Towards a catalog and software library of mapping methods

Hans-Christoph Steiner
at.or.at
New York, NY, USA
hans@at.or.at

ABSTRACT

Mapping has been discussed for decades, yet there is not standard catalog of mapping methods. The Mapping Library for Pd is a fledgling library of mapping primitives with the aim of cataloging existing mapping methods. Also included are techniques for conditioning sensor data to make it usable in the context of instrument design.

1. INTRODUCTION

Everything should be made as simple as possible
- but no simpler. - Albert Einstein

Mapping for instruments has been discussed for decades. There have been a huge range of ideas touted, and many instruments built and tried. Shared elements and ideas are repeated, and re-implemented again and again. The foundations of audio have been long since codified with the standard unit generators, with few audio software packages disregarding them. They have become a basic language to express ideas in that realm, whether in software or hardware. There is no catalog of fundamental mapping algorithms, and little work has been done to build the foundations into software.

With no standard framework, instrument designers are constantly reinventing the wheel, re-implementing mapping algorithms whenever creating a new instrument. Many mapping operations are very common, so it makes sense to have a software library that includes these operations. Even for uncommon and more complex operations, having a library of mapping methods allows the instrument designer to rapidly test a wide variety of mapping ideas without having to implement them, and even derive some inspiration from scanning the contents of a mapping library and rapidly interchanging elements.

In a similar vein, there are a myriad of methods of conditioning sensor data. When using sensors, there is often noise and errant results. Many useful sensors do not produce linear data, or even easily modeled curves. There are many techniques for conditioning this data to make it straightforward to use. These techniques are well established in the realm of electronics [8] [12]. Many of these ideas are very useful in the context of instrument design and can be applied in the software realm. Yet there seems to be no catalog of software algorithms for conditioning, especially when talking about applying them to instrument design.

This paper aims to start the discussion of what these mapping primitives are and describes work towards building a library of these ideas: the Mapping Library for Pd. Algorithms for processing sensor data are included because many of them are also used in mapping data, such as applying transfer functions to shape the data. This library is also intended to become a catalog of mapping techniques which can freely be implemented in other ways, with the hope of developing some standardized terminology.

2. PREVIOUS WORK

I started my work on the topic of mapping with the [hid] toolkit [9], which mainly focused on streamlining the process of getting data from game controllers for creating instruments. The [hid][1] object already provides access to a wide range of devices. In addition, there is work underway on supporting sensorboxes in a way that follows standards laid out by the [hid] toolkit objects. The other key part of that work is the objects for mapping data. Cyrille Henry, who is a major contributor to the software library, had developed a collection of Pd objects for sensor processing techniques. It is these sets of objects that the Mapping Library for Pd is based on.

There have been a couple of attempts at building frameworks for creating musical instrument mappings. Two notable packages come from IRCAM. "MnM is a set of Max/MSP externals... providing a unified framework for various techniques of classification, recognition and mapping for motion capture data, sound and music." [2] An earlier attempt from IRCAM is the ESCHER toolkit for jMax [11] which is a set of objects to address various problems of mapping. Goudeseune presented his mapping technique based on high dimensional interpolation he calls "simplicial interpolation" [4] . While this is an interesting technique that shows promise, it only addresses a specific approach to mapping and is not broken up into more generally useful modules. Both of these are complex systems which show promising results, but they seem to address the opposite end of the spectrum that this paper is addressing. They are built from complicated objects and take a mathematical approach to mapping. While

[1]a word in square brackets denotes a Pd object

Permission to make copies of all or part of this work for any purpose are granted under a Creative Commons Attribution-ShareAlike license: `http://creativecommons.org/licenses/by-sa/2.0/`
NIME 2006, June, 2006 Paris, France
Copyright 2006 Copyright remains with the author(s).

mathematics are an integral part of mapping, the user need not experience it in that way.

3. DESIGN IDEAS

When talking about mapping, we are almost always talking about computer software, anything from custom C code to Pd patches to application preferences. Therefore it makes sense to implement a catalog of mapping methods as a software library. Mapping is generally represented firmly within the realm of mathematics. By abstracting the math into software objects, mapping can be approached as a system of logic. Software derives its vast power from the encapsulation of ideas and the reuse of code. Many complex mapping algorithms can be encapsulated into software objects, opening up new opportunities for exploration. One need not understand much about an algorithm within an object in order to insert it into a program and play with it. This way of interacting is much more like how many musicians learn to play an instrument: they play with it and see what they can figure out. Having encapsulated software objects, mapping can be more in this spirit of play, rather than purely a separate, studied effort. Also, catalog of mapping primitives aids in teaching and standardized terminology allows more fluid discussion and exchange of mapping ideas.

This paper covers explicitly defined mapping methods, where the instrument designer directly controls each aspect of the mapping. Other papers cover methods involving generative techniques or neural networks[3]. While such systems might be based on the same mapping primitives as other methods of mapping, it is difficult to derive the mapping since it is only represented internally to the map-generating process.

There is still a lack of a set of commonly agreed upon primitives for the building of mappings. There are many great ideas about mapping, but lack reusable implementations. The goal of the Mapping Library for Pd is to first provide a set of mapping primitives, then to build more complex objects using the primitives. This modular approach has a number of advantages: the code is easy to read since its based on encapsulated ideas, code can be easily tweaked or repurposed since its all written within Pd, and as more objects get written, it becomes easier to write higher level objects. These objects be general, making as few assumptions as possible for how they should be used. This idea is key to the design of Pd itself. TCP/IP was famously designed this way, and it has proven itself to be useful in ways far beyond the original creators' intentions.

In some ways, this library is a return to basics. Many interesting yet complex methods of mapping have been proposed and discussed. It seems that its too soon to be moving onto such complex methods when the basics are not clearly established. Software has become an integral part of designing new instruments, and it is rare for a new instrument these days to have absolutely no software component. Mapping should then be codified in software beyond being written about. Not only is software functional, but it is also a highly effective method of communicating the ideas related to mapping, arguably more effective that written language. There are numerous clearly defined ideas in audio synthesis which are implemented in most computer music software these days. Much how the standard audio unit generators encapsulate the mathematics used in synthesizing audio, a mapping ideas should also be laid out and encapsulated.

As with the [hid] toolkit, the Mapping Library uses the data range of 0 to 1 wherever possible for the reasons outlined in the paper on that software. Almost all of the mapping objects expect input and output data in the same range. This range is applied everywhere wherever possible, including to somethings that might disturb mathematicians, like [polar], which converts cartesian coordinates to polar coordinates. Even the angle is scaled to the range of 0 to 1. This allows other mapping objects to be used after the conversion without having to rescale the data. Though the objects are designed to work within this range, many of them also work beyond that range. For example, [spiral] has an infinite range, with 1 representing one full rotation.

Another realm of mapping which has not yet mentioned are the issues of processing sensor data to make it usable. Sensors are subject to noise, errant glitches, and unique properties which make them difficult to use. Like the world of instrument mapping, there is not a standard catalog of sensor primitives in the realm of software. Another promising area of research is data stream processing. The Stanford STREAM group has created a standard query language for data streams[10]. This language is oriented towards typical database applications, but many of their techniques could be useful in processing sensor data stream. A key part of the Mapping Library effort is to create software that encapsulates these techniques and make them usable to instrument designers.

4. EXAMPLES OF MAPPING OBJECTS

There are already quite a few objects implemented, here are some selected examples: control rate filters ([iir], [fir], [mpfilter], [lop]); basic transfer functions ([curve], [curve_power], [gaussian]); interpolation ([wave], [interpolate]); sensor data conditioning ([hysteresis], [local_min], [local_max]); testing ([test_n], [box], [stream_presense]); and, ranging and sizing ([resample], [upsample], [downsample], [clip], [distance]). At the most basic level, the Mapping Library includes objects for generating various curves over a range. [curve] is an object that provides a continuum of curves starting from 0 for linear. For positive numbers, the curve is a power curve, for negative numbers, its a root curve. For people who want to use standard functions, [gaussian], [sinusoid], [curve_power], [curve_root], [curve_exp], and [curve_log] are provided. Scaling is another common operation, so there is [autoscale] which dynamically scales input data to an output range, [notescale] which scales 0 to 1 to the specified range of integer note values. Other ranging objects include [local_min], [local_max], [min_n], and [max_n]. Objects with the "_n" suffix mean that they take an numeric argument which control how many previous values that object should consider (i.e. apply the function to n elements). Averaging

The mapping objects are built from the ground up from quite primitive operations into higher level objects. For example, the [spiral] object is built using the [polar] object, which is in turn built using the [vector] object, which uses [radians-¿mapping]. The objects names have been carefully chosen to accurately represent the idea with a minimum of confusion. Commonly used words for certain methods were adopted wherever possible, for example with [diverge] for one-to-many and [converge] for many-to-one since these are words commonly used to describe such mappings [5]. Some unusual words are used, like [disjoin], because it makes sense with its opposite operation: [join]. This is just a fledgling

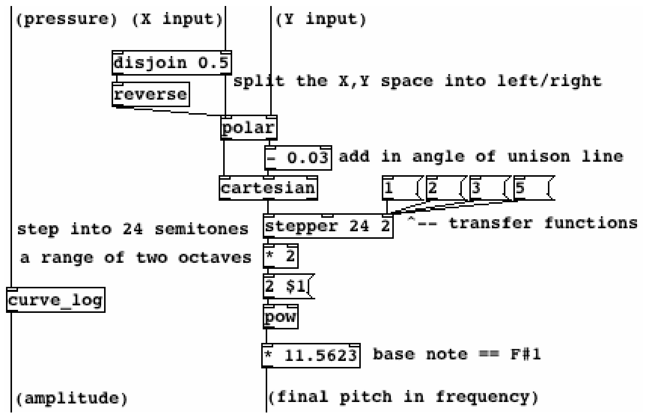

Figure 1: A Pd patch showing the mapping of The Ski's pitched mode.

effort, so the names are bound to change as things develop.

5. BREAKDOWN OF EXISTING MAPPINGS

To explore these ideas, the mappings of two specific instruments are analyzed and reproduced using the Mapping Library for Pd. There is a huge variety of new instruments and wide range of ideas for mapping. These instruments where chosen because they have been played extensively, performed in concert, and each instrumentalist has achieved a high level of fluency with his instrument. Instruments that are regularly played will have a more honed mapping, and the spotlight of performance is excellent at drawing exposing problems.

5.1 The Ski Angular Mode

With Huott's Ski [6], he outlines four different modes for mapping the tactex pads to controlling samples: linear, polar, angular, and linear velocity. All of these can be easily implemented using exisiting mapping objects. For example, here is how to implement angular mode. First, [autoscale] automatically scales in the incoming data to a floating point range of 0 to 1. The scaled cartesian coordinates from the tactex pad are then fed to the [polar] object, which converts the data to a magnitude and angle. The angle is output in the range of 0 to 1 instead of the more usual -π to π. The allows other mapping objects to be easily chained after the [polar] object.

5.2 The Ski Pitched Mode

The front pads are used in a pitched mode, this layout is diagramed in Figure 4 of Huott 2002. First, the pads are split into left and right sides using [disjoin]. The left side is reversed using [reverse]. The cartesian coordinates are converted to polar, then the angle of the unison line is created by subtracting 0.03 from the angle (represented from 0 to 1 not -π to π). The data is then converted back to a rotated cartesian plane, and the Y value is taken to control frequency. The range is split into 24 semitones and a x^2 transfer function is applied to allow glissando between notes. The range is multiplied by 2 to span 2 octaves, then converted to frequency with a base note of $F^{\#}1$. The pressure data is curved logarithmically to control amplitude.

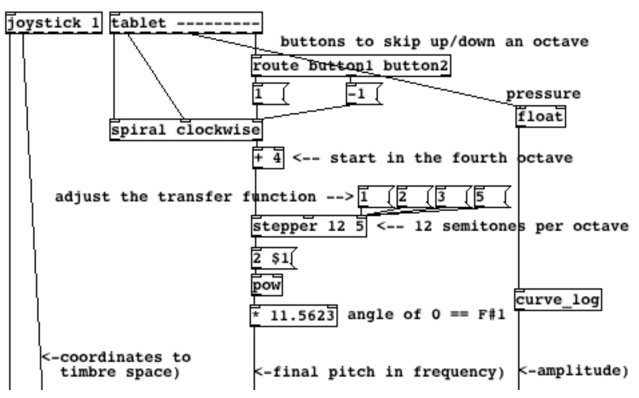
joystick 1
tablet ---------
buttons to skip up/down an octave
route button1 button2
1
-1
pressure
spiral clockwise
float
+ 4
<-- start in the fourth octave
adjust the transfer function -->
1
2
3
5
stepper 12 5
<-- 12 semitones per octave
2 $1
pow
curve_log
* 11.5623
angle of 0 == F#1
<-coordinates to timbre space)
<-final pitch in frequency)
<-amplitude)

Figure 2: A Pd patch showing the mapping of the Voicer.

5.3 The Voicer

Kessous developed his Voicer [7] using a tablet and a joystick as controllers. The tablet controls the pitch and amplitude while the joystick navigates a timbre space of vowel sounds. Pitch is derived from the polar angle, in a spiral. The [spiral] object does this using [polar] to convert to polar coordinates, then it keeps track of rotations. Since the [spiral] object has a "clockwise" argument in the patch, the data increases in the clockwise direction, rather than counter-clockwise as is usual with polar coordinates. [stepper] converts the linear angle data from [spiral clockwise] into a stepped line. The first argument of 12 creates 12 steps within the range of 0 to 1. Each step locally curved according to the transfer function specified by the last argument/inlet. This curved by taking the input and raising it to the power specified by the transfer function argument/inlet. The output of [stepper] is then converted into frequency values, with an angle of 0 equal to the beginning of the $F^{\#}4$ segment. In Arfib, et at, 2002, Figure 5 displays a graph of the output of the [stepper] object. It is built using other Mapping Library objects: [segment], [curve_power], and [desegment]. [segment] is in turn built using [disjoin], and [desegment] is built using [join]. For the amplitude control, the pressure in taken from the pen using the [tablet] object, which outputs all axes in the range of 0 to 1. The pressure data is then curved logarithmically using [curve_log], to match the human perception of amplitude.

6. CONCLUSION

The process of analyzing these two instruments has affirmed many of the existing ideas in the Mapping Library for Pd, and has exposed a number of weaknesses and omissions. Many interesting approaches to mapping, such as a multi-layered approach discussed by Arfib[1], Kessous, Wanderley, do not seem inherently compatible with these current objects. This is just the beginning, but these two examples clearly demonstrate that there is promise to this approach to mapping. With a flexible toolbox of mapping methods, instrument designers can experiment more fluidly with mapping ideas. With a catalog of mapping methods, we can more easily discuss new mapping ideas and demonstrate them using code.

7. FUTURE WORK

Now that the basic foundation has been laid, a more objects will be created to work towards completing a catalog of sensor processing and mapping methods. Also, following up on the [hid] toolkit, we aim to create a framework for working with raw sensors and sensor boxes, and making them interoperate with game controllers and the Mapping Library for Pd. This will lead away from a focus on music and hopefully towards a more generally useful library, contributing towards visual instruments, mapping for interactive installations, robotics, or whatever needs data mapped to controls.

8. ACKNOWLEDGMENTS

Cyrille Henry contributed in a number of key ways. He was involved in the formations of the original ideas, and helped me focus my ideas through discussions about mapping, and wrote a wealth of objects for the Mapping Library.

9. REFERENCES

[1] D. Arfib, M. Couturier, L. Kessous, and V. Verfaille. Strategies of mapping between gesture data and synthesis model parameters using perceptual spaces. *Organised Sound*, 7(2):127–144, 2002.

[2] F. Bevilacqua, R. Muller, and N. Schnell. MnM: a Max/MSP mapping toolbox. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME05)*, Vancouver, Canada, 2005.

[3] A. Cont, T. Coduys, and C. Henry. Real-time gesture mapping in pd environment using neural networks. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME04)*, Hamamatsu, Japan, 2004.

[4] C. Goudeseune. Interpolated mappings for musical instruments. *Organised Sound*, 7(2):85–96, 2002.

[5] A. Hunt and M. Wanderley. Mapping performer parameters to synthesis engines. *Organised Sound*, 7(2):97–108, 2002.

[6] R. Huott. An interface for precise musical control. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*, Dublin, Ireland, 2002.

[7] L. Kessous. Bi-manual mapping experimentation, with angular fundamental frequency control and sound color navigation. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME02)*, Dublin, Ireland, 2002.

[8] R. Pallàs-Areny and J. G. Webster. *Sensors and Signal Conditioning*. John Wiley and Sons, 2nd edition, 2001.

[9] H.-C. Steiner. [hid] toolkit: a unified framework for instrument design. In *Proc. of the Conference on New Interfaces for Musical Expression (NIME05)*, 2005.

[10] The STREAM Group. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin*, 26(1), March 2003.

[11] M. Wanderley, N. Schnell, and J. B. Rovan. Escher-modeling and performing composed instruments in real-time. *IEEE Systems, Man, and Cybernetics*, 1998. `http://intl.ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=727836`.

[12] J. S. Wilson, editor. *Sensor Technology Handbook*. Elsevier, 2005.