Synthesizer Modules

Below is a brief description of the synthesizer modules used in my instrument design.

Oscillator 1 The Oscillator 1 generates sound through oscillators (digital VCOs) under the hood. The module provides control of the oscillator frequency in steps of $\pm$ 48 semitones. It has a button enabling hard sync with Oscillator 2, where Oscillator 1 is controlled. There is a shape knob that crossfades between triangle, sawtooth, and pulse waveshapes. Pulse width affects the pulse shape. Oscillator 1 also provides a separate triangle sub-octave oscillator accessible from the Mixer module.

Oscillator 2 The Oscillator 2 module provides everything from Oscillator 1 except for the sync toggle and sub-octave. Additionally, it has control for fine-tuning its oscillator frequency to detune it with Oscillator 1. It also has a button for making it into an Low-Frequency Oscillator (LFO) and one for turning the keyboard tracking off, making the oscillator frequency independent of the played note's pitch.

Mixer This module provides individual volume control of oscillators 1, 2, the sub-octave, and a white-noise generator.

High-Pass Filter This is a two-pole, 12dB per octave resonant high-pass filter, with knobs for the cutoff frequency and resonance. There is bipolar control of the modulation from the Filter Envelope, with a button enabling velocity modulation of the envelope amount. The keyboard tracking has three states: Off/Half/Full. This affects the cutoff frequency relative to the pitch of the note being played.

Low-Pass Filter This is a four-pole, 24dB per octave resonant low-pass filter, otherwise with similar control as the High-Pass Filter.

Filter Envelope This module provides control of the Attack, Decay, Sustain, and Release (ADSR) stages of an envelope generator modulating the filters.

Amplifier Envelope This module similarly provides control of the ADSR stages of the envelope generator modulating the output volume (digital VCA) of the synthesizer voice. There is also control for the modulation amount from the Amplifier Envelope, affecting the maximum volume, with a button enabling velocity modulation of the envelope amount.

Appendix E

Design & Prototyping Diary

A diary logging the steps in the iterative instrument design and implementation process in Pure data.

E.1 Oct/Nov 2023 - Initial design phase

- Started the process of designing the synthesizer in Pd, inspired by the Sequential Prophet-6
- Chose which modules to implement and which to leave out
- Begun thinking about how to implement features in Pd
- Prototyped abstractions of modules from the Prophet 6 with GUIs:
    - oscillator_1
    - oscillator_2
    - mixer
    - low-pass filter
    - high-pass filter
    - filter_envelope
    - amplifier_envelope
- Oscillator 1 + 2 makes sound with features resembling the Prophet 6

E.2 11/12/2023

- Started a p6_voice.pd sketch, making a GUI with the module abstractions
- Reworked Oscillator 1 + 2
    - Added Graph-On-Parent GUI feature
    - Reworked the logic and input rescaling of parameters
    - Added a semi-global "bit" selector for future use with MIDI 2.0 or testing/comparing with 14-bit CC or OSC
        - Reworked scaling objects to receive "bit" variable input
- Tested with a Akai LPK25 MIDI keyboard
    - Mockup MPE through the TouchKeys software works with Pd
    - We have sound in the oscillator 1 + 2 modules.

Reflections and considerations

The p6_voice.pd sketch is acting both as the start of a "voice card" and also the overall synthesizer panel. At some point I have to separate those parts, by making a voice card abstraction to be cloned for each of the six voices and the overall GUI panel. I also have to consider which module is part of the voice card, and which is global. Maybe I should keep the Graph-On-Parent abstractions for the GUI, and move or abstract the logic inside to a voice card implementation.

E.3 12/12/2023

- Started working on the GUI layout and send/receive architecture of signals in the p6_voice.pd sketch

E.3.1 Iteration v0.1 – Oscillators

- p6_voice plays a single voice (monophonic) with the following modules:
  - oscillator_1
  - oscillator_2
- Regular MIDI 'notein' changing the pitch of the oscillators only. No noteoff yet.
- Moved 'bit' selector from individual modules to global p6_voice
- Exchanged else/buttons (used for aesthetic purposes) with vanilla toggles, because of initialization issues.
- Reworked and added Mixer module
- Moved Sub-Oscillator to Oscillator 1 module for simplicity
- Added volume adjustment to mixer (sub-oct too low, noise too high)
- Swapped else/mix4~ with vanilla [*~] since linear volume control was preferred
- Reworked Low-Pass Filter module
  - Implemented bipolar Envelope amount (Range +/- 48 semitones)
    - Implemented Velocity modulation of Envelope amount
  - Implemented Keyboard Tracking (Off/Half/Full) with C4 as center frequency
- Added Low-Pass Filter to p6_voice
- Undid the volume adjustments to mixer after low-pass filter was added. The energy was in the (now tamed) high frequencies.
  - It seems that the preset output volume appear balanced when low-pass filtering is added, except when "open".

Reflections and considerations

I need to make a block diagram of the synthesizer design. This could be beneficial to me in the design process as well. At a later stage, a diagram is needed for the complete instrument as well. Low Resonance takes the energy out of the sound. Look into that. The externals part of the "Pure Data: Electronic Music and Sound Design" book might address and/or solve this.

E.4 13/12/2023

- Added color-coding to inlets (yellow), receives (green) and outlets (purple) in p6_voice.pd
- Removed "hard-wired" inlets to modules, the connection between modules, which should be constant.
  - Only inlets controlling parameters (knobs/buttons) are now used.
- ** Found a bug with the Env_amount range (+/- 4 oct), which fades out the sound(?!) if every knob is at max.
  - Temporarily fix på reducing the range to +/- 3 oct
- Reworked and added the Amplifier Envelope module
  - Implemented Env Amount and Velocity modulation
  - A/D/R with 1-20000 ms range

E.4.1 Iteration v0.2 – Basic Single Voice

- p6_voice plays a single voice (monophonic) with the following modules:
  - oscillator_1
  - oscillator_2
  - mixer
  - low-pass filter (mo filter_envelope modulation yet)
  - amplifier_envelope
- Regular MIDI 'notein' with 'noteoff' through amp envelope.
- Changed the filter-typ to [else/svfilter~] for better resonance response.
  - 2x 2-pole in series
  - Implementation of Chamberlin's state-variable filter algorithm
- Removed the resonance control of the second filter, it should only control the first in the series.
- Reworked and added Filter Envelope module
  - A/D/R with 1-20000 ms range
- Updated [pd reference] of every module
- Reworked the cutoff logic in the Low-Pass Filter module
- Implemented Filter Envelope modulation to Low-Pass Filter module

E.4.2 Iteration v0.3 – Single Voice

- p6_voice plays a single voice (monophonic) with the following modules:
  - oscillator_1
  - oscillator_2
  - mixer
  - low-pass filter
  - filter_envelope (updated filter frequency logic and added envelope modulation)
  - amplifier_envelope
- Regular MIDI 'notein' with 'noteoff' through amp envelope.

Reflections and considerations

There is pops and glitches in sound if turning knobs when note is playing (specifically amp env amount). Consider adding lines to smooth parameter changes (slew/lag). *With iteration v0.3, the synthesizer finally starts to feel expressive, although it is still monophonic, and played with a small MIDI keyboard.* The next step is to add polyphony and MPE support.

E.5 14/12/2023

- Added Master Volume module
- Shelved an instance in a 'MIDI' folder, to mark the transition to MPE
- Implemented MPE through the pd_mpe external (still monophonic patch)
  - Reworked and renamed the [mpeTest] abstraction from *pd_mpe* to [mpe_voice]
    - Added copyright notice
  - Added sends for each parameter
- Added pitch-bend support, with global PB range selector
- Added [glide2pitchbend] abstraction for glide control of frequency
- Tested 'slide' and 'press' (aftertouch) modulation
  - Slide: [-64, 63]
  - Press: [0, 127]
- Added preliminary modulation input for lpf cutoff frequency (to not override knob, but modulate relative to its position)
  - This has to be reworked for every parameter to be modulated

E.5.1 Iteration v0.4 – MPE

- p6_voice is MPE-enabled and plays a single voice (monophonic) with the following modules:
    - oscillator_1
    - oscillator_2
    - mixer
    - low-pass filter
    - filter_envelope
    - amplifier_envelope
    - master_volume
- Tested with a ROLI Seaboard Block
- Glide simulates pitch-bend
- Slide modulates Osc1 Pulse Width
    - Overrides knob
- Press modulates lpf cutoff frequency
    - Modulates relative to knob (not really, but relative to the modules inlet)

Reflections and considerations

I need to implement MPE objects into my synthesizer patch, as in map the individual MPE voices' output to my synthesizer parameters. First up is the 'glide', analogous to pitch-bend, which is a core functionality in MPE. The 'noteoff' architecture needs to adapt to the MPE object as well.

Pitch-bend control is added to make use of the MPE 'glide' parameter. To enable for 'glide' modulation of other parameters as well, a [glide2pitchbend] abstraction was added. The other MPE control parameters 'slide' and 'press' (aftertouch) is also tested. Cutoff modulation works fine, Pulse Width modulation might have som artifacts. Still considering a [line] object to smooth parameter changes.

NB! The latest pd_mpe git activity was committed on Apr 29, 2019. The current build is for win/mac 'i386;x86_64' CPU architectures. This build does not support the new Apple M1/M2/M3 chips, which is in the UiO MCT Synthroom. Considering making a build for the 'arm64e' CPU architecture.

E.6 19/12/2023

- Bugfix: Connecting [r~ osc2] to sync input of Oscillator 1
- Bugfix: MPE glide / pb_range typo
- Restructuring Oscillator 1 + Oscillator 2

E.6.1 Iteration v0.4.1 – MPE (+ sync)

- Bugfix to Iteration v0.4
    - Sync input of Oscillator 1 is now connected to [r~ osc2]
- Restructured the Oscillator 1 + 2 modules with abstractions per function

- Started the process of separating the GUI from the logic of modules
  - Added a _GUI folder containing the GUI patches with the logic removed.
  - The old patches, is now without GUI and graph-on-parent, with just the logic.
  - The input scaling is happening in the GUI patch.
  - The GUI module patches are "global", on the layout of the main synthesizer patch.
  - The logic module patches are "local" per voice, to be called in each clone of the voicecard patch.
- Reworked Oscillator 1 (separated GUI + logic)
- Started with a [pd voicecard] subpatch, which later will be saved as a separate file.
- Reworked Oscillator 2 (separated GUI + logic)
- Reworked Mixer (separated GUI + logic)

Reflections and considerations

There has been a bug in the previous iterations, where the [r~ osc2] inlet was not connected to the sync-input of the oscillator on Oscillator 1. Because the Oscillator 2 is halfway up in the mix by default, I didn't notice the bug. Considered fixing for iteration 0.3 and 0.4, but decided on not to temper with previous iterations after they were finished. The bug will be fixed in iteration 0.5. - Should Oscillator 2 level attenuate the signal to Oscillator 1 when in Sync mode?

E.7 20/12/2023

- Reworked Low-Pass Filter (separated GUI + logic)
- Reworked Filter Envelope (separated GUI + logic)
- Reworked Amplifier Envelope (separated GUI + logic)
- Because of the local $0-prefix to send/receive objects, I have to incorporate the modules logic to the voicecard abstraction.
- Reworked the voicecard abstraction to not call the synthesizer module logic abstractions, but copy/pasted this logic into subpatches instead.
- Added polyphonic support to the synthesizer!

E.7.1 Iteration v0.5 – Polyphony (global modulation)

- The synthesizer now works with polyphonic note input
- Every voice currently modulates every other voice (except pitch glide)
  - Polyphonic synthesizer with "monophonic modulation"
  - The modulation is currently done in the GUI module abstractions, and needs to be moved to the logic subpatches of the voicecard abstraction.

Reflections and considerations

The synthesizer is now polyphonic, which brings up another consideration regarding the modulation. I've thought about that it is not necessary to have visual feedback of the knob of a parameters modulation. But now I see that it is not wanted, as the modulation needs to be individual per voice. *There might be wish for global modulation through a future LFO, but that will be reflected upon at a later stage.* For now, I have to move the modulation of parameters to the synthesizer module logic inside the voicecard.pd abstraction. This also means that there will be need for a separate scaling of the modulation input range, as the scaling is now happening in the GUI patch to make the knobs work in higher resolution than the MIDI 7-bit range. (This was already an conscious choice in making the synthesizer more future-proof and compatible with higher resolution MIDI 2.0, OSC or 14-bit CC.) The scaled modulation input signal will be added/subtracted to the "knob" control signal. There should possibly also be a "depth" control per modulation source or destination.

Considering making a matrix for modulation routing!

E.8 21/12/2023

- Moved the modulation logic from the main patch into the voicecard.pd abstraction to make the modulation polyphonic (individual modulation per voice)
- Added a small modulation matrix
  - Sources: Slide / Press
  - Destinations: Off / Osc1 PW / LPF Cutoff
  - Added depth control for each modulation (source)

E.8.1 Iteration v1 – It's alive!

- A 6-voice polyphonic synthesizer with MPE and per-voice modulation!
- There is a small modulation matrix for flexible modulation routing between a few sources and destinations
- Synth modules:
  - oscillator_1
  - oscillator_2
  - mixer
  - low-pass filter
  - filter_envelope
  - amplifier_envelope

E.9 04/01/2024

Reflections and considerations

The first "official" v1-iteration is tailored to the ROLI Blocks. This explains the Slide/Press naming of modulation sources, but which easily can be interchanged with the y-axis control and surface area control parameters of the TouchKeys. *The naming convention should probably be changed in a later iteration, to more "neutral" controller-agnostic labels.*

The Modulation Matrix is great for testing individual parameters, both sources and destinations for modulation. It is also "performer-friendly" in that it is easy to route/map the modulation sources to a synthesizer parameter and set its depth. However, this negates the possibility of having one modulation source affect multiple destinations. *The modulation matrix affords One-To-One and Many-to-One mapping strategies, but not One-to-Many.* One solution to this could be to have two or more instances of each source (with individual depth-knobs). This could work with only a few modulation sources (or DoF of the instrument) in use, but I suspect it would become messy when adding more modulation sources. This limitation furthermore hinders the making of a higher-level intermediate abstract mapping layer. Such an intermediate layer would be beneficial for the designer–performer, enabling custom instrument iterations where a saved pd patch instance of the synthesizer could be the equivalent of a synthesizer sound patch/preset. Such an intermediate layer would be "messier" to implement, aesthetically speaking, than a transparent and clean modulation matrix, and should at least be placed in a dedicated sub-patch "environment". A modulation matrix is as mentioned a great tool when designing and prototyping new features. As for the performer-friendliness, I think this would make the instrument more available to a wide selection of performer–users, enabling instant, although limited, expressivity, by affording the possibility to set the modulation matrix to their individual preferences. It is also easy to test different ideas out, in a "one-dimensional" sense, with the restriction of using a source only once, while any destination parameter potentially could be used for every source in use.

E.10 08/01/2024

Reflections and considerations

The modulation logic needs to be changed so that the response is either unipolar or bipolar according to the design of the control source parameter. *As of now, the slide control modulation (which by default is routed to Oscillator 1 PW) is unipolar, i.e., only adding positive values to the modulation.* The slide parameter feels on the other hand to be inherently bipolar, centering around a middle value (64), with the possibility to slide values up or down. As of now, where the slide control only adds positive values to the PW modulation, this only affords a full modulation range if the Osc1 PW knob is set to 0. (Which in turn would "jump" the PW modulation to around the middle value for each time a note is pressed, if the depth is set to 100%.) Instead, the slide control parameter signal should be transformed to center around 0 [-64, 63], to be a bipolar modulation source. *The press modulation on the other hand (which by default "opens" up the low-pass filter by increasing the cutoff frequency), is inherently unipolar and feels right as it currently is set up.* The default value of a press is 0, and pressing harder (using the polyphonic aftertouch CC) adds to this value.

Care and consideration has to be taken in setting up the different control parameters and treating their data before routed to their modulation destination, either unipolar or bipolar. To implement this, I'm making sub-patches to transform the bipolar control sources from [0, 127] to [-64, 63]. The unipolar control sources are used as is, as adding the value of the knob controlling the destination synthesizer parameter with the control source value was already the design choice used. This addition will (when depth is set to 100%) in effect range from subtracting 50% to adding 50% for bipolar modulation, and range from adding 0% to 100% for unipolar modulation.

Regarding the actual range of the modulation, every synthesizer parameter value gets clipped to a certain range depending on the type of parameter. This means that, e.g., having a low-pass filter frequency already set to max before being modulated by the press source would not add more or otherwise affect the frequency. In a future bipolar modulation design, the Osc1 PW knob set to the minimum value before being modulated negatively by the slide source would similarly not affect the PW — only and upwards slide adding a positive value would.

I could consider adding the depth ranging from [-1, 1] instead of [0, 1] to make the unipolar modulation be negative as well. But on the other hand, this negative-to-positive modulation possibility is already a hard-coded design feature of the ENV AMOUNT ot the filter(s) and amplifier envelope. Maybe keeping the unipolar positive, and otherwise only letting the bipolar sources modulate "negatively" would more than suffice.

E.11 10/01/2024

- Shifted the values of the slide modulation source from [0, 127] to [-64, 63] to make it bipolar
- Expanded the modulation matrix destinations, now including:
  - [OFF]
  - Osc1 Shape (Implemented)
  - Osc1 PW (Implemented)
  - Osc2 Shape (Implemented)
  - Osc2 PW (Implemented)
  - Mixer Osc1 Level (Implemented)
  - Mixer Osc2 Level (Implemented)
  - Mixer SubOct Level (Implemented)
  - Mixer Noise Level (Implemented)
  - LPF Frequency (Implemented)
  - LPF Resonance
  - Filter Envelope Attack
  - Filter Envelope Decay
  - Filter Envelope Sustain
  - Filter Envelope Release
  - Amplifier Envelope Attack
  - Amplifier Envelope Decay
  - Amplifier Envelope Sustain
  - Amplifier Envelope Release
- Started implementing the modulation destination to the Press control source, according to the list above.

E.12 11/01/2024

- Finished implementing all modulation destinations to the Press control source
- Finalized the Slide control modulation
- Added the High-Pass Filter module
- Fixed temporal modulation (Envelope A/D/R) destinations by removing the logarithmic scaling of modulation signal
- Made the Glide control (hard-coded to pitch-bend) selectable as a small part of the Modulation Matrix: Off / Pitch-Bend
- Added Strike modulation source (Note-On Velocity)

- Added Lift modulation source (Note-Off Velocity)
- Added logic for individual CC input per voice for modulation
- Reworked the Glide to include Pitch Glide and Vibrato from the TouchKeys
  - Renamed old Glide to 'Pitch Glide/Vibrato'
- Added a new Glide control source (TouchKeys X-axis: CC 75)
- Added a new Area control source (TouchKeys Contact Area: CC 76)
- Added a new Distance control source (TouchKeys 2-Finger Distance: CC 77)

E.12.1 Iteration v2 – Modulation Matrix + HPF

- The synthesizer now has a Modulation Matrix with 7 sources and 20 destinations (+ Off)
  - There is also a toggle for Pitch Glide/Vibrato (hard-coded to pitch(-bend functionality))
- The synthesizer also has a High-Pass Filter module
- Bi-polar modulation control with the Slide and Glide control sources
- Minor bugfixes and improvements (details found in this diary)

Reflections and considerations

When modulating the (Filter) Envelope Attack, nothing happens. As change in the attack length will not affect the attack curve after a note has triggered. Is this because of the ADSR implementation in the [else/adsr~] object? When modulating the (Filter) Envelope Decay, there is some unexpected behavior including what sounds like re-triggering of envelope. This happens when instantly applying a lot of pressure (from the Press source) to make the decay time longer. If you loosen up the pressure (and probably further when you (involuntarily) press harder again) something akin to a re-trigger occurs, also making the decay curve of the envelope stuck in a fixed position – which sounds like it has arrived at a higher sustain level than what is set. This is somewhat also affected by the release value in some unknown way. Modulating the (Filter) Envelope Sustain works as expected, affecting the Cutoff Frequency (according to the set Env Amount). Even though the other temporal parameters (Attack/Decay) have their quirks, modulating the (Filter) Envelope Release works as expected. The same observations apply to the Amplifier Envelope, except: When modulating the (Amplifier) Envelope Sustain, there are artifacts introduced to the sound. This can be tamed by using the [vline] object, but needs a considerable length to be eliminated. An compromise of 150 ms is used.

Update: There was a considerable improvement when removing the linear to logarithmic scaling of the modulation signal. The Attack modulation is now working. Still, on the Decay modulation, there will be a sudden dip after the otherwise working modulated (prolonged) decay curve before landing on the sustain level. So even if there is a new decay curve, it will end with a dip down to the sustain level. (This dip might be in the curve of the Decay knob position.)

Modulating certain parameters introduces artifacts to the sound, which can be controlled with using a [vline] object (a slew/lag generator) to smooth out the changes. This however, will introduce some (fading) latency to the modulation or knob turn. If the slewing has to be long, this would affect the responsiveness of the modulation control. *Where is the threshold of balancing the audio quality with the perceived control intimacy?*

I might sometimes get «in the zone» when designing or implementing features in the synthesizer. I might intuitively get an idea of how to add or fix a feature, without necessarily having a conscious answer to why (just yet). If it doesn't work, I will then analyze why it did not and try to fix it. Or change the approach. Even if it works, I will find out why it did, to learn from it. This experience reminds me of being «in the zone» when performing and improvising. You don't necessarily think consciously about which scale or which rhythmical motifs to use. There is an intuitive approach made possible by all the hours of practice and experience in the craft. When this happens during the design time, I like to think that this is because my knowledge of and experience with synthesizers are elevating my more limited programming experience.

E.13 16/01/2024

- Tested the synthesizer with the TouchKeys in the MCT Synthroom
  - An external sound-card was needed, as the TouchKeys need a MIDI DIN-cable for merging sensor and MIDI data. The USB-connection was just for power.
  - The TouchKeys in "standalone" (triggering MIDI with the sensors) didn't work very well, possibly because of communication issues regarding MIDI note-off messages. (I don't think it sends velocity 0 messages as note-off, but rather actual note-off messages.)

E.14 18/01/2024

- Removed the [+ 64] 'ROLI offset' to the pitch glide control (pitch-bend), which was added to compensate for what seemed like a slight off-center pitch deviation with the ROLI Lightblock controller.

E.15 22/01/2024

- Added a _testing-folder and a patch for testing 7-bit versus 14-bit CC modulation resolution
  - The bit selector works for changing to 14-bit!
  - The logic for receiving 14-bit values through two CC channels could probably be improved (proof-of-concept)

Reflections and considerations

With the y-axis modulating "resolution-sensitive" parameters such as the LPF cutoff frequency, the 7-bit resolution sound 'steppy' and 'glitchy', while the 14-bit resolution sounds smooth. The cutoff and envelope modulation was turned all the way of, while the resonance was boosted high, to really hear the frequency response of the filter being modulated. However, there is some "re-triggering" issues when operating in 14-bit mode. The logic of combining two 7-bit CCs to one 14-bit value probably needs to be fine-tuned, but this works as a proof-of-concept and to compare the resolution.

E.16 23/01/2024

Reflections and considerations

My assumptions about the TouchKeys X-axis controlling the "Pitch Bend" mapping was wrong, it with its Y-axis. No major implications for the synthesizer, as the functionality works anyway. I have to readjust my mind though, between the glide (x-axis) of ROLI, and the Pitch Bend (y-axis) of TouchKeys. Although I think it feels much more natural to control pitch-bends sideways, I guess it makes sense since the TouchKeys keys get depressed as keyboards do.

E.17 24/01/2024

- Started with "The Knob"
  - A knob on the front GUI controls the amount of 'black box' modulation happening in the voicecard.pd abstraction
  - Adds a many-to-many mapping strategy
- Ideas:
  - "Backwards" [metro] controlling "sample & hold"-ish modulation of frequency
    - Random modulation of S&H modulating "frequency"
    - Frequency is the cutoff frequency of a low-pass filter using resonance (it's not modulating pitch per se)
    - "Backwards" More rapid metro changes when knob is low and less when it is set high
  - Using modulus of 127 when combining input parameters, instead of the clipping happening in the module logic
  - Use negative numbers as well, for decreasing parameter values
  - line with metro-modulated time between changes in randomized modulation

---

E.18 25/01/2024

- Added "The Knob" to the front panel GUI
- It has a [pd the_knob] sub-patch, providing 5 random numbers
  1. Random number 0-127 every 20 ms
  2. Random number 0-127 every 100 ms
  3. Random number 0-127 every 500 ms
  4. Random number 0-1000 every 2000-50 ms (2000, when knob is off down to 50 when knob is fully on)
  5. Random number 0-1000 every 50-2000 ms (50, when knob is off up to 2000 when knob is fully on)
- Made a template for setting up the_knob mappings (in the _tools folder)
- Added a [pd knob_modulation] sub-patch to the modulation area of the voice card abstraction
  - Set up a test-mapping with random elements and many-to-many mappings

E.18.1 Iteration v3 – The Knob

- The synthesizer now has a Knob for ’black box’ complex many-to-many mappings
- The knob ships with 5 random number generators, of different speeds
- A template abstraction for setting up mappings are included in the _tools folder
- A quick test with of a mapping idea acts as a proof-of-concept

Reflections and considerations

It didn’t work as expected with the S&H idea as is, but there is lots of potential here! At a later time, I have to fix the modulation matrix interfering (when turning off destinations).

Acronyms

ADC analog-to-digital converter. 9

ADSR Attack, Decay, Sustain, and Release. 10, 14, 29, 34, 67

ALSA Advanced Linux Sound Architecture. 55

AM Amplitude Modulation. 66

AMEI Association of Musical Electronics Industry. 12, 51, 53

API application programming interface. 19

ARP Alan Robert Pearlman. 3

ASCII American Standard Code for Information Interchange. 62

AU Audio Unit. 45

CC Control Change. 8, 9, 14, 15, 29, 30, 36, 46, 53, 58, 61

CEO Chief Executive Officer. 14

CNMAT Center for New Music and Audio Technologies. 16, 17, 57

CV Control Voltage. 10, 11

DAW Digital Audio Workstation. 45, 50, 53, 54

dB decibel. 66, 67

DIN Deutsches Institut für Normung. 50

DIY Do It Yourself. 17

DMI digital music instruments. 7, 19, 20

DoF degrees of freedom. 1, 7–10, 29, 37, 38

DSP digital signal processing. 10

ELSE EL Locus Solus' Externals. 34

EQ Equalizer. 50

EWI electronic wind instrument. 16

FM Frequency Modulation. 66

GM General MIDI. 12, 49

GUI Graphical user interface. 11, 17, 24, 27, 34–36, 38–40

HCI Human-Computer Interaction. 19, 21

HCU 100-Cent Unit. 53

Hz hertz. 28, 30, 58

IC Integrated Circuit. 9

IEEE Institute of Electrical and Electronics Engineers. 62

IMA International MIDI Association. 11

IMS Interactive Music Systems. 55

IRCAM Institut de Recherche et Coordination Acoustique/Musique. 11

JR Jitter Reduction. 15, 50, 51

JSON JavaScript Object Notation. 50, 62

kBd kilobaud. 13

L/A Linear Arithmetic. 65

LED light-emitting diode. 19, 34

LFO Low-Frequency Oscillator. 10, 29, 41, 66

LPF Low-Pass Filter. 10

LPG Low Pass Gate. 10, 66

LPK Laptop Performance Keyboard. 34

MCT Music, Communication and Technology. 4

MIDI Musical Instrument Digital Interface. i, 1, 2, 4, 5, 7–9, 11–17, 19, 21, 23–26, 28–31, 34–36, 41, 43–47, 49–58, 61, 62

MIDI-CI MIDI Capability Inquiry. 14, 15, 49–52

ML machine learning. 17

MMA MIDI Manufacturers Association. 12, 14, 17, 44, 51, 53

MPDL Music Parameter Description Language. 16, 57

MPE MIDI Polyphonic Expression. i, 1, 2, 4, 5, 7–9, 12, 14, 15, 29, 33, 34, 36, 41, 43–46, 49, 53, 54

ms milliseconds. 11, 13, 58, 65

MSB Most Significant Bit. 51, 58

MSP Max Signal Processing. 11, 17, 24

NAMM National Association of Music Merchants. 11, 14–16, 49

NIME New Interfaces for Musical Expression. 7, 9, 17, 63

NRPN Non-Registered Parameter Numbers. 53

NTP Network Time Protocol. 61, 62

OS Operative System. 50, 53, 55

OSC Open Sound Control. 5, 7–9, 16–18, 31, 34, 37, 43, 45, 61–63

OSF Open Science Framework. 5, 41

Pd Pure Data. i, 2, 4, 5, 7, 9, 11, 17, 19, 21, 24, 25, 31, 33–38, 41, 43, 45, 46

PhM Physical Modeling. 66

PM Phase Modulation. 66

ps picosecond. 61

RM Ring Modulation. 66

RPN Registered Parameter Numbers. 53

RPNC Registered Per-Note Controller. 53

SDK Software Development Kit. 45, 54

SMF Standard MIDI Files. 16

STFT short-time Fourier transform. 65

SysEx System Exclusive. 50, 51

TCP Transmission Control Protocol. 63

TMA The MIDI Association. 12, 15, 23, 45

UC Berkeley University of California, Berkeley. 16, 17

UMP Universal MIDI Packet. 15, 16, 46, 50–52, 62

URL Uniform Resource Locator. 61, 62

USB Universal Serial Bus. 34, 35, 40

USI Universal Synthesizer Interface. 12

VB Visual Basic. 24

VCA Voltage-Controlled Amplifier. 3, 10, 65–67

VCO Voltage-Controlled Oscillator. 3, 10, 66

VST Virtual Studio Technology. 45, 54

Wi-Fi Wireless Fidelity. 15

XML Extensible Markup Language. 62

XPath XML Path Language. 62

ZIPI Zeta Instrument Processor Interface. 5, 7, 16, 17, 43, 57–59

Bibliography

[1] International MIDI Association (IMA). *MIDI Musical Instrument Digital Interface Specification 1.0*. North Hollywood, 1983.

[2] The MIDI Association. *MIDI 2.0 Profiles Configuration and Property Exchange NAMM 2021 Session*. URL: https://www.youtube.com/watch?v=_NEhlOo1paU (visited on 11/15/2023).

[3] The MIDI Manufacturers Association. *MIDI 1.0 Detailed Specification*. Version 4.2.1. Los Angeles, CA, Feb. 1996. URL: https://midi.org/specifications/midi1-specifications/midi-1-0-core-specifications/midi-1-0-detailed-specification-2 (visited on 11/14/2023).

[4] The MIDI Manufacturers Association. *MIDI Manufacturers Association (MMA) Adopts New MIDI Polyphonic Expression (MPE) Enhancement to the MIDI Specification*. Press release. Los Angeles, CA, Jan. 2018. URL: https://www.midi.org/images/easyblog_articles/361/MPE-2018-NAMM-Show-Release.pdf (visited on 11/15/2023).

[5] The MIDI Manufacturers Association and Association of Musical Electronics Industry (AMEI). *Common Rules for MIDI-CI Profiles*. Version 1.1. June 2023. URL: https://www.midi.org/specifications/midi-2-0-specifications/midi2-core/common-rules-for-midi-ci-profiles (visited on 11/15/2023).

[6] The MIDI Manufacturers Association and Association of Musical Electronics Industry (AMEI). *MIDI Polyphonic Expression*. Version 1.1. Apr. 2022. URL: https://midi.org/specifications/midi1-specifications/mpe-midi-polyphonic-expression (visited on 11/14/2023).

[7] The MIDI Manufacturers Association and Association of Musical Electronics Industry (AMEI). *Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol*. Version 1.1.2. Nov. 2023. URL: https://www.midi.org/specifications/midi-2-0-specifications/midi2-core/universal-midi-packet-ump-and-midi-2-0-protocol-specification (visited on 11/15/2023).

[8] Marije Baalman. *Composing interactions: an artist's guide to building expressive interactive systems*. eng. Rotterdam: V2_Publishing, 2022. ISBN: 978-90-828935-4-0.

[9] Marije A. J. Baalman. "Interplay Between Composition, Instrument Design and Performance." en. In: *Musical Instruments in the 21st Century*. Ed. by Till Bovermann et al. Singapore: Springer Singapore, 2017, pp. 225–241. ISBN: 978-981-10-2950-9. DOI: 10.1007/978-981-10-2951-6_15. URL: http://link.springer.com/10.1007/978-981-10-2951-6_15 (visited on 11/29/2023).

[10] Marije A. J. Baalman. "Mapping the question of mapping." In: *Proceedings of the 5th International Conference on Live Interfaces (ICLI)* (2020). Colloquial paper. URL: https://api.semanticscholar.org/CorpusID:232216513 (visited on 11/28/2023).

[11] Mads Bye. "MIDI Polyphonic Expression: A Bridge Between Acoustic and Electronic Instrument Experience?" MA thesis. University of Oslo, 2018.

[12] Scott deLahunta. *Invisibility/corporeality*. Essay. 2001. URL: https://noemalab.eu/ideas/essay/invisibilitycorporeality/ (visited on 12/10/2023).

[13] Emmanuel Favreau et al. “Software Developments for the 4X Real-time System.” In: *Proceedings of the 1986 International Computer Music Conference, ICMC*. Den Haag, The Netherlands: Michigan Publishing, Oct. 1986, pp. 369–373. URL: https://hdl.handle.net/2027/spo.bbp2372.1986.074 (visited on 12/01/2023).

[14] Adrian Freed and Andrew Schmeder. “Features and Future of Open Sound Control version 1.1 for NIME.” In: *NIME*. June 2009, pp. 116–120.

[15] Jeffrey E. F. Friedl. “Mastering regular expressions - powerful techniques for Perl and other tools.” In: *Powerful techniques for Perl and other tools*. 1997. URL: https://api.semanticscholar.org/CorpusID:38230303 (visited on 11/21/2023).

[16] Michaela Gläser-Zikuda. “Self-Reflecting Methods of Learning Research.” en. In: *Encyclopedia of the Sciences of Learning*. Ed. by Norbert M. Seel. Boston, MA: Springer US, 2012, pp. 3011–3015. ISBN: 978-1-4419-1427-9. DOI: 10.1007/978-1-4419-1428-6_821. URL: http://link.springer.com/10.1007/978-1-4419-1428-6_821 (visited on 11/28/2023).

[17] John D. Gould and Clayton Lewis. “Designing for usability: key principles and what designers think.” en. In: *Communications of the ACM* 28.3 (Mar. 1985), pp. 300–311. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3166.3170. URL: https://dl.acm.org/doi/10.1145/3166.3170 (visited on 02/13/2023).

[18] W. Vann Hall. “Conquering the Midi Muddle.” en. In: *Music Educators Journal* 73.4 (Dec. 1986), pp. 26–29. ISSN: 0027-4321, 1945-0087. DOI: 10.2307/3397899. URL: http://journals.sagepub.com/doi/10.2307/3397899 (visited on 11/15/2023).

[19] Julian Hamelberg. *Applications and Implications of MIDI 2.0*. Master thesis. Massachusetts Institute of Technology, 2023.

[20] Simon Holland et al., eds. *Music and Human-Computer Interaction*. en. Springer Series on Cultural Computing. London: Springer London, 2013. ISBN: 978-1-4471-2989-9. DOI: 10.1007/978-1-4471-2990-5. URL: https://link.springer.com/10.1007/978-1-4471-2990-5 (visited on 12/06/2023).

[21] Simon Holland et al., eds. *New Directions in Music and Human-Computer Interaction*. en. Springer Series on Cultural Computing. Cham: Springer International Publishing, 2019. ISBN: 978-3-319-92068-9. DOI: 10.1007/978-3-319-92069-6. URL: http://link.springer.com/10.1007/978-3-319-92069-6 (visited on 12/06/2023).

[22] David Miles Huber. *The MIDI Manual: A Practical Guide to MIDI within Modern Music Production*. 4th ed. New York: Routledge, Oct. 2020. ISBN: 978-1-315-67083-6. DOI: 10.4324/9781315670836. URL: https://www.taylorfrancis.com/books/9781317368304 (visited on 11/14/2023).

[23] Andy Hunt and R. Kirk. “Radical user interfaces for real-time control.” In: vol. 2. Feb. 1999, 6–12 vol.2. ISBN: 0-7695-0321-7. DOI: 10.1109/EURMIC.1999.794755.

[24] Andy Hunt, Marcelo M Wanderley, and Ross Kirk. “Towards a model for instrumental mapping in expert musical interaction.” In: *ICMC*. Sept. 2000.

[25] Alexander Refsum Jensenius. *Sound Actions: Conceptualizing Musical Instruments*. en. The MIT Press, Dec. 2022. ISBN: 978-0-262-37220-6. DOI: 10.7551/mitpress/14220.001.0001. URL: https://direct.mit.edu/books/book/5513/Sound-ActionsConceptualizing-Musical-Instruments (visited on 02/13/2023).

[26] Alexander Refsum Jensenius and Michael J. Lyons, eds. *A NIME Reader*. Vol. 3. Current Research in Systematic Musicology. Springer International Publishing, 2017. ISBN: 978-3-319-47213-3. DOI: 10.1007/978-3-319-47214-0. URL: http://link.springer.com/10.1007/978-3-319-47214-0 (visited on 03/13/2023).

[27] Kevin Karplus and Alex Strong. “Digital Synthesis of Plucked-String and Drum Timbres.” In: *Computer Music Journal* 7.2 (1983), p. 43. ISSN: 01489267. DOI: 10.2307/3680062. URL: https://www.jstor.org/stable/3680062?origin=crossref (visited on 03/13/2023).

[28] Sequential LLC. *Prophet-6 Operation Manual.* Version 2.1. San Francisco, CA, Feb. 2021. URL: https://sequential.com/wp-content/uploads/2021/02/Prophet-6-Operation-Manual-2.1.pdf (visited on 01/15/2024).

[29] Gareth Loy. “Musicians Make a Standard: The MIDI Phenomenon.” In: *Computer Music Journal* 9.4 (1985), pp. 8–26. ISSN: 01489267. DOI: 10.2307/3679619. URL: https://www.jstor.org/stable/3679619?origin=crossref (visited on 11/14/2023).

[30] Peter Manning. “The Development of the MIDI Communications Protocol.” In: *Electronic and Computer Music.* 4th ed. Oxford University Press, Mar. 2013, p. 267. ISBN: 9780199746392. DOI: 10.1093/acprof:oso/9780199746392.003.0014. eprint: https://academic.oup.com/book/0/chapter/149079727/chapter-ag-pdf/44988318/book_5836_section_149079727.ag.pdf. (Visited on 11/14/2023).

[31] Keith McMillen. “ZIPI: Origins and Motivations.” In: *Computer Music Journal* 18.4 (1994), pp. 47–51. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3681357 (visited on 12/03/2023).

[32] Keith McMillen, David Simon, and Matthew Wright. “A Summary of the ZIPI Network.” In: *Computer Music Journal* 18.4 (1994), pp. 74–80. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3681359 (visited on 12/03/2023).

[33] Keith McMillen, David L. Wessel, and Matthew Wright. “The ZIPI Music Parameter Description Language.” In: *Computer Music Journal* 18.4 (1994), pp. 52–73. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3681358 (visited on 12/03/2023).

[34] Andrew McPherson. “2012: TouchKeys: Capacitive Multi-touch Sensing on a Physical Keyboard.” In: *A NIME Reader.* Ed. by Alexander Refsum Jensenius and Michael J. Lyons. Vol. 3. Series Title: Current Research in Systematic Musicology. Cham, Switzerland: Springer International Publishing, 2017, pp. 419–432. ISBN: 978-3-319-47213-3. DOI: 10.1007/978-3-319-47214-0_27. URL: http://link.springer.com/10.1007/978-3-319-47214-0_27 (visited on 03/13/2023).

[35] Andrew McPherson. *Discontinuing TouchKeys production.* URL: https://groups.google.com/g/touchkeys/c/D4UdrNOwPN4/m/H5cGO60zAwAJ (visited on 11/29/2023).

[36] Andrew McPherson. *TouchKeys: Software Manual.* Version 0.2. Sept. 2015. URL: https://code.soundsoftware.ac.uk/attachments/download/1579/touchkeys_manual_0.2.pdf (visited on 12/04/2023).

[37] Andrew Mcpherson and Youngmoo Kim. “Design and applications of a multi-touch musical keyboard.” In: *Proceedings of the 8th Sound and Music Computing Conference, SMC 2011* (July 2011).

[38] F. Richard Moore. “The Dysfunctions of MIDI.” In: *Computer Music Journal* 12.1 (1988), pp. 19–28. ISSN: 01489267. DOI: 10.2307/3679834. URL: https://www.jstor.org/stable/3679834?origin=crossref (visited on 11/14/2023).

[39] Per Anders Nilsson. “A Field of Possibilities: Designing and Playing Digital Musical Instruments.” PhD thesis. University of Gothenburg, 2011. ISBN: 9789197847780. URL: https://books.google.com/books?id=jlc4jwEACAAJ (visited on 11/27/2023).

[40] Miller Puckette. “Pure Data.” In: *International Computer Music Conference (ICMC).* Sept. 1997.

[41] Miller Puckette. “Pure Data: another integrated computer music environment.” In: *Proceedings of the Second Intercollege Computer Music Concerts.* Tachikawa, 1996, pp. 37–41.

[42] Miller Puckette. "The Patcher." In: *International Conference on Mathematics and Computing*. 1988. URL: https://api.semanticscholar.org/CorpusID:45525159 (visited on 12/01/2023).

[43] Ivan Reese (Host). "Max/MSP & Pure Data: Miller Puckette." In: *Future of Coding*. Podcast episode. May 2020. URL: https://omny.fm/shows/future-of-coding/47-maxmsp-pure-data-miller-puckette (visited on 12/01/2023).

[44] Curtis Roads. *The Computer Music Tutorial*. MIT press, 1996.

[45] Allen Strange. *Electronic Music: Systems, Techniques, and Controls*. Second edition. Toronto, Canada: Responsive Ecologies Lab with the assistance of Toronto Metropolitan University, 2022. ISBN: 978-1-77417-031-1.

[46] John Sweller. "Cognitive load theory, learning difficulty, and instructional design." en. In: *Learning and Instruction* 4.4 (Jan. 1994), pp. 295–312. ISSN: 09594752. DOI: 10.1016/0959-4752(94)90003-5. URL: https://linkinghub.elsevier.com/retrieve/pii/0959475294900035 (visited on 11/29/2023).

[47] M.M. Wanderley and P. Depalle. "Gestural Control of Sound Synthesis." In: *Proceedings of the IEEE* 92.4 (Apr. 2004), pp. 632–644. ISSN: 0018-9219. DOI: 10.1109/JPROC.2004.825882. URL: http://ieeexplore.ieee.org/document/1278687/ (visited on 11/28/2023).

[48] M.M. Wanderley, N. Schnell, and J. Rovan. "ESCHER-modeling and performing composed instruments in real-time." In: *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*. Vol. 2. San Diego, CA, USA: IEEE, 1998, pp. 1080–1084. ISBN: 978-0-7803-4778-6. DOI: 10.1109/ICSMC.1998.727836. URL: http://ieeexplore.ieee.org/document/727836/ (visited on 11/28/2023).

[49] Kristian Wentzel. *Myo My – That keyboard sure tastes good with some ZOIA on top*. Blog post. May 2022. URL: https://mct-master.github.io/motion-capture/2022/05/20/kriswent-extending-the-keyboard-through-motion-capture-and-modular-synthesis.html (visited on 01/23/2024).

[50] David Wessel and Matthew Wright. "Problems and prospects for intimate musical control of computers." In: *Computer music journal* 26.3 (2002), pp. 11–22.

[51] Matthew Wright. "A Comparison of MIDI and ZIPI." In: *Computer Music Journal* 18.4 (1994), pp. 86–91. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3681361 (visited on 12/03/2023).

[52] Matthew Wright. "Answers to Frequently Asked Questions about ZIPI." In: *Computer Music Journal* 18.4 (1994), pp. 92–96. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3681362 (visited on 12/03/2023).

[53] Matthew Wright. "Examples of ZIPI Applications." In: *Computer Music Journal* 18.4 (1994), pp. 81–85. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3681360 (visited on 12/03/2023).

[54] Matthew Wright. *OpenSoundControl Specification*. Web publication. Version 1.0. 2002. URL: https://opensoundcontrol.stanford.edu/spec-1_0.html (visited on 11/20/2023).

[55] Matthew Wright and Adrian Freed. "Open SoundControl: A new protocol for communicating with sound synthesizers." In: *International Computer Music Conference (ICMC)*. 1997, pp. 101–104.

[56] Matthew Wright, Adrian Freed, and Ali Momeni. "2003: OpenSound Control: State of the Art 2003." In: *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*. Ed. by Alexander Refsum Jensenius and Michael J. Lyons. Vol. 3. Series Title: Current Research in Systematic Musicology. Cham, Switzerland: Springer International Publishing, 2017, pp. 125–145. ISBN: 978-3-319-47214-0. DOI: 10.1007/978-3-319-47214-0_9. URL: https://doi.org/10.1007/978-3-319-47214-0_9 (visited on 11/21/2023).

[57] Matthew J. Wright et al. *ZIPI: Proposal for a New Networking Interface for Electronic Musical Devices.* Berkeley, California: University of California, Department of Music, Center for New Music & Audio Technologies, Jan. 1991. URL: https://esf.ccarh.org/ccarh-wiki/ZIPI.pdf (visited on 11/15/2023).