RICE UNIVERSITY
Ultra-Low Latency Control of Large Vibrotactile Arrays for Haptic Interactions

By
Brandon Thomas Cambio

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
Master of Science

APPROVED, THESIS COMMITTEE

Marcia O'Malley
Chair, Stanley C. Moore Professor of Mechanical Engineering

Benjamin Fregly
Professor of Mechanical Engineering and CPRIT Scholar in Cancer Research

Daniel Preston
Assistant Professor of Mechanical Engineering

HOUSTON, TEXAS
April 2020
ABSTRACT

Ultra-Low Latency Control of Large Vibrotactile Arrays for Haptic Interactions

by

Brandon Cambio

Research in haptic devices has led to them becoming ubiquitous. One of the most popularly used and researched types of cutaneous haptic feedback is vibration. Though, current implementation options are either limited in their usability for complex, such as virtual reality, applications, very expensive, or require high technical expertise. Syntacts is a audio-based software and hardware framework that lowers the financial and technical barriers to entry into vibration haptics. The software is available in C++, C#, and Python languages to be easy to implement by a variety of researchers. It has been designed to natively help users create complex cues, sequence them in time, and do spatialization on an array of motors. The hardware is the Syntacts Amplifier board that bridges the gap between the digital audio devices and the vibration motors. Benchmarking concluded that Syntacts generates and renders cues with lower latency than current vibration systems at comparable or much lower financial cost. The entire framework is open-source and available at github.com/mahilab/Syntacts.

DISCLAIMER: The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.
1 Introduction

1.1 Haptics ................................. 1
  1.1.1 Kinesthetic Haptics .................. 2
  1.1.2 Cutaneous Haptics .................. 2
1.2 Skin Mechanics ........................... 2
1.3 General Types of Cutaneous Haptic Cues ................ 5
  1.3.1 Normal Force .......................... 5
  1.3.2 Shear Forces .......................... 6
  1.3.3 Vibration ............................. 6
  1.3.4 Thermal .............................. 8
  1.3.5 Multi-modal Interfaces ............... 8
1.4 Motivation .............................. 8
1.5 Contributions ........................... 10

2 Mechanics of Vibrotactile Hardware .......................... 11

2.1 Vibration Motors .......................... 11
  2.1.1 Eccentric Rotating Mass Actuators ............... 12
  2.1.2 Linear Resonance Actuators ............... 12
  2.1.3 Voice Coil Actuators ............... 13
  2.1.4 Piezoelectric Actuators ............... 14
2.2 Creating the Signal .............................................. 14
   2.2.1 Waveform Generators ......................................... 15
   2.2.2 Commercial Closed-Box Solutions ............................... 15
   2.2.3 Commercial I2C Solutions ...................................... 16
   2.2.4 Audio Based Solutions ....................................... 17
2.3 Amplification .................................................. 19
   2.3.1 Linear Amplifiers ........................................... 19
   2.3.2 PWM Amplifiers ........................................... 19
2.4 Summary .................................................... 20

3 Syntacts ...................................................... 21
   3.1 Software ..................................................... 21
      3.1.1 Programming Interface ...................................... 22
      3.1.2 Interfacing Devices ....................................... 23
      3.1.3 Creating Effects with Signals ............................. 24
      3.1.4 Sequencing Signals ....................................... 26
      3.1.5 Rendering the Signal ...................................... 27
      3.1.6 Spatialization and Realtime Modifications ............... 28
      3.1.7 Saving and Loading Signals ................................. 31
      3.1.8 Graphical User Interface .................................. 31
      3.1.9 Dependencies ............................................ 32
      3.1.10 Translation to Other Programming Languages ........... 33
   3.2 Hardware ..................................................... 34
      3.2.1 Sound Cards / Digital-to-Analog Converters ................ 34
      3.2.2 Hardware Drivers and Buffer Sizes ......................... 36
      3.2.3 Amplifiers ............................................ 38

4 Comparison and Verification .................................... 42
   4.1 Latency Test Setup ........................................... 42
4.1.1 Test Rig ................................................................. 42
4.1.2 Vibrotactile System Setup ......................................... 43
4.1.3 Latency Testing Protocol ........................................... 45
4.2 Latency Benchmarking Results ....................................... 47
4.3 Overall Comparison .................................................. 48

5 Syntacts Demonstrations ................................................. 50
  5.1 One Framework to Rule Them All .................................. 50
  5.2 Design Your Cues, Feel Them Move ............................... 51

6 Conclusion ............................................................... 53

Bibliography ............................................................... 55
Illustrations

1.1 Visualization of mechanoreceptors in skin [1] . . . . . . . . . . . . . . 4
1.2 HapticClench squeeze mechanism . . . . . . . . . . . . . . . . . . . . 5
1.3 Examples of haptic stretch devices. (a) Rice Haptic Rocker linear
stretch mechanism [2]. (b) Twist haptic device [3] . . . . . . . . . . . 6

2.1 Generalized flowchart of vibration system . . . . . . . . . . . . . . . 11
2.2 Video game controller with DualShock Vibration System highlighted 12
2.3 Internal diagram of LRA [4] . . . . . . . . . . . . . . . . . . . . . . . 13
2.4 Texas Instruments DRV2605LEVM-MD . . . . . . . . . . . . . . . . 17

3.1 Signals created in Listing 2 . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.2 Sequenced Signals created in Listing 3 . . . . . . . . . . . . . . . . . 27
3.3 The Spatializer created in Listing 5 . . . . . . . . . . . . . . . . . . . 30
3.4 Fan example - this figure demonstrates the variable design paradigms
that can be used within Syntacts to create realistic vibration cues. . . . 30
3.5 Syntacts GUI - The left-hand side demonstrates cue design. Users
drag, drop, and configure Signals from the design Palette to the
Designer workspace. The Signal is visualized and can be played on
individual channels of the opened device. The right-hand side show’s
the GUI’s spatializer (background) and track-based sequencer
(foreground) interfaces. Once designs are complete, they can be saved
for later loading from any of the programming APIs. . . . . . . . . . 32
3.6 Syntacts In Use - This figure demonstrates one possible implementation and use of Syntacts hardware and software, using Tasbi [5] as an example. The top-half shows an audio-device (MOTU 24Ao) connected to two Syntacts amplifier boards that have been integrated into Tasbi control units. Each Tasbi bracelet incorporates six radially spaced LRA tactors, for a total of twelve utilized audio channels. The audio device interfaces to a PC (not shown).

3.7 Example consumer-grade sound cards. (a) Creative SoundBlaster Audigy Rx (b) Asus Xonar U7 MKII

3.8 Mean Windows audio driver API latencies with standard deviation. Data collection methods are described in Sec. 4. For reference, the dashed line indicates the perceptional threshold of visual-haptic simultaneity [6].

3.9 Latency increase with increase in buffer size

3.10 Noise comparison on parallel analog signal wire from TASBI [5] use-case

3.11 Syntacts Amplifier Board, both on its own and integrated into a system from TASBI [5]

3.12 Side view of Syntacts amplifier V1.1

3.13 Side view of Syntacts amplifier V3.1

4.1 Software protocols to compare latency

4.2 Latency testing protocols

4.3 Latency as a function of channels rendered, measured as the time from software triggering to the detection of tactor acceleration. Only four channels are shown for the EAI control unit since this is its max.
5.1 Syntacts spatialization demonstration - Right side shows user dragging target on computer, left side shows vibrations rendered on tactor array. Opacity of green tactors is directly tied to amplitude of vibration.
Tables

1.1 Summary of mechanoreceptors in skin [1]. † Multiple types of stimuli may activate each receptor type, stimulus described is optimal to elicit response. ................................................................. 3

4.1 Comparison of Tactor Control Methods ‡ Amplifiers, such as the Syntacts amplifier, are also necessary for audio methods and usually cost an additional $50 to $100 USD. † Latency not measured because Macaron is meant to design and test cues, not play in real time. † Only 4 channels can be played simultaneously. .......................... 48
Code Examples

1. Querying hardware information and using devices . . . . . . . . . . . . 23
2. Creating, mixing, and playing Signals . . . . . . . . . . . . . . . . . 25
3. Sequencing Signals in time . . . . . . . . . . . . . . . . . . . . . . . 27
4. Rendering the Signals . . . . . . . . . . . . . . . . . . . . . . . . . 28
5. Spatializing tactor arrays and modify parameters in realtime . . . . . . 29
Chapter 1

Introduction

Through the advancement of ubiquitous technology, such as mobile phones and computer interaction devices, the term “haptics” has become household. Simple haptic displays have been included in consumer video game controllers since the Sony DualShock controllers released in 1997 [7]. Cellular phones have also included haptics in the form of vibration since the 1990s. Despite the term becoming increasingly popular through the technologies being released to the public, the general public is only exposed to a small portion of the technology that exists. This thesis begins by introducing haptics in the broadest sense before motivating the work contained herein and exploring the state of the art surrounding the topic.

1.1 Haptics

In the most basic sense, haptics “refers to the sense of touch” [8]. A more specific definition in the scope of engineering is that “haptics is a tactile feedback technology which recreates the sense of touch by applying forces, vibrations, or motions to the user” [9]. Of course there are numerous ways to provide information through touch to the human, so haptics is generally divided into two categories: kinesthetic and cutaneous.
1.1.1 Kinesthetic Haptics

Kinesthetic haptic feedback has to do with the body using muscles and joints to understand the forces and torques applied to it. For example, when somebody picks up a metal block they expect to have to apply certain forces to the block to move it. If the block weighs significantly more or less than expected, the person may experience a different sensation than they were expecting. This is an example of how the body uses kinesthetic information to interact with the environment. The Immersion Force Feedback Joystick and Novint Falcon are examples of commercial haptic devices that provide kinesthetic feedback to the user to provide information about the environment, virtual or real, in which they are working.

1.1.2 Cutaneous Haptics

Cutaneous haptics has to do with interacting with the skin to provide information to the user. A very common instance of this is a cellular phone vibrating in the pocket of the user to alert the user that they have received a message. A more advanced example would be a device squeezing an amputee’s arm to signal the position of their prosthetic without requiring that they constantly look at it to determine its position. This thesis largely focuses on cutaneous haptic cues, so more discussion on the methods and current devices will be provided below.

1.2 Skin Mechanics

Cutaneous haptic devices use the skin as the means of conveying information to the user in a similar way that graphical interfaces use the eyes and audio devices use the ears. The skin is a large and complex organ for sensing that contains specialized
Table 1.1: Summary of mechanoreceptors in skin [1]. † Multiple types of stimuli may activate each receptor type, stimulus described is optimal to elicit response.

receptors to sense the world around it. The receptors that sense touch and motion are known as mechanoreceptors. The skin contains different types of mechanoreceptors that are adapted to sense different types of touch. Table 1.2 contains a basic summary of relevant mechanoreceptors. Notice that the types of mechanoreceptors that are present is based on the skin type, glabrous or hairy, largely due to the mechanical design of the receptor. Understanding the human sensing elements helps inform the design of devices meant to activate them. Later in this section I will discuss existing devices, but notice the general design of haptic cues falls into squeeze, stretch, or vibration categories which coincide with the optimal stimulus types for many of the mechanoreceptor types.

Figure 1.1 is included to visually consider why the mechanoreceptors function as they do. Merkel cells are the first type of slow-adapting cells. They are superstruc-
tures around the nerve endings that transmit pressure to the nerve endings, which is why they respond well to indentation or normal force. Ruffini endings are aligned and attached to folds of skin, so when these folds are moved, such as by a shear force on the skin, they activate. They are the second type of slow adapting receptors. The first of the fast adapting receptors are the Meissner corpuscles, which are fluid-filled, ellipsoid structures oriented normal to the surface of the skin within the papillae that can sense skin motion and low frequency vibration. The other rapidly adapting receptors are the Pacinian corpuscles, which are also fluid filled structures, but with a more “branched” shape that sense higher frequencies that can penetrate deeper into the skin because they are located in the deep dermis layer [10]. The free nerve endings are typically not targeted in cutaneous haptics because they are largely activated by noxious mechanical stimuli that would be at least unpleasant to the user. Likewise, the mechanoreceptors that are activated by hair fibers are not often targeted specifically.

Figure 1.1: Visualization of mechanoreceptors in skin [1]
1.3 General Types of Cutaneous Haptic Cues

Because the body reacts to certain mechanical movements to the skin, researchers try to target these movements when designing devices. This means the devices that exist in research and commercially can be generally sorted into a few categories. The devices referenced below are included only as examples for the reader, there are numerous other devices that have been created in each category.

1.3.1 Normal Force

Devices that generate normal forces are commonly referred to as squeeze devices. Often these devices wrap completely around an appendage of the user so that the radius of the “band” can be decreased to squeeze the skin underneath. These cues can be generated as binary, on/off, or along a continuum to provide information to the user. An example device that uses squeeze is HapticClench [11], shown in Figure 1.2. This is a wrist-based device that uses a shape memory alloy to actuate the device. Squeeze devices often produce shear forces coupled to the normal forces, though some new devices have tried to decouple the normal and shear forces [5].

Figure 1.2: HapticClench squeeze mechanism
1.3.2 Shear Forces

Shear forces are often used in haptics in two different modalities: linear stretch and twist. Linear stretch devices generally move a contact point on the skin in a direction along the surface. An example of a device that does this is the Rice Haptic Rocker shown in Figure 1.3a. Twist devices contact the skin and rotate around an axis normal to the skin. An example of this device is the twist module developed in Bark et. al. [3] and shown in 1.3b. This module uses two contact points on an end effector to rotate around a central axis to cause a “twisting” sensation to the user.

Figure 1.3 : Examples of haptic stretch devices. (a) Rice Haptic Rocker linear stretch mechanism [2]. (b) Twist haptic device [3]

1.3.3 Vibration

Vibration type haptic cues often of great interest to researchers and thus have been used in a large number of devices. Vibration devices oscillate the skin at some frequency to create the sensation. Part of the draw to vibration is the size of the
potential cue set from a single actuator and the ability to use multiple actuators. Also, these actuators do not require the same mechanical grounding that is associated with normal and shear cue devices. The use of vibration has recently exploded from basic alert cues to cues that convey rich information or methods that create pseudo-realistic sensations for users.

Discrete, as opposed to continuous, vibrotactile cues can convey rich information in a variety of ways. One common method is to use multiple tactors spaced on the user so that researchers can take advantage of spatial and temporal elements in their cue design. The MISSIVE uses four tactors spaced around the arm to increase the cue set available from the device [12]. Other devices have used multiple tactors to provide navigation information [13]. The vibrations themselves have also become more complex, but this discussion is reserved for Section 1.4.

Another way that researchers have begun to use vibration is in pseudo-haptic displays of virtual environments. As virtual reality (VR) devices become more popular, users want to feel even more immersed in the virtual environment. To do this, devices like Tasbi [5] use event-related haptic signals to convey a sense of what users visually see as haptic information on their skin. Timing between the visual and haptic presentations to the user is critical, and is a challenge that researchers must face to create realistic effects. The TESLASUIT (teslasuit.io) is another example of technology that uses vibration to enhance the realism in VR. Outside of VR, users have tried to enhance storytelling experiences to readers by providing pseudo-realistic effects such as rain [14].
1.3.4 Thermal

Though so far this thesis has largely focused on mechanical means to create cutaneous haptic sensations, other means have been used. One of the major methods is using Peltier elements to employ heating and cooling of the skin as haptic cues. Thermal haptics have been tested on the face in virtual reality [15] and on the hand to determine the limits of perception [16].

1.3.5 Multi-modal Interfaces

Researchers have recently become interested in combining modalities from the above sections into multi-modal devices. This works effectively because the different modalities focus on triggering different receptors and creating distinct sensations. Research has been accomplished showing that using multiple types of cues can be more effective than a single type [17], though providing these cues can interfere with one-another and makes device design more complex [18]. One example of a multi-modal device is MISSIVE, which uses rich haptic cues to convey language through phonemes [12].

1.4 Motivation

Each of these types of haptic cues have evolved over time through research. Vibration has been of particular interest to researchers due to the many options present that may help increase the size of the cue set. As was discussed in Section 1.3.3, some devices simply vary the length of the cue or its spatial position to convey different information. More complex waveforms have been implemented in some research, such as overlaying a low-frequency sawtooth wave onto a sine wave (Equation 1.1), to provide different sensations, simulate lower frequencies, or prevent perceptual adaptation. The research
that used this signal mapped absolute error exponentially to the amplitude of the signal [19]. The voltage of the signal is defined by $v$, $t$ is time in seconds, and $error_{abs}$ is the absolute pose error of a prosthetic.

$$v = \sin 250t \ast (10t)(0.5e^{error_{abs}})$$ (1.1)

Similarly there has been research into using specific patterns to increase the distinguishable cue set to determine what helps humans differentiate one vibration cue from another [20].

Despite the growth in complexity of vibration cues or their uses in VR, few cue design and implementation tools have been made available to researchers. Commercial solutions tend to have a predetermined library of cues, which limits researchers to using some variation of these cues. Some resources have been created to help with cue design, but they have been limited in their implementation capabilities. A full review of systems available to researchers is presented in Section 2.2.

Based on these limitations, researchers are limited to making their research fit within the parameters of the available tools or creating a custom tool that specifically fits their needs. Using these commercial tools can mean adapting entire projects around these tools and possibly a large financial investment. On the other hand, developing custom tools means spending large amounts of time to design and build an entire system that fits the project needs.

The work in this thesis was motivated knowing that researchers needed a useful, flexible tool that could design and implement vibration haptics without the large investment or specialized implementation that commercially available tools currently provide.
1.5 Contributions

The following chapters of this thesis will describe in detail a system that fills the need for an open-source vibrotactile cue design and implementation system. Chapter 2 will discuss the current state of the art of vibrotactile haptics hardware. Chapter 3 presents the Syntacts framework, including both the hardware and software components. Finally, Chapter 4 describes the verification testing done on the Syntacts framework and compares Syntacts to other implementation options.

The specific contributions of this thesis through the Syntacts framework are:

- Ultra-low latency cue generation and rendering available in C++, C#, and Python languages
- Graphical user interface designed for complex cue design and editing
- 8-channel, linear power amplifier board to amplify analog signals from an output device to be rendered on the vibration motor
- Technologically and financially accessible tool allowing entry for cross-disciplinary researchers to implement haptic vibration into new and existing projects
Chapter 2

Mechanics of Vibrotactile Hardware

Many systems that use vibration motors are composed of three general parts: the signal generator, signal conditioning, and mechanical output. The generalized process is shown in Figure 2.1. The mechanical output is often in the form of a vibration motor, which physically moves at a desired frequency to trigger the mechanoreceptors in the skin. These motors are driven by electrical signals that are generated by some device according to user-defined parameters. These signals are often generated by low voltage or low power components, so they must be conditioned before being sent to the motor. Each of these components of a vibration system are discussed below.

2.1 Vibration Motors

As previously discussed, vibration motors are the portion of the system that converts the electrical signals into mechanical vibrations for the user to feel. There are many different options readily available at varying cost from commercial sources. These options can largely be sorted into four major categories based on how they work and

Figure 2.1 : Generalized flowchart of vibration system
their signal requirements. This section is meant to provide a general overview of vibration motors, not to be a full review of all available motors.

2.1.1 Eccentric Rotating Mass Actuators

Eccentric Rotating Mass Actuators (ERMs) were the first type of vibration motor popular in haptics. An example of these motors is shown in Figure 2.2 inside a video game controller from Sony that was released in 1997 [7]. ERMs are DC motors with an eccentric mass, as the name would imply, attached to the shaft. These are controlled by a DC signal to spin the mass at different speeds to create vibrations. This means that amplitude and frequency are directly coupled. There are many options that exist for these type of motors, including fully enclosed and different sized versions.

![Figure 2.2: Video game controller with DualShock Vibration System highlighted](image)

2.1.2 Linear Resonance Actuators

Another common type of vibration motor is the linear resonance actuator (LRA). This type of actuator has become popular due to its small size and being completely enclosed. These actuators are essentially a spring-mass system, shown in Figure 2.3,
where a electromagnet is excited by an oscillating signal that forces the mass against the spring causing oscillating motion, therefore vibration.

![Figure 2.3: Internal diagram of LRA [4]](image)

Due to their mechanical design, LRAs generate the highest amplitude, measured by normal acceleration, within a narrow range of frequencies, compared to other motors. This frequency is calculated by the manufacturer according to the principles of simple harmonic motion. Equation 2.1 shows that this frequency is a function of the spring constant, $k$, and the mass of the moving mass, $m$. The motor can be driven at off-resonant frequencies, but it will generate lower acceleration amplitudes.

$$f = \frac{1}{2\pi} \sqrt{\frac{k}{m}}$$  \hspace{1cm} (2.1)

### 2.1.3 Voice CoilActuators

Voice-coil actuators are similar to LRAs in that they involve linear motion of a mass to generate vibration. The major difference is that voice coils do not use a spring, instead
use the polarity of the magnet to drive it in the different directions. Examples of these tactors are Engineering Acoustics C-2 (https://www.eaiinfo.com/product/c2/) and TacHammer (https://nanoport.io/haptics/) products. The mechanical design generally makes these on the largest in size of the vibration motors, but they are also typically more powerful than LRAs.

2.1.4 Piezoelectric Actuators

Piezoelectric actuators are different from other actuator types because they generate vibration through the electrical signal being applied to electrodes, which causes deformation of the electrodes. This deformation is felt as vibration when it is applied against the skin. These devices can be ultra-thin as they are thin pieces of metal, but do require voltages at and above 24V.

2.2 Creating the Signal

One of the central issues in using vibration in haptics is how to generate the electrical waveforms that are required to properly oscillate the vibration motor that mechanically vibrates the skin of the user. There are a number of factors that make choosing the proper equipment challenging, including waveform shape, number of channels, and noise minimization. A summary of waveform generating methods follows as a survey of the current options.

Actuating these motors usually, but not always, uses an open-loop control approach where the desired input-output relationship is known for each specific type of motor. The control signal must be generated by some device that is well-suited for the application.
2.2.1 Waveform Generators

Possibly the simplest implementation approach is using a waveform generator connected directly to the tactor. This is easy to do because the waveform generator is purpose-built to create these shapes. Using the datasheet for the tactors, it is possible to set the characteristics of the signal on the waveform generator and simply play it. These can be a robust tool that creates samples at greater than 100 MHz, though this is a much higher rate than necessary to drive vibration motors.

The waveform generator is not designed as an implementation tool; therefore, it can only control one to four tactors, depending on the type, and only play the cue that is created manually on screen. This is useful for initial testing and possibly some initial cue design, but has limited further usefulness.

2.2.2 Commercial Closed-Box Solutions

There are a few commercial solutions to the problem of controlling vibration motors. One of these is the tactor control unit from Engineering Acoustics, which is the recommended controller for driving their voice-coil actuators that are commonly used in haptics research. Note that the hardware can only drive four tactors at the same time. It is capable of interfacing with Windows, Linux, or Android devices and has eight output ports to control up to eight tactors independently.

The primary limitation of this method is the high upfront cost of the controller itself. Not all researchers who are trying to implement vibration motors into their projects are willing or able to invest large sums of money (on the order of $2,000+), which limits the research that is done in the arena.
2.2.3 Commercial I²C Solutions

Texas Instruments makes a purpose-built integrated circuit (IC) controller called DRV2605L for Eccentric Rotating Mass motors (ERM) Linear Resonant Actuators (LRA). This controller boasts seven different modes to control the sinusoidal output wave amplitude. Another feature of the device is that it is capable of closed-loop control by detecting the back-EMF caused by the motor in ERMs and the motion of the magnet in LRA motors. This feature allows the chip to detect the resonant frequency of the tactor as to ensure that the tactor is always outputting the strongest vibration it can. The chip is capable of driving the tactor at an off-resonant frequency using the open loop PWM-control architecture, as well as varying the amplitude to create different cues.

The recommended way to minimize the latency of this chip is to pre-select the cue from the library so that all that needs to happen to accelerate the cue is to send the trigger signal. The library contains 123 variations of cue components, of which 7 can be sequenced at a time. For implementations where there may be more than one type of cue that the user may choose to actuate, such as different types of physical buttons or a VR environment, this latency could be enough to lessen the realism that the device is meant to create.

This system is commonly used for prototyping due to low cost and availability of components. However, the system requires I²C protocol in order to program the chip, which is an extra step of programming necessary if the designer is using multiple devices that may not also include I²C. This may be accomplished for an Arduino device and premounted chips from Adafruit for prototyping; however, full implementation requires creating a custom circuit board to mount the chips themselves, which is difficult to do properly and can become very expensive.
Texas Instruments also offers the DRV2605LEVM-MD (Evaluation Module-Multiple Drivers) that functions both as an evaluation module for the chips and various vibration motors, and as an example of how to construct a multi-driver system. Shown in Figure 2.4, the board features a microcontroller, multiplexer, and eight DRV2605L vibration motor drivers. It also exposes pins for \text{I}^2\text{C} communication to the components.

### 2.2.4 Audio Based Solutions

Another method to control vibration that has become popular recently is using audio interfaces. Digital audio devices, many examples of which will be discussed in Chapter 3, are designed to take in a signal and output it to various numbers of channels, which are usually speakers. Though the signals that audio devices usually handle are much more complex than the signals that are used to drive vibration motors, the hardware
is ready out-of-the-box to handle these oscillating signals. There exist a few prior examples of systems that have specifically taken advantage of audio techniques

**Macaron**

One example of a system that relies on audio techniques is Macaron [21]. This is a web-based vibrotactile haptic editor with examples that allows users to edit amplitude and frequency of a sine wave, play it over hardware or save it as a file. It is meant to allow users to design and test cues on the fly, as they demonstrated in their publication.

Macaron is built on the Web Audio API, which is a tool created to allow web developers and musicians to include audio into their projects. Currently, Macaron only supports a single channel of audio output and is not designed to be incorporated into other projects.

**Stereohaptics**

Stereohaptics is another system that relies on audio technologies to create haptic displays [22]. It is based on Max programming language and designed to deliver haptic vibrations through off the shelf audio components. Stereohaptics includes a custom PCB to facilitate two channels of output and a single input channel for feedback. The framework is scalable to include multiple stereohaptics boards to allow for more tactors. Using this framework requires an active subscription to use the Max software, which adds an ongoing cost for use.
2.3 Amplification

When working with electrical signals, especially when they are generated by a computer, they will need to be amplified in some way before they are capable of properly driving vibration motors. Power amplifiers are useful to increase the available power in the system so that the full power, voltage and/or current, is not drawn through the computer. These amplifiers are divided into classes based on their output type and construction. There are more types than listed here, but the below are the most common.

2.3.1 Linear Amplifiers

The first of these types of amplifiers are linear amplifiers. Class A amplifiers allow current to flow continuously through the circuitry to avoid the distortion of turning components on and off. Class B amplifiers comprise > 99% of all audio amplifiers and operate using two complementary transistors to create the amplified signal. Class AB amplifiers are a combination of the above classes that pairs the low-distortion of Class A with the better efficiency of Class B [23].

2.3.2 PWM Amplifiers

Another common type of amplifier contains no linear components and is called Class D. These amplifiers use components that are switched at ultrasonic frequencies, using pulse width modulation (PWM), to approximate the input wave [23]. These have become increasingly popular due to their very high efficiency, so they work well with batteries. A difficulty of using these amplifiers in larger systems is they operate on digital circuitry, which can create noise issues if used in conjunction with linear circuitry.
2.4 Summary

Having thoroughly reviewed examples of vibration motors, the currently available systems to control the LRA and voice coil types, and common amplifiers available to accomplish signal conditioning, I found a gap between the available systems and their new uses. As discussed in Section 1.4, the available systems lacked the cue design and system integration functions necessary to easily be used in complex, immersive environments such as VR. Thus, a new system to design cues and integrate haptics with new and existing systems was necessary to allow more researchers to experiment with haptic vibrations without limiting their research to the current systems or having to create their own system.
Chapter 3

Syntacts

Syntacts was created based on using the current systems outlined in Chapter 2 and driven by the needs of both Tasbi [5] and MISSIVE [12]. Through thorough review of haptic vibration research, we designed Syntacts to meet a variety of needs and be easy to use for researchers in a large variety of disciplines. Syntacts’ primary goal is to provide a flexible, code-oriented, interface that can be easily integrated with existing software and applications. The following chapter discusses the software capabilities and hardware designs included in the Syntacts framework in detail.

3.1 Software

The Syntacts software library was created with the goal of being as intuitive as possible while facilitating necessary capabilities to fulfill the requirements set forth. The requirements we felt we needed to fulfill to create a useful software tool were:

- a user friendly API that integrates with existing code
- direct access to external sound card devices and drivers
- flexible and extensive waveform synthesis mechanisms
- the ability to generate and modify cues in realtime
- spatialization of multi-channel tactor arrays
• saving and loading cues from a user library

• compatibility with existing file formats and synthesizers

• a responsive GUI for cue design and playback

• created with open-source programs and libraries

The final product fulfills all of these requirements and more. The following sections discuss how the software is implemented and its functionality. Syntacts is completely open-sourced with a permissible MIT license. Source code and compiled binaries are available for download from github.com/mahilab/Syntacts.

3.1.1 Programming Interface

The library is written in C and C++ to facilitate accessing low-level drivers and maximizing performance. Additionally, bindings are currently provided for C# and Python. The former is particularly useful for integrating Syntacts with Unity Engine for creating 3D virtual environments, while the latter allows for high level scripting and interactivity (e.g. with Jupyter notebooks). Integration with other languages is possible via C shared library (i.e. DLL) loading. Code presented in this section is taken from the Python binding, but the native C++ API and C# binding are nearly identical in their syntax and usage.

Development environments and compilers that can handle each of these languages are available free of charge. This means that using the software has no up front or ongoing cost like other systems.
3.1.2 Interfacing Devices

Syntacts will interface with virtually any audio card on the commercial market. The API allows users to enumerate and select devices based on specific drivers, a feature typically unique to professional commercial software. While Syntacts can open devices under any audio API, users should be mindful of the considerations discussed in Section 3.8, favoring low latency options such as ASIO and WASAPI. The code in Listing 1 is an example of the available options when interfacing with the digital audio devices. Further discussion of these devices can be found in Section 3.2.1.

```python
# create an audio context
session = Session()

# enumerate connected hardware
for dev in session.getAvailableDevices():
    print(dev.index)  # e.g. 6
    print(dev.name)  # e.g. MOTU Pro Audio
    print(dev.maxChannels)  # e.g. 24
    print(dev.api)  # e.g. ASIO
    ...  # etc.

# open device 6 with 24 channels at 48 kHz
session.open(6, 24, 48000)

# your code here
session.close()  # close device
```

Listing 1: Querying hardware information and using devices

Library usage begins with creating an audio context, or Session. A Session opens communication with a requested audio device and starts an output stream to it in a separate processing thread. Within a Session, there is a command that can search for
the devices available on the computer. The computer assigns each device an index, which is how the device is later referenced in the open command. Each device has several attributes, including name, available channels, API, and sample rate, that describe communicating with each device. The open command tells the session which device to open and includes options on how to communicate with the device. When the thread is finished using the device, it is good practice to close the device. This is because some audio drivers can only support a single device at a time, so to prevent conflicts devices should be closed when not in use.

3.1.3 Creating Effects with Signals

Making the vibration waveform itself is accomplished through abstract classes called Signals. Signal classes define a temporal sampling behavior and length, which may be finite or infinite. A wide variety of built-in Signals are available in Syntacts. For example, the classes Sine, Square, Saw, and Triangle implement typical oscillators with normalized amplitude and infinite duration, while Envelope, ASR, (Attack, Sustain, Release), and PolyBezier define amplitude scaling functions with finite duration. Signals can be mixed using basic arithmetic operations. The act of multiplying and adding Signals can be thought of as an element-wise operation between two vectors. Multiplying two Signals yields a new Signal of duration equal to the shortest operand, while adding two Signals yields a new Signal of duration equal to the longest operand. Gain and bias can be applied to Signals with floating point scalar operands as well.

In Listing 2, the Signals `sqr` and `sin` are implicitly of infinite length, while `asr` has a length of 0.3 s. Multiplying `sqr` by `sin` yields another infinite Signal with a 100 Hz square carrier wave, amplitude modulated with a 10 Hz sine wave (`sig1`). This Signal can further be given shape and duration by multiplication with `asr` to
yield the finite Signal `sig2`. The Signal `sig3` represents another form of modulation through summation instead of multiplication. While the examples here only demonstrate passing scalar arguments to Signal constructors, some Signals can accept other Signals as their input arguments. For instance, it is possible to pass `sin` as the frequency argument to `sqr`’s constructor, yielding a form of frequency modulation. The modularity of the API allows users to create a wide variety of effects with minimal code. Syntacts can also be easily extended with custom user-defined Signals simply by creating classes which define the functions `sample` and `length`.

```python
sqr = Square(100)  # 100 Hz square
sin = Sine(10)  # 10 Hz triangle
asr = ASR(0.1,0.1,0.1)  # attack, sustain, release

# basic examples mixing the Signals above
sig1 = sqr * sin
sig2 = sig1 * asr
sig3 = 0.5 * (sqr + sin) * asr

# play Signals on channel 0 and 1
session.play(0, sig1)  # plays until stopped
session.play(1, sig2)  # plays for 0.3 seconds
...
session.stop(0)  # stop sig1
```

Listing 2: Creating, mixing, and playing Signals

The signals in Figure 3.2 correspond to the signals in Listing 2.
3.1.4 Sequencing Signals

Multiple Signals can be concatenated or sequenced temporally to create patterns of effects using the insertion, or left-shift, operator. Consider the examples in Listing 3. First, two finite Signals \texttt{sigA} (0.3 s) and \texttt{sigB} (0.4 s) are created. Signal \texttt{sig4} demonstrates their direct concatenation, resulting in a 0.7 second long vibration where \texttt{sigB} is rendered immediately after \texttt{sigA}. Delay and pause can be achieved through the insertion of positive scalar operands, as shown in \texttt{sig5}. Inserting negative scalars moves the insertion point backward in time, allowing users to overlay or fade Signals into each other as in \texttt{sig6}. Sequences of Signals can also be sequenced as in \texttt{sig7}. 

\begin{figure}[h]
  \centering
  \includegraphics[width=\textwidth]{signals.png}
  \caption{Signals created in Listing 2}
\end{figure}
sigA = Sine(100) * ASR(0.1,0.1,0.1) # 0.3 s
sigB = Sine(50) * ADSR(0.1,0.1,0.1,0.1) # 0.4 s

sig4 = sigA << sigB # 0.7 s
sig5 = 0.1 << sigA << 0.2 << sigB # 1.0 s
sig6 = sigA << -0.1 << sigB # 0.6 s
sig7 = sig4 << sig5 << sig6 # 2.3 s

session.play(2,sig7)

Listing 3: Sequencing Signals in time

Figure 3.2 : Sequenced Signals created in Listing 3

3.1.5 Rendering the Signal

Once the desired signal has been created, rendering the signal on the tactor is done by playing the signal. Listing 4 contains examples of playing the signal. The first signal, inf, is a basic sine wave with infinite length and sig is inf with an envelope applied to it. Signals can either be played over all channels simultaneously or a single channel. If a subset of channels are needed to play the same cue, the play function
can be used inside a loop. If a signal does not have a defined length, the \texttt{stop} or \texttt{stopAll} command must be used to silence the channel. It is good practice to use a \texttt{stop} command with finite length cues as well to ensure there is nothing playing on a channel.

\begin{verbatim}
inf = sin(175)  # sine wave with infinite length
sig = inf * asr(1,1,1)  # sine wave with ASR envelope, length 3s

session.playAll(inf)  # play cue on all channels
session.stopAll  # stop playing on all channels
session.play(3, sig)  # play signal on single channel
sleep(sig.length)  # sleep script for length of signal
\end{verbatim}

Listing 4: Rendering the Signals

3.1.6 Spatialization and Realtime Modifications

In addition to playing Signals on discrete channels, multiple channels can be mapped to a normalized continuous 1D or 2D spatial representation with the Spatializer class. Generally, users would configure a Spatializer according to the layout of a physical tactor array. Channel positions can be set individually or as uniformly spaced grids. Only channels within a \texttt{target} zone are played, and their volume is scaled according to a specified drop-off law (e.g. linear, logarithmic, etc.) based on their proximity to the target location. By moving the target location in, for example, a \texttt{while} loop, users can create sweeping motions and the illusion of continuous space with their tactile arrays (Listing 5, Fig. 3.3).

Other parameters, such as master volume and pitch, can be modified in realtime
spatial = Spatializer(session)  # 2D Spatializer
spatial.createGrid(4, 6)        # 4 rows by 6 cols
spatial.setPosition(18, 0.1, 0.8) # move channel 18
spatial.setRadius(0.3)          # effect radius
spatial.setTarget(0.2, 0.1)     # location to play
spatial.setRollOff('linear')    # roll-off law
spatial.play(sig1)              # play inf Signal

# modification in a loop
while condition:
    ...
    spatial.setTarget(x, y)
    spatial.setVolume(v)
    spatial.setPitch(p)
spatial.stop()

Listing 5: Spatializing tactor arrays and modify parameters in realtime

for Spatializers or individual channels. This offers users the ability to move beyond playing discrete, pre-designed cues, to instead modifying continuous cues in response to conditions within the application. For example, consider the VR application in Fig. 3.4. In addition to pre-designed haptic effects that are triggered for specific events (such as button clicks), a continuous haptic effect is rendered when the user’s hand is within the fan air stream. Volume, pitch, and spatialization are changed based on hand proximity and orientation, and the fan speed, respectively. Figure 3.4 illustrates the C# binding of the Syntacts API being used to provide haptic effects for a virtual fan built in Unity Engine. Two usage paradigms are in effect. The first leverages pre-designed, finite Signals for knob detents (designed in the Syntacts
GUI and loaded at runtime) and button contact events (created programmatically on-the-fly, parameterized by hand approach velocity). The second paradigm uses an infinitely long Signal for the fan air stream. The volume and pitch of this Signal are modified in realtime based on the user’s hand location and the fan speed, respectively. One-dimensional spatialization is used to target only the tactors which are oriented toward the fan in a continuous fashion.

Figure 3.4 : Fan example - this figure demonstrates the variable design paradigms that can be used within Syntacts to create realistic vibration cues.
3.1.7 Saving and Loading Signals

User-created Signals can be saved to disk and reloaded at a later time using the free functions `saveSignal` and `loadSignal`. The default file format is a binary representation of the serialized Signal. That is, instead of saving all individual audio samples, only the parameters needed to reconstruct the Signal at runtime are saved. This method results in considerably smaller files which can be loaded more quickly on the fly than typical audio file formats. Nonetheless, Syntacts can still export and import WAV, AIFF, and CSV file formats for interoperability with existing haptic libraries.

3.1.8 Graphical User Interface

Syntacts also includes a graphical user interface (GUI) to help demonstrate the functions of Syntacts and to allow users to design and iterate on cues in a visual workspace. It is distributed as an executable program that can be used as a standalone application on computers so that users can design cues without installing all portions of Syntacts should they not need to for any reason.

The center portion of this GUI contains three tabs: Designer, Sequencer, and Spatializer. On the designer tab, users can drag Signal options from the Pallette on the left. These options can then be edited in the Designer tab and the visualizer below the tab will update in real time to visualize the waveform. The current signal can be played by clicking on a channel number, or on all channels by clicking on the channel number at the top. The Sequencer tab allows cues to be sequenced in time and then played in the same way as the Designer tab. Finally, the Spatializer is a visual version of the spatialization tools available in Syntacts and discussed in Section 3.1.6.

Once a cue is designed, the GUI can save or export the cue under the Library
tab on the left of the application. Waveforms can also be loaded into the library by
dragging and dropping them onto the Library.

Figure 3.5 : Syntacts GUI - The left-hand side demonstrates cue design. Users
drag, drop, and configure Signals from the design Palette to the Designer workspace.
The Signal is visualized and can be played on individual channels of the opened
device. The right-hand side show’s the GUI’s spatializer (background) and track-
based sequencer (foreground) interfaces. Once designs are complete, they can be
saved for later loading from any of the programming APIs.

3.1.9 Dependencies

Syntacts takes advantage of various third-party software. In addition to some stan-
dard packages included in C++, Syntacts depends on:

- **portaudio** - cross-platform, open-source, audio I/O library that Syntacts uses
to interact with audio drivers and devices available at [www.portaudio.com](http://www.portaudio.com)
- **cereal** - C++ library for serialization, allowing the use of XML and JSON
  formats available from [uscilab.github.io/cereal](http://uscilab.github.io/cereal)
- **mahi-gui** - open-source C++ software to construct the GUI available from
  [github.com/mahilab/mahi-gui](http://github.com/mahilab/mahi-gui)
Note that all of these dependencies are free and open-source so that no part of Syntacts software requires cost or subscription.

3.1.10 Translation to Other Programming Languages

Many labs who would be the target users for Syntacts likely would prefer to use other coding languages than C++ due to their tools already being implemented in other languages. One of the primary goals of Syntacts is to lower the technical barrier to entry for vibration haptics, and to accomplish this goal Syntacts should be available to use in languages people are already comfortable using. For this reason, Syntacts is currently available in C# and Python languages. C# is used by the Unity Engine to create traditional and virtual-reality video games. The amount of research currently happening in this field, discussed in Section 1.4 shows that there is a need for C#. In addition, Python is a relatively simple language that is popular in research and implementation so this is included to reach the widest audience possible.

The implementation is done through a dynamically linked library (DLL) in Microsoft or .dylib on MacOs. This DLL is created in the C language and made available in the other language files. Both C# and Python include a file that translates this into the respective language in a way that best mirrors the other languages. There are some functional components that do not appear in these languages (such as frequency modulation), but the vast majority of the library remains. In the future, there are plans to add Matlab functionality, but at the time of this publication it is still in development.
3.2 Hardware

The hardware that is specific to Syntacts includes the Sound Card and Syntacts Amplifier. In this section I discuss the available options for the Sound Card and then the custom-made amplifier for this audio-based system. An example hardware setup is shown in Figure 3.6.

![Figure 3.6: Syntacts In Use - This figure demonstrates one possible implementation and use of Syntacts hardware and software, using Tasbi [5] as an example. The top-half shows an audio-device (MOTU 24Ao) connected to two Syntacts amplifier boards that have been integrated into Tasbi control units. Each Tasbi bracelet incorporates six radially spaced LRA tactors, for a total of twelve utilized audio channels. The audio device interfaces to a PC (not shown).]

3.2.1 Sound Cards / Digital-to-Analog Converters

The most important piece of hardware for audio based control is the digital-to-analog converter (DAC) device. The DAC is responsible for converting digitally represented waveforms, like music files, to analog signals to be played though headphones or speakers. Virtually all PCs have a DAC integrated into the motherboard which outputs two analog signals through a headphone or line out jack (typically a 3.5mm phone jack) for left and right audio channels. If no more than two vibrotactors are
needed, use of the built-in headphone jack may be sufficient for some users.

Driving more than two channels generally requires a dedicated DAC, or sound card. The least expensive options are consumer grade surround sound cards, which can be had in typical PCI-e or USB interfaces. Up to six tactors can be driven with 5.1 surround-sound cards, while up to eight can be driven with 7.1 surround sound cards. We have found this to be viable solution if consideration is given to differences between channel types (e.g. subwoofer channels are usually tuned for lower impedance loads than speaker channels). Offerings from Creative Soundblaster and Asus are among the most readily available choices.

![Sound Cards](image)

Figure 3.7: Example consumer-grade sound cards. (a) Creative SoundBlastser Audigy Rx (b) Asus Xonar U7 MKII

There also exist professional grade audio interfaces with more than eight outputs, such as the MOTU UltraLite-mk4 and 16A with 12 and 16 channels, respectively. However, professional devices may also feature other I/O channels (e.g. MIDI, S/PDIF, etc.) that increase cost and are of little use to driving tactors. For even higher channel counts, the purely analog output MOTU 24Ao is a popular choice [24, 25]. A single unit provides 24 simultaneous output channels, and up to five units
can be connected using IEEE Audio Video Bridging (AVB) to drive 120 vibrotactors if desired.

The number of channels that a device can control greatly affects the overall cost of the project. Cost comparison is included in Table 4.3. Before purchasing a device, users should evaluate the maximum number of tactors they desire to control to ensure they invest the minimum financially while being able to fulfill their project requirements.

3.2.2 Hardware Drivers and Buffer Sizes

![Image of latency graph with various sound card APIs and their latencies]

Figure 3.8: Mean Windows audio driver API latencies with standard deviation. Data collection methods are described in Sec. 4. For reference, the dashed line indicates the perceptual threshold of visual-haptic simultaneity [6].

An extremely important consideration in sound card selection is the device’s driver API support. An API describes a digital audio transmission protocol, and most drivers support many different APIs. Windows standardizes at least four first-party
APIs: WDM-KS, WASAPI, MME, and DirectSound. As shown in Fig. 3.8, not all APIs are created equally. Because MME, which exhibits highly perceptible latency, is usually the default API, it could be easy to conclude that audio is insufficient for realtime haptics. Steinberg’s third-party ASIO driver is widely considered to be the most performant option, but it is often only implemented by professional grade equipment. Regardless, API selection is a rather opaque setting under Windows, and appropriate software is usually required to select the preferred driver API (see Section 3.1).

![Figure 3.9 : Latency increase with increase in buffer size](image)

Another important consideration is audio buffer-size, or the number of audio samples sent on every transmission to the device. A buffer size of 256 is typical, but some devices can go as low as 16, offering further reduced latency (Fig. 3.9). The smallest buffer size is preferable, but some computers will be limited by their speed to a certain buffer size. If the buffer is set too small, random “pops” or “clicks” will be felt on the tactor. The optimal buffer size for each hardware setup can be determined through in place testing.
3.2.3 Amplifiers

Audio DACs typically output a low-power signal at what is called “line-level” because they expect that the output device will amplify the signal before it is actually played. Vibrotactors are similar to typical 8 to 16 Ω speakers, and therefore require amplification. As discussed in Section 2.3, amplifiers are divided into different classes based on how they operate. To drive vibration motors, Digital Class D amplifiers are the most common. They expect an analog input signal and output an amplified version of the signal with pulse-width modulation (PWM). This type of amplification tends to be very power efficient, but high frequency PWM switching can add large amounts of electrical noise to a system. This issue is especially common when designing for arrays of vibrotactors, where multiple naively implemented Class D amplifiers can create enough noise to be physically felt. The initial amplifier used with Syntacts was the TPA3110 Stereo Class-D amplifier chip. In order to fulfill our need at that time, three of these chips were connected to control six channels of vibration motors. In the use case in Figure 3.6, the amplification is done on the grounded surface and the signal transmitted to the motors via a cable. In the same cable is an analog signal from a force sensor within the device. When these amplifiers were powered on, noise could be felt on the vibrotactors and electrical noise rendered the signal on the force measurement unusable.

Class A, B, and AB amplifiers are linear amplifiers. These amplifiers tend to have much lower efficiency than the Class D, which can lead to heat problems if their thermal design is overlooked. However, because they do not constantly switch at high frequencies, they introduce considerably less noise into the overall system. Figure 3.10 shows the relative magnitudes of the electrical noise on the force measurement between Class D amplifiers, Class AB amplifiers, and no amplifiers. These data were taken
Figure 3.10: Noise comparison on parallel analog signal wire from TASBI [5] use-case from a wire adjacent to the signal carrying wires in a standard HDMI cable. The signal wires only had the amplifiers attached, but nothing was played. The measured wire carried a 1V analog signal to simulate a force measurement and illustrate the detrimental effects of noise. The signal itself had a maximum of 2.2mV of noise. When the Class AB amplifier was turned on, the maximum noise value increased to 3.1mV. In comparison, the Class D amplifier increased the noise to 15.3 mV. As seen in Figure 3.10, the signal with the Class D amplifier turned on could be large enough, especially while using fine equipment, to make the signal nearly useless. This noisy signal is the same noise that could be felt in the vibrotactors while attached to TASBI. As another example, we noticed considerable noise emission from the C2 tactors and EAI control unit (which also utilizes switching amplifiers) in MISSIVE [12] during EEG measurements.

In addition to these findings, we saw that a stable power supply is critical to the amplifier’s ability to condition the signal. Batteries or linear power supplies provide much more stable power than typical switch-mode power supplies. Despite the potentially increased cost, using these more stable means of power supply also
contribute to minimizing the noise in a system.

Figure 3.11 : Syntacts Amplifier Board, both on its own and integrated into a system from TASBI [5]

Based on these difficulties and limited commercial options for high density output, we designed the purpose built, eight channel Syntacts Amplifier board (Fig. 3.11). It is based on the TI TPA6211A12 3.1W audio power amplifier IC, featuring a Class AB architecture and fully differential input and outputs that eliminates noise issues we’ve experienced with commercial options. Using a differential output amplifier also removes the negative side of the vibration motor from ground. Removing the tactors from ground isolates them from noise on the ground plane that could be felt on the tactors themselves. The board is designed for 8-channels to minimize the number of boards needed for large projects and take advantage of standard connectors.

Figure 3.12 : Side view of Syntacts amplifier V1.1
In order to make these boards most useful to the widest range of digital audio devices there are two versions of the board. The first, shown in Figure 3.12 uses a DSUB-25 connector arranged in the standard AES-59 configuration as the signal input connector. This type of connector is standard for users who make their own cables as well as the output from the MOTU 24Ao. The other version of this board, shown in Figure 3.13, uses standard headphone jacks as the signal input, known as 3.5mm jacks. The output of both of these versions it a standard IDC header that accommodates ribbon cables or prototyping wire. It also requires 5-5.5V supplied to the screw-gate terminal blocks to power the amplifiers.

Designs for the two versions are available online along with data sheets and assembly instructions. We also provide bundled CAD files and parts lists correctly formatted for submission to online turn-key PCB manufactures, where the board can be built for roughly $50 USD each in sufficient quantities. Components can be ordered from online parts suppliers. In addition, the components used on this board are sized for human assembly with a soldering iron or reflow oven.
Chapter 4

Comparison and Verification

In this chapter, we evaluate Syntacts against two of the commercially available control options discussed in Section 2.2: the EAI control unit, and the TI DRV2605LEVM-MD evaluation board. These two commercial options were chosen because they are each meant to implement multiple vibration motor channels from a computer interface. Both are commercially available to the public and often used in research or other applications.

4.1 Latency Test Setup

Latency is a critical measure of a system’s ability to render cues, especially for time sensitive applications such as VR. For high density tactile arrays, latency can increase with the number of channels simultaneously played since each subsequent channel adds more processing or transmission time. If multiple channels are played at once, the last actuated channel may lag the first actuated channel by several milliseconds depending on the overall implementation. For this reason, we chose to benchmark latency as a function of the number of channels played.

4.1.1 Test Rig

The test setup was designed to match the factory testing rig for the Mplus ML1040W LRA vibrators that were used. An accelerometer (TE Connectivity 4000A-005) was attached perpendicular to gravity on a 100g block of acrylonitrile butadiene styrene
(ABS) sitting on a layer of polyurethane foam to mitigate external vibrations. The accelerometer sent differential-ended analog acceleration signals to the Quanser QPID data acquisition board. This board was chosen due to its ability to collect data at extremely fast rates.

4.1.2 Vibrotactile System Setup

Each of the systems required a different means of implementation. To establish a fair comparison, each system was designed to minimize latency, as reported by the manufacturer. Figure 4.1 visually demonstrates the protocol the testing followed for each system.

![Software protocols to compare latency](image)

Figure 4.1 : Software protocols to compare latency
Syntacts was used by including the library into the testing project and using the commands available from the library as it is designed to be used. The cue that was played was created during the testing just as it may be in actual usage. The cue was then sent to the digital audio device to convert the cue to an analog form and render it through the Syntacts amplifier board to the vibration motor. Syntacts software was configured to control a MOTU 24Ao under the ASIO driver API and a buffer size of 16. Later the digital audio device was swapped for other options to collect data for comparison, though the audio API and buffer size remained constant.

The Engineering Acoustics Tactor Control Unit and software were set up similarly to Syntacts. The provided development kit from EAI was included in the main project and the commands were used. In this case the commands were sent to the Tactor Control Unit, which output a signal to the vibration motor. The cue was created by setting a frequency, amplitude (gain), and duration on each channel, then played directly after.

The Texas Instruments system needed different implementation than the previous systems due to its use of I^2C communication protocol. In order to communicate from a standard computer output port, an Arduino Uno board was necessary as an intermediary. In order to accomplish this, the Arduino device was connected to the main program by opening the communications port and using serial communication. The Arduino was then connected to the TI DRV2605LEVM-MD via the standard two I^2C communication lines, as discussed in Section 2.2.3. A pre-compiled file that contained the necessary commands to set the cue and play the cue had been written to the Arduino so that only two bytes needed to be communicated by the main program to send a command to the DRV chip. According to the manufacturer’s datasheet, loading and playing a cue from the library onboard the chip is the way to use the chip
with minimal latency. Therefore, the first command sets the proper cue, and the play command renders the cue. Due to the delay in retrieving the cue from memory, the cue was initially loaded before any testing and each trial simply ensured the proper cue was loaded instead of reloading each trial.

4.1.3 Latency Testing Protocol

The testing was done using a specific protocol to ensure each test was accomplished the same way. As shown in Figure 4.2, the testing began in the main C++ executable where the type of test and test options could be specified. The executable sent commands to set and play to the vibration system selected as described in Section 4.1.2. The selected system sent a signal to the vibration motor attached to the test rig. The accelerometer attached to the test rig then sent acceleration data to the data acquisition device (DAQ). The main program collected this data and controlled timing.

![Figure 4.2: Latency testing protocols](image-url)
Each trial adhered to the following sequence:

1. Reset timer to zero and begin recording acceleration data
2. Create new cue to be played
3. Play cue
4. Continue collecting data for 1.5 seconds
5. Stop collecting data and save file

This C++ testing application, also available on GitHub, controlled the experiments and ran 100 trials for each device. Data was collected with a Quanser QPID digital acquisition device polled at 50 kHz to ensure accurate capture of the oscillating waveform. All systems rendered a 178 Hz sine wave between ±5V with a duration of 1,000 ms to ensure equity between systems. This waveform was chosen because it was available within the TI cue library and could be easily replicated with the other systems.

Finally, the data were reduced latency as the time from calling the functions to create and play a cue on \( n = [1, 8] \) tactors until the acceleration caused by the last actuated tactor exceeded 0.015 g. Accelerometer data were reduced to find the mean and standard deviation of the latency for each system and number of channels played (Fig. 4.3).
4.2 Latency Benchmarking Results

![Latency vs Channels Rendered](image)

Figure 4.3: Latency as a function of channels rendered, measured as the time from software triggering to the detection of tactor acceleration. Only four channels are shown for the EAI control unit since this is its max.

The Texas Instruments system has the highest latency for a single tactor, but does not increase latency through four tactors. After the fourth tactor, the average latency and standard deviation increase, possibly due to I²C multiplexer components, but again stays constant after five tactors. The Arduino likely contributes most to this latency, but since it represents a very plausible implementation, we consider it a fair comparison. The EAI system has lower latency than the TI system for one and two tactors, but the latency linearly increases with number of channels played to greater than the TI system, and as noted cannot play more than four channels. The Syntacts system has significantly lower latency than either of the commercially available systems tested and does not seem to be a function of channels played, so the system could expand to larger tactor arrays without delays. Though not shown, we measured similar latency values for the MOTU 24Ao with 24 channels played simultaneously.
Table 4.1: Comparison of Tactor Control Methods. Amplifiers, such as the Syntacts amplifier, are also necessary for audio methods and usually cost an additional $50 to $100 USD. Latency not measured because Macaron is meant to design and test cues, not play in real time. Only 4 channels can be played simultaneously.

4.3 Overall Comparison

Whole-system comparisons of the vibrotactile control methods tested are summarized in Table 4.3. The different programming APIs show the extent to which hardware can be integrated within software. The GUI column lists the different functionality of the included graphical user interfaces. Synthesizers are able to create cues, sequencers have the ability to organize cues in time on one or more channels, and spatializers allow users to specify the center of vibration for an array of tactors. The right side of this table describes example hardware available for use in each category. The minimum average latency column reports the data from the tests described above as the average value for the 100 trials completed for each hardware combination using the minimum latency approach with a single channel.

The integrated-circuit (IC) approach is effective as, in the current configuration, the latency was consistently low and could handle eight tactor channels. It also has
a relatively low financial cost to users. A limitation that exists with this setup is the library of cues. The chip contains a library of 123 cues that can be sequenced with up to 7 in a row at any point. However, there is little control over these cues if users want to make specific cues or tune cues to be “realistic” in a VR application scenario. In addition, retrieving cues from memory does have a non-negligible time-cost that users would need to work around to maintain the low latency performance. Another option is using the audio-to-vibe functionality included on the chip that can render audio signals on the vibration motor, but this would require more infrastructure than the setup tested here.

The closed-box controller solution yielded low latency for a single channel, but linearly increased with more channels. In addition, though with tactor channels can be controlled, only four can be played at the same time. Depending on the implementation, this may be problematic if users need to use more than four channels simultaneously.

Finally, Syntacts showed overall comparable latency values to the commercially available solutions. The hardware listed only represents a small subset of the possible options, but as can been seen Syntacts allows users to select audio devices based on output needs and cost. For just $50 USD and the cost of an amplifier, researchers can interface a 7.1 surround sound card with Syntacts to achieve a complete 8 channel setup comparable in performance to the $2,250 USD EAI Universal Controller. Though rendering more than 8 channels with audio comes at a cost, it can still be done for much less that the cost of multiple EAI controllers and is considerably more manageable than implementing an integrated circuit based design. In addition, the choice of hardware revealed little change in latency since each used the ASIO API discussed in Section 3.2.2.
Chapter 5

Syntacts Demonstrations

There are two demonstrations that have been developed to showcase the abilities native to Syntacts. Both of these employ the GUI developed and deployed with Syntacts. The following sections describe the demonstrations designed for Syntacts.

5.1 One Framework to Rule Them All

Commercial companies have released a wide variety of tactors, and many companies have their own control systems to use. Syntacts is flexible to run any tactor, LRA or voice-coil, that uses an oscillating signal to drive the vibration. In this demonstration, users interact with the GUI above on a laptop computer. The GUI allows users to design single cues, load and save cues, and organize a series of cues in time. The user is encouraged to design and sequence their own cues through the GUI and render them on the various vibration motors. This computer will be connected to different types of tactors, some examples of which are:

- Engineering Acoustics C2 Tactor (Voice Coil Actuator)
- MPlus 1040 (LRA)
- TacHammer Carlton (Linear Magnetic Ram - Similar to Voice Coil)

Users design and play cues on each tactor so that they see the flexibility of using this software with different hardware, meaning they can integrate this software into
current projects without having to change to new tactor hardware. In addition, users see the ability of Syntacts to control each of these types of motors and how easy it can be to edit the cues in real-time to optimize them on different tactor types.

5.2 Design Your Cues, Feel Them Move

![Syntacts spatialization demonstration](image)

Figure 5.1: Syntacts spatialization demonstration - Right side shows user dragging target on computer, left side shows vibrations rendered on tactor array. Opacity of green tactors is directly tied to amplitude of vibration

This demonstration allows users to design and feel their cues in real time on a custom-designed vibrotactile array. They use the GUI shown on the computer in Figure 5.1 to create their own complex cues and play them in real time on the hand-sized array of 24 linear resonant vibration actuators (LRAs) on the table in Figure 5.1. Users can also use the spatializer in the GUI to feel their cues move around in space by simply filling the grid with the available tactors in their real-world locations. When the cue is rendered through the spatializer, Syntacts will calculate the appropriate amplitudes for each tactor based on the location. The figure shows a cue dragged
from one area of the array to another while the output of the vibrotactors respond in real time. This shows users how easy it is for a much large audience of users to flexibly design and implement complex haptic effects using Syntacts.
Chapter 6

Conclusion

The field of haptics is rapidly evolving and becoming increasingly complex. One of the most common forms of cutaneous haptic feedback is vibration. Vibration devices take advantage of rapidly adapting mechanoreceptors in the skin to convey information to the user. In the past, this information was relatively simple alert-type cues, such as a cellular phone buzz for a received message. Since then, research has generally looked at expanding the cue set from each vibration motor or designing more complex devices to take advantage of one or more of these vibration motors. The research on expanding the cue set has evaluated what waveforms humans can differentiate or using patterns of vibrations to communicate information. Using arrays of vibration motors, or more advanced motors has typically been the focus of researchers creating more complex devices.

The contributions of this thesis are the creation and evaluation of a software and hardware framework that built to lower the technical and financial barrier to entry into cutaneous vibration haptics. This framework, called Syntacts, is completely open-source and available to users on GitHub. The software portion is a C++ API built that takes advantage of audio engineering techniques to render ultra-low latency vibration cues. The API natively includes functionality that has been commonly used in research, as well as new design, time sequencing, and spatialization for vibration arrays. In order to make the library more accessible to a wider range of users, Python and C# functionality has been added. Finally, a graphical user interface (GUI) is
included for cue design and editing, as well as general demonstration. The hardware is a custom designed Syntacts Amplifier, which is an 8-channel linear amplifier designed to interface with digital audio devices and vibration motors. All of this information, as well as an implementation guide, can be found on the GitHub repository.

This system was benchmarked against commonly used vibration control systems. We chose to compare against a commercially available closed-box system, the Engineering Acoustics Tactor Control Unit, as well as a I²C solution, the Texas Instruments DRV2605L. Syntacts is capable of lower latency than either commercial solution at the same or lower cost. Syntacts is a flexible implementation option for users to accomplish complex vibrotactile research as they can select the hardware setup necessary for their research and spend only what is necessary.
Bibliography


July 2019.


[21] O. S. Schneider and K. E. MacLean, “Studying design process and example use


