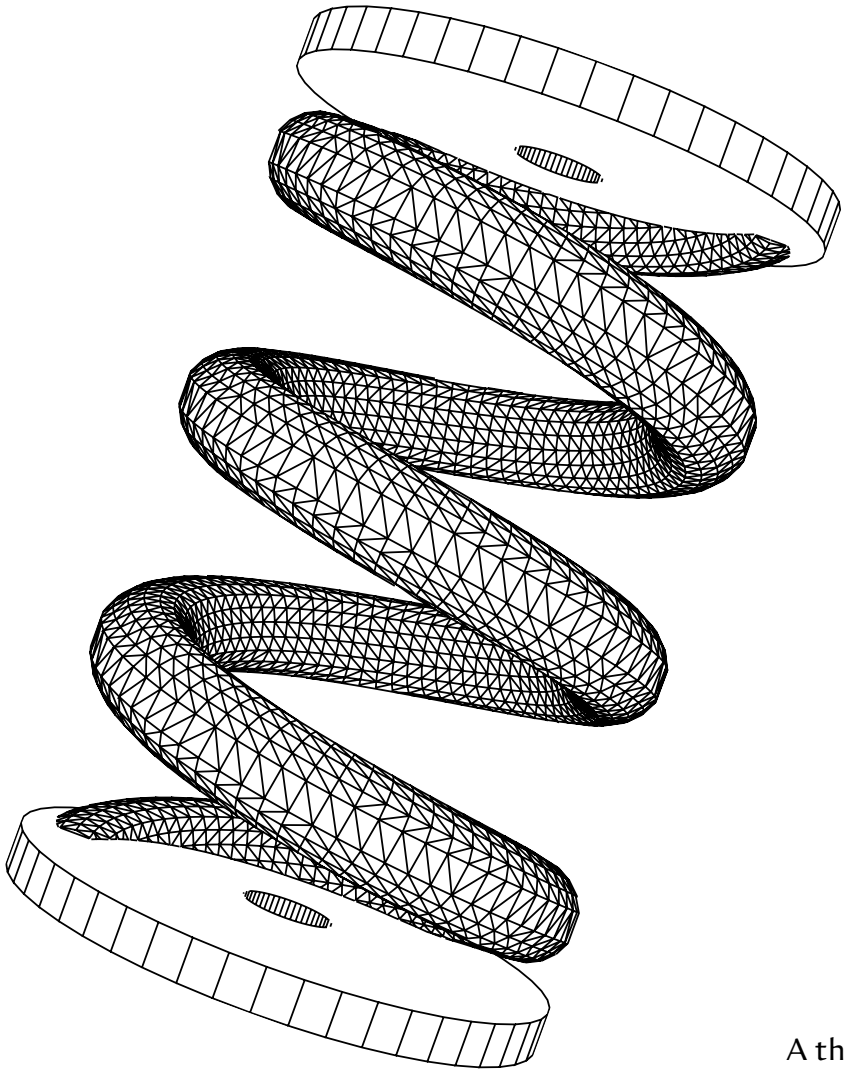
MAHWWHAM

a self-contained digital whammy bar add-on for electric guitars



Yü Fang

Schulich School of Music
McGill University
Montréal, Québec, Canada

April 12, 2024

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

you are viewing a plank bage

Abstract

THIS thesis introduces the design and redesign process of MAHW, a digital–mechanical whammy bar add-on for electric guitars. It is intended to serve as a more customizable and easier-to-fabricate substitute for the traditional mechanical vibrato systems on guitars. As a musical interface, MAHW combines the *mechanics* of a whammy bar and *digital* modeling of vibrato using a pitch-shifter.

The necessity for the new musical interface is discussed in the beginning chapter, by analyzing the problems present in existing vibrato designs. Following that is a summary and an evaluation of the two major design iterations that MAHW went through, transforming it from a simple, wired musical controller to a standalone, portable system that can process sound on its own. The hardware/sensor design and the signal processing algorithm are explained in detail. Special attention is given to how the redesign reduces the friction in using the system while keeping the fabrication complexity low. Several interesting techniques were discovered during the redesign process, such as the use of 3D-printed springs as a reproducible custom passive force feedback mechanism and a trick to model the nonlinear dissonant characteristic of a mechanical vibrato system. The contributions and limitations of the interface are discussed in the end to provide an insight into the next steps for the research and MAHW.

Résumé

Cette thèse présente le processus de conception et de reconception de MAHW, un système de vibrato numérique et mécanique pour les guitares électriques. Il est conçu comme substitut plus personnalisable et plus facile à fabriquer pour les vibratos mécaniques traditionnels sur les guitares. En tant qu'interface musicale, MAHW combine la *mécanique* de la barre de vibrato (whammy bar) et la modélisation *numérique* du vibrato à l'aide d'un pitch-shifter, qui modifie la hauteur du son.

La nécessité de cette nouvelle interface musicale est discutée dans le premier chapitre, en analysant les problèmes présents dans les conceptions des vibratos existants. On aborde ensuite un résumé et une évaluation des deux itérations principales de la conception de MAHW, qui l'ont transformé d'un contrôleur musical naïf et câblé en un système autonome et portable qui peut traiter le son tout seul. La conception de matériel et de capteurs ainsi que l'algorithme de traitement du signal sont expliqués en détail. On accorde une attention particulière à la façon dont la reconception réduit la friction dans l'utilisation du système, tout en maintenant une faible complexité de fabrication. Au cours du processus de reconception, plusieurs techniques intéressantes ont été découvertes, telles que l'utilisation de ressorts imprimés en 3D comme mécanisme passif, personnalisable et reproductible de retour de force, ainsi qu'une astuce pour modéliser le caractère dissonant et non linéaire du vibrato mécanique. Enfin, on discute les contributions et les limites de l'interface afin de donner un aperçu des prochaines étapes de la recherche et de MAHW.

Acknowledgements (Part I)

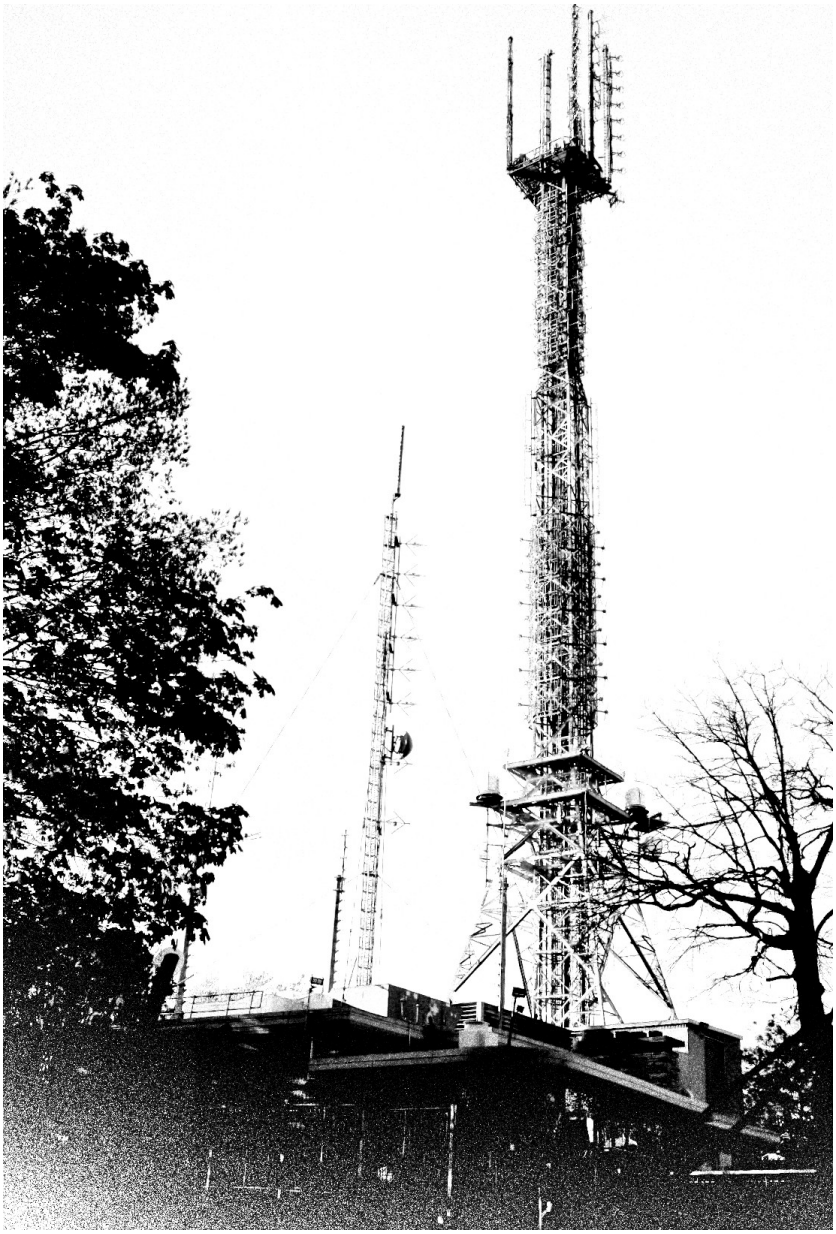


Figure 0: The *elephant* in the city (May 14, 2023).

Desk of Contents

List of Figures

List of Tables

Chapter 1: Overture

Musicians always have a fanatic obsession about things they can't do on the musical instruments they play. Pianists are notoriously[^1] fascinated by the idea of playing vibrato on a piano keyboard that, ever since people figured out how to build a piano out of electronics [21], they immediately put their heart into the stubborn idea of adding pitch-bending to the piano keyboard. Some of the earliest attempts include the movable keyboard design of the ondes Martenot [50] (since model 4, 1932) and the Ondioline [32] (1939), the sub-keyboard vibration transducer of the Экводин/Ekvodin [84] (since V-9, 1958), as well as the pitch wheel of the Minimoog [80] (1964). These designs eventually evolve into commercial products today, such as the TouchKeys [56] (2013), the ROLI Seaboard[^3] (2013), the Expressive E Osmose[^4] (2019), and the ROLI LUMI Keys[^5] (since the 2021 update), to satisfy pianists' compulsive urge to perform vibrato on a keyboard.

which, in fact, is something lost[^2] in the evolution from the clavichord [11] to the piano.

For more about the history of keyboard vibrato, refer to Mieda's book [85] and Lamb's thesis [49].

Guitarists are very glad that they are lucky enough to wield the forbidden magic of pitch-bending on their own instrument, until they realize [7] that no matter how hard they bend the string, the pitch only goes upward, never downward, not to mention the even darker magic of pitch-bending an entire chord. Frustrated, they once again embark on a forever journey of finding new ways to make their instrument more "expressive".

NOTE

Technically, it is possible to do downward chord bending without any modifications to a guitar. The trick is to not bend the string but to bend the neck—just push the body of the

[^1] Piano Vibrato: https://www.youtube.com/watch?v=sg9vp2uXS3c
[^2] Sabathil & Son Clavichord demonstration: "Die Bebung" (vibrato): https://www.youtube.com/watch?v=7oCGNwDokT0
[^3] ROLI Seaboard: https://roli.com/products/blocks/seaboard-block-studio-edition
[^4] Expressive E Osmose: https://www.expressivee.com/2-osmose
[^5] ROLI LUMI Keys: https://roli.com/products/blocks/lumi-keys-studio-edition

guitar against your body and, using the picking hand as a pivot, bend the neck forward with the fretting hand,

```
                    ↓
+----+================================***+
|_________________/                  ↑
```

though the downside is that the picking hand is locked from doing anything else while acting as a pivot. The pitch-shifting range is also limited to around only 20 cents on the low E string and around 10 cents on the high E string[^6]. Anything beyond that would be too scary that it might break the neck, especially on an acoustic guitar.

Among these frustrated guitarists, Clayton Orr Kauffman (a.k.a. Doc Kauffman) came up with a mechanical design [47] in 1929 that finally brought accessible downward pitch-bending to the guitar (and other similar stringed instruments). This was not the first vibrato design on guitar—it is possible to find even earlier designs by Albert J. Forrest [33] in 1897, James A. Burchit [14] in 1904, and Robert Tom Sawyer [67] in 1926, but these designs were no more than fancy extensions of the neck-bending technique and mostly limited to only raising the tension in the strings. They were not as commercially impactful as Kauffman's design.

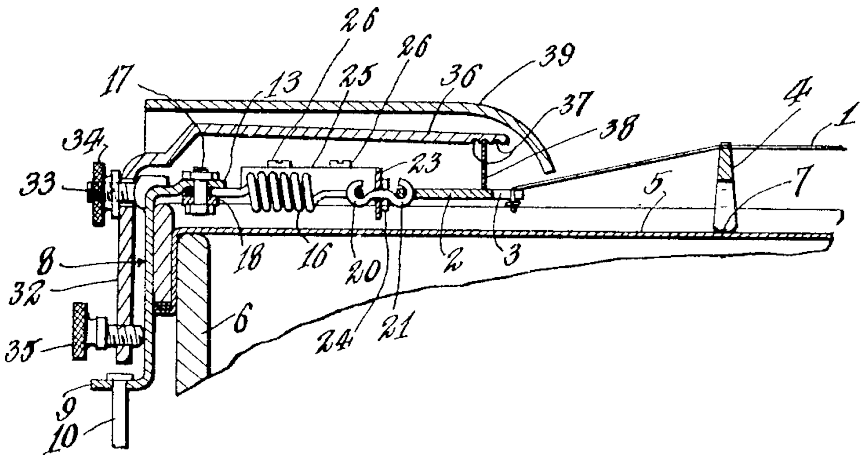


Figure 1.1: Kauffman's mechanical vibrato design illustrated in his patent [33]. In public domain.

Kauffman's vibrato design (Figure 1.1) works by stretching and contracting the strings *mechanically* using a movable tailpiece, which is linked to a lever arm to control the string tension and raise or lower the pitches of the notes played.

[^6] Listen to `https://mahw.krj.st/neckbend.wav`

Unlike most of today's vibrato designs, the lever arm is operated sideways, in the same direction as the hand movement when strumming a chord. According to Kauffman, it was supposed to feel more *natural* that way, but apparently the perpendicular motion became more popular later. Gibson's Sideways Vibrola tailpiece (Figure 1.2) is one of the few modern vibrato designs that still follow the same control gesture as the original Kauffman design.

Figure 1.2: Gibson Sideways Vibrola, one of the few modern vibrato designs similar to the original Kauffman design. The springs are missing from the figure. Image use authorized from Crazyparts.

Nonetheless, the core of Kauffman's design—using a lever arm to stretch and contract the strings—still became the basis of almost all the modern mechanical vibrato systems on guitars, the most famous ones being the Paul Bigsby's Bigsby vibrato tailpiece [8] and Clarence Leo Fender's floating tremolo system on the Stratocaster [25] and the Jazzmaster [27]. It is usually known to guitarists as a *whammy bar*, a *vibrato arm*, or a *tremolo arm* if they prefer to live in Leo Fender's universe.

Many early inventors used "vibrato" and "tremolo" interchangeably, but apparently Fender knew the difference (refer to his patents) and still insisted marketing vibrato systems [28] on guitars as tremolo and tremolo systems [26] on guitar amps as vibrato to this day.

Doc Kauffman's innovation was undeniably a big step forward in the history of guitar. Its successors realized millions of people' dream to indulge in the joy and "expressiveness" of vibrato on a guitar. However, mechanical vibratos are especially intricate systems. They always involve a movable bridge (Figure 1.3) that needs to mount with the extraordinary tension [22] from guitar strings and maintain such tension to keep them in tune.

The movable bridge in a mechanical design causes quite a few headaches for guitarists, such as[7]

1. a loss of body resonance
2. tuning instabilities
3. bridge-rubbing noises (usually in floating bridge designs, such as a Fender Jazzmaster[8])

[7] The list just ridiculously keeps getting longer ever since I bought a cheap Squier Jazzmaster.
[8] Fender Jazzmaster: https://www.fender.com/en-US/electric-guitars/jazzmaster/

4. bridge moving causing intermittent grounding problems at the bridge posts when the bridge height is low (in floating bridge designs)
5. bridge rubbing causing frequent high-E string breakage
6. dissonant chords due to uneven string tension (but can be desirable sometimes)
7. string tension coupling (changing the tuning of one string requires retuning all other strings)

though despite all of these, the *real concern* for myself, which eventually leads to the exploration of **MAHW**, was the frustration to find a usable and accessible vibrato system for *mirrored* or *left-handed* electric guitars—the manufacturers of these mechanical vibrato systems more than often left out a small group of unfortunate human beings who were mistakenly born into a *mirrored*[^10] universe. It is not that a mirrored vibrato system doesn't exist *at all*, but finding the one you need requires some good luck outside the United States, especially during the pandemic shortage, when the exploration of **MAHW** started.

not only vibrato manufacturers. Fender famously wires log potentiometers[^9] on mirrored guitars incorrectly to this day, causing headaches for many people.

Figure 1.3: The movable bridge design of Duesenberg Les Trem II. Image use authorized from Duesenberg Guitars.

Being able to fabricate a customized (namely, mirrored) vibrato design, thus, became a real concern for me. The amount of mechanical stress on a mechanical vibrato system restricts that it can only be machined from a piece of metal, making it extremely difficult and costly to fabricate a robust, customized design from scratch. The challenge is not so much about designing the vibrato system on paper—there are plenty of working designs already (despite usually being mirrored)—

[^9] Reverse logarithmic pots: https://www.homeoftone.co.uk/blogs/news/reverse-logarithmic-pots-as-a-left-handed-player-are-they-right-for-me
[^10] Mirror: https://mahw.krj.st/mirror.png

but rather the ability to source the material and master the machining skills and tools to fabricate one from an existing design.

Determined that the mechanical route is neither practical nor helpful for the peers trapped in the mirrored universe, I started a journey to search for an easier way to implement vibrato on guitars—possibly not as robust and convenient as a mechanical design, but anyone with some skills should be able to fabricate it, without the need of a bulky, dusty lathe or milling machine, accessible only to a lucky few who happen to own a garage or have access to an institution with a mechanical lab. Hence, I decided to experiment with an alternate hybrid design, named **MAHW**, that combines the *mechanics* of a whammy bar and *digital* modeling of vibrato using a pitch-shifter. Because the hybrid design no longer needs to alter or maintain the string tension, it would drastically reduce the material strength requirement and the fabrication complexity, making customizations more accessible.

A digital–mechanical hybrid whammy bar design is not a new idea. In an attempt to solve the "typical" mechanical vibrato problems, Peter J. Walker developed and launched a commercial digital–mechanical vibrato system called Virtual Jeff PRO[^11] in 2015. The vibrato system uses the same lever bar design as a mechanical whammy bar, but instead of altering the string tension, it applies a pitch-shifting effect on the output audio signal from the guitar to obtain a similar pitch-bending effect.

The mechanical design of Virtual Jeff is ingenious and intricate [78], but the product comes with its own limitations, too, the most important one for me being that it does not come with a mirrored variant yet (as of 2023). Besides that, the whammy bar attachment does not operate on its own. The audio processing and pitch-shifting are performed on a separate effect pedal, to which the whammy bar attachment is connected via a long cable or a wireless transmitter (sold separately). Both of these options bring some frictions to using the digital whammy bar—in the wired case, the user needs to maintain an extra cable coming out of the guitar and a power cable for the effect pedal; in the wireless case, there will be the typical concerns about wireless reliability [79] and lithium battery deterioration (batteries die over time).

will be discussed later in Chapter 3.

Should I spend hundreds if not thousands of hours learning and practicing an opaque new control interface, making it part of my brain, my *second voice*, if I have no idea if I could customize it to my need in the future

You can't really trust wireless any more if your ex-neighbor owned a microwave oven that kills all your Wi-Fi and Bluetooth connections at dinner time.

[^11] Virtual Jeff Pro: https://www.fomofx.com/

NOTE

I somehow learned to use the whammy bar quite differently from most people, probably because I figured out the technique on the prototypes of **MAHW** before I got my hands on a real whammy bar. I press it with my wrist, not my palm or fingers. I find this technique gives me more control and less movement on the hands when bending the attack of chords when strumming.

Refer to `https://mahw.krj.st/mahw/` to see how it works. I haven't found anyone else operating the whammy bar in a similar way, maybe because it looks ugly and often accidentally puts the guitar out of tune, but it kind of works if you can appreciate a dissonant sound.

or get it repaired when it breaks years down the road, especially when it contains a mechanical component under stress? This is a challenge faced by all the commercial "new" digital musical control interfaces and probably why a majority of musicians are fairly conservative when it comes to adopting a new digital musical instrument, such as the Haken Continuum [38] or the LinnStrument [52], into their regular musical practices. It is not unusual for a commercial digital musical instrument to become obsolete and unsupported. When reacTable [44] ceased their commercial operations in 2022, all the existing reacTable musicians were put at risk of losing their *second voice*. Any musical control interface with a digital component or integrated circuits (ICs) will become almost impossible to repair without an official support.

In fact, the software and app of reacTable were obsolete long before that, only runnable on an ancient Mac OS and iOS system version.

ASIDE

This seems to be a real concern for touring musicians utilizing new digital musical instruments. Recently, I happened to have a chance to chat with Chad Matheny (Emperor X[12]) before a show, and he mentioned that the lead-free solder joints (the regular lead solder is banned in the European Union) in Teenage Engineering OP-1[13] were so unreliable that he had to bring some extra IO boards ordered from iFixit on the road for quick replacements, in case anything goes wrong. Now that the IO boards for the OP-1 are out of stock on iFixit, the repair will become much more difficult and time-consuming. It is not like guitar strings or violin bows in that multiple suppliers are able to provide the consumable part.

For a traditional musical instrument like the guitar, we have a range of different designs and build qualities, from a DIY Partscaster to a Masterbuilt. Any major city with a decent musician population will most likely have a few luthiers scattered around, who custom build and maintain these instruments to keep them alive. A community exists not only for the musicians but also for the makers, yet the scene appears to be more centralized when it comes to a new digital musical interface—the concept of the interface and the only implementations often tie to the same entity: a single person, a single lab, or a single company. As found out by [16], out of the 245 papers about musical interface development in the *Proceedings of the New Interfaces for Musical Expression (NIME)* from 2018 to 2020, only two were about replicating an interface designed by someone else, creating an alternate implementation. The maker community of these kinds of new musical interfaces just don't tend to have the interest to reiterate and improve on an existing design without a proven success. Due to the complexity of digital design, the lifespan of a new digital musical interface essentially ends when the central entity loses interest in maintaining the interface (as exemplified by the reacTable before).

Exceptions do exist. One of my favorites is the monome norns[14] community. There is an official build of norns, but many people build their own compatible norns using open-source implementations such as fates[15] and norns shield[16].

For a musical interface design to be reimplementable or reproducible, people often advocate better documentation as a panacea [15, 16], but I always feel there is something more than that. People can certainly design a custom IC chip for their new musical interfaces using software like Cadence[17] (not uncommon for large corporations like Yamaha or Roland), but even if we have

[12] Emperor X: https://emperorx.bandcamp.com/
[13] Teenage Engineering OP-1: https://teenage.engineering/products/op-1/original
[14] monome norns: https://monome.org/docs/norns/
[15] fates: https://github.com/okyeron/fates
[16] norns shield https://github.com/monome/norns-shield
[17] Cadence: https://www.cadence.com

the schematic for the chip, not everyone have the budget to etch them on a silicon wafer.

ASIDE

I contacted quite a few manufacturers for the feasibility of creating a custom component for a design. Many of them cold reject that university research projects won't have the budget for that. Custom components will become cheap at a scale, but not usually for a one-off prototype. If we want a power spring to build a retractor, we can easily salvage one from a cheap tape measure or a retractable badge holder, and build the prototype around whatever the dimension and force of that only power spring we got, but fabricating a spring with the same specification for a replica is not a trivial thing to do.

I once had an idea of using the $7 USD Amazon Basics wired computer mouse as a fast and reliable supplier of a relative distance sensor, but the solution always feels wrong to me when considering that, first, not every country has Amazon and, second, the replicability of the design will be tied to a single third-party manufacturer never changing the sensor of their product.

As we mentioned before, for the mechanical whammy bar, the bottleneck is not about the design but the fabrication complexity and the ability to source the material. Digital musical interface designers often use whatever components they have on hand for the design. For example, I chose to use Teensy 3.2 in the first prototype of **MAHW** (introduced in Chapter 2) only because it was conveniently lying in my drawer, and I did not think about how it was an overkill for my application or how it might get deprecated soon. This is the least resistant way to design a prototype against rotting, but without putting more care putting into the design stage, choosing which components and fabrication technologies to use or avoid, replicating the design will be very difficult even with documentation and parametric source files.

Designing an interface for replication, or maybe *designing an interface to be redesigned and improved by someone else*, is a separate, more time-consuming, process at the design stage. It is more than just a mindless design using whatever the designer has on hand, maybe a compression spring salvaged from a shampoo dispenser (Figure 3.1), then documenting it, and hoping the person reimplementing it best luck to use the exact same shampoo. There is a virtue in not needing to consider these kinds of manufacturing-level component dependency details and being able to iterate fast on new ideas in an academic setting, but in order for an interface to be truly useful for the majority of independent and hobbyist artists, who are often short on budget to access profes-

After all, as proven by the Toaster Project [75], it's insanely difficult to truly make something from scratch nowadays.

sional tools and have limited engineering background, there is also a need for self-imposed low-tech [6], low-fabrication-complexity designs. On one hand, the low fabrication complexity will provide an affordable and sustainable way for people to obtain or replicate the interface. On the other hand, the low-tech solution itself, creatively composing simple elements to achieve a non-straightforward task, is also a valuable engineering lesson accessible even to an artist without a formal engineering training. Amateurs build and create to learn and share new skills—this is a constant theme in the DIY community [29, 76]. When people understand how their tool works internally, they could bring their own expertise and creativity to the design, and it becomes possible for them to adapt the design to their own needs, instead of relying on someone else, who doesn't really understand their priorities, to take into account their niche habits. This is important for a musical interface to be improvable, customizable, and sustainable.

Ever since I moved to Montréal, I can't even count how many horror stories I've heard from struggling artists about their finance.

As we shall see in the following chapters, the latter approach of a low-fabrication-complexity design is going to govern many decisions in the design process of **MAHW**. After all, **MAHW** was born in the frustration of the poor customizability and DIY fabricability of existing guitar vibrato systems. There is no point for me in building yet another intricate system that no one else could reimplement and improve upon.

Hence, after all these justifications, we have now established the big goal of the project: designing **MAHW** as a digital–mechanical hybrid alternative to a traditional vibrato system for guitars, with a hidden constraint in mind to keep the fabrication complexity as low as possible. The following chapters will be documenting the design for two prototypes of **MAHW** and evaluate whether and how they accomplish the goal.

Chapter 2: The First Take

The first prototype of **MAHW** (Figure 2.1) was built as a course project in the span of a few weeks. It was put together with a joystick sensor, some 3D-printed components, and a Teensy 3.2 microcontroller to send the sensor signals to a computer via a wired serial connection to control the audio processing.

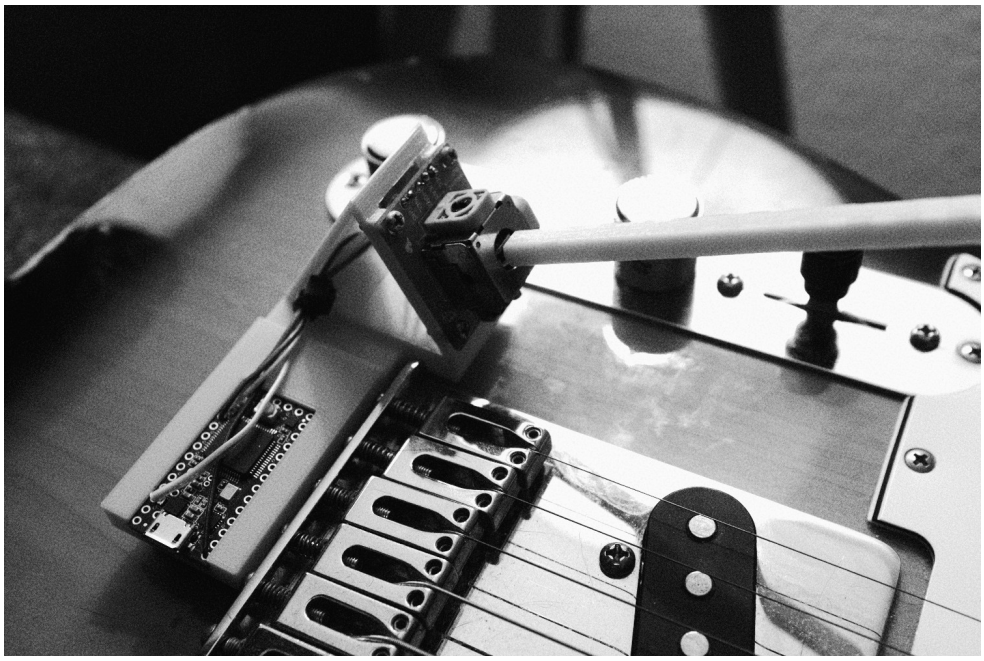


Figure 2.1: The first prototype of **MAHW**.

The design of this prototype is minimal, pretty much a barebone *copy of shape* from an actual mechanical whammy bar. It does not yet have the capability to process the guitar audio signal on its own, so it has to rely on an external computer and an audio interface to perform the signal processing.

Basically, the thought process is just, if they look similar, they should function similarly.

The very first thought that comes into mind when designing **MAHW** is that the controller must be detachable from the guitar. Electronics fail often, and no one can design a controller in one shot, so it doesn't make sense to glue the very first design permanently onto the only guitar I

had back then. Velcro fastener tapes turn out to be an essential tool in the prototyping process of **MAHW**, probably the only thing that continued into the second prototype. One side of the tape is semi-permanently glued onto the guitar (Figure 2.2), and the other side is glued onto the controller. In this way, I can swap and test out different designs easily and take off the controller when I don't use it or when I need to put the guitar into a bag. Although the glue from the Velcro is quite strong, it can still be removed with some force or a blade.

In fact, I'm not using a fastener from Velcro™ exactly. It's a stronger version called Scotch™ Extremely Strong Fasteners, though many people refer to this product as a Velcro.

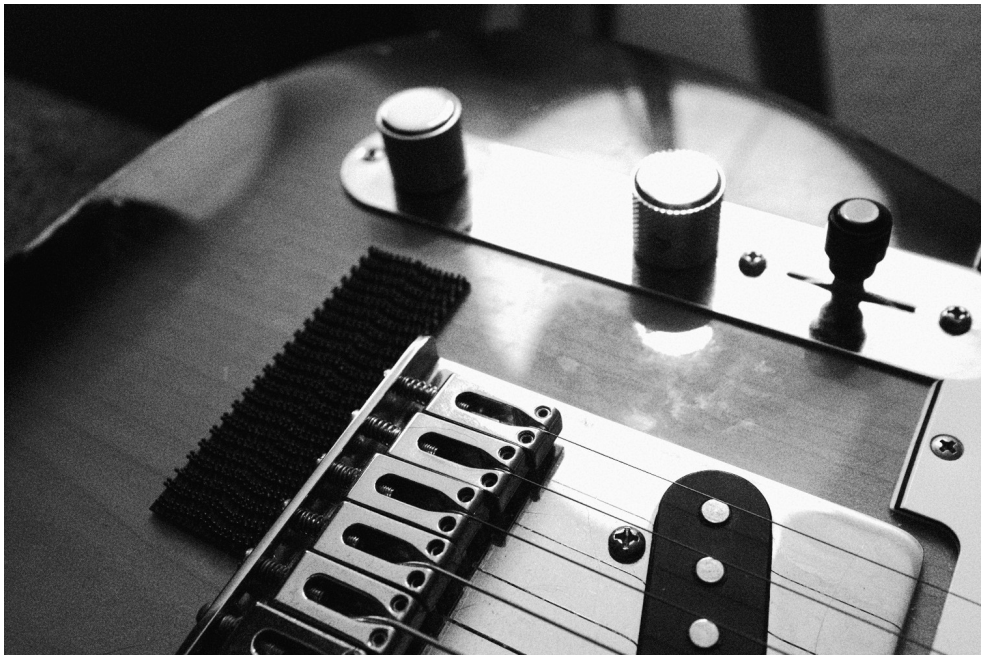


Figure 2.2: Velcro fastener tapes are used to mount **MAHW** onto the guitar.

The first generation of the controller **MAHW** consists of two parts: the actual controller hardware that is installed on the guitar and a software component on the computer. The audio signal from the guitar and the sensor signals from the controller are sent to a computer, and then the software component process the guitar audio using the sensor signals as parameters to produce the output audio signal (Figure 2.3).

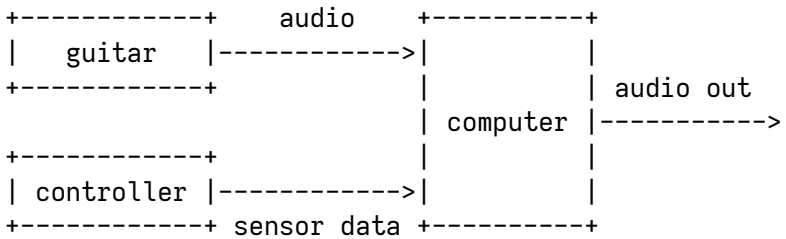
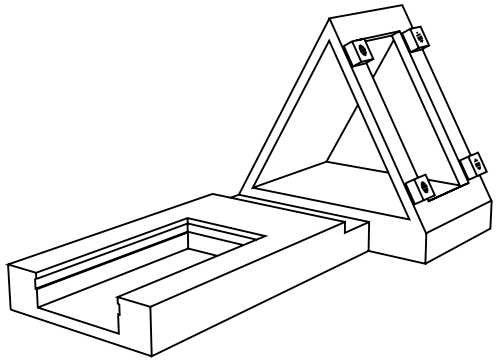


Figure 2.3: The flow diagram of the first generation **MAHW**.

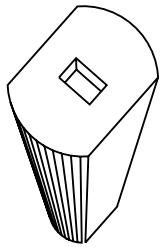
I will introduce these two components separately in the following sections.

2.1 THE CONTROLLER HARDWARE

The hardware of the controller consists of four major components: a 3D-printed casing (Figure 2.4a), a joystick sensor, a Teensy 3.2 microcontroller, and a 3D-printed whammy bar arm (Figure 2.4b) installed on the joystick. The full assembly can be seen in Figure 2.1. Notice that all prototypes of **MAHW** are designed for a mirrored guitar, but they can be easily adapted to a normal guitar by printing a flipped model.



(a) The casing



(b) The whammy bar arm

Figure 2.4: 3D models of the hardware component.

The most crucial yet tricky part of the hardware design is to simulate the *press* gesture of a mechanical whammy bar. It is tricky in that we not only need a way to measure the amount of *press*—which can be quantified either as the tilt/rotation angle of the arm, the force applied on the arm, the distance from the arm tip to some reference point, or maybe something else—but the arm also needs to have some kind of return-to-center mechanism.

Return-to-center (RTC) is an essential mechanism in any manual vibrato design, no matter digital or mechanical, on a piano keyboard or on a guitar. Our ears can tolerate some level of dissonance from the out-of-tune nature of vibrato, as long as it eventually resolves into something more in-tune and consonant, and the RTC mechanism is exactly what brings the instrument back in tune when the vibrato is disengaged. Without an RTC mechanism, the vibrato design is pretty much unusable because the instrument will be out-of-tune all the time.

vibrato amount controlled by the user, not by a low-frequency oscillator.

See noise rock bands such as Yuck[^1] and Fleeting Joys[^2] for some examples.

[^1] Yuck: https://yuck.bandcamp.com/track/yr-face
[^2] Fleeting Joys: https://fleetingjoys1.bandcamp.com/track/while-im-waiting

NOTE

The pitch wheels on many early electronic synthesizers, notably the original Minimoog, do not have a spring-loaded *automatic* RTC mechanism, but the potentiometer used for the pitch wheel usually has a center detent to offer a tactile feedback when it is returned to the center. The spring-loaded automatic RTC design apparently become the more popular option later so that Moog had to redesign the pitch wheel to include a spring-back mechanism in the 2022 reissue of the Minimoog Model D [65].

The joystick sensor happens to be a good candidate for this task. It satisfies both of these requirements at once. The sensor measures how much the lever deviates from the center position and convert that to two voltage values as the output (in X and Y direction since the joystick can move in two directions, though only one direction is necessary for our purpose). If we hook up an extension stick as the whammy bar arm to the joystick, the deviation can then be interpreted as the vertical or horizontal tilt angle of the arm. Additionally, because joystick sensors already have springs built-in, the lever will automatically return to the center position when the arm is released.

though the reliability of joysticks' RTC mechanism over time is another story, as seen in Nintendo's troubles with Joy-Con drifting [61].

The joystick, hence, became the only sensor present in the first prototype of **MAHW**. The exact model of the joystick sensor used here is Top-Up JT9S-1P-B10K-50°-P-OGR-16Y, which is distributed by SparkFun as SparkFun COM-16273 and separately comes with a breakout PCB board SparkFun BOB-09110 for easier mounting and wiring. There are plenty of other options, probably not hard to salvage one from an old game controller. The joystick sensor used here actually produces three outputs: the vertical tilt, the horizontal tilt, and a click button triggered when you push the lever against the metal casing. However, only the vertical tilt is strictly needed to simulate a mechanical whammy bar. The other two sensor outputs can provide extra flexibility for the user, though they are not used in the demo. The joystick is mounted to the casing (Figure 2.4a) using the PCB breakout board and four M2 mounting screws.

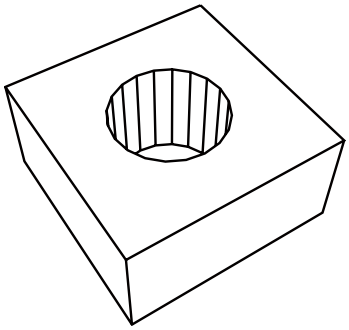
For the *press* gesture sensor, one of the other ideas at the time was to use a more traditional rotary potentiometer to measure the rotation angle. It is a more widely available sensor, but most potentiometers inconveniently do not have an automatic RTC mechanism and the friction force of the rotation is less documented and standardized. Also, as a mechanical component, potentiometers often develop a poor contact over time and need to be cleaned. Many piano vibrato designs do use potentiometers, such as the pitch wheel of the Moog Sonic Six [31] and the Weston Pre-

cision Audio PRO2021 synthesizer [82], with an additional torsion spring to implement the RTC mechanism, but it requires an intricate mechanical design and a way to fabricate or source the torsion spring in the first place. The center detent potentiometer design from the original Minimoog is also an option, but the potentiometer needs to be manually restored to center. It might work for a monophonic synth operated with one hand most of the time, but it is unlikely for the user experience to be great on a two-handed instrument like guitar. I did not pursue this path due to the time constraint on the first prototype.

```
hole_d = 2.05;
cube_d = 4; cube_h = 2;

difference() {
  cube([cube_d,cube_d,cube_h],center=true);
  cylinder(h=7,d=hole_d,$fn=20,center=true);
}
```

(a) The OpenSCAD script



(b) The rendered model

Figure 2.5: An example model written in OpenSCAD.

The 3D models for the casing and the whammy bar arm are both generated from parametric code written in OpenSCAD[^3] and printed using a Prusa i3 3D printer. An advantage of OpenSCAD is that it is a completely script-based (Figure 2.5) open-source CAD software and the GUI is optional. Some people might find it confusing compared to a traditional CAD software, but the parameters are just variables in a text file, so it is not essential to know how OpenSCAD scripting works or how to use a CAD software in order to make adjustments to the model. The codebase of OpenSCAD is open and a lot smaller than most GUI-based open-source and commercial CAD software, so software deterioration [62] is less likely to become a problem years down the road, which is good for our goal to keep the customization complexity low.

For comparison, the C++ codebase for OpenSCAD is ~150k lines of code, and it's more than 2000k for FreeCAD.

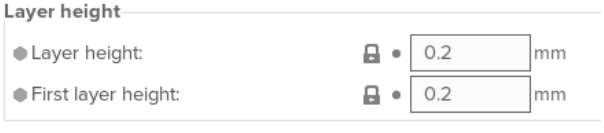


Figure 2.6: Layer height setting for PrusaSlicer.

When printing the 3D model for the casing, it might be necessary to use a slightly higher layer

[^3]OpenSCAD: https://openscad.org/

height precision, around 0.20 mm or lower (Figure 2.6). This is because the vertical precision of a 3D printer is generally worse than the horizontal precision, so the slanted parts in the model, i.e. the mounting holes for the PCB breakout board, might not fit the screws properly without an increased precision.

at least for a FDM 3D printer like Prusa i3.

The assembly requires no glue at all. The joystick is soldered onto the PCB breakout board, and the breakout board is fixed onto the casing by screws. There is a groove on the casing (Figure 2.4a) designed to hold the Teensy 3.2 microcontroller. The Teensy slides perfectly into the groove and is held in place by friction. The tilt angle outputs of the joystick sensor are connected to two analog pins of Teensy and the click button output of the joystick is connected to a digital pin. The whammy bar arm (Figure 2.4b) also has a groove that fits the lever on the joystick and is held in place by friction.

NOTE

For any friction-held component, the clearance gap for the grooves or the holes is important and needs to be determined experimentally. If there is no clearance, the component won't fit in the space, and if the clearance is too large, the friction won't be enough to hold the component in place. I generally start with a clearance of around 0.1 mm and tweak it from there.

2.2 THE SOFTWARE

The assembled **MAHW** controller is connected to a computer with a 2-meter micro-USB to USB data cable. The micro-USB end of the cable is plugged into Teensy, providing power and transmitting the sensor data to the USB end of the cable, which is connected to the computer. When **MAHW** is mounted on a guitar, the audio output of the guitar also needs to connect to the computer via an audio interface. Hence, the software program on the computer will receive two signals, one is the sensor data and one is the guitar audio signal (refer back to Figure 2.3).

The use of a *data* cable is important here. Some cables only supply power and don't transmit data.

The joystick sensor data read by Teensy is sent to the computer via serial communication. The sensor data, consisting of vertical tilt (**v**, 13 bits), horizontal tilt (**h**, 13 bits), and select button (**s**, 1 bit), adds up to more than 1 byte in total, so to transmit it with serial, it is necessary to use a packet framing protocol to delimit the boundary of a multi-byte packet. There are standardized packet framing protocols such as SLIP [66] and COBS [17], but a custom, simpler protocol is used here—the sensor data is packed into a packet of 4 bytes

```
1vvvvvvv 0svvvvvv 0hhhhhhh 00hhhhhh
```

The beginning byte has a most significant bit of 1 and the trailing bytes have a most significant bit of 0. Without this distinction, it would be impossible to determine the beginning of a 4-byte packet because the transmission might start from a byte in the middle of a packet.

The computer side of the program is written mostly for Linux, though it might run under other UNIX-compatible systems. The serial data is read using Linux kernel's ACM driver. Basically, Teensy as a serial device will be made available as a file descriptor under the path **`/dev/ttyACM0`** or something similar. The usual **`read`** UNIX system call will be able to read the serial data from the file with some setup (see [73] for details).

The audio processing part, and in fact many tools used, in the first **MAHW** prototype took ideas from Paul Bachelor's instrument Contrenot [5]. The main inspiration is the use of a domain-specific language (DSL) to handle the parameter mapping and routing, but there are also other things such as the serial communication and OpenSCAD for modeling. A few months before **MAHW** started, I was asking him for help with a new DSL chapter for projet µ[^4]. His response inspired and helped me created µsporth[^5], a stack-based audio programming language written in 500 lines of C. Because µsporth is easily extensible and the stack-based nature is convenient to construct a parameter mapping graph, the first **MAHW** prototype ended up using a custom extended version of µsporth for guitar signal processing. In fact, the use of µsporth even continued into the early development of the second prototype to test out sensors and digital signal processing (DSP) algorithms.

which is a rewrite of Bachelor's audio programming language Sporth [4] originally prepared to be used in projet µ.

Figure 2.7: JACK routing for µsporth.

The software component of **MAHW** is based on the real-time JACK front-end of µsporth. JACK Audio Connection Kit[^6] is used to handle the audio routing from and to the sound card (Figure 2.7). µsporth is a separate project from **MAHW**, not the focus of this thesis, so I will only provide a one paragraph crash course on the language:

Refer to projet µ for more details about JACK.

[^4] projet µ: https://mu.krj.st/
[^5] µsporth: https://tig.krj.st/usporth/
[^6] JACK Audio Connection Kit: https://jackaudio.org/

The syntax of μsporth is based on FORTH [60]. A μsporth script consists of a series of *words* separated by spaces. Each *word* in μsporth is called a *unit generator* (ugen). A *ugen* generates or processes values at audio rate. For example, the script

```
440 sine
```

produces a sine wave in the speaker at 440 Hz. Here, the **`440`** is the argument of **`sine`**, which sets the frequency of the sine oscillator. For every audio sample, the **`440`** constant ugen pushes a value of 440 onto *an implicit stack*. The **`sine`** ugen pops one value on the stack, which gives 440, then uses the popped value as the frequency to compute a new sample of sine wave, and finally pushes the computed sample onto the stack (Figure 2.8). After running through all the ugens, the value on the stack in the end will be the output sample sent to the speaker for playback.

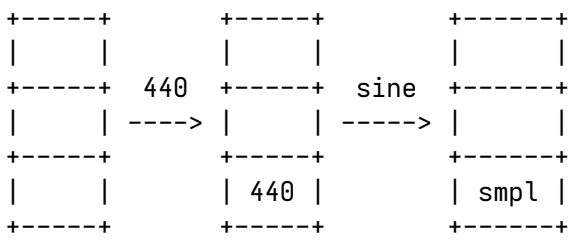


Figure 2.8: Visualization of ugens acting on the implicit stack.

For MAHW, the original μsporth is extended with a new ugen named **`wham`**. The **`wham`** ugen takes a string as its argument, either one of **`"vert"`**, **`"horz"`**, or **`"sel"`**, and pushes the corresponding sensor value onto the stack. **`"vert"`** corresponds to the vertical tilt, **`"horz"`** corresponds to the horizontal tilt, and **`"sel"`** corresponds to the select button. The tilt values are scaled between -1 and 1 and the select button is a binary value of either 0 or 1. For example, if we replace **`440`** in the previous script by **`"vert" wham`**,

```
"vert" wham sine
```

running the script will give us a sine oscillator frequency-controlled by the vertical tilt of the whammy bar. The **`biscale`** ugen can be used to linearly scale a value between -1 to 1 to a new range, specified by its two arguments, so the script

```
"vert" wham
0 440 biscale
sine
```

generates a variable frequency sine wave where the minimum position of the vertical tilt gives a

frequency of 0 Hz and the maximum position gives a frequency of 440 Hz.

Instead of controlling the frequency of an oscillator, the sensor value from the whammy bar can be fed into a ugen named **`pitshift`** to pitch-shift an input signal. The **`pitshift`** ugen takes two arguments from the stack, an input audio sample and a shift ratio factor (2.0 means shift frequency by a factor of 2, e.g. a 440 Hz signal will become 880 Hz), and produces a pitch-shifted version of the input as the output.

How much should we pitch-shift the guitar signal? Turns out the answer might be more complicated than you expect. The mechanical whammy bar is technically not a perfect pitch-shifter, because each string will detune by a different amount of semitones due to variations in string thickness, tension, and length. Generally, lower strings will detune faster than higher strings [71].

See [36] for the physics of string bending.

There is an easy experiment to demonstrate this effect. On a guitar with a mechanical whammy bar, pick the low and high E strings at the same time, then gradually engage the whammy bar. If you listen carefully, you should hear a beating sound similar to what you hear when tuning an out-of-tune guitar by ear. The beating is caused by the waveform interference of two adjacent-frequency waves. The high E string is beating with the harmonics of the low E string. The beating will be faster and faster as you detune more and more.

make sure the guitar is in tune before engaging the whammy bar.

This effect is quantified in Figure 2.9. The high-E and low-E strings are picked separately, detuned to the maximum using the whammy bar, then restored to the normal tension. The estimated frequency over time is plotted in the three figures. The first plot is the Hz frequency difference from the base frequency (82.41 Hz for low E and 329.63 Hz for high E string). The second plot is the ratio with respect to the base frequency. The third plot is the semitone difference from the base pitch. As we can see, although the high E string appears to be easier to detune than the low E string under the frequency scale, in the more musical semitone scale, the detune of the high E string is actually 0.5 semitones less than the low E string. This is because pitches are more spaced apart in the frequency scale at high frequency, so the same frequency difference would imply a lower pitch change when the referenced base frequency is higher. As a result, the low E string can detune by around 1.1 semitones at maximum, but the high E string can only detune by 0.5 semitone at maximum. The discrepancy here hence creates the beating sound.

The implication of this phenomenon is that any pitch-bent chord will become dissonant under a mechanical vibrato system. It is very tricky to model this nonlinear dissonant behavior accurately without a hexaphonic pickup, though the importance to model this behavior in the first place

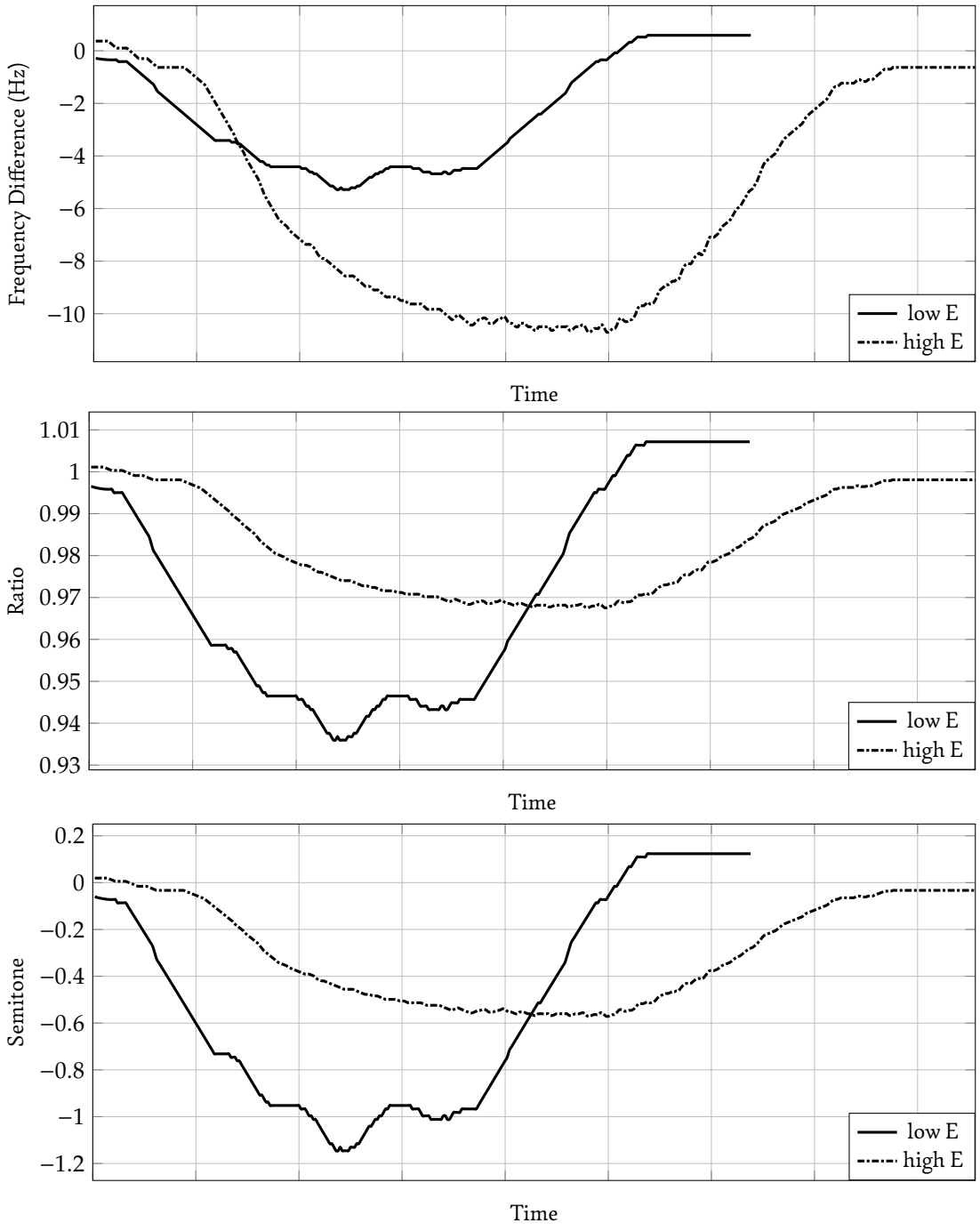


Figure 2.9: Pitch bend analysis of mechanical whammy bar (audio recorded using a Squier Jazzmaster, and pitch estimated using the **`pitch`** function in MATLAB).

can be up to debate. Ned Steinberger spent years to develop a mechanical vibrato system called TransTrem [71] just to avoid the out-of-tune nature of pitch-bent chords, so in some sense the artifact can be considered yet another "imperfection" of the mechanical whammy bar system. The audible difference compared to a clean pitch-shifter is minimal[7] if you don't try to play with the beating effect on purpose. There are tricks to simulate the beating effect though, we will revisit this later in Chapter 5. For the first **MAHW** prototype, only a clean pitch-shifter is used.

With the data in Figure 2.9, it is also clear to us that the maximum pitch-bend amount should be around 1 semitone, at least if we try to simulate the mechanical whammy bar from Jazzmaster. Hence, the script becomes

Other vibrato systems might have better detune capabilities. Using a different string gauge also changes the detuning curve.

```
in
"vert" wham
1 -1 biscale stor
pitshift
```

In the script above, the **`in`** ugen pushes the input audio sample from the guitar to the implicit stack, and the **`stor`** ugen converts semitones to pitch-shift ratio factor using the formula $r = 2^{s/12}$, where s is the shift in semitones and r is the ratio. An alternative is to control the pitch-shift ratio factor directly

Note that the **`biscale`** here makes a downward tilt pitch down the guitar, to be consistent with a mechanical whammy bar.

```
in
"vert" wham
1.05 0.94 biscale
pitshift
```

In fact, the two scripts do not differ that much functionally, because the mapping $f(s) = 2^{s/12}$ is almost a straight line near the no-shift condition $s = 0$. However, semitones generally make more sense in a musical context if the user wants to modify the shift amounts. Both of these approaches uses a close-to-linear mapping from the sensor value to the pitch-shift factor. This is not a physics-based mapping. One could certainly work out the math just like [36] to obtain a more accurate mapping relationship between the sensor value and the pitch-shift factor, but I don't think that is critical to get the sound I was chasing for. For a detune effect, with a pitch-shift less than one semitone, the close-to-linear mapping already sounds and feels very similar to a mechanical whammy bar.

[7] See the demo for a comparison: https://mahw.krj.st/comp/

Initially, the **pitshift** ugen is implemented using the Rubber Band library[8], which is an open-source pitch-shifting library, but its latency is quite high, so alternatively I modified the pitch-shifter from STK [19] to be a ugen. It is a standard pitch-shifting algorithm that works by cross-fading between two taps of an interpolating delay line . It has some problems that will be mentioned later, but it is already a working pitch-shifter.

See Section 7.9 of [63] and Section 5.1 for details.

With the advantage of a domain-specific language, the first **MAHW** prototype has convenient way to experiment with audio effects and different mapping options for the sensor data. For example, the script

```
"vert" wham
12 1 biscale
sine 0.2 1 biscale
in *
```

can be used to process the input audio signal with a tremolo effect (amplitude modulation) controlled by the whammy bar. In a similar fashion, the script

```
in dup
"vert" wham
1000 30000 biscale smooo
65536 delay
0.5 * +
```

implements a variable-length delay controlled by the whammy bar. Refer to the **ugens.lua** file in µsporth for a documentation of all the ugens implemented.

The software, firmware, and the 3D-printing models for the first **MAHW** prototype can be found on tig[9].

2.3 EVALUATION

The first prototype of **MAHW**, as a proof-of-concept, does achieve the downward chord bending sound I was looking for from a mechanical vibrato system, at least confirmed the assumption that the simple combination of a bar plus a variable-rate pitch-shifter is indeed sufficient to simulate a mechanical whammy bar.

[8] Rubber Band: https://breakfastquay.com/rubberband/
[9] mahw_v1: https://tig.krj.st/mahw_v1

3d print	$1-20
teensy 3.2	$23.20
joystick breakout	$2.10
joystick	$3.95
4 x M2 screws	$0.46
velcro	$3.97
total	$34.68-$53.68 USD

Figure 2.10: Cost breakdown of the first **MAHW** prototype (in 2023).

The total cost of the controller is around $35–$54 USD (Figure 2.10) plus some hooking wires and solder. If a 3D printer is available to the builder, the models will cost pennies, but otherwise ordering from a supplier would be more expensive. The cost of Teensy 3.2 takes up a majority of the budget, but it is an overkill and any Arduino-compatible microcontroller can be used as a substitute. In any case, the controller is still more affordable than most commercial mechanical or digital vibrato systems. For example, the Bigsby B5 vibrato tailpiece costs $149 USD and the Virtual Jeff PRO costs $599 USD as of late 2023.

2.3.1 Usability analysis

Despite being cheaper, the first prototype of **MAHW** is far from a usable replacement for a mechanical vibrato system on guitars.

The most concerning problem is that the controller must rely on a computer to function properly. In fact, it must be a UNIX system because the software component relies on JACK and some system calls for the serial communication, both of which are designed for UNIX, or Linux in particular, which is not something used very common in the general musician community. It will be less of a problem if the operating system requirement is only applicable to the maker of the interface, but in this case, it is the user who needs to understand and use the system, and it is going to be a huge learning curve for them, making the controller less helpful for other people. Besides, computers and computer software are often unreliable and they deteriorate [62]. PipeWire[^11] was already on the way to replace JACK as the low-latency audio solution on Linux during the development of the first prototype of **MAHW**. PipeWire still supports JACK applications, but given the relatively small user base and steep setup learning curve of JACK, it is not impossible that one day JACK becomes obsolete and the software component of **MAHW** has to be rewritten. In addition, because the controller is wired to the computer, it is impossible to use the controller

It's not fun to watch the Windows system update screen together with your audience for half a show (a true story from a thisquietarmy[^10] show a few years ago).

[^10] thisquietarmy: https://thisquietarmy.bandcamp.com/
[^11] PipeWire: https://pipewire.org/

more than a meter away from a computer. The set-up steps alone are cumbersome enough to bring so much friction into using, let alone practicing hundreds of hours and performing in public with, the hybrid vibrato system.

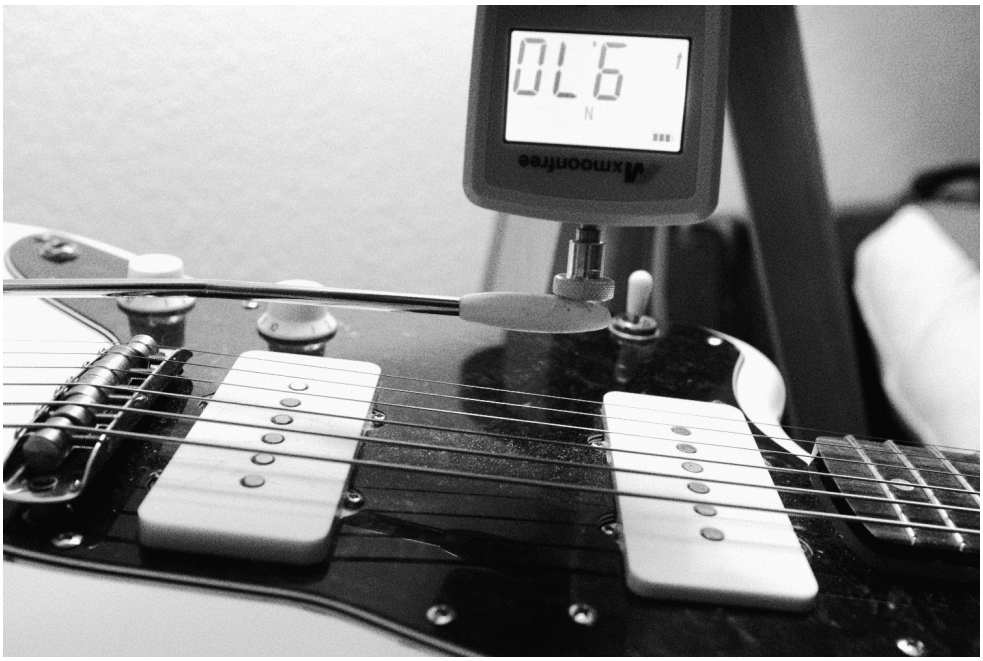


Figure 2.11: Measuring setup for the force response. The gauge is pushed to the maximum.

Apart from the dependency on a computer, the mechanics of the whammy bar needs to be re-designed. While the joystick sensor have a convenient RTC mechanism, it is not the best solution to model a whammy bar. The springs in the joystick are not designed to handle an 18-cm long bar attached to the lever. The whammy bar is too heavy compared to a thumbstick that sometimes the lever will not return to the center position properly. It is so wobbly that even shaking the guitar will trigger the vibrato. For comparison, a force gauge registered around 10 N to push the arm to the maximum on a Jazzmaster (Figure 2.11), but the force required to push the arm on **MAHW** is so low that it is registering 0.00 N on the gauge.

The ergonomics of the controller is also a problem. The arm of **MAHW** is much taller than a mechanical whammy bar due to the angle at the casing. For comparison, on a Jazzmaster, the tip of the whammy bar is 5.5 cm above the pickguard, while it is 12.3 cm on the first **MAHW** prototype. It is very difficult to strum chords and do pitch-bending at the same time. This gesture is very important for my envisioned use case.

Another problem is the quality of the pitch-shifting algorithm. I did not spend a lot of time on

this during the first prototype. The simple pitch-shifting algorithm modified from STK suffers from a transient doubling issue. Because the algorithm works by cross-fading between two taps of a delay line, there is an audible echo after the direct sound. This is particularly noticeable for percussive sounds with a prominent transient, such as a muted or tapped note. It was hence necessary to spend more time on the tuning and experimentation of other algorithms in the next prototype.

2.3.2 Fabrication complexity analysis

Even though the first **MAHW** prototype was built with only six hardware components in total, almost nothing compared to a more complex musical interface design, hardware and software rot already started to cause problems in the fabrication process, less than two years since the design was completed.

Besides the situation with JACK mentioned before, Teensy 3.2 was notably discontinued in 2023 due to the semiconductor industry deprecating the 90 nm silicon process and causing shortage in the chips used by Teensy 3.2. The microcontroller was also long out of stock before the official discontinuation. The joystick used by the first **MAHW** prototype also became out of stock on SparkFun and appears to be replaced by another model with a thumbstick cover pre-installed on the lever. It is unclear whether the thumbstick cover can be removed and whether the whammy bar stick can fit the new model without modification. The manufacturer Top-Up does not appear to distribute the component to individual customers, so it would be difficult for any DIY makers to repeat the design.

In order to better identify and compare these fabrication-related issues in later designs, it is thus necessary to conduct an analysis on the fabrication complexity of the design and identify which components are more susceptible to fabrication failures so we can be more cautious in component selection.

I will qualitatively divide the DIY fabrication complexity of components into three categories: **`Safe`**, **`Risk`**, and **`Bottleneck`**. The detailed description of each category is listed in Table 2.1. The basic idea to define these bottleneck components is that resolving a single point of failure requires redesigning multiple components in the design. The classification is not rigorous or mutually exclusive in any way to cover all the scenarios, but should be useful enough to compare design decisions and steer the component selection in the right direction.

For example, does Mouser and Digi-Key count as separate suppliers? Is using a closed-source dependency solely to support a new platform problematic?

For the design of the first **MAHW** prototype, a component-wise analysis will thus give

Complexity	Hardware	Software
Bottleneck	Difficult to fabricate without specialty machines and only available from a single individual-facing supplier. Failure to source one risks redesigning other components.	Core functionality depends on any opaque, closed-source run-time dependency.
Risk	Only available from a single supplier, but can be replaced or exactly replicated by custom fabrication without affecting other components.	All run-time dependencies are open-source, but only available on a single platform.
Safe	Available from multiple suppliers selling to individuals, or can be automatically fabricated from an affordable machine (like a desktop-size 3D printer or CNC machine).	All dependencies are open-source and available on multiple platforms.

Table 2.1: A qualitative classification of DIY fabrication complexity.

3D-printed Models (**Safe**) Easy to fabricate given a 3D-printer, which are automatic and usually affordable. Fabrication requires filaments, but they are available from multiple suppliers.

Teensy 3.2 (**Bottleneck**) Difficult to fabricate and available from a single supplier. Can be replaced by Teensy 4.0 without modifications to the hardware design, but in the case that the sole supplier becomes unavailable, changing to other Arduino-compatible microcontrollers can require modifying the mounting design of the 3D model.

Joystick (**Bottleneck**) Difficult to fabricate and available from a single supplier. Can be replaced by any joystick sensor but requires redesigning the models.

Joystick Breakout (**Risk**) Difficult to fabricate and only a single supplier for the exact joystick model used, but plenty of suppliers offer custom PCB manufacturing. Requires knowledge of PCB design when the single supplier collapses.

M2 screws (**Safe**) Difficult to fabricate but multiple suppliers available.

Velcro (**Safe**) Difficult to fabricate but multiple suppliers available. Can be replaced by other brands.

µsporth (**Risk**) All dependencies are open-source, but only available on Linux using JACK.

In total, the design has 2 bottleneck components and 2 at-risk components. All the bottleneck

components has possible replacements, but they require changing the 3D model, so replicating the design concept should not be a problem if the builder has some 3D modeling skills.

2.3.3 Customization analysis

Besides the fabrication, there is another aspect of the design to be considered. How much and how easy can the design be customized if the user is not satisfied with the experience?

For the software and audio effect aspect of the design, the customization is fairly easy. Because of the use of µsporth, changing the parameter mapping and audio effects can be done without recompiling the software. New effects, like a better pitch-shifting algorithm, can be added if the user knows how to program in C. There is a limitation though, because the controller does not offer a MIDI [43] or OSC [83] output, it has limited interoperability with third-party synthesizers and guitar pedals, so it is not possible to use it with a commercial pitch-shifter. This feature is not difficult to add using an open-source library, such as liblo[^12], but everything has to be implemented by the user.

Hot swapping is even possible.

On the mechanical side of the design, the customization options are more limited. If the user is not satisfied with the current almost non-existent, less than 0.01 N force feedback, the options are to either shrink the whammy bar to reduce the distance to the fulcrum or find a joystick with stronger springs, but since joysticks are not designed for this application, it is not always easy to compare the spring strength.

The joystick has a 1.6 N operating force but it is reduced by the leverage of the extension bar (discussed later in Figure 4.9).

If the user wants to customize the height of the whammy bar, it is possible to change the angle of the triangle on the casing (Figure 2.4a) or shrink the whammy bar arm, though because the joystick has a 50° travel (Figure 2.12), the height cannot be too low, or the arm might hit the pickguard when fully depressed.

2.3.4 Setting the goals

Given these considerations, the goals for the next prototype were then determined. In order to reduce the friction to using the digital–mechanical whammy bar, the next prototype needed to be completely self-contained—audio in, audio out, no computer involved. This would require moving the entire pitch-shifting processing to the microcontroller.

The second goal was to experiment with alternate pitch-shifting algorithms and see if they could fix the transient doubling problem and improve the audio quality.

[^12]liblo: https://github.com/radarsat1/liblo

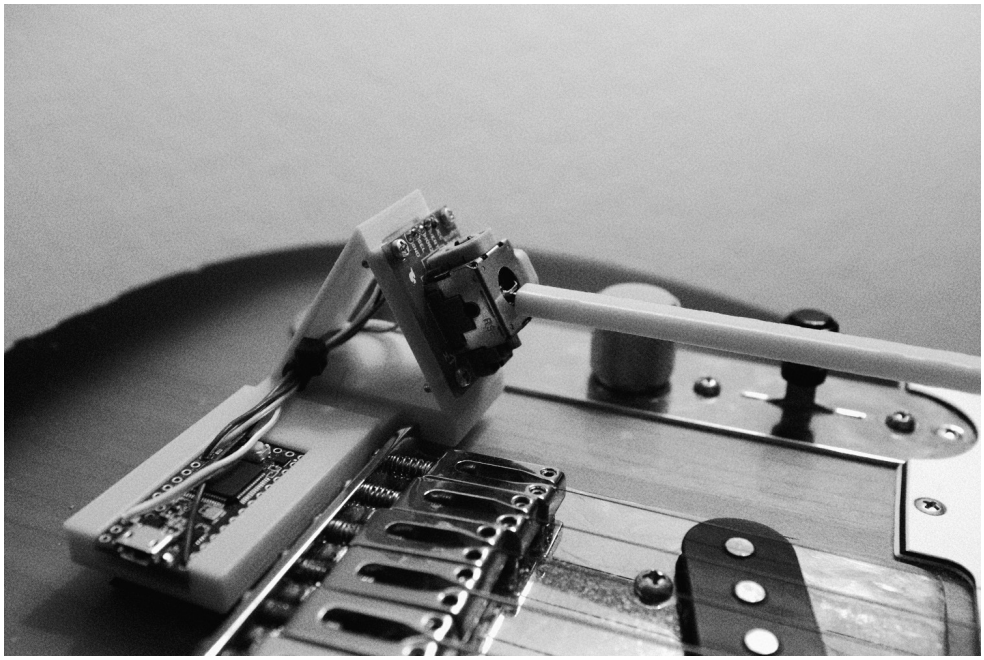


Figure 2.12: The height of the whammy bar is limited by the travel of the joystick.

The final goal was to experiment with alternate mechanical and sensor configurations to improve the robustness and force feedback of the mechanical system. The implementation of these added goals should ideally not increase the fabrication complexity substantially and allow an easier customization.

Chapter 3: Intermission

GIVEN the goals to design a better mechanical system, it was necessary to take a pause and review some existing mechanical and sensor designs to look for inspirations. As mentioned before, the main challenge in the mechanical design of a vibrato system is the return-to-center mechanism. It is not hard to get something moving. The difficult part is to get it return back to where it was.

Figure 3.1: RTC using compression spring in shampoo dispenser.

Luckily, the use of RTC is not restricted to a vibrato design. It is so ubiquitous that even many everyday household items contain it, like a shampoo dispenser (Figure 3.1). In fact, we could directly salvage the spring from there and use in the design if we don't care about reproducibility, but, again, it is unlikely for other people to use the same shampoo and the durability of the spring in a disposable bottle is questionable—the spring is already having trouble pushing the nozzle back up without wiggling. However, we do see a typical design pattern on how to mount a compression spring. It is using a small segment of the tube, matching almost exactly as the inner diameter of the spring, at the base as the retainer to keep the spring in place.

You never expect what you can discover when bored sitting on a toilet.

This design is very similar to the Duesenberg Les Trem (Figure 1.3), Bigsby vibrato tailpiece [8], except both designs have the spring retainer at the top as part of the whammy bar. For Les Trem, the lever is attached directly to the bridge to control the bridge height. For Bigsby, the lever bar is

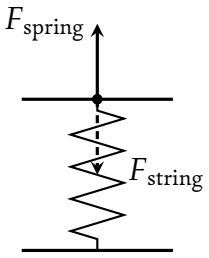


Figure 3.2: Simplified force diagram of Les Trem, Bigsby, and Jazzmaster designs.

rotating a roller bridge that pulls the strings forward or backward to modulate the tension. These two designs heavily depend on the tension of the guitar string to keep the compression spring in place. Without strings attached, the compression spring will be floating in the spring holder. The spring can keep in place only because the string tension applies a downward force on the spring (Figure 3.2). The spring is compressed far enough so that the restoring force balances out the downward force from the strings. Whenever this equilibrium is broken by an external force, either the spring or the string will drag the system back to equilibrium when the force is released. It is worth noting that only the pitch-down action is restored by the spring tension. The pitch-up action (pulling the whammy bar up) is restored by the string tension. Without strings attached, this design cannot restore the pull-up motion of the whammy bar.

The Jazzmaster vibrato design [27] also uses a compression spring and is in equilibrium with the string tension. The working principle is very similar. Tilting the tailpiece forward lowers the height of the bridge and compresses the spring, while tilting the tailpiece backward raises the bridge and increases string tension. The notable difference with the previous two designs is that the spring is retained by a long screw passing through its center. The design does not require the presence of strings to hold the spring in place.

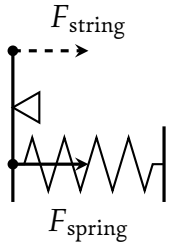


Figure 3.3: Simplified force diagram of the Stratocaster vibrato design. The pivot is actually on the front side of the guitar body.

Besides compression springs, there are also many vibrato designs that use extension springs, such as the Stratocaster vibrato design [25]. It is still a balanced system between the strings and the

spring, but there is a screw on the baseplate tied to the guitar body that acts as a pivot (Figure 3.3). Because the spring is on the other side of the pivot, hooked to a sustain block, it must be an extension spring and apply the restoration force in the same direction as the strings.

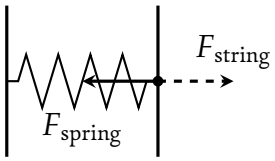


Figure 3.4: Simplified force diagram of the Stetsbar.

Extension springs are also used in the Stetsbar [72], but the springs are hooked directly to the movable bridge plate. The design is very similar to Figure 3.2, just in a different direction (Figure 3.4). A unique point in Stetsbar's is that the pivot of the lever arm is located at the center of the bridge, so the entire bridge piece is symmetric except for the lever arm. The same bridge piece can be installed on either a normal or a mirrored guitar, and the only thing that needs to be switched is the lever arm.

A problem of using these mechanical vibrato designs directly in a digital design is that they all have some dependency on the tension from guitar strings. For example, if we remove the force from strings in Les Trem, the spring cannot even stay in place and restore a pull-up motion. However, there are also plenty of digital whammy bar designs to take note from. The history of digital whammy bars is far older than Virtual Jeff PRO, just in a different form, not as an add-on but part of an instrument. The earliest commercial product was a guitar-like synthesizer called SynthAxe [2] in 1985. Less than 100 units were made [70], but the synth even made its way into Michael Jackson's Bad World Tour from 1987 to 1989, performed by Christopher Currell[^1]. The mechanics of SynthAxe is not clear to me, but one thing they mentioned in the patent is the use of a linear Hall effect sensor to measure the distance of depress. There is also similar projects like the You Rock MIDI Guitar[^2], though the project is essentially dead, and its design is not documented anywhere.

Yet another example of a digital musical instrument died without support.

The Guitar Hero game controller [74] is probably the most widely sold product equipped with a digital whammy bar. Its mechanical design is very interesting. It uses an extension spring to function as a torsion spring, similar to the pitch wheel design in Moog Sonic Six [31] and

[^1] Michael Jackson - Bad Groove (Interlude Instrumental) | Bad World Tour | Live At Brisbane | 1987: https://www.youtube.com/watch?v=lV1kKdnB-rg

[^2] You Rock Guitar: https://www.kickstarter.com/projects/yourockguitar/yrg-pro-professional-grade-midi-guitar

Weston Precision Audio PRO2021 [82] (mentioned before in Section 2.1). This design requires a quite a few mechanical pieces. The whammy bar is tied to a semicircular base connected to a potentiometer. Two arms will be rotated along with the base, but the two arms can only rotate in one direction and are blocked in the other direction by a stopper (Figure 3.5). If the whammy bar is pressed down, one of the arms will be rotated while the other is blocked by the stopper, stretching the spring away from its equilibrium, hence creating a restoring force. The scenario will be the similar for the opposite rotation.

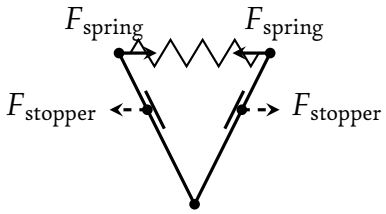


Figure 3.5: Guitar Hero controller's mechanic design (again, simplified). The right arm can only move to the right. The left arm can only move to the left.

Actual torsion springs are used by Virtual Jeff PRO [78] and by the whammy bar controller created by Martin Kristoffersen and Trond Engum [48]. The working principle of the torsion spring is similar to the Guitar Hero controller, but the two arms are replaced by the two terminals of the torsion spring. Virtual Jeff PRO uses a linear Hall effect sensor to measure the rotation angle. The whammy bar on Virtual Jeff PRO rotates a shaft mounted with two magnets, and the Hall effect sensor will pick up any changes in the magnetic field as the shaft rotates. Kristoffersen and Engum's controller uses a simpler design with an Alps Alpine RK08H miniature potentiometer to measure the rotation angle.

Figure 3.6: Flat leaf spring used in the Maestro Vibrola.

Some people might believe that springs can only be manufactured in a coil shape, but it is not

always the case. Anything that exerts a restoring force when deformed can function and be called a spring. Most materials exhibit some kind of elasticity. The Maestro Vibrola is a good example that takes advantage of this property (Figure 3.6). The entire vibrato is just a piece of bent metal sheet forming a pivot with the guitar body. Pushing the whammy bar deforms the metal sheet and loosens the string tension, but the metal will restore to its original shape and reset the tension when the whammy bar is released.

The *elasticity* of materials—this property turns out to play an important role in the second prototype of **MAHW**, which will be discussed in the following chapters.

Chapter 4: The Second Take: Hardware

Realizing the importance and potential of elasticity was the starting point of the second prototype of **MAHW**—elasticity not only applies to the usual spring materials, such as carbon steels and alloy steels, but also 3D-printable materials like PLA and PETG, etc. This revelation led to a mechanical design that is based on a custom designed, 3D-printed spring (Figure 4.1).

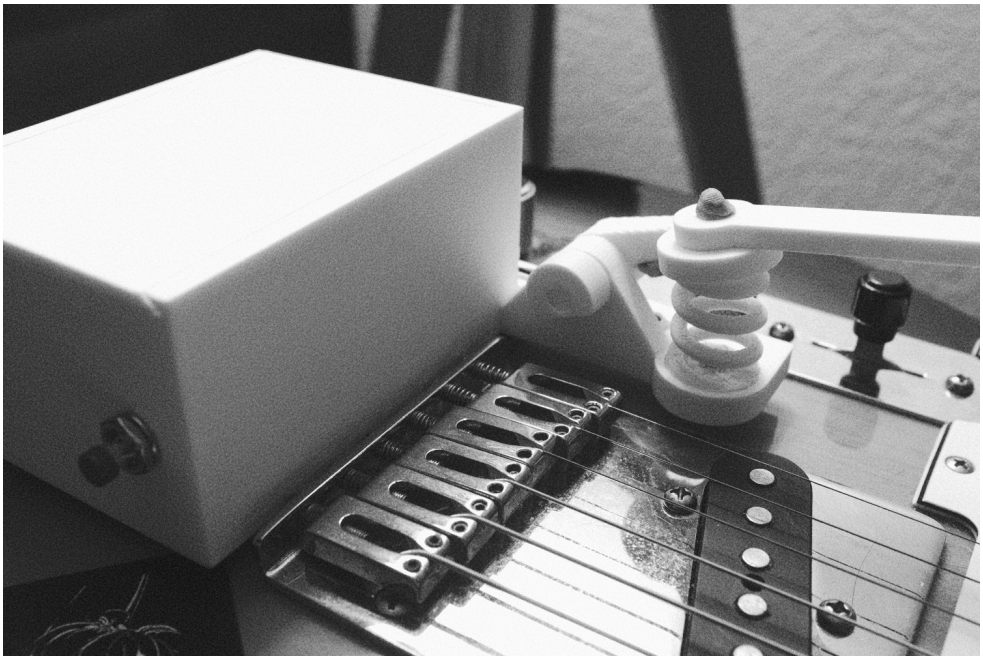


Figure 4.1: The second prototype of **MAHW**.

4.1 3D-printed springs

As discussed previously in Chapter 3, many different RTC designs can function equally well, yet the most challenging part in designing an RTC mechanism is to find a suitable and reproducible spring for the design. Factors such as the length, wire diameter, and spring diameter directly determine the force feedback and the ergonomics of the design.

The springs inside of most mechanical vibrato systems are custom designed to counter-balance the tension of guitar strings. While it is not difficult to order or salvage a spring specially designed for mechanical vibrato systems and use it in a digital vibrato design, because there are no guitar strings providing the counter-balancing force to hold the spring compressed, the length of the spring would be too long, which is not ideal to create a better ergonomics. It would be desirable to have a large selection of custom springs to experiment with the optimal force feedback and spring length.

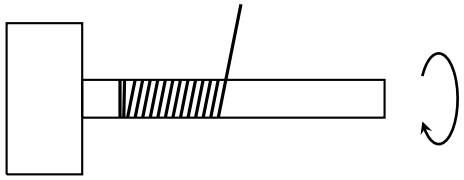


Figure 4.2: Spring winding on a mandrel using a lathe.

It is possible to find kits of springs readily available online and in hardware stores, though they rarely come with the length and force feedback desired by the designer. Winding a custom spring is also an option. This is traditionally done by rotating a piece of music wire (high carbon spring steel) on a mandrel using a lathe [41] (Figure 4.2). Because operating a lathe is a manual process, there is no guarantee that the resulting spring is repeatable. The tension in the steel is also quite dangerous for home fabrication. Manufacturers such as Lee Spring[^1] provide the service to wind custom springs using CNC machines, but this is typically more expensive and takes weeks to iterate designs.

Figure 4.3: 3D-printed springs.

The solution to these problems in **MAHW**, however, came from a surprise discovery that it is possible to print working springs using a regular 3D printer (Figure 4.3). A seemingly hard material can be made flexible by solely changing the shape of the object.

I was just printing out spring models to check dimensions.

It is ridiculous how far the "similar look gives a similar function" philosophy is going.

[^1] Lee Spring: https://www.leespring.com/

These 3D-printed springs have a limited range of extension, but they work fine as compression springs. The 3D printing technique provides an automatic way to fabricate these custom springs consistently, greatly reducing the fabrication complexity of the design. With the flexibility of 3D printing, it is also possible to add mounting holes directly on the springs to simplify the design of connectors—something simply not possible with metal springs.

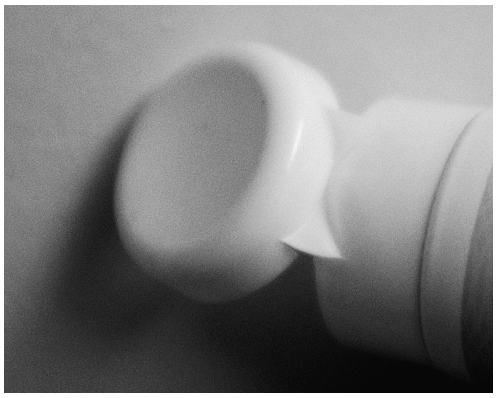


Figure 4.4: Compliant mechanisms are used on face cleanser bottles to provide a spring-like closing force when the cap is half open to prevent leaking.

This is certainly not a new discovery. The concept of *compliant mechanisms* [40] has been used in many interactive mechanical designs to provide spring-like behaviors using solely the flexibility of materials, usually plastic or metal, but also applicable to micro-structures using photolithography. Some examples include snap buckles, one-piece bottle caps (Figure 4.4), and, when taken to the extreme, even a Nerf gun[^2]. In particular, 3D-printed spring designs can be seen in many academic projects such as MyoSpring [51] and Ondulé [39], though both projects are not related to musical interface design. 3D-printed compliant mechanisms have the potential to create very unique custom force-feedback mechanisms while keeping the fabrication reproducible, but they are severely under-explored in the interface design of commercial digital musical instruments and controllers, which is dominated by an almost cliché usage of buttons, knobs, and sliders .

referencing to the name of the book *PUSH TURN MOVE* by Kim Bjørn [9].

4.1.1 Fabrication of 3D-printed springs

There are some tricks involved when printing a compression spring using a 3D printer. Because the coils of a spring are hanging in the air without any support underneath, when printed with an FDM 3D printer, it is necessary to let the slicer generate some support structures for the overhang coils . The spring model in the second **MAHW** prototype is sliced using PrusaSlicer[^3]. The

Support might be unnecessary on an SLA 3D printer. I didn't try this.

[^2]World's Smallest Nerf Gun Shoots an Ant: https://www.youtube.com/watch?v=9c2NqlUWZfo
[^3]PrusaSlicer: https://www.prusa3d.com/page/prusaslicer_424/

default support style in PrusaSlicer does not work well for springs with small gaps. It is better to use the **Snug** option instead of the default **Grid** (Figure 4.5).

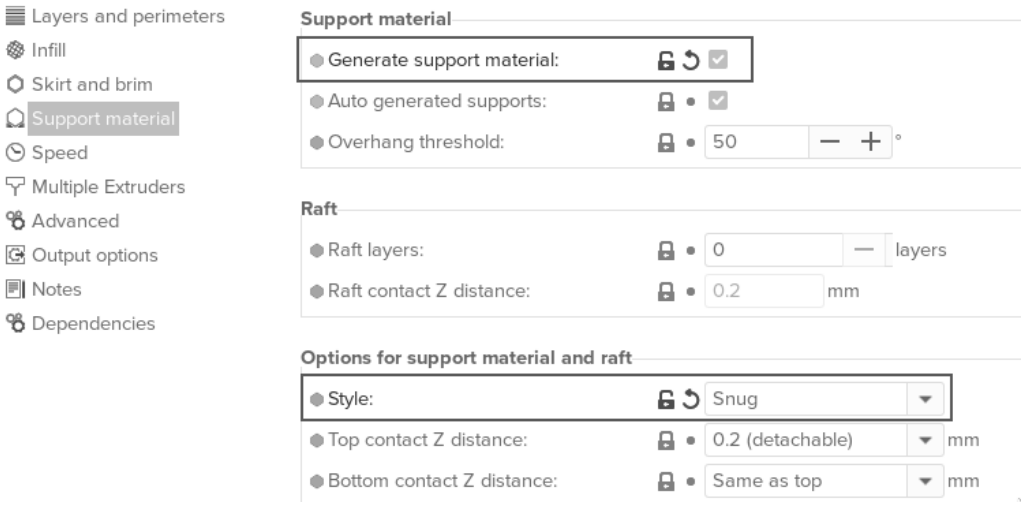
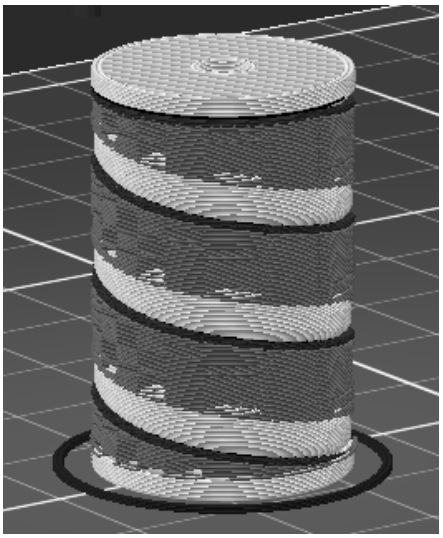
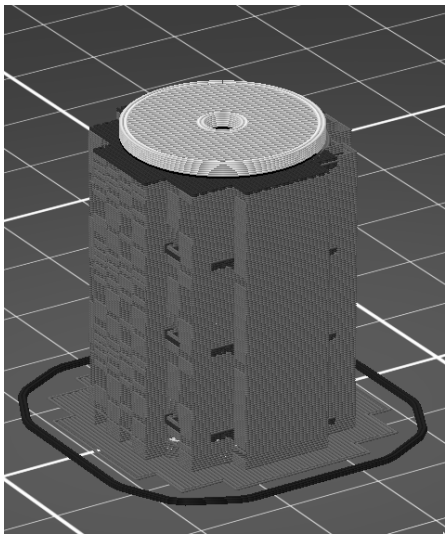


Figure 4.5: Support options for printing springs in PrusaSlicer.

This option forces the slicer to only generates support material right below the gaps (Figure 4.6a) instead of surrounding the gaps (Figure 4.6b). This will make removing the support material much easier later and also consume less support material. Notice that the spring must be placed upright. It will not work when printed sideways.



(a) Support generated by **Snug**.

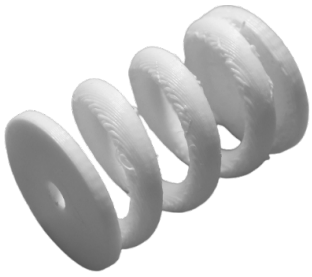


(b) Support generated by **Grid**.

Figure 4.6: Support structure (colored dark) comparison in PrusaSlicer.

(a) 3D-printed spring before removing the support.

(b) Removing the support with a knife.



(c) 3D-printed spring after removing the support.

Figure 4.7: Removing the support material from the 3D-printed spring.

The spring can be printed with the regular PLA filament. After the printing is finished, the spring will look something like Figure 4.7a. It is still not usable at this point since the support needs to be removed. This can be done using a kitchen knife. Just carve along the coils and the support material can be peeled off (Figure 4.7b). With some patience, the support material can be removed almost perfectly. After peeling off the support, the spring will look like Figure 4.7c. Notice that the spring in **MAHW** is customized with flat bases and mounting holes. If the base is not flat, manual support structures might be needed to align the spring vertically.

4.1.2 Spring customization

The spring model used for **MAHW** is modified from an OpenSCAD script written by Chris Wallace[4]. I added bases and mounting holes to the model for easier mounting. However, there are more customizations possible than these. The entire force feedback of the spring can be changed by tuning outer diameter, wire diameter, number of coils, and pitch (distance between coils) of the spring.

The force feedback of the 3D-printed spring can be characterized by the spring rate

See Chapter 4–1 of [13] for details.

$$k = \frac{F}{y},$$

which determines the amount of restoring force F from the spring when compressed by a distance of y from the rest position (Figure 4.8). The higher the spring rate k, the greater the force feedback F.

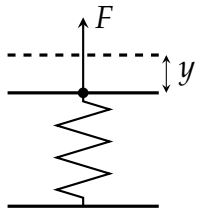


Figure 4.8: Compressed spring by a distance of y.

The spring rate k can be approximated by

See Chapter 10–3 of [13].

$$k \approx \frac{d^4 G}{8 D^3 N},$$

where d is the wire diameter of the spring, D is the outer diameter of the spring, N is the number of (active) coils, G is the material's modulus of rigidity.

See Chapter 3–8 of [13].

[4] Spring model: https://github.com/KitWallace/openscad/blob/master/spring.scad

Assuming the modulus of rigidity G is constant for the material, we can conclude that increasing the wire diameter d, reducing the outer diameter D, or reducing the number of coils N can increase the force feedback from the spring, and vice versa. However, it should be noticed that increasing the number of coils N might also increase the length of the spring, which can lead to a longer compress distance y and hence a greater force feedback F when fully compressed.

To verify this, I printed seven springs customized with different parameters and measured the restoring force of each spring when fully compressed with a force gauge. Spring 1 is the reference, and it is the spring used in the second prototype of **MAHW**. Spring 2 and 3 have a different wire diameter d, but other parameters are the same. Spring 4 and 5 have a different number of coils N. Notice that the pitches are also changed accordingly to maintain the spring height. Spring 6 and 7 has a different outer diameter. The result of the experiment is documented in Table 4.1.

Number	Outer Diameter (mm)	Wire Diameter (mm)	Pitch (mm)	# Coils	Force (N)
1	8	1.8	9	3.5	8.6
2	8	**2.1**	9	3.5	12.5
3	8	**1.2**	9	3.5	2.3
4	8	1.8	15	**2.1**	broken
5	8	1.8	7	**4.5**	6.8
6	**11**	1.8	9	3.5	4.5
7	**6**	1.8	9	3.5	17.4

Table 4.1: Impact of spring parameters on the fully-compressed force feedback.

We can see that the force feedback does follow the intuition from the formula—increasing the wire diameter gives a higher force feedback, decreasing the wire diameter gives a lower force feedback, etc.—though the exact numbers might be different since 3D-printed wires are not exactly as dense as metal. Spring 4 is broken because the pitch is too high. The angle of the spiral becomes so steep that the material is no longer flexible (just consider that, if the number of coils approaches 0, the spring essentially becomes an incompressible stick). Spring 7 is also very hard to compress and on the edge of breaking. Again, it is because the spiral becomes steeper (if the outer diameter approaches 0, the spring also becomes a stick).

In the second prototype of **MAHW**, the whammy bar tip and the spring approximately form a 6:1

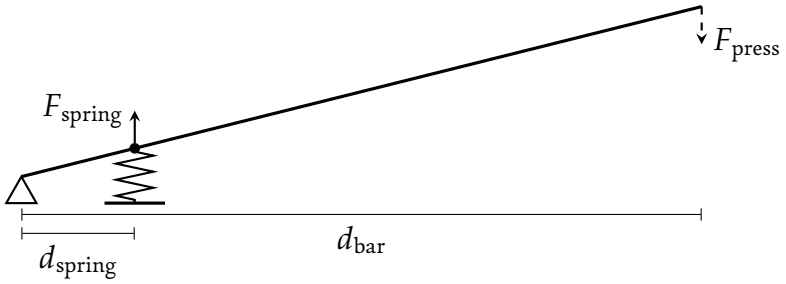


Figure 4.9: Lever formed by the spring and the hinge, where $d_{\text{bar}}/d_{\text{spring}} \approx 6$.

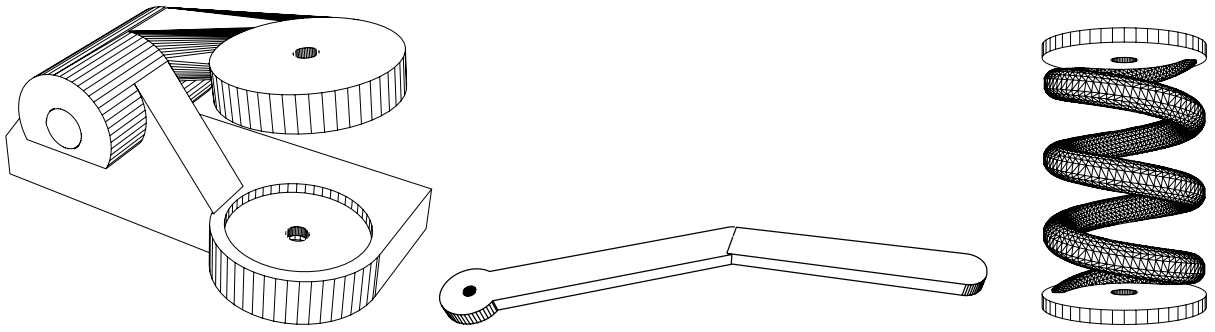
leverage (Figure 4.9), so the actual force feedback from the whammy bar is around

$$\frac{8.6\,\text{N}}{6} \approx 1.43\,\text{N}.$$

It is still around 7 times smaller than a mechanical whammy bar, but already much better than the first prototype registering 0.00 N on the force gauge. Tweaking the parameters can slightly improve this force feedback, but it is not likely to match the 10 N force feedback from a mechanical whammy bar. More research needs to be done on compliant mechanisms in the future to come up with a better design for the spring. Maybe a different 3D-printing material would also help.

4.2 HINGES

The mechanical part of the second **MAHW** prototype is constructed with five 3D-printed components (Figure 4.10). The final assembly is shown in Figure 4.1. The shape of the design is mostly based off the Bigsby vibrato tailpiece [8], though the mechanics are quite different.

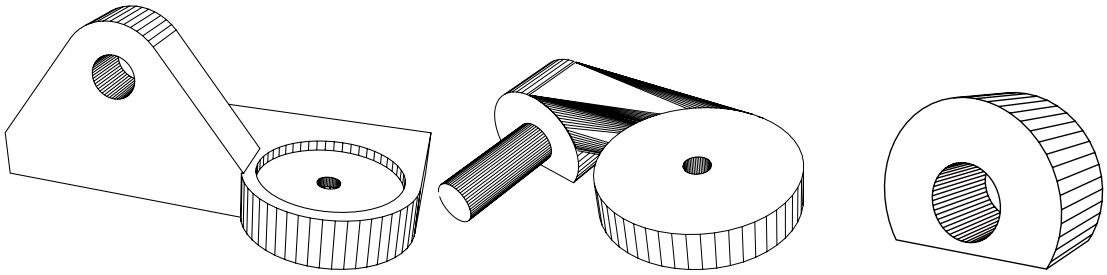


(a) The base. **(b)** The whammy bar. **(c)** The spring.

Figure 4.10: Mechanical part of the second **MAHW** prototype.

The base (Figure 4.10a) of the mechanical part forms the hinge of the whammy bar. It is constructed with three separate components (Figure 4.11). The shaft of the arm (Figure 4.11b) is

inserted in the bore of the base plate and is sealed by friction on the other end with a cap (Figure 4.11c). Some glue can be applied to the cap to enforce the seal, but it is not strictly necessary. The rotation of the shaft is smooth enough that no bushing is needed at the bore.



(a) The base plate. **(b)** The arm segment. **(c)** The hinge cap.

Figure 4.11: The base of the second **MAHW** prototype.

In fact, a 3D-printed hinge does not have to be in separate pieces. As long as there is some clearance between the shaft and the bore, the hinge can be printed in one piece and the shaft will naturally separate with some twisting. However, in the second **MAHW** prototype, the arm segment (Figure 4.11b) needs to be printed with the spring mounting base facing down (Figure 4.12), so it is more convenient to separate the arm segment and the cap.

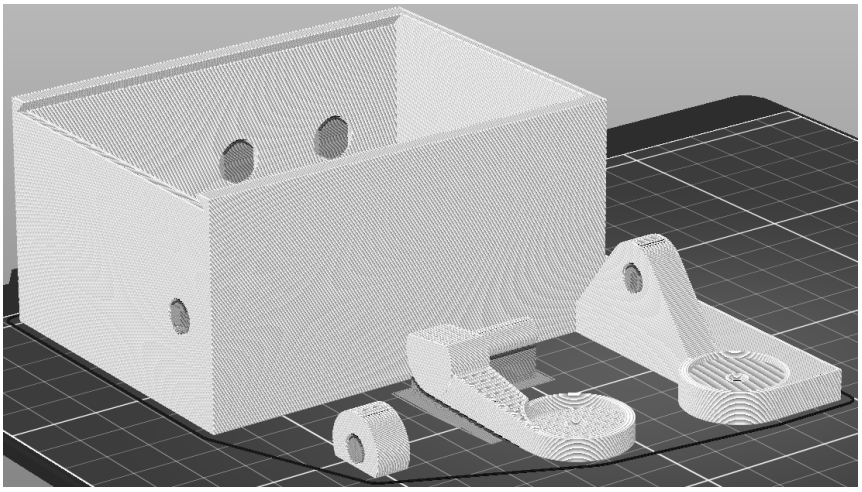


Figure 4.12: Orientation of components in PrusaSlicer. The **Snug** style is used to generate the support.

The base, the spring, and the whammy bar form another hinge, screwed together by an M3 bolt-nut (see Figure 4.1), similar to the hinge construction in scissors (Figure 4.13). Tightening the bolt makes the whammy bar harder to rotate sideways, and vice versa. I applied some Blu-Tack to the nut in case it becomes loose and falls off.

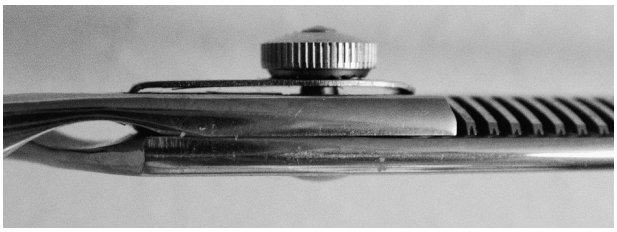


Figure 4.13: Hinge construction in scissors.

The bottom of the spring is fixated to the base by another set of bolt-nut. There is a hole at the bottom of the base (Figure 4.14) to conceal the nut and prevent it from sticking out. This nut needs to be fastened using a pair of tweezers.

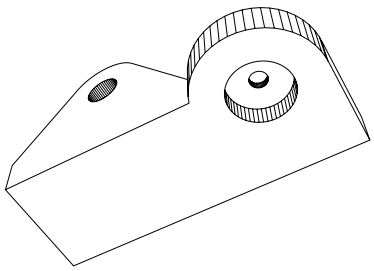


Figure 4.14: Bottom view of the base.

Notice that in the second **MAHW** prototype, both ends of the spring are fixed. This is different from many mechanical vibrato designs mentioned in Chapter 3, because here there is no string tension to hold the spring in place. This is needed to provide a restoring force when the whammy bar is pulled up by a small amount.

4.3 SENSOR

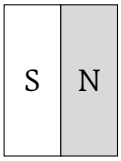
Unlike the first prototype, the second prototype uses a separate sensor to measure the distance between the whammy bar and the base. The mechanical design of the lever arm leaves a movable space between 1.0 to 1.7 centimeters between the spring and the base (Figure 4.15).

To capture such a small distance with high resolution, a Texas Instruments DRV5056A1 ratiometric linear Hall effect sensor is attached to the base of **MAHW** using Blu-Tack, and an 8mm×3mm axial nickel magnet is attached right above it on the arm of the hinge. This combination forms a high-resolution distance sensor. Blu-Tack is a removable adhesive used heavily in the second prototype of **MAHW**. Although it is removable, the adhesion becomes stronger over time, making it very convenient for prototyping as a replacement for glue.

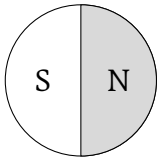
It is very important that the magnet here is an axially magnetized magnet (Figure 4.16a), where

Figure 4.15: Sensor arrangement of the second **MAHW** prototype.

the face of the disk has only one polarity, as opposed to a diametrically magnetized magnet (Figure 4.16b), where the face has two polarities. They look exactly the same, so people often get confused. The diametric magnet is used with a rotary Hall effect sensor to accurately measure angles. This sensor can be found in the foot pedal design of Electrosteel [69]. Because diametric magnets are much harder to source than axial magnets as individual makers, the linear Hall effect sensor is a much better option in terms of reproducibility.



(a) Axial disk magnet.



(b) Diametric disk magnet.

Figure 4.16: Types of magnet.

A linear Hall effect sensor measures the flux density of a magnetic field, which is proportional to the distance between the magnet and the sensor, despite not in a linear fashion. Generally, the sensor will have a higher sensitivity at a closer distance, though the non-linearity becomes less important when the operating distance is in a small range of less than 1 cm The A1 variant of DRV5056 is the most sensitive version of the Hall effect sensor, reacting to magnetic flux at 200 mV/mT. A different sensor variant can be used to tweak the responsiveness and the taper of the whammy bar. If the spring is shorter or the magnet is swapped to a neodymium magnet, a lower sensitivity, higher dynamic range variant would be more suitable.

See Equation 5 in the data sheet [42].

Figure 35 in the data sheet [42].

Unlike the most basic four terminal Hall element [64], DRV5056 only has three terminals—**VCC**, **GND**, and **OUT** (Figure 4.17). It has a built-in amplifier to amplify the sensor output to a voltage level

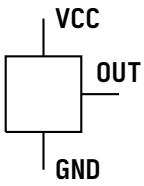


Figure 4.17: Sensor pin configuration of DRV5056.

directly proportional to **VCC** (hence the name ratiometric), maximized at around 3.3 V. The output of the sensor can be directly connected to any analog pin of a microcontroller, an Electro-Smith Daisy Seed[5] in the case of the second **MAHW** prototype. An optional 100 nF capacitor can be placed between **VCC** and **GND** to reduce power supply noise. Because the sensor is not attached to a PCB, a surface mount ceramic capacitor can be soldered directly between the **VCC** and **GND** leads of the sensor.

The combo between DRV5056A1 and nickel magnet is almost a perfect sensor for the distance in our interest. The sensor outputs around 3.2 V at 1 cm distance and around 1.2 V at the 1.5 cm resting distance, yielding around 2 V of dynamic range[6]. The exact voltage of the resting position can float up and down due to gravity, depending on whether the controller is laid down or tilted sideways, so there needs to be an adjustment and a normalization procedure to prevent DC drift and make the reading more consistent. This is done by first sampling the resting voltage, subtracting the sampled average from a reading, and then finally normalizing it by the range.

This simple scheme works, but it can also be replaced by a more sophisticated scheme, such as the regularization in Plaquette[7].

```
float base = 0;
float range = 1;

static void readjust()
{
  base = 0;
  for (int i = 0; i < 1000 ; i++)
    base += hw.adc.GetFloat(0);  // max of GetFloat = 1
  base /= 1000;
  range = 1-base;
}
```

The adjusted value is hence

```
float shift = (hw.adc.GetFloat(0)-base)/range;
```

[5] Electro-Smith Daisy Seed: https://www.electro-smith.com/daisy/daisy
[6] See https://mahw.krj.st/hall.mp4 for a demo of the sensor.
[7] Plaquette: http://sofapirate.github.io/Plaquette/regularizing.html.

This readjustment routine is triggered during startup. Because it doesn't make sense for the user to reboot the device just to readjust the sensor, I had to add a push button to the second prototype of **MAHW** (Figure 4.18) connected to a digital pin of the Daisy Seed. This is the only button in the entire design.

Figure 4.18: The kill switch on the second prototype of **MAHW**.

It would be a waste of sensor input if the button is used solely for sensor readjustment, so I came up with the idea to make it dual-function as a kill switch. Pressing the button mutes the guitar sound temporarily, and releasing it brings the sound back. This is a very popular modification on regular electric guitars to achieve a glitchy sound. It is just implemented digitally in **MAHW** with a smoothed envelope on the volume.

The readjustment procedure will be triggered when the button is held down for more than 3 seconds. Because the guitar is conveniently already muted by the kill switch when the button is pressed down, there won't be any audible artifacts from the readjustment. Usually, the user does not need to perform readjustments regularly, but the option is there in case the user does need to reset the resting voltage without restarting the device.

In addition to the adjustment, there is another line of defense against pitch drifting by setting a dead zone for the adjusted sensor value.

```
if (shift < 0.08 && shift > -0.04)
  shift = 0;
```

The thresholds are chosen empirically to prevent the wobbles from body movement. Because the audio processor is running a pitch-shifting algorithm, the discontinuity in sensor values is not a big problem. It suffices to perform the dead zoning without any kind of smoothing.

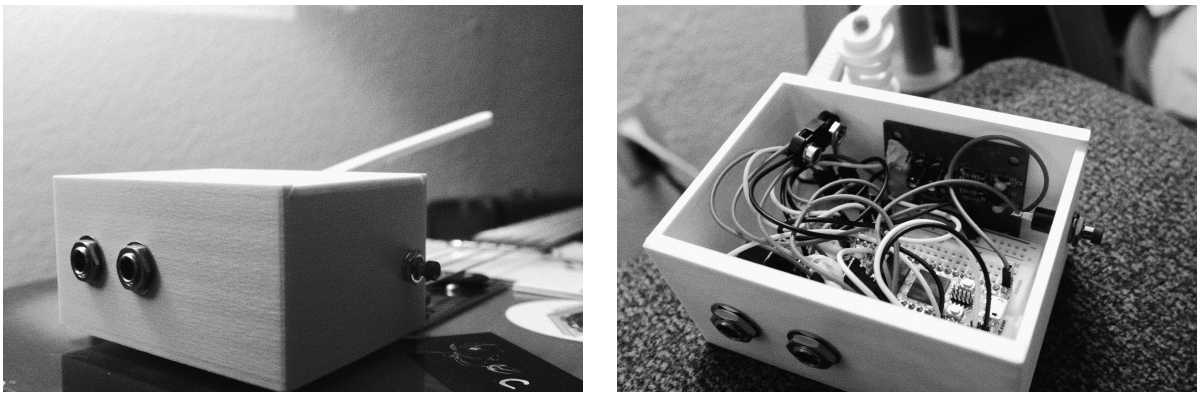
The linear Hall effect sensor is advantageous in that it has zero mechanical components and only the sensor needs to be powered (the magnet is completely passive). The response time is around 10 µs for DRV5056, which leaves plenty of room for audio processing to achieve the typical 10 ms maximum action–sound latency in musical interface controls [81, 57]. However, if a linear Hall effect sensor is not available, it is also possible to consider replacing it with a reflexive infrared sensor, such as Vishay CNY70, or an infrared photodiode/phototransistor with a daylight blocking filter, though one thing I notice about these infrared sensors is that the light blocking filters on them are not perfect. Even the LED blink on the Daisy Seed can be captured by the sensor when the distance is close enough, so light interference might become a problem with a more difficult lighting situation on stage. Alternatively, a force sensing resistor (FSR) or a Velostat pressure sensor can be placed below the spring to measure the pressing force instead of the distance, but a problem with this approach is that it only senses the downward bending motion and the sensor will be under mechanical stress all the time.

A photoresistor will not work here due to its millisecond-level response time and light interference.

4.4 AUDIO I/O

As a goal set in Section 2.3.4, the audio processing of the second **MAHW** prototype becomes completely self-contained. The device runs on a 9V battery and takes the guitar input directly (Figure 4.19a). It becomes a truly plug-and-play interface. The only extra thing needed is a short 1/4″ cable to connect the guitar output to the input jack on the controller. The controller is automatically powered on when the output jack is plugged in, so there is not even a power switch .

Battery life discussed later in Chapter 6.



(a) Output on the left. Input on the right.

(b) The internal circuits.

Figure 4.19: Audio input and output of **MAHW**.

The audio path of the second **MAHW** prototype is shown in Figure 4.20. The guitar output is connected to the input jack. The high impedance guitar signal is converted to a low impedance

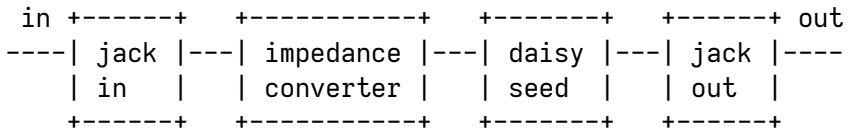


Figure 4.20: Audio path of **MAHW**.

signal with a custom-built impedance converter and is sent to the analog-to-digital converter (ADC) on the Daisy Seed (pin 16). The Daisy Seed reads the sensor signal from the Hall effect sensor and applies a pitch-shifting algorithm (discussed in Chapter 5) to the input signal. The output signal will be sent to the output jack via a digital-to-analog converter (DAC) at pin 18.

4.4.1 Audio jacks

The audio jacks of the second **MAHW** prototype have some hidden features to improve the controller's user experience. The input and output jacks of **MAHW** are not regular mono 1/4″ audio jacks. There are some special mechanical switch designs built into the jacks.

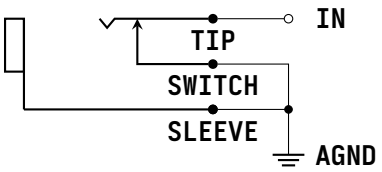


Figure 4.21: Input jack wiring. The connection between **SWITCH** and **TIP** will be open when the input is plugged in.

The input jack used here is a Switchcraft 112AX (tip closed). It can be replaced by a Switchcraft 12A, just in a different casing. This particular audio jack is used to mute the input audio when nothing is plugged in at the input. Switchcraft 112AX has an extra **SWITCH** terminal that is connected to a mechanical switch. The **SWITCH** terminal by default is connected to the **TIP** when nothing is plugged in, and the connection will be broken when an input jack is inserted. If we ground the **SWITCH** terminal to the **SLEEVE** (Figure 4.21), the audio input **IN** will be grounded by default to prevent humming. The input **IN** will become the actual audio signal only when an input jack is plugged in.

The output jack here is a Switchcraft 113X (tip open and isolated "make" circuit) or a Switchcraft 13. It is used to power on the device when an output jack is connected. Switchcraft 113X has two extra switch terminals **SWITCH1** and **SWITCH2** (Figure 4.22). The two switch terminals are isolated from the audio terminals **TIP** and **SLEEVE**. They are open circuit by default, and a connection will

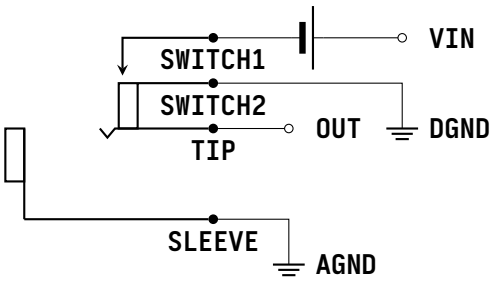


Figure 4.22: Output jack wiring. The **SWITCH1** and **SWITCH2** terminals will be closed when the output is plugged in.

be made between **SWITCH1** and **SWITCH2** only when an output jack is plugged in. If we connect one of the switch terminals to the negative end of the battery and connect the other switch terminal to the digital ground of Daisy Seed, the switch essentially becomes the power switch of the microcontroller.

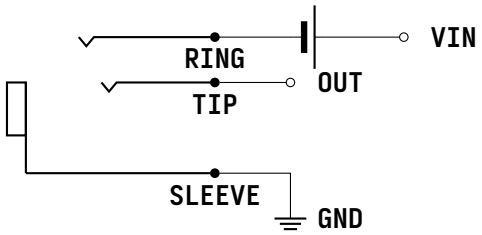


Figure 4.23: Traditional audio jack power switch wiring. The **RING** and **SLEEVE** will be connected when an mono jack is plugged in.

Notice that the output jack power switch design in MAHW is a bit different from the traditional audio jack power switch design commonly found in guitar pedals. In the traditional design, the audio jack is a standard stereo jack and the **RING** and the **SLEEVE** terminals will be connected when a mono jack is plugged in (Figure 4.23). The problem with this approach is that the unregulated power ground and the signal ground share the same path. This is fine for a low-speed audio circuit, but for a high-speed digital circuit, the power/digital ground is going to contain a lot of switching noise, and it is necessary to control the return path of the audio signal to avoid the digital return path. Hence, we need an isolation between **DGND** and **AGND**. They will be connected elsewhere in the circuit with a ferrite bead to choke the high-frequency digital noise.

4.4.2 Impedance converter

The Daisy Seed is a very self-contained digital signal processing module, and all it needs to process an audio signal is to connect the input pins 16/17 to an audio input and the output pins 18/19 to an audio output. The I/O wiring is usually the only external circuitry when using the Daisy

Seed, but there is a subtle problem when the input signal is coming from the magnetic pickup of an electric guitar.

It is not that the audio codec on the Daisy Seed cannot receive any signal at all, but rather, if we bypass the input signal to the output on the Daisy Seed

```
void process(AudioHandle::InputBuffer in, AudioHandle::OutputBuffer out,
             size_t nframes)
{
  for (size_t i = 0; i < nframes; i++) {
    out[0][i] = in[0][i];
    out[1][i] = in[1][i];
  }
}
```

and compare it with the original raw guitar signal, the Daisy-bypassed signal sounds comparably darker, or lack of high frequency content[8]. This will be a problem for a pitch-shifter because ideally the signal chain should be transparent and not coloring the input signal at all when there is no pitch-shifting.

The root of this problem is that most ADCs do not have a high input impedance. They are mainly designed to accept line out signals with a low output impedance of a few hundred ohms. The audio codecs used by Daisy, Asahi Kasei AK4556 (in Rev 4) and Wolfson WM8731 (in Rev 5), only have an input impedance of only around 30 kΩ.

or lower according to the data sheet of the chips [46, 59].

For a passive magnetic guitar pickup, the output impedance can range from around 5 kΩ for a single coil pickup to around 10 kΩ for a humbucker pickup [45]. The 30 kΩ input impedance of the ADC is already too low compared to the usual 1:10 output-to-input impedance ratio for impedance matching.

However, the coloration of the audio signal is not about the resistive nature of the circuit. If the circuit is truly resistive (Figure 4.24), the impedance mismatch of the source impedance R_s and load impedance R_ℓ would only attenuate the output voltage V_ℓ compared to the input voltage V_s

$$V_\ell = \frac{R_\ell}{R_s + R_\ell} V_s.$$

This is not a big problem if it is what we receive in the digital domain. The gain loss can be easily compensated by a simple multiplication on a DSP processor, only at a cost of losing some

[8] Listen to http://mahw.krj.st/imp.wav for a comparison. The first part is the raw guitar signal. The second part is the bypassed signal from Daisy. The two parts are loudness matched.

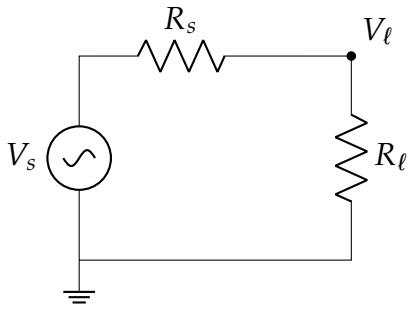


Figure 4.24: Purely resistive voltage divider.

dynamic range.

The unique problem with the magnetic pickups on electric guitars is that the inductance and the parasitic capacitance are what defines the *tone* of a pickup. While a magnetic pickup is just a piece of wire wrapped on a magnetic core in a loop, it is much more than the resistance of the wire. The looped topology of the wire gives the pickup an inherit inductance, and because the looped wires are stacked next to each other, the sandwich of conductor–insulator–conductor also leads to parasitic capacitance. A magnetic pickup is better modeled as an RLC circuit [45] (Figure 4.25), where L_s is the inherit inductance of the pickup, usually ranging from 1–10 H, and C_p is the parasitic capacitance in the circuit. The pickup itself contributes around 70–100 pF of stray capacitance, but the instrument cable from the guitar to the ADC can add another 50–100 pF of capacitance, so the combined C_p is around 120–200 pF.

For example, see the specification of Hosa cables[9].

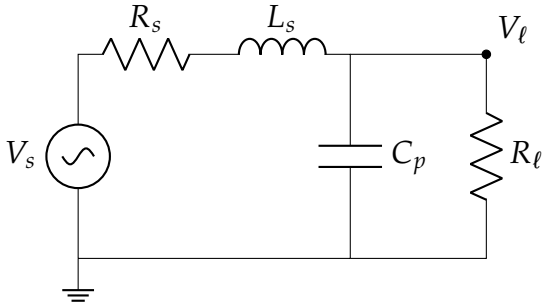


Figure 4.25: RLC model of magnetic pickups.

The LC component of the circuit model produces a characteristic resonant peak in the frequency response of the pickup. However, this resonant peak is highly dependent on the load impedance R_ℓ. If the load impedance is too low, the resonant peak will be tamed down into a regular high-cut filter, and the cut-off frequency for the high-cut filter will be lower and lower as the load

[9] Hosa Edge Guitar Cable CGK-000: https://hosatech.com/wp-content/uploads/2020/12/CGK-000-Specifications.pdf

impedance decreases.

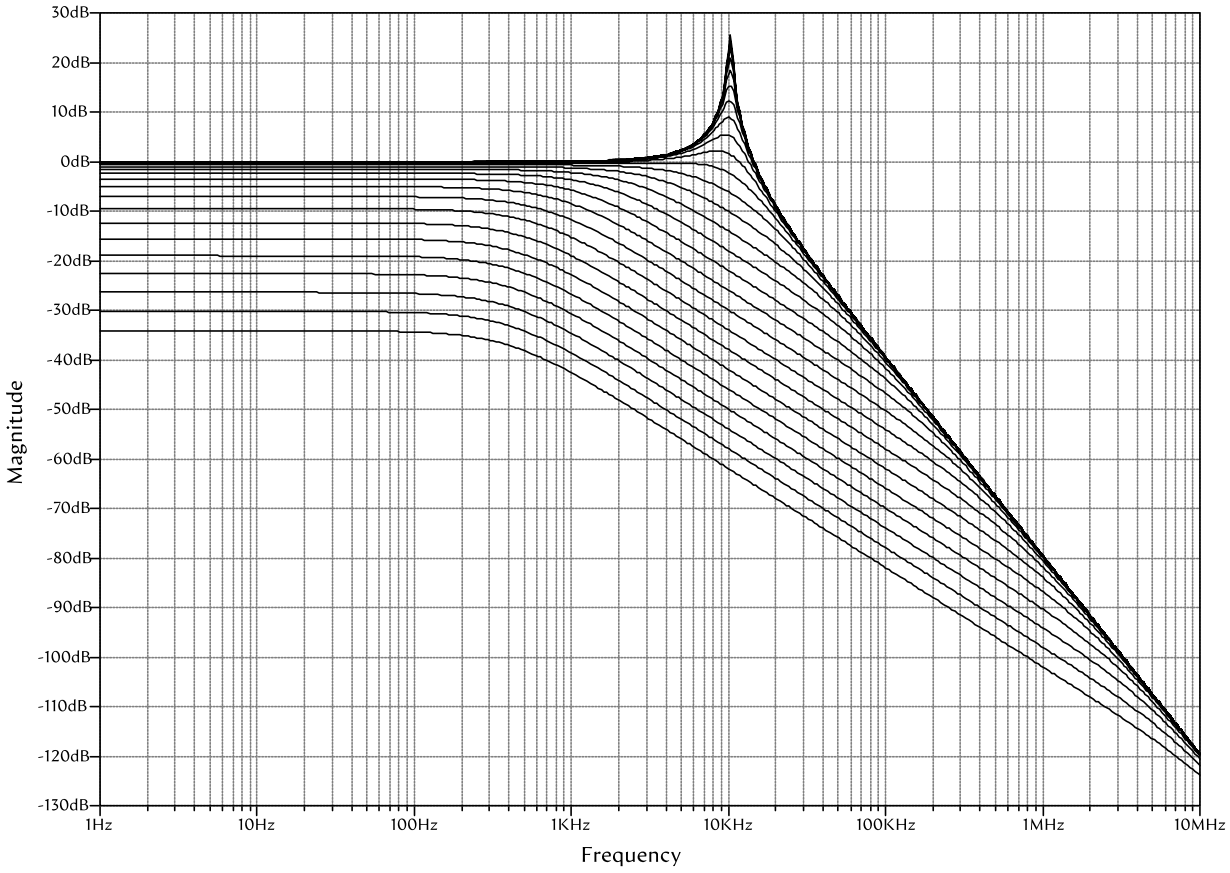


Figure 4.26: Frequency response of V_ℓ as a function of load impedance R_ℓ swept from 100Ω to 10 MΩ at 5 points per decade. The bottom line is the 100 Ω load impedance, and the top line is 10 MΩ. The impedances of any adjacent lines in between are separated by a factor of $10^{1/5} = 1.58$.

This effect can be visualized by sweeping the resistor value of R_ℓ and plot the output frequency response at V_ℓ compared to V_s. Figure 4.26 is produced by choosing $R_s = 5$ kΩ, $L_s = 2$ H, $C_p = 120$ pF, and sweeping R_ℓ from 100 Ω to 10 MΩ. The frequency response for the 100 Ω load impedance is at the bottom and the one for the 10 MΩ load impedance is at the top. To retain the characteristic resonant peak from the pickup, R_ℓ needs to be greater than around 500 kΩ. This requires an even larger input impedance than what the usual 1:10 output–input impedance ratio suggests.

There are two ways to solve the problem. The first one is to ignore the problem all together and market the controller as having a "warm tone". The second one is to boost the input impedance of the ADC with an *impedance converter*, or a *buffer*, placed before the ADC input. The second

approach is apparently the better option for **MAHW**.

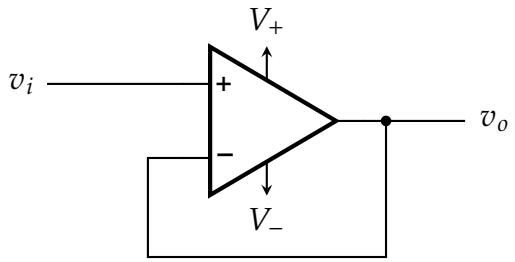


Figure 4.27: Op-amp voltage follower.

There are many impedance converter designs using bipolar junction transistors (BJT), op-amps, and field effect transistors (FET). The most common design for this is a voltage follower using an op-amp (Figure 4.27), though if we want to use the regulated 3.3 V output from Daisy Seed, the op-amp selections are quite limited [10]. Typical op-amps do not reach the entire range of voltage supply, so headroom becomes a problem (need at least 2 V for guitars), especially for a humbucker pickup with a higher output. If we stick with 3.3 V, the op-amp has to be rail-to-rail, operate at low-voltage, and have a high input impedance. If we use the 9V supply, we have to add an extra voltage regulator. With these many constraints, using an op-amp unnecessarily complicates the design.

Regular jellybean op-amps like TL072 breaks down below around 4 V.

Many low-voltage op-amps only have around 100 kΩ input impedance.

The simpler way is to use a BJT in an emitter follower (common collector) configuration (Figure 4.28). This design is used by many guitar pedals, such as the Ibenez Tube Screamer [24].

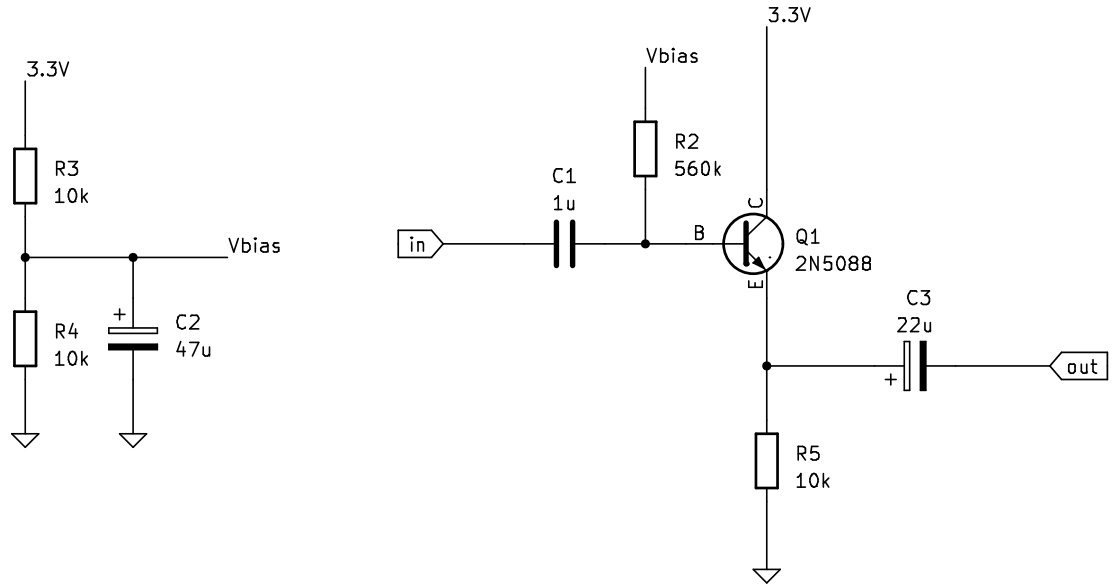


Figure 4.28: Basic BJT emitter follower.

Because 2N5088 is a high-gain transistor ($h_{fe} > 350$), the circuit can boost the input impedance to nearly the value of **R2**, around 560 kΩ (see the Tube Screamer Analysis [24]), and it only loses around 0.6 V relatively to the 3.3 V voltage rail, giving around 2.7 V of headroom. Notice that the bias here is not carefully designed. It should be above the midpoint voltage $3.3/2 = 1.65$ V, but in this design the bias is slightly below the midpoint due to the non-zero base current from the transistor, thus the headroom is further reduced slightly. However, the design appears to work fine in practice and requires less resistor types, so there is no need to over-engineer.

This naïve textbook circuit design is good enough for most people, unless you decide to use the controller in downtown Montréal. There is a giant FM antenna (Figure 0) right in the center of the city, on top of Mont Royal (Figure 4.29). This monstrosity interferes with any transistor devices near the McGill campus, including the naïve BJT emitter follower.

It affects op-amp voltage follower as well.

Figure 4.29: Antenna location in the city.

NOTE

I swept the entire FM and AM radio spectrum to match the radio interference I got from the BJT. The coordinate of the station can be found by searching the matched frequency online, which ended up to be on top of the mountain. This is why residents in the neighborhood of McGill are often confused and frustrated by their vintage and hand-made audio equipments picking up FM radio signal. The answer is right outside their window.

Figure 4.30: Breadboard wire picks up 82 mV peak-to-peak interference at around 98 MHz.

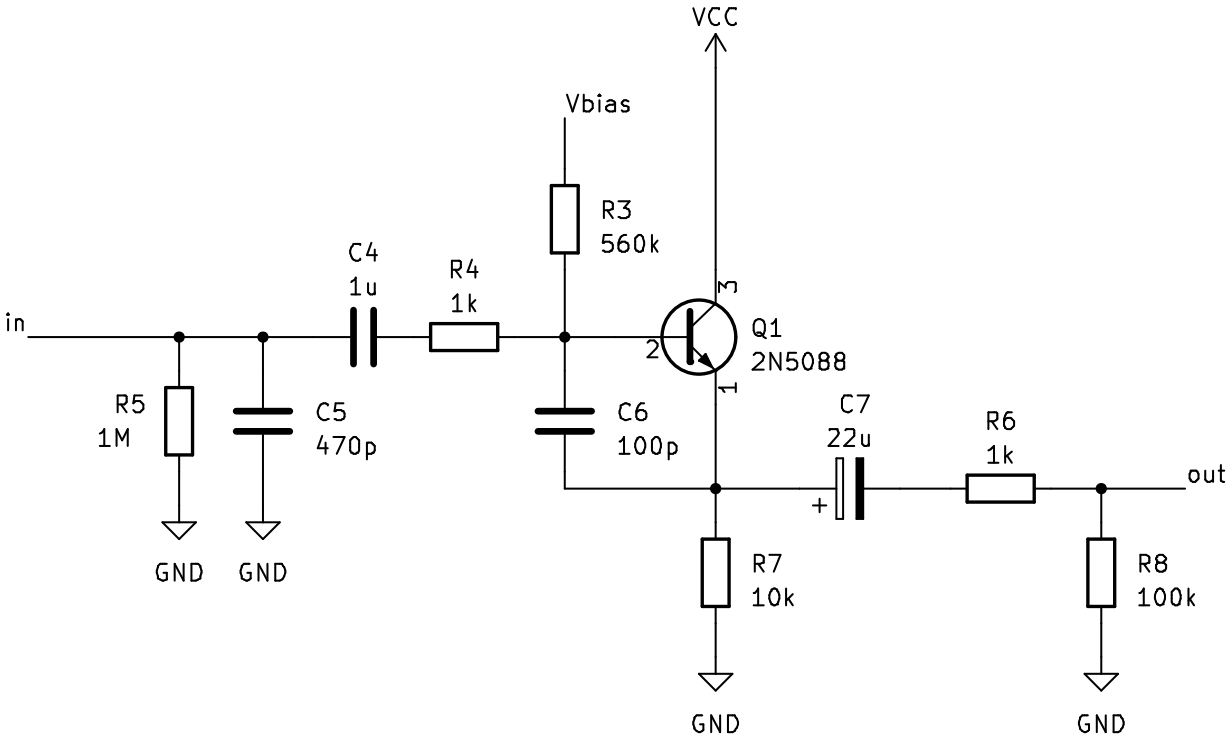
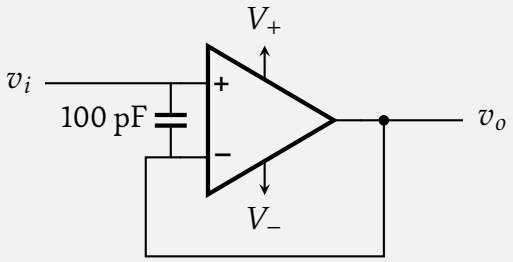


Figure 4.31: Enhanced BJT emitter follower.

The antenna is so close to the McGill campus that even a simple breadboard wire can pick up around 50-100 mV of FM radio frequency signal (Figure 4.30). To make the circuit usable, some additional RF filtering needs to be added (Figure 4.31). The basic idea of the fix is to reduce the effective bandwidth of the amplifier. The most important component in the new design is **C6**, the capacitor between the base and the emitter of the transistor. This gives RF signals a low impedance path to bypass the non-linear transistor p–n junction, preventing the RF demodulation. Effectively, it acts as a low-pass filter in the RF frequency range. When picking the capacitor value, it is necessary to use SPICE[^10] to plot the frequency response at output to verify that the capacitor doesn't attenuate the audio frequency range (< 20 kHz) by too much, around 1 dB at maximum.

NOTE

For an op-amp design, the fix that works for me is to put a pF range capacitor between the positive and negative input terminal, physically placed as close to the terminals as possible.



This design can be found in the schematic of Yamaha HS5/7/8 speakers [20].

In addition to the RF filtering, some other protections are added to the circuit as well. The base resistor **R4** is added to prevent RF oscillation [18]. **R6** is added for current limiting. **R5** and **R8** are added as a discharge path for the capacitors in case of capacitive loads. **C5** is an additional RF filtering at input, though this one is not as important as the base–emitter capacitor.

To best minimize parasitic inductance, a custom PCB is made for the impedance converter (Figure 4.32). The only thing important in the PCB layout is the location of RF filtering capacitors. The base–emitter capacitor **C6** needs to be as close to the transistor as possible to reduce loop area and prevent parasitic inductance. The filtering will not be as effective if **C6** is far away from the transistor. Other parts of the layout are not critical due to the low frequency nature of audio signals (relative to digital signals).

[^10] For example, ngspice: https://ngspice.sourceforge.io/

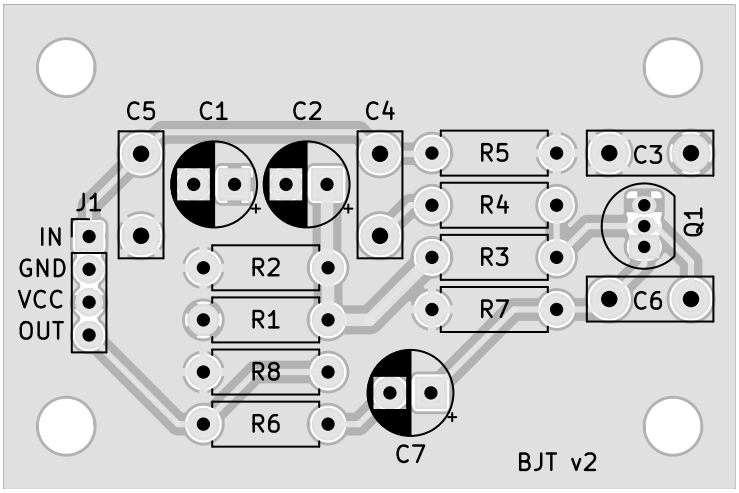


Figure 4.32: PCB of the impedance converter board.

The **IN** terminal of the PCB is connected to the input jack, **VCC** is connected to the 3.3 V regulated power pin of Daisy Seed (pin 21), and **OUT** is connected to the audio input pin of Daisy Seed (pin 16). There is no impedance conversion necessary at the output since the output impedance of Daisy's audio codec is low enough that no high frequency loss is audible. However, I did notice the audio codec of Daisy Seed Rev 5 was picking up RF signal from the output audio cable. It was necessary to put a 100 pF capacitor between the audio output pin (pin 18) and analog ground (pin 20) to filter out the radio interference. Again, the location of the RF filter capacitor is important. It needs to be placed close to the transistor device. The filter will not work when the capacitor is placed on the output jack.

Rev 4 doesn't have this problem.

Chapter 5: The Second Take: Pitch-shift

After discussing the hardware component of the second **MAHW** prototype, the only thing left to introduce is the firmware. Unlike the first prototype, µsporth is no longer used for sensor and audio effect mapping. Despite its flexibility and convenience for live-coding, µsporth does not quite fit into the plug-and-play design of the second prototype. Updating the script becomes cumbersome when the editor and the interpreter are not on the same device. The computer and the microcontroller have to be connected by a USB cable to upload a new script. To reduce friction in using the controller, there should be ideally no configuration or scripting needed for the setup. Hence, the only thing left on the Daisy Seed firmware is the sensor reading and the signal processing algorithm. The sensor processing was already introduced in Section 4.3, so this chapter is going to focus on the pitch-shifters implemented in the second **MAHW** prototype. Notice that there were multiple pitch-shifting algorithms implemented in the second prototype. This chapter will cover the designs that work fairly well with **MAHW** and their caveats.

For a pitch-shifting algorithm to be usable in **MAHW**, there are some very specific requirements. Because the algorithm is used in an interactive controller, it has to run in real-time, ideally having a low latency (not too far from the 10 ms guideline [81]). To simulate the pitch-bending effect, the algorithm must support smooth, variable pitch-shifting rates, not discrete steps. Because most people use pitch-bending as an embellishment, not something used very frequently, the rate of the pitch-shifter will be fixed at 1 (no shift) most of the time. Therefore, the pitch-shifter should ideally not color the sound when there is no shift. Even if there is a shift, undesirable artifacts from the pitch-shifter should be minimized. Finally, the implementation of the pitch-shifter should be simple and efficient enough that even a microcontroller can run it.

The MCU of Daisy Seed is STM32H750, which is an Arm Cortex-M7 running at a clock rate of 480 MHz.

Most open-source pitch-shifting libraries, such as Rubber Band[^1] (based on phase vocoder [30],

[^1] Rubber Band: http://breakfastquay.com/rubberband/

used in FFmpeg[^2]) and SoundTouch[^3] (based on WSOLA [77], used in Firefox[^4]), have trouble satisfying these goals all at once. They are mostly designed for video and audio streaming, so the action/audio latency and variable rate are not a concern. From my testing, the latency of Rubber Band and SoundTouch are over 100 ms even at the lowest quality configuration, which makes them unusable in an interactive setting. Even if the latency is tolerable, the amount of external dependencies they have is going to cause a headache when compiling for a microcontroller. Therefore, I ended up have to implement my own pitch-shifter.

An easy way to check if the latency is good enough is to strum a guitar at funk tempo and check if you can follow a beat. I generally can tolerate around 20 ms latency at maximum.

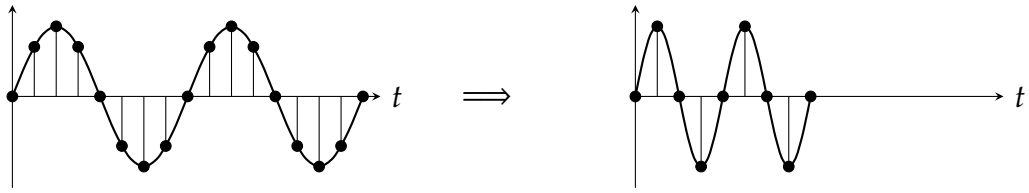


Figure 5.1: Reading signals at a different rate scales the frequencies but also changes the duration.

The history of pitch-shifters can go back to almost 100 years ago. Engineers at that time were mostly interested in reducing the bandwidth requirement for signal transmission, so a natural solution was to shift down all the frequencies by some factor to compress the spectrum before transmission, then shift up on the receiving end to recover the signal. The duration of the signal needs to be kept the same, or there will be no net gain in compression. One of the earliest methods to implement this technique is documented in the 1924 patent by Norman R. French and Manvel K. Zinn [34]. It uses a rotating pickup circulating a "sound pipe" (similar to a delay line or a buffer) at a different playback speed to scale the frequency using Doppler's effect (Figure 5.1). The signal is fed into the "sound pipe" at the regular playback speed, but due to the rotary construction of the pick-up, segments are skipped on the transmitting end and repeated on the receiving end, so the overall duration is not changed. This important insight is shared by almost all the time-domain pitch-shifters—to shift down a signal, the reading speed is slower than the playback speed, so some segments need to be discarded; to shift up a signal, the reading speed is faster than the playback speed, so some segments need to be repeated. Different pitch-shifting algorithms just handle the transitions differently.

Later analog designs, such as the optical implementation based on a sound film projector in Dennis Gabor's seminal paper series *Theory of Communication* in 1946 [35], mostly followed the same

[^2]FFmpeg: https://github.com/FFmpeg/FFmpeg
[^3]SoundTouch: https://www.surina.net/soundtouch/
[^4]Firefox: https://searchfox.org/mozilla-central/source/dom/media/AudioStream.cpp

rotary pick-up principle, but for the transitions, an additional windowing is added to smooth out the transition and multiple taps of the signal are overlapped to reduce the amplitude modulation from windowing. A history of these pre-digital pitch-shifters can be found in Marlens's 1966 paper [54].

The rotary pick-up design even makes into the digital era of audio processing. The first real-time digital processor for pitch-shifting was the Eventide H910 Harmonizer in 1974. It is based on the same principle as the analog rotary pick-up, but the digital implementation is achieved by cross-fading multiple taps of a delay line [3]. This algorithm is marketed as a "glitching" pitch-shifter due to its amplitude modulation artifacts that are introduced by the imperfect cross-fading of out-of-phase signal segments. However, because the artifacts of this algorithm are fairly musical and the implementation is simple, it is used ubiquitously in commercial products (can be easily recognized by the artifacts).

A review of digital pitch-shifters can be found in [23].

as opposed to the later "de-glitched" H949 Harmonizer in 1979, which is based on cross-correlation to determine the best transition splice point [1].

5.1 ALGORITHM: XFADE

This is also the pitch-shifting algorithm used in the first prototype of **MAHW**. The initial code was based on the pitch-shifter of STK [19], but the STK implementation is very echoy and not particularly usable for guitars. However, I later noticed that it is possible to tweak the implementation a little bit to get a more usable effect. To introduce the tweak, it is necessary to first understand how the algorithm works.

The cross-fade pitch-shifting algorithm is based on a fractional delay line. The most basic setup is a buffer with a write pointer **wp**. Whenever a new audio sample comes in, it is written to the buffer at the location specified by the write pointer **wp**, and then **wp** increments by 1 so that the next sample will be written to the next slot[5]. For a regular delay line with a fixed delay, the read pointer **rp** always trails behind **wp** by a fixed amount d

$$\mathbf{rp} = \mathbf{wp} - d.$$

There will be wrapping necessary in practice when implemented with a circular buffer.

In this case, the read pointer **rp** moves 1 sample per tick, so the signal is read at the normal playback rate, neither compressed nor expanded (first plot of Figure 5.2). The reading speed of the signal can be changed by varying this delay time d (i.e. simulating a moving pick-up). If the delay time d increases by 0.5 sample every tick, then the read pointer **rp** will only move by 0.5 sample per tick. This results in a slower reading speed, hence shifting down the signal by an octave (sec-

Sample value at fractional positions will be interpolated from adjacent samples. See fractional delay discussions in [68].

[5] See https://mu.krj.st/delay/ for a detailed introduction on delay lines.

ond plot of Figure 5.2). Similarly, if the delay time d decreases by 1 every tick, the read pointer will move by 2 samples per tick, resulting in a faster reading speed and shifting up the signal by an octave (third plot of Figure 5.2).

```
fixed delay
0   1   2   3   4   0   0    tick 0  (d=3)
    ^rp         ^wp
0   1   2   3   4   5   0    tick 1  (d=3)
        ^rp         ^wp
0   1   2   3   4   5   6    tick 2  (d=3)
            ^rp         ^wp


delay increases by 0.5 every tick (shift factor=0.5)
0   1   2   3   4   0   0    tick 0  (d=3)
    ^rp         ^wp
0   1   2   3   4   5   0    tick 1  (d=3.5)
      ^rp           ^wp
0   1   2   3   4   5   6    tick 2  (d=4)
        ^rp             ^wp


delay decreases by 1 every tick  (shift factor=2)
0   1   2   3   4   0   0    tick 0  (d=3)
    ^rp         ^wp
0   1   2   3   4   5   0    tick 1  (d=2)
            ^rp     ^wp
0   1   2   3   4   5   6    tick 2  (d=1)
                    ^rp ^wp
```

Figure 5.2: Read and write pointers at different moving delay conditions.

Following the trend, it is possible to generalize that the increment of delay Δd and the shift factor α are related by the formula

$$\Delta d = 1 - \alpha.$$

From Figure 5.2, we can notice a problem. If $\Delta d \neq 0$ and the buffer is a finite circular buffer, then the read pointer **rp** will necessarily catch up with the write pointer **wp**. When this happens, there will be a discontinuity in the output signal, because the output suddenly jumps from the latest sample to the oldest sample in the buffer (or the reverse).

The solution in the cross-fade algorithm is to apply a window function to the output samples read from the delay line (Figure 5.3). The edge of the window always trails **wp**, so the output is close to 0 when **rp** approaches the discontinuity at **wp**. Although the discontinuity problem is fixed, the

window introduces an amplitude modulation to the output signal. To address this problem, an additional tap from the delay line is added to the output. The two taps are separated by an offset equal to half the buffer length, so the amplitudes of the window at the two read pointers always add up to 1. Essentially, when one of the read pointers reached the maximum of the window, the other will be at the minimum, creating a cross-fading effect. This cannot eliminate the amplitude modulation artifacts entirely, but it will be much milder than using one tap of delay.

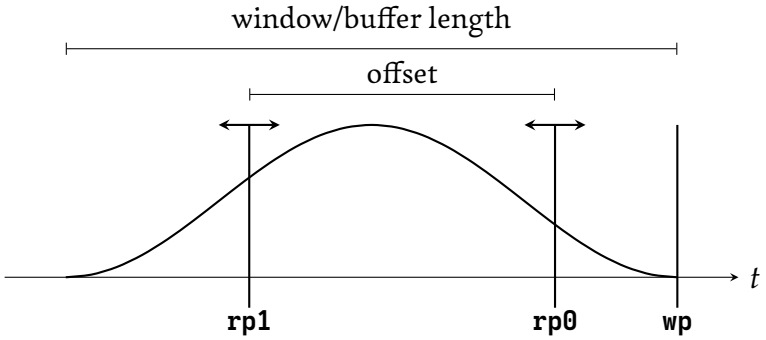


Figure 5.3: Cross-fade pitch-shifting algorithm.

Hence, the cross-fade algorithm has two parameters that can be tuned. One is the offset length between the two taps of delay, and the other is the window function. The offset controls both the delay between two read pointers and the cross-fade length. If the offset is too large, the two taps of delay will be recognized as two separate events, which makes the output sound very echoy. If the offset is too short, the frequency of the amplitude modulation artifacts will rise into the audible frequency range and make the artifact more audible and dissonant, especially with extreme pitch-shifting factors. The window function is less important in this situation from my experience. Any symmetric smooth functions with both ends tapering to 0 should work, though there are some general considerations about the cross-fade on the difference between amplitude-preserving and energy-preserving, as discussed by Geraint Luff [53] and Robert Bristow-Johnson [12].

The limit for perceiving sounds as two separate events is around 40 ms [55].

In STK, the offset length is 2500 samples and the window function is a triangular window. The main reason for the echoy artifacts was that the offset length was way too large. 2500 samples is around 56.7 ms in the 44100 Hz sampling rate, which is much larger than the 40 ms threshold to distinguish two separate events. After reducing the offset length to 1000 samples, the echo becomes much more subtle. Additionally, the window function is changed to a Hann window

$$\text{hann}(d) = 0.5 - 0.5\cos(\pi d) \quad d \in [-1, 1]$$

indexed by the delay d, since it is differentiable and hence a smoother function than the triangular

window.

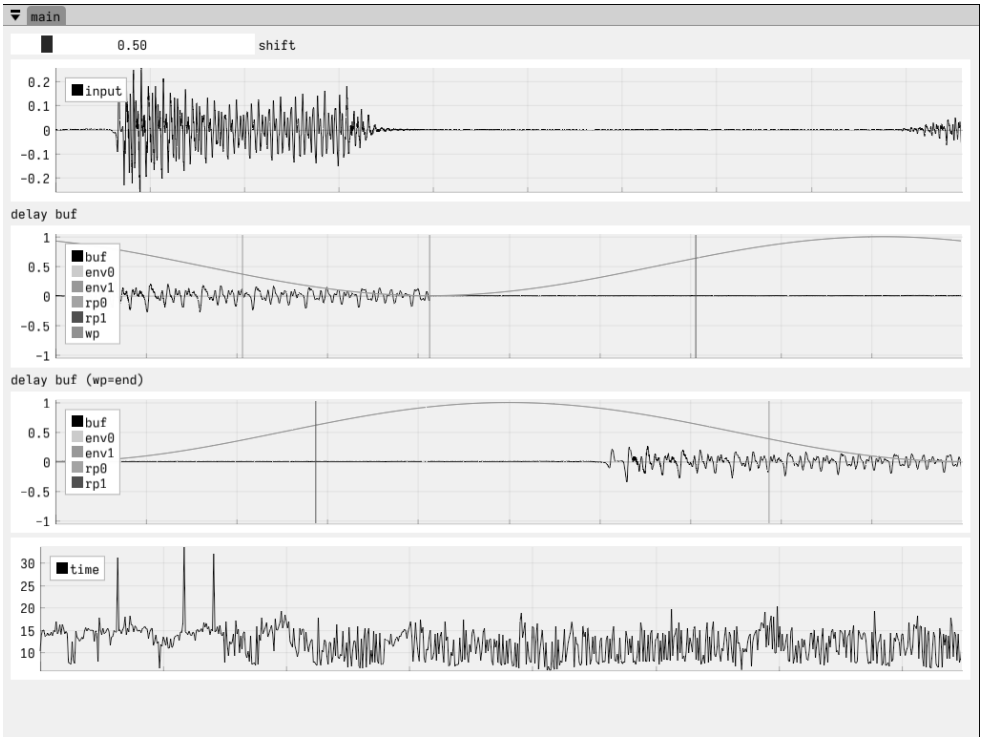


Figure 5.4: Demo program for the cross-fade pitch-shifter. The **`shift`** slider controls the pitch-shift factor. The first plot is the input signal. The next two plots are visualizations of the delay buffer, similar to Figure 5.3, to indicate which samples are currently read, scaled by which envelope value. The last plot is a performance counter to track the computation time of each tick.

A demo of the cross-fade pitch-shifting algorithm can be found on tig[^6]. It has a few visualizations of the delay line buffer for easier understanding and tweaking the algorithm (Figure 5.4).

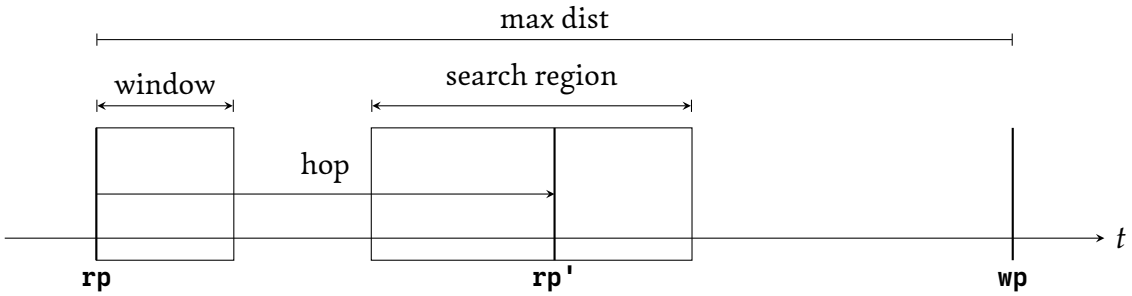
The improved cross-fade algorithm already sounds much better than the STK pitch-shifter used in the first **MAHW** prototype. However, an inherent problem of this algorithm is that even if the pitch-shifting factor is 1, i.e. no shift, the output still contains two taps of delay, creating a comb filtering artifact or, if the delay is large, an echo. It is not the same as the input signal. This is not ideal for **MAHW**, because, as mentioned before, most of the time the pitch-shifter remains at the no-shift condition. The audio quality at the no-shift condition is equally important as the pitch-shifting artifacts.

[^6]xfade: https://tig.krj.st/pitshift_gui/file/main_xfade.cc

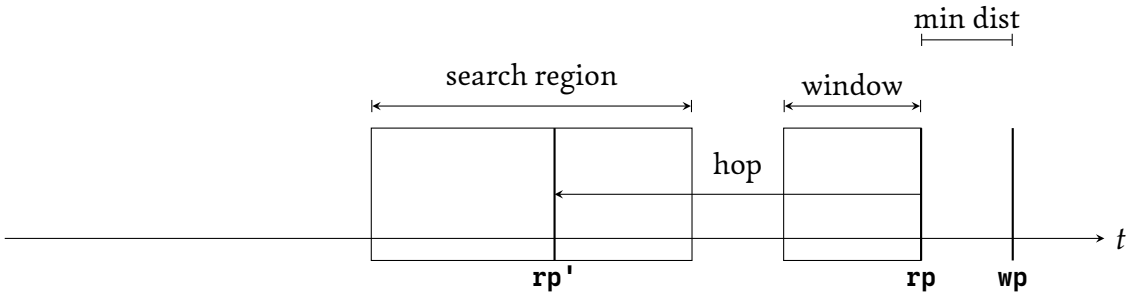
5.2 ALGORITHM: DELAY HOP

But this is an advantage of the delay hopping algorithm proposed by Azadeh Haghparast, Henri Penttinen, and Vesa Välimäki [37]. For this algorithm, the output will be exactly the same as the input when the pitch-shifting factor stays at 1. The algorithm has a few problems at extreme pitch-shifting factors, but it happens to suit the requirement of MAHW very well, where the pitch-shifting is only momentary and the factor is close to 1.

The delay hopping algorithm follows the same principle as the cross-fade algorithm until the handling of the read pointer **rp** catching up with the write pointer **wp**. Instead of using a window function to taper the discontinuity, this algorithm hops the read pointer **rp** whenever it gets too close or too far from the write pointer **wp** so it makes sure that the read pointer **rp** never catches up with the write pointer **wp** (Figure 5.5).



(a) Jump forward when **rp** falls behind (pitch down).



(b) Jump backward when **rp** catches up (pitch up).

Figure 5.5: Delay hopping pitch-shifting algorithm.

The best landing position of the read pointer **rp** is selected in a search region. For every possible landing position **rp'** in the search region, a window of samples near the original read pointer **rp** is compared with the samples near the potential landing position **rp'**. Whichever position has the most similar samples to the original **rp** will be chosen as the landing position for the hop. In

the original paper, the similarity is measured using the average magnitude difference function

$$\text{AMDF}[k] = \frac{1}{L_{\text{win}}} \sum_{n=0}^{L_{\text{win}}-1} |x[n] - y[n+k]|,$$

where L_{win} is the window length, x is the window near the original **rp**, and y is the search region. Basically, the function answers the following question: at which offset k does y have the most similar waveform as x[7]? The offset k that leads to the most similar windows of samples, characterized by the minimal difference in amplitude, will have the smallest AMDF value. Because we are only interested in the minimum of AMDF, the division by L is optional if the window length L is kept the same. This metric is very similar to cross-correlation, but for cross-correlation it is the maximum that indicates the highest similarity.

The fast computation of AMDF depends on cross-correlation, as we will see later on.

The hop will take place gradually in the form of a cross-fade. For a short period of time after a hop happens, samples from both the original read pointer **rp** and the hop destination **rp'** are present in the output, but the signal from the hop origin gradually fades out and the signal from the hop destination gradually fades in. The cross-fade does bring the same problem of echoy artifacts, but it is rare that a hop happens exactly after a transient attack, so in practice it sounds cleaner than the previous algorithm.

There are many parameters in this algorithm. Many of them depend on the bandwidth of the signal to be pitch-shifted. The parameters used in **MAHW** is tuned for a standard tuning 21-fret guitar, where the lowest note is E2 (82.41 Hz) and the highest note is C♯6 (1108.73 Hz), excluding harmonics. But a bass will need a different set of parameters due to its lower frequency range.

The first parameter is the maximum and minimum distance between the **wp** and **rp**. The minimum distance d_{min} needs to leave enough room for cross-fading and fractional delay interpolation (1 sample for linear interpolation), and the maximum distance d_{max} needs to be small enough that the latency doesn't become too large. The paper [37] suggests using

$$d_{\text{min}} = (\alpha - 1)L_{\text{xfade}} + 1$$
$$d_{\text{max}} = 2T_{\text{lowest}},$$

where α is the pitch-shifting factor, L_{xfade} is the cross-fade window length, and T_{lowest} is the period of the lowest frequency, which for a guitar is around $1/82.41 = 12$ ms. However, it is better to

[7] See projet μ for a detailed introduction to AMDF: https://mu.krj.st/assignments/wave_s.html

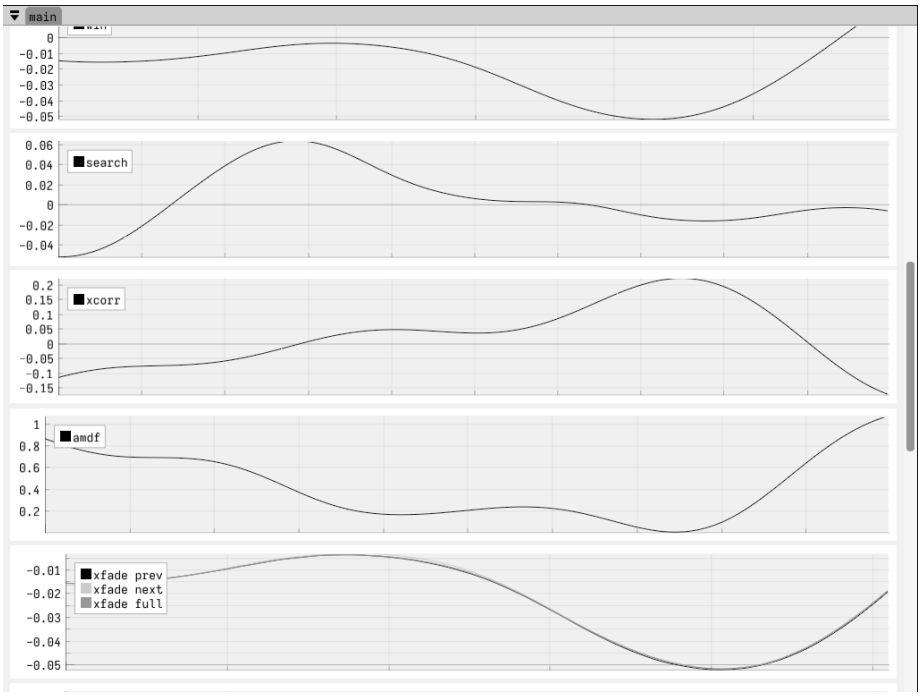


Figure 5.6: Tweak the lowest frequency until a minimum can be seen in the AMDF plot (the 4th plot from the top; the x-axis is the offset) for the lowest note.

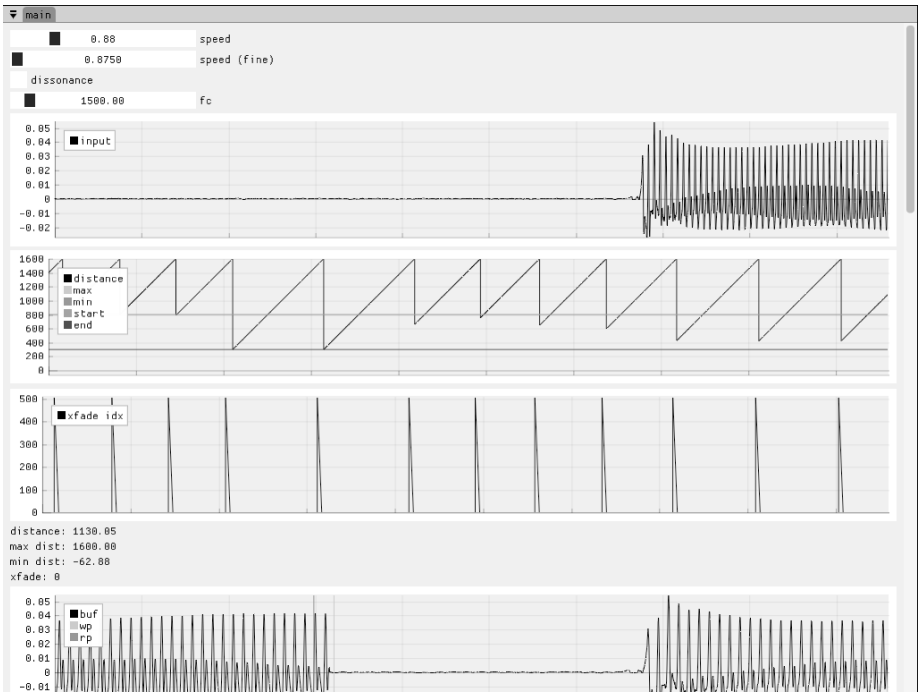


Figure 5.7: The latency (sawtooth-like **`distance`** in the 2nd plot from the top; the x-axis is time) is constantly changing, so the actual latency is less than the maximum.

tune it by looking at the AMDF plot in real-time and make sure there is a minimum in AMDF when the lowest note is played (Figure 5.6)[8]. The lowest frequency is determined to be around 60 Hz from the empirical tuning. This leads to a maximum latency of around 33 ms, but the the actual latency is always changing during the pitch-shifting (Figure 5.7) and usually less than this number. Plus, there is a trick introduced in Section 5.3 to further reduce the perceived latency.

For the reference window length L_{win} and search region length L_{search}, the paper suggests using

$$L_{\text{win}} = (3/8)T_{\text{lowest}}$$
$$L_{\text{search}} = (5/8)T_{\text{lowest}}.$$

The starting point of the search region is chosen to be one period T_{lowest} back from the **wp**. For the cross-fade length, it is chosen to be 1024 samples from tuning.

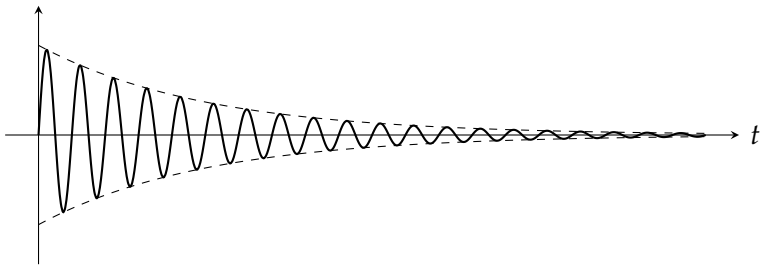


Figure 5.8: In the case of a decay envelope, the best splice position will be dependent on the magnitude of the waveform.

The waveform used for similarity comparison is low-passed using a second order Butterworth filter to reduce the impact of harmonics and aims to emphasize the continuity in the lowest partial, which brings another parameter to tune. The cutoff frequency should be close to the maximum frequency of the instrument, around 1100 Hz. The original paper [37] also advises normalizing the reference window and the search region to eliminate the effect of amplitude variations (such as in the attack and release envelope), though I don't find the normalization necessary and in some cases, such as a decay envelope (Figure 5.8), the signal magnitude of the reference window will be important to determine the best hop destination to avoid causing a sudden magnitude change.

The last parameter is the cross-fade window. I notice that the cross-fade window does not matter that much if the two waveforms are similar or correlated, or if the waveforms are very complex (like a complete mix) and have many high frequency content, but the choice of window is im-

[8] See https://mahw.krj.st/pitshift.gif for a time-varying version.

portant when the waveforms are simple (like a single instrument) and they are not completely correlated. This happens when a suspended or seventh chord is played, where the period of the bass note does not necessarily align with other notes in a chord, leading to a much longer overall period than the lowest frequency. In this case, the energy-preserving cross-fade curve from [53]

$$w(x) = (x(1-x)(1+1.4186x(1-x))+x)^2 \quad x \in [0,1]$$

produces a less noticeable artifact. The artifact is still audible, but because the aimed use case of **MAHW** is momentary pitch-shifts, this is less a problem in practice.

When porting the algorithm to an embedded platform, an important optimization needs to be implemented. Without it, the complexity of the AMDF computation is quadratic $O(L_{\text{win}}L_{\text{search}})$ and there is no way an embedded platform can compute it at a low buffer size. There is a trick to optimize the computation to $O(L_{\text{search}}\log(L_{\text{search}}))$, but it is somehow not mentioned in the original paper.

The trick is to compute the average square difference function (ASDF) instead of the AMDF

$$\text{ASDF}[k] = \sum_{n=0}^{L_{\text{win}}-1} (x[n] - y[n+k])^2.$$

This metric is very similar to the AMDF, but it computes the squared difference between x and delayed y instead of the absolute difference. The extrema of ASDF and AMDF will occur at the same offset k, but the advantage of using ASDF is that the formula can be transformed for fast computation

$$\begin{aligned}\text{ASDF}[k] &= \sum_n (x[n] - y[n+k])^2 \\ &= \sum_n (x^2[n] - 2x[n]y[n+k] + y^2[n+k]) \\ &= \sum_n x^2[n] - 2\sum_n x[n]y[n+k] + \sum_n y^2[n+k].\end{aligned}$$

The first term $\sum_n x^2[n]$ is the total energy of the reference window. It does not depend on the offset k so it only needs to be computed once. The third term $\sum_n y^2[n+k]$ is the energy of each compared window at offset k in the search region. It can be computed by first computing of the energy of the compared window at offset $k = 0$, caching the value. Then for each offset k onwards, inherit the cached energy from the previous offset $k-1$, but subtract the dropped sample from the previous offset window and add the new sample from the current offset window.

```
0 1 2 3 4 5 6 7 8  search region
|-------|          k=0
  |-------|        k=1
^drop     ^new
```

It can be implemented in code by

```
float ysq[L_search] = {0};
for (size_t k = 0; k < L_win; ++k)
  ysq[0] += y[k] * y[k];
for (size_t k = 1; k < L_search; ++k)
  ysq[k] = ysq[k-1] + y[L_win+k]*y[L_win+k] - y[k-1]*y[k-1];
```

In the second term, $(x \star y)[k] = \sum_n x[n]y[n+k]$ is the cross-correlation between x and y. It is possible to compute it for all the offset k at once in $O(L_{\text{search}} \log(L_{\text{search}}))$ time using the convolution theorem with the Fast Fourier Transform (FFT)

$$x \star y = \mathcal{F}^{-1}\left\{\overline{\mathcal{F}\{X\}}\mathcal{F}\{Y\}\right\},$$

as long as x and y are zero-padded to be longer than $L_{\text{win}} + L_{\text{search}}$ to prevent time-aliasing. On Daisy Seed, the Fourier transform is implemented using the pffft library[^9].

This is everything needed to implement the delay hopping algorithm. An interactive demo can also be found on tig[^10]. Compared to the cross-fade algorithm, the delay hopping algorithm has a slightly longer delay (though tunable), but the output is not colored at no-shift condition and the pitch-shifting is less echoy due to only having a single read pointer most of the time. It is the algorithm used all the demo of the second **MAHW** prototype. Besides this, there is also a phase vocoder[30] pitch-shifter implementation on tig[^11], though the pitch-shifting can contain some inharmonicity at a low FFT size (for low latency and performance) and modulating the pitch-shifting factor can cause some phase issues that are tricky to fix. Due to the unsatisfactory audio artifacts, it did not make into the final firmware implementation of **MAHW**.

5.3 DISSONANCE

As mentioned previously in Section 2.2, a distinct characteristic of a hardware vibrato system is the non-uniform pitch-shifting on different strings, where, for example, the high E string and the low E string will be shifted by different amounts when the whammy bar is pushed to the extreme. This results in a characteristic dissonant sound when pitch-bending a chord. While it is up to

[^9] pffft: https://bitbucket.org/jpommier/pffft/
[^10] hop: https://tig.krj.st/pitshift_gui/file/main_hop.cc
[^11] pvoc: https://tig.krj.st/pitshift_gui/file/main_pvoc.cc

debate whether the dissonance is truly desired, this behavior can be modeled with some tricks.

The main insight of the trick is that we can pitch-shift different frequency bands of the signal by different amounts to simulate the non-uniform pitch-shifting. This will generate a slight detune between the fundamental frequency and the harmonics, creating a beating effect. This is not a perfect or physics-based simulation, since on a real mechanical vibrato, the same pitch on different strings are in the same frequency band, but they will be detuned differently, and there won't be any dissonance when only a single string is plucked. However, this trick does bring some of the characteristic dissonance sound into a clean pitch-shifter.

Simulating that would require a real-time chord vs. single note classifier.

Recall from Figure 2.9 that the high-E string is much harder to bend than the low-E string, so naturally we can think about applying more pitch-bending to the low-frequency range than the high-frequency range. The simplest solution is hence to split the signal into two bands using crossover filters, one high-pass and one low-pass, then only apply the pitch-bending to the low-pass output, leaving the high-pass output untouched (Figure 5.9). We could add another pitch-shifter at a slower bending rate in the high-pass path, but the frequency beating is already audible without any shift.

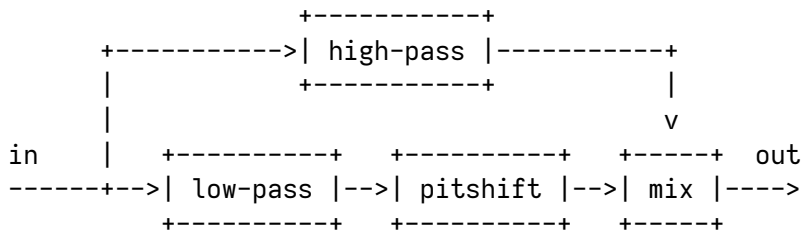


Figure 5.9: Signal path to simulate the mechanical vibrato dissonance behavior.

This is implemented in the delay hopping demo[^12] (Figure 5.6). The **dissonance** checkbox toggles this behavior and the **fc** slider sets the crossover frequency. The crossover frequency controls how much dissonance is introduced to the output signal. A low crossover frequency (100–1500 Hz) will leak more fundamental frequencies into the bypass path and creates a more dissonant sound, because the beating will happen on the lowest partials with the most energy. Lower crossover frequencies will leak too little signal into the pitch-shifter that the pitch-bending effect is no longer audible.

A moderately high crossover frequency (5000–15000 Hz) creates almost no dissonance, though I notice there is still an advantage to set a crossover frequency this high. A high crossover fre-

[^12]hop: https://tig.krj.st/pitshift_gui/file/main_hop.cc

quency naturally separates the transient and tonal parts of the input signal. The transient attack is encoded in the high-pass signal, and the sustained tonal part is encoded in the low-pass signal. By keeping the transient attack of the signal away from the pitch-shifter delay, the apparent latency of the overall system can be greatly improved. The latency is no longer 20–30 ms of the pitch-shifter but rather the group delay of the high-pass filter, which can be almost ignored.

Any crossover frequencies in between (1500–5000 Hz) will have both the dissonance characteristic and the latency reduction benefits. Moving the slider in this range mainly controls how much tonal content is leaked into the direct path with no pitch-shift. The dissonance is not changed by much. I personally find 1500 Hz to be a balanced crossover frequency to include a moderate amount of dissonance to the pitch-shifter. If the dissonance is not desired by the user, the pitch-shifter can still benefit from a perceptual latency reduction by setting the crossover frequency to around 5000 Hz.

Although adding the crossover alters the input signal at no-shift condition, mainly introducing a 20–30 ms delay in the low-pass path, the difference is minimal if the crossover frequency is not too low that it introduces too much transient into the pitch-shifter. The separation between the tonal and transient components naturally prevent a similar echo artifact from the cross-fade pitch-shifting algorithm. Hence, adding the dissonance modeling to the pitch-shifter will not degrade the audio quality noticeably. It has an additional benefit of reducing the impact of latency. However, there is no hardware switch on **MAHW** to toggle the dissonance. The firmware needs to be recompiled if the dissonance modeling needs to be turned on or off.

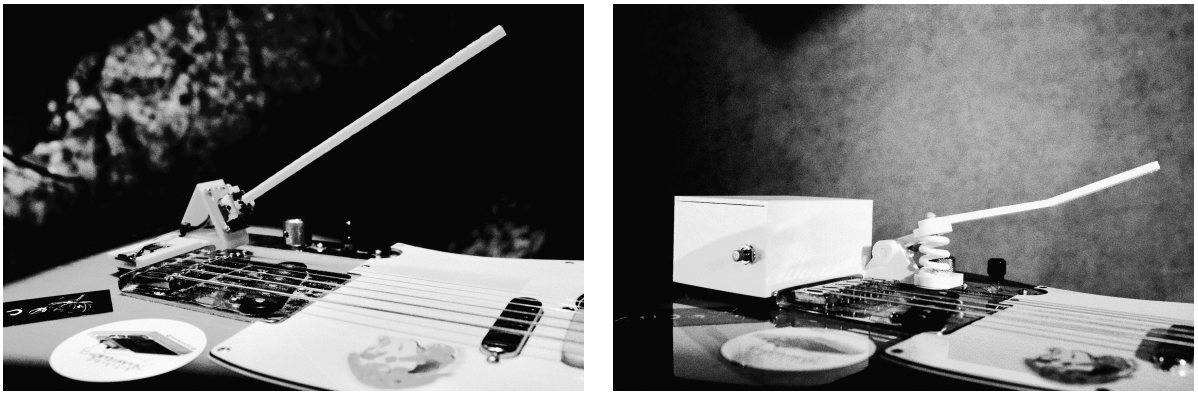
The firmware and 3D printing models of the second **MAHW** prototype can be found on tig[^13].

[^13] mahw_v2: https://tig.krj.st/mahw_v2/

Chapter 6: The Second Take: Evaluation

THE second prototype of **MAHW** addressed many usability problems from the first prototype. With the redesigned mechanism, the arm height is now reduced to a more usable range so that chord bending is now possible[^1]. The tip of the arm is reduced from 12.3 cm in the first prototype to 6–9 cm in the second prototype (Figure 6.1). This is very close to the 5.5 cm arm height on a Jazzmaster vibrato system.

The height will change when the arm is rotated.



(a) The first **MAHW** prototype.

(b) The second **MAHW** prototype.

Figure 6.1: Comparison of arm height.

Additionally, the pitch-shifter of the second prototype is greatly improved, with two algorithms to choose from and an additional option to model the dissonance from a mechanical vibrato. The echo artifacts from the STK pitch-shifter are no longer present in the new implementations. Because the second prototype does not depend on computers or any software frameworks, only the Daisy Seed hardware, software rot will be less a problem. The firmware runs on the microcontroller baremetal, i.e. without any operating systems, so it is not going to have similar issues to the first prototype, where a particular operating system and audio framework is required to run the audio processing software/firmware. As long as the firmware is compiled, it will likely run on the hardware forever.

[^1] See the demo: `https://mahw.krj.st/comp/`

The self-contained audio I/O also drastically reduces the setup overhead for the controller. The only extra thing needed to use the second **MAHW** prototype is a short 1/4″ audio cable which connects the guitar output jack to the audio input of the controller. There is not even an on/off switch. The controller will be powered on automatically when the output jack is connected.

The current draw of the controller is around 110 mA at 9 V, most of which comes from the Daisy Seed, so an average 9 V battery lasts around 4–5 hours powering **MAHW**. A good reference for the battery life requirement on the stage is the 7-hour battery life of the Boss WL-20 guitar wireless system[^2]. The battery life of **MAHW** is slightly less than that, but it might be longer if used with a large-capacity rechargeable 9 V lithium battery. A problem with Daisy Seed is that the microcontroller needs a minimum of 4 V to operate, so the standard 3.7 V lithium battery or the 1.2 V NiMH/NiCd rechargeable battery cannot power the controller directly and I had to use the 9V battery. An additional DC–DC boost converter or a series battery connection might be needed if we want to avoid the energy waste of stepping down the 9 V supply using the linear voltage regulator in Daisy Seed. However, to support a more complicated circuit, a dedicated PCB needs to be designed. Alternatively, a 5 V USB portable charger can be used instead of a battery, but it requires an additional USB cable, which adds some friction to the setup.

such as Duracell 9V battery with a capacity of 580 mAh.

3d print	$2-30
daisy seed	$27.95
input jack	$3.10
output jack	$4.59
pcb	$0.40
pcb components	$1.50
magnet	$0.15
hall sensor	$1.28
2 x M3 bolt-nuts	$0.20
velcro	$3.97
blu-tack	$3.14
battery clip	$0.20
momentary push button	$1.00
400-pt breadboard	$4.76
total	$54.24-$82.24 USD

Figure 6.2: Cost breakdown of the second **MAHW** prototype (in 2023).

The total cost of the second prototype is around $54.24–$82.24 USD (Figure 6.2) plus hooking wires and solder. Some components might be cheaper with a bulk pricing, but, still, the controller

[^2]Boss WL-20: https://www.boss.info/us/products/wl-20_wl-20l/specifications/

is already very affordable compared to commercial mechanical vibrato systems, costing at least around $150 USD in 2023.

6.1 USABILITY ANALYSIS

Although there have been many improvements to the second **MAHW** prototype, such as the self-contained audio I/O, lower arm height, better force feedback, more customizability, improved user experience, redesigned pitch-shifting algorithms, it is far from a production-ready controller. There are two aspects of this problem.

The first one is still the ergonomics. Although 3D-printed springs provide a replicable and automatic way to fabricate an RTC mechanism, the maximum force feedback from the plastic spring is still not comparable to a metal spring, around 7 times weaker. The 3D-printed spring is not rigorously analyzed for fatigue, so it is unclear how long the spring will last. Compliant mechanisms can support a high number of cycles, but only if the mechanical design properly limits the material fatigue [58]. This is something that requires more mechanical engineering experience. While the spring can be easily replicated by 3D printing, replacing it from the controller would require unscrewing the two bolt-nut fasteners with a tweezer. It is time-consuming and requires some patience, which might become a problem if the spring fails on the stage. It is necessary to perform more research on compliant mechanisms and mechanisms in general to better characterize and improve the spring design, making it more robust and easier to replace. This would also bring opportunities to further reduce the arm height to an appropriate level.

At least I know the spring is not safe from cats chewing.

The second part is the reliability of electronics. Currently, only the impedance converter is designed on a PCB board. Other parts of the circuits are connected via messy jumper wires or soldered wires. This not only takes a lot of space but also very susceptible to connection failures. It is very difficult to design a low-noise, large-scale audio circuit on a breadboard, especially when long wires pick up radio interferences easily. The wiring would be much easier and reliable if the entire audio path is designed on a PCB. A complete PCB would also allow for a more complex circuit design, better power management, and noise control. Noise control is especially important for the Daisy Seed since the microcontroller is built on a small footprint and does not handle the isolation for digital noise interference very well. It might be worth designing our own board for the microcontroller and the audio codec, since the STM32H750 in Daisy Seed is excessively powerful for the application and has a high current draw. The noise floor performance of the audio codec in Daisy Seed Rev5 is also problematic compared to Rev4. Designing the circuit from scratch would enable a better control on the noise and battery life. By eliminating the Daisy Seed,

the circuit compartment on the second **MAHW** prototype can also be made much thinner and less intrusive to the user.

6.2 FABRICATION ANALYSIS

Compared with the first prototype, the most risky joystick mechanism has been replaced with a more customizable and replicable mechanism based on a 3D-printed spring. The construction of the mechanism is more complicated than the first prototype, but most of the components are custom-made so they are safe from supplier collapsing. A component-wise analysis is given as follows:

3D-printed Models (**`Safe`**) Easy to fabricate given a 3D-printer, which are automatic and usually affordable. Fabrication requires filaments, but they are available from multiple suppliers. The post-processing of 3D-printed springs is slightly more complicated, but it is still doable even with a kitchen knife.

Daisy Seed (**`Bottleneck`**) Difficult to fabricate and available from a single supplier. Can be replaced by Teensy + Audio Shield, but this requires changing the firmware.

Audio jacks (**`Risk`**) Switchcraft 112AX has many alternative suppliers, but alternatives for the isolated switching Switchcraft 113 are harder to find. Some Fender Amps appear to be using a compatible jack[^3], and there is also a 3.5 mm variant of the jack from CUI Devices available with part number SJ-63043H, but in case a Switchcraft 113 is not available, a dedicated power switch can be added.

PCB (**`Safe`**) Moderately difficult to fabricate but nowadays a custom 2-layer PCB fabrication is cheap from many suppliers.

PCB components (**`Safe`**) Difficult to fabricate but multiple suppliers available. Resistors, capacitors, and BJTs are common components available at almost any electronic shop. The model of the BJT is not important. It can be replaced by any npn BJT with a compatible pin configuration.

Magnet (**`Safe`**) Difficult to fabricate but multiple suppliers available. The size of the magnet does not matter that much.

Hall sensor (**`Safe`**) Difficult to fabricate but multiple suppliers available, such as the Honeywell SS49E. The sensitivity might be different, but it can be tuned by either replacing the magnet or

[^3] 1/4″ Jack - Stereo, PC Mount, for Fender® Amps '88 - '99: https://web.archive.org/web/20210729044206/https://www.tubesandmore.com/products/14-jack-stereo-pc-mount-fender-amps-88-99

the code.

M3 bolt-nuts (**Safe**) Difficult to fabricate but multiple suppliers available.

Velcro (**Safe**) Difficult to fabricate but multiple suppliers available. Can be replaced by other brands.

Blu-Tack (**Safe**) Difficult to fabricate but multiple suppliers available. Can be replaced by other brands.

Battery clip (**Safe**) Difficult to fabricate but multiple suppliers available.

Momentary push button (**Safe**) Difficult to fabricate but multiple suppliers available. The hole on the 3D model might require some adjustment, but it is not going to change the whole box design.

Breadboard (**Safe**) Difficult to fabricate but multiple suppliers available. The one I am using is BusBoard BB400.

Firmware (**Safe**) The firmware runs baremetal, only depending on pffft for FFT computations, but it is a single-file library, so it can be considered to be part of the source code.

The more cautionary design of the second **MAHW** prototype reduces both the bottleneck component and at-risk component to one. The output jack uses a specialized isolated-switching design so it is slightly harder to source, but the special jack is only used to eliminate the on/off switch. This minor design detail is not very critical to the overall functionality of the controller, and it can be replaced with a dedicated on/off switch. The only bottleneck is the Daisy Seed, but at the moment it is one of the few widely available embedded platforms for audio processing. Teensy with an audio shield is another option, but the dimensions are much larger. Some ESP32 development boards have an audio codec, but none of the boards provide a wired audio input, only hardwired microphones, so they are not usable without modifications to the PCB. Bela[^4] can also be an option, but it is not using a microcontroller but a 1 GHz microprocessor (Texas Instruments AM3358), which has a much higher current draw and entails a shorter battery life.

It might be worth designing a custom PCB for the audio processing, but it does not necessarily reduce the fabrication complexity of the controller. The risk of managing the lifespan and supply chain of the microcontroller and the audio codec will be passed down to the designer. The PCB

[^4]Bela: https://bela.io/

design will need constant revision to take care of end-of-life components, similar to the many revisions of Daisy Seed, but it will be the designer's job to do the revision. This demands more PCB design skills from the designer and increases the barrier of entry for redesign and maintenance, which does not contribute to the longevity of the musical interface. Using a third-party audio platform is likely the best one can do for a DIY-friendly project.

6.3 CUSTOMIZATION ANALYSIS

An advantage of the 3D-printed mechanism in the second **MAHW** prototype is that every aspect of the user experience can be customized, both the mechanical and audio effect designs. If the whammy bar arm is too high, the user can either reduce the height of the 3D-printed spring or reduce the length of the whammy bar. If the force feedback is not enough from the spring, the spring rate k can be slightly increased by widening the wire diameter of the 3D-printed spring (c.f. Table 4.1). To change the response curve of the pitch-shifter, modifications can be made either in the firmware or the hardware—the Hall effect sensor can be swapped to a different sensitivity and the magnet can be customized to use a different material, even with a different diameter, to change the strength of the magnetic field. Using Blu-Tack as the semi-permanent glue makes it easy to swap out these components. If the user doesn't like the artifacts from the pitch-shifter, there are also multiple algorithms to choose from. The location of the electronic compartment can even be changed by changing a parameter in the OpenSCAD script for the base if the user thinks it hinders the hand from muting the string.

The main limitation of customization is the strength of 3D-printing materials. It is not possible for a naïvely designed spring made from PLA to compete with the force feedback from a metal spring. More research needs to be done on compliant mechanisms to figure out how to design a stronger, yet easily fabricable spring. Right now, the 3D-printed spring used in **MAHW** is still very wobbly due to the weaker material strength. Similarly, the thickness of the arm is also restricted by the material strength. If the handle is too thin, it will warp easily under stress. Changing the 3D-printing material can help a bit, but it is still far from the strength of metal.

Customizing the model would require some knowledge of OpenSCAD and tweaking the audio processing algorithm would require knowledge of C and C++. Both technologies are fairly low-level and can bring some barriers to entry for redesigners. It can be necessary to provide more documentation on the software side of the design to simplify the learning curve.

Chapter ⌖: Onwards

The overarching goal of the project **MAHW**, as established in Chapter 1, is to build a low fabrication/customization complexity, digital–mechanical alternative to a mechanical guitar vibrato system. It attempts to unlock some interesting embellishments, such as downward chord bending and harmonics bending, that are normally difficult to do on a fixed-bridge guitar.

The first prototype of **MAHW**, made of a simple joystick and a Teensy, confirmed the concept that the simple combination of a bar plus a pitch-shifter was enough to model a mechanical vibrato system. However, as the first prototype, it suffered from various usability problems, such as reliance on a computer, unreliable mechanisms, ergonomics, pitch-shifting quality, and component shortage.

A second design iteration was hence commenced to improve **MAHW** to be a more robust, standalone device, pushing it further towards its goals. With 3D-printed mechanisms and a self-contained audio path in the design, the second **MAHW** prototype was able to greatly improve the usability of the device without introducing too much complexity into the fabrication process. In fact, the switch from a pre-manufactured joystick to customized, 3D-printed mechanisms made the second **MAHW** prototype even more customizable than the first one.

The second prototype not only achieved all the goals set previously in Section 2.3.4, but the design process of the second prototype also inspired and provided an example of using compliant mechanisms to implement a customizable yet easily fabricable force feedback mechanism, in the form of a 3D-printed spring. Several DSP tricks, such as the optimization for the delay hopping algorithm and the trick to model dissonance from a mechanical vibrato system, were also discovered in the redesign process. However, as discussed in Chapter 6, the second **MAHW** prototype is still not a perfect control interface, far from production-ready.

It is very difficult to simplify the user experience even further when all it requires to use the controller is to plug in an extra cable. Nonetheless, for next steps in the research, the ergonomics

of the controller do have room for improvement. How to increase the force feedback of a 3D-printed spring to more than 50 N while keeping the compression distance? Can we increase the force feedback by reducing the leverage of the arm? Is it possible to combine the hinge and the spring into one piece? If the spring is destined to suffer from wear and tear, how to make it easily replaceable, even on the stage? The answer to these questions will help further reduce the arm height and improve the stability and repairability of **MAHW**, but further research into compliant mechanisms will be necessary to provide an answer.

Another possible angle for improvements would be, if we discard the shape and the gesture of a traditional mechanical whammy bar, is there another natural gesture to perform a pitch bend, and what sensor is needed to capture that? I am already using the mechanical whammy bar quite differently from most people. Maybe the "traditional" gesture is just a legacy of how mechanical vibrato systems were built, not actually the most natural and efficient gesture to add embellishment? We can already find many alternative designs, such as the touch panel in Mosi Nova[^1], the bend angle sensor in [7], but the possibilities definitely don't stop here.

Finally, many aspects of the circuit component can be improved. What would happen if the controller's battery died during a performance? Right now the audio output becomes silent completely, but it is not the best solution when the musical interface only acts as an embellishment for the input sound. A better solution is to bypass the input to the output when the battery dies and only switch to DSP when the interface is powered on. It is also desirable to have a charging circuit for the battery. But to implement these complex behaviors, an integrated PCB needs to be designed for the entire audio path, not just the impedance converter. The PCB can be necessary to reduce the wire clutter in the current design and improve the noise control capabilities. With cleaner wiring, the electronic compartment can be made much thinner, shrinking **MAHW** to be a more portable device and perhaps attachable to other musical instruments.

[^1] Mosi Nova: https://mosiaudio.com/products/nova

Acknowledgements (Part II)

FIRST of all, I would like to thank all the nurses, doctors, surgeons, and physicians at CHUM (le Centre hospitalier de l'Université de Montréal) who took care of me for the past two years. It wouldn't be possible for me to imagine how to live as a normal person again without everyone's help. I also want to thank Ari for staying and chatting with me in the hospital the night before my surgery and Gülan, Colin, and Ann Nastatia for taking care of Nila and Nulla while I was away in the hospital, as well as everyone who came to visit me during my most vulnerable times.

et qui ont essayé de m'apprendre le français en même temps.

I would like to thank Dr. Marcelo Wanderley and Prof. Erin Gee for taking the time to read the thesis and providing me with valuable feedback. I want to thank everyone whom I have met and who has interacted with me, either directly or indirectly, in the past few years. Every one of you, in one way or another, shapes and sculpts something inside my brain, leading to the realization of the thesis. I would also like to acknowledge CIRMMT (Centre for Interdisciplinary Research in Music Media and Technology) for the partial financial support towards this project.

Lastly, in terms of the presentation of the thesis, a majority of the text in this thesis is typeset in Garibaldi. I would like to thank Henrique Beier for designing this typeface and commenting on my *maze*. The headers are typeset in Artifex Hand CF. I would like to thank Connary Fagen for designing the typeface and providing me with a discount. I would like to thank Crazyparts and Duesenberg Guitars for creating the photos in Figure 1.2 and Figure 1.3 and allowing me to use them in the thesis.

Bibliography

[1] Anthony Agnello, "Method and apparatus for producing two complementary pitch signals without glitch," U.S. Patent 4369336, Jan. 21, 1981.

[2] William A. Aitken, Anthony J. Sedivy, and Michael S. Dixon, "Electronic musical instrument," U.S. Patent 4658690, May 9, 1984.

[3] Eventide Audio, *Flashback #4.2: H910 Harmonizer® Pt. 2*, https://web.archive.org/web/20240104192348/https://www.eventideaudio.com/50th-flashback-4-2-h910-harmonizer-the-product/, 2016.

[4] Paul Batchelor, *Sporth*, https://web.archive.org/web/20240104192643/https://pbat.ch/proj/sporth.html, 2015.

[5] Paul Batchelor, *Contrenot*, https://web.archive.org/web/20230924044846/https://pbat.ch/proj/contrenot/, 2016.

[6] Rekka Bellum and Devine Lu Linvega, *Keynote at NIME 2022*, https://web.archive.org/web/20240104192940/https://git.sr.ht/~rabbits/nime2022/tree/main/item/slides, 2022.

[7] Adan L. Benito Temprano and Andrew P. McPherson, "A TMR Angle Sensor for Gesture Acquisition and Disambiguation on the Electric Guitar," in *Proceedings of the 16th International Audio Mostly Conference*, Association for Computing Machinery, 2021, pp. 256–263.

[8] Paul A. Bigsby, "Tailpiece vibrato for string instrument," U.S. Patent D169120, Nov. 15, 1952.

[9] Kim Bjørn, *PUSH TURN MOVE*. Bjooks, 2017, ISBN: 978-87-999995-0-7.

[10] Dennis Bohn, *High-Quality Low-Voltage Audio*, https://web.archive.org/web/20240104193101/https://www.ranecommercial.com/legacy/hilovolt.html.

[11] Bernard Brauchli, *The Clavichord* (Cambridge Musical Texts and Monographs). Cambridge University Press, 1998, ISBN: 978-0-521-63067-2.

[12] Robert Bristow-Johnson, *Algorithms for finding seamless loops in audio*, https://web.archive.org/web/20240104193116/https://music-dsp.music.columbia.narkive.com/C8IESiuZ/algorithms-for-finding-seamless-loops-in-audio#post24, 2010.

[13] Richard G. Budynas and J. Keith Nisbett, *Shigley's Mechanical Engineering Design*, 11th ed. McGraw-Hill, 2020, ISBN: 978-0-07-339821-1.

[14] James A. Burchit, "Tremolo attachment for musical instruments," U.S. Patent 775327, Mar. 24, 1904.

[15] Filipe Calegario, João Tragtenberg, Johnty Wang, Ivan Franco, Eduardo Meneses, and Marcelo Wanderley, "Open Source DMIs: Towards a Replication Certification for Online Shared Projects of Digital Musical Instruments," in *HCI International 2020 – Late Breaking Papers: User Experience Design and Case Studies*, Springer International Publishing, 2020, pp. 84–97.

[16] Filipe Calegario *et al.*, "Documentation and Replicability in the NIME Community," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2021.

[17] Stuart Cheshire and Mary Baker, "Consistent overhead byte stuffing," in *Proceedings of the ACM SIGCOMM'97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1997, pp. 209–220.

[18] Michael Chessman and Nathan Sokal, "Prevent emitter-follower oscillation," *Electronic Design*, vol. 24, no. 13, pp. 110–113, 1976.

[19] Perry Cook and Gary Scavone, "The synthesis toolkit (STK)," in *Proceedings of the International Conference on Mathematics and Computing*, 1999, pp. 299–304.

[20] Yamaha Corporation, *HS8 / HS7 / HS5 Service Manual*, 2013.

[21] Simon Crab, *120 Years of Electronic Music*, https://web.archive.org/web/20230724115909/http://120years.net/.

[22] D'Addario & Co., *A complete technical reference for fretted instrument string tensions*, https://web.archive.org/web/20240104193334/https://www.daddario.com/globalassets/pdfs/accessories/tension_chart_13934.pdf.

[23] Jonathan Driedger and Meinard Müller, "A review of time-scale modification of music signals," *Applied Sciences*, vol. 6, no. 2, 2016.

[24] ElectroSmash, *Tube Screamer Analysis*, https://web.archive.org/web/20240104193423/https://www.electrosmash.com/tube-screamer-analysis.

[25] Clarence L. Fender, "Tremolo device for stringed instruments," U.S. Patent 2741146, Aug. 30, 1954.

[26] Clarence L. Fender, "Amplifier with tremolo," U.S. Patent 2817708, Jan. 16, 1956.

[27] Clarence L. Fender, "Floating tremolo and bridge construction for lute-type musical instruments," U.S. Patent 2972923, Nov. 6, 1958.

[28] Clarence L. Fender, "Guitar incorporating inertial vibrato device," U.S. Patent 3241418, Jun. 5, 1964.

[29] Alexandre Marino Fernandez and Fernando Iazzetta, "Circuit-bending and DIY culture," *Keep it Simple, Make it Fast! An approach to underground music scenes*, vol. 1, pp. 17–28, 2015.

[30] James L. Flanagan and Roger M. Golden, "Phase vocoder," *The Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, 1966.

[31] R.J. Folkman, *The Moog Sonic Six Technical Service Manual*. Moog Music Inc., 1973.

[32] Forgotten Futures, *About—Ondioline*, https://web.archive.org/web/20240104193439/https://ondioline.com/about.

[33] Albert J. Forrest, "Stringed musical instrument," U.S. Patent 607359, Sep. 14, 1897.

[34] Norman R. French and Manvel K. Zinn, "Method of and apparatus for reducing width of transmission bands," U.S. Patent 1671151, Nov. 10, 1924.

[35] Dennis Gabor, "Theory of Communication. Part 3: Frequency Compression and Expansion," *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, vol. 93, no. 26, pp. 445–457, 1946.

[36] David Robert Grimes, "String theory—the physics of string-bending and other electric guitar techniques," *PLOS ONE*, vol. 9, pp. 1–9, Jul. 2014.

[37] Azadeh Haghparast, Henri Penttinen, and Vesa Välimäki, "Real-time pitch-shifting of musical signals by a time-varying factor using normalized filtered correlation time-scale modification (NFC-TSM)," in *Proceedings of the 10th International Conference on Digital Audio Effects*, 2007, pp. 10–15.

[38] Lippold Haken, Ed Tellman, and Patrick Wolfe, "An Indiscrete Music Keyboard," *Computer Music Journal*, vol. 22, no. 1, pp. 30–48, 1998.

[39] Liang He, Huaishu Peng, Michelle Lin, Ravikanth Konjeti, François Guimbretière, and Jon E. Froehlich, "Ondulé: Designing and Controlling 3D Printable Springs," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, Association for Computing Machinery, 2019, pp. 739–750.

[40] Larry L. Howell, *Compliant Mechanisms*. Wiley, 2001, ISBN: 978-0-471-38478-6.

[41] Tim Hunkin, *SPRINGS - The Secret Life of Components*, https://www.youtube.com/watch?v=dethuGX2IGs, 2021.

[42] Texas Instruments, *DRV5056 Unipolar Ratiometric Linear Hall Effect Sensor*, https://web.archive.org/web/20240104193408/https://www.ti.com/lit/ds/symlink/drv5056.pdf, 2020.

[43] International MIDI Association, *MIDI Musical Instrument Digital Interface Specification 1.0*. 1983.

[44] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner, "The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces," in *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 2007, pp. 139–146.

[45] Thomas Jungmann, "Theoretical and Practical Studies on the Behavior of Electric Guitar Pick-ups," B.Sc. thesis, Helsinki University of Technology, 1994.

[46] Asahi Kasei, *AK4556*, https://web.archive.org/web/20240104193919/https://www.akm.com/content/dam/documents/products/audio/audio-codec/ak4556vt/ak4556vt-en-datasheet.pdf, 2015.

[47] Clayton O. Kauffman, "Apparatus for producing tremolo effects," U.S. Patent 1839395, Aug. 19, 1929.

[48] Martin Kristoffersen and Trond Engum, "The whammy bar as a digital effect controller," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2018.

[49] Roland O. Lamb, "The Seaboard: discreteness and continuity in musical interface design," Ph.D. dissertation, Royal College of Art, 2014.

[50] Jean Laurendeau, *Maurice Martenot, luthier de l'électronique*. Louise Courteau Éditrice, 1990, ISBN: 978-2-7010-2237-6.

[51] Stephen Shiao-ru Lin, Nisal Menuka Gamage, Kithmini Herath, and Anusha Withana, "MyoSpring: 3D printing mechanomyographic sensors for subtle finger gesture recognition," in *Sixteenth International Conference on Tangible, Embedded, and Embodied Interaction*, 2022.

[52] Roger Linn, "LinnStrument and other new expressive musical controllers," *The Journal of the Acoustical Society of America*, vol. 134, no. 5, pp. 4053–4053, Nov. 2013.

[53] Geraint Luff, *A cheap energy-preserving-ish crossfade*, https://web.archive.org/web/20240104194020/https://signalsmith-audio.co.uk/writing/2021/cheap-energy-crossfade/, 2021.

[54] William S. Marlens, "Duration and Frequency Alteration," *Journal of the Audio Engineering Society*, vol. 14, no. 2, pp. 132–139, 1966.

[55] Stephen McAdams and Carolyn Drake, “Auditory Perception and Cognition,” in *Stevens’ Handbook of Experimental Psychology*, 1st ed. John Wiley & Sons, Ltd, 2002, ch. 10, pp. 397–452, ISBN: 978-0-471-21442-7.

[56] Andrew P. McPherson, “TouchKeys: Capacitive Multi-Touch Sensing on a Physical Keyboard,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2012.

[57] Andrew P. Mcpherson, Robert Jack, and Giulio Moro, “Action–Sound Latency: Are Our Tools Fast Enough?” In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2016, pp. 20–25.

[58] Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski, “A Computational Design Tool for Compliant Mechanisms,” *ACM Transactions on Graphics*, vol. 36, no. 4, Jul. 2017.

[59] Wolfson Microelectronics, *WM8731 / WM8731L*, https://web.archive.org/web/20240104194112/https://www.mouser.ca/datasheet/2/76/WM8731_v4_9-1141834.pdf, 2012.

[60] Charles H. Moore and Geoffrey C. Leach, “FORTH – A Language for Interactive Computing,” *Amsterdam: Mohasco Industries Inc.*, 1970.

[61] Nintendo Co., Ltd., 第８０期　定時株主総会　質疑応答（要旨）, https://web.archive.org/web/20240104194144/https://www.nintendo.co.jp/ir/pdf/2020/qa2006.pdf, 2020.

[62] David Lorge Parnas, “Software Aging,” in *Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society Press, 1994, pp. 279–287.

[63] Miller Puckette, *The Theory and Techniques of Electronic Music*. World Scientific, 2007, ISBN: 978-981-270-077-3.

[64] Edward Ramsden, *Hall-Effect Sensors: Theory and Application*. Elsevier, 2011.

[65] Ben Rogerson, *The iconic Moog Minimoog Model D synth is back in production and looking better than ever*, https://web.archive.org/web/20240104194304/https://www.musicradar.com/news/moog-minimoog-model-d-2022, 2022.

[66] John Romkey, *Nonstandard for transmission of IP datagrams over serial lines: SLIP*, RFC 1055, 1988.

[67] Robert Tom Sawyer, “Stringed musical instrument,” U.S. Patent 1747650, Oct. 25, 1926.

[68] Julius O. Smith, *Physical Audio Signal Processing: for Virtual Musical Instruments and Audio Effects*. 2010.

[69] Jeffrey Snyder, Davis Polito, and Matt Wang, "The Electrosteel: An Electronic Instrument Inspired by the Pedal Steel Guitar," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2023.

[70] Alec Stansfield, *The SynthAxe*, https://web.archive.org/web/20240104194454/http://www.alendi.co.uk/synthaxe.html.

[71] Ned Steinberger, "Tremolo mechanism for an electric guitar," U.S. Patent 4632005, Oct. 1, 1984.

[72] Eric P. Stets, "Tremolo device for stringed musical instrument," U.S. Patent 5392680, Mar. 4, 1994.

[73] Michael R. Sweet, *Serial Programming Guide for POSIX Operating Systems*, https://web.archive.org/web/20220612040637/https://www.cmrr.umn.edu/~strupp/serial.html, 1999.

[74] Johnnie Tam, Jamie Jien-Mei Yang, Theodor R. Lange, and John Yao, "Tremolo bar input for a video game controller," U.S. Patent 7435178, Apr. 12, 2006.

[75] Thomas Thwaites, *The Toaster Project: Or A Heroic Attempt to Build a Simple Electric Appliance from Scratch*. Princeton Architectural Press, 2012, ISBN: 978-1-61689-119-0.

[76] Nicholas Torretta and Hessam Pakbeen, "Empowerment through design-doing experiences: workshops on nurturing creative makers for sustainability," *Keep it Simple, Make it Fast! An approach to underground music scenes*, vol. 1, pp. 59–75, 2015.

[77] Werner Verhelst and Marc Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech," in *Proceedings of the 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1993, pp. 554–557.

[78] Peter Joseph Walker, "Vibrato arm and system," U.S. Patent 10978029 B2, Mar. 20, 2015.

[79] Johnty Yizhong Wang, "Analysis of Wireless Interface Latency and Usability for Digital Musical Instruments," Ph.D. dissertation, McGill University, 2021.

[80] Sophie Weiner, *Minimoog: The First Truly Portable Synthesizer*, https://web.archive.org/web/20240104194403/https://daily.redbullmusicacademy.com/2017/10/instrumental-instruments-minimoog/, 2017.

[81] David Wessel and Matthew Wright, "Problems and Prospects for Intimate Musical Control of Computers," *Computer Music Journal*, vol. 26, no. 3, pp. 11–22, 2002.

[82] Devin Weston, *Project PRO2021 – A Pro-One keyboard synth*, https://web.archive.org/web/20240104194422/https://modwiggler.com/forum/viewtopic.php?p=3000304%5C#p3000304.

[83] Matthew Wright and Adrian Freed, "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers," in *Proceedings of the International Computer Music Conference*, 1997, pp. 101–104.

[84] Андрей А. Володин, *Электронные Музыкальные Инструменты*. Энергия, 1970.

[85] 三枝文夫, 電子楽器　過去・現在・未来. ミュージックトレード社, 2021, ISBN: 978-4-908357-26-8.