Improving Data-Driven Design and Exploration of Digital Musical Instruments

Christopher Laguna
University of California, San Diego
La Jolla, California
United States of America
cplaguna@ucsd.edu

Rebecca Fiebrink
Goldsmiths University of London
New Cross, London
SE14 6NW
United Kingdom
r.fiebrink@gold.ac.uk

CHI 2014, April 26–May 1, 2014, Toronto, Ontario, Canada.
ACM 978-1-4503-2474-8/14/04.
http://dx.doi.org/10.1145/2559206.2581327

Abstract
We present Gesture Mapper, an application for digital musical instrument designers to rapidly prototype mappings from performer gestures to sound synthesis parameters. Prior work [2] has shown that using interactive supervised learning to generate mappings from user-generated examples can be more efficient and effective than users writing mapping functions in code. In this work, we explore new ways to improve on data-driven design of interactive systems, specifically by proposing new mechanisms for rapid exploration and comparison of multiple alternative mappings. We present a conceptual structure for interactive mappings, a basic framework for generating mappings from more diverse types of user-specified constraints than are supported by supervised learning, and the new Gesture Mapper user interface for mapping exploration and comparison.

Author Keywords
User interfaces; Digital musical instruments; Design tools.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—Interaction Styles

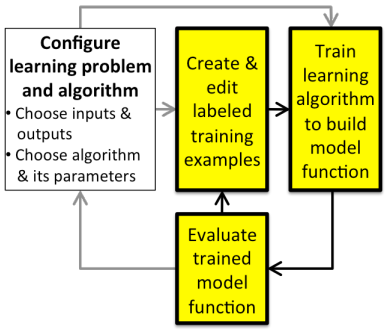

Figure 1: User workflow for applying interactive machine learning to building mapping functions in the Wekinator [1]. After an initial configuration step, users typically focus on iterating through the steps highlighted in yellow: adding or removing labeled training examples, re-training to build a revised mapping function, and evaluating the revised mapping by running it in real-time while performing gestures and listening to resulting instrument sound.

Introduction

Digital musical instrument (DMI) designers must decide what it means to play an instrument: how will a performer move, and what sounds will be produced in response? Inherent in the design of a new instrument is the challenge of defining an appropriate *mapping* from a musician's actions (often represented as features extracted from sensors tracking physical gestures, e.g., USB game controllers or on-body sensors) to parameters driving real-time sound synthesis (e.g., pitch, volume, timbre) [4]. Sometimes an instrument builder has strong preconceived notions about how the input gestures should be mapped to output parameters, for example when the digital instrument will mimic the behavior of an existing acoustic instrument. Often, however, digital instrument builders strive to create completely unique instruments, and they may not initially have clear ideas about what input gestures to use or how they should affect the instrument's sound. These creators need to not just *implement* a mapping, but to also *design* the instrument, determining what subset of the infinite number of possible interactions it will support. Our goal in Gesture Mapper is to improve the design process for this latter group. A longer-term goal of this research is to develop data-driven design tools that can be applied to designing other real-time systems, such as gaming and accessible interfaces.

We begin this paper with an overview of the Wekinator, a tool for creating DMI mappings using interactive supervised learning. We motivate our current work by identifying important design subtasks—deciding *what* to build, and comparing alternative designs—that are not well-supported by Wekinator. Then, we describe new approaches to supporting these tasks while maintaining the user's ability to drive the design by providing example data and high-level constraints, rather than by programming.

Background and Motivation

The Wekinator [2] was designed to facilitate the creation of DMIs and other real-time interactive systems. It allows users to define mappings from an arbitrary set of real-time input features (e.g., describing a musician's gestures with a sensor interface) to an arbitrary set of real-time output parameters driving sound synthesis, animation, or similar processes. (Inputs are sent to Wekinator from an external feature extractor, and outputs are sent to the sound synthesis engine or similar environment, using the OpenSoundControl protocol.) Wekinator enables users to create mappings using interactive supervised learning: users create a labeled training set by demonstrating example input gestures and labeling them with the appropriate sound synthesis parameter values. A supervised learning algorithm infers the mapping from input features to output parameters using the training data. The user can test the mapping by moving and listening to the instrument's sound in real-time. The typical Wekinator workflow (Fig. 1) involves an iterative process of creating training data, evaluating the mapping produced by that data, and refining the mapping by modifying the data.

Resnick et al. [5] have written about how technology can support creative work in diverse domains, including design and the arts. One of the guidelines for creativity support tools emphasized by [5] is that tools should support exploration of many alternative designs, backtracking when design changes are unsuccessful, and "sketching" new designs without specifying unknown or irrelevant low-level details. Fiebrink et al. [3] observed composers using Wekinator to build new

musical instruments and found that, in line with the proposals of [5], instrument designers value exploration, rapid prototyping, and "sketching" from incomplete specifications. Fiebrink et al. also showed that interactive supervised learning allowed composers to explore, prototype, and sketch more quickly and effectively compared to building mappings by writing code. Because changing the training data was all that was needed to change a mapping, and changing the data was relatively fast and easy, users could explore many designs and experiment with diverse ideas.

Generating mappings from user-supplied examples rather than from code also allows more effective communication of embodied knowledge [2] and better accessibility to non-programmers. However, we believe there is room to improve on these methods as implemented in prior work, based on the following two observations:

(1) Instrument building has properties of a "wicked" design problem [6], an ill-defined problem wherein a problem "definition'' is found only by arriving at a solution. An instrument designer may not have a clear plan for how the instrument should behave (or what the mapping should look like), especially at early stages of the design process. Wekinator-style supervised learning imposes unnecessary constraints on designers by forcing them to hypothesize appropriate output labels for each example input in the training set.

(2) A designer may have many ideas about how input *might* map to output, and about what input devices or features to use in the first place. He or she may need to develop several ideas in parallel and then explore them side-by-side in order to decide how to proceed. In fact, recent research has pointed to the positive effects of parallel exploration on design quality [1]. In contrast, interactive supervised learning as supported in Wekinator encourages a linear design process, in which one single idea is iteratively refined through successive modifications to a dataset. New mechanisms should encourage experimentation with radically different mappings and feature sets, and support comparing alternative mappings in quick succession.

System Overview

Our system implements new approaches for generating mappings without requiring users to fully specify instrument structure or to generate labeled data, and new user interfaces to encourage side-by-side creation, modification, and evaluation of alternative mappings.

Operational Definition of a Mapping

Our creation of new methods and interfaces has required a more formalized definition of what a *mapping* entails. We define a mapping as a set of functions, one per output parameter. Each function computes its output using some subset of available inputs. Each function has one of several possible *types* (e.g., a regression model, classifier, or threshold detector), and the set of feasible types is determined by properties of the input and output (e.g., whether the output is continuous or discrete). A specific function of a given type can be instantiated by choosing a precise parameterization (e.g., specifying the regression coefficients or threshold value). The instantiating parameters can be set manually, inferred from an example dataset, or randomly generated.

The design of a mapping function for each output synthesis parameter can therefore be viewed as a sequence of decisions: the choice of which inputs are used in computing the output; the choice of function

type; and the choice of parameters to instantiate a specific function of that type. Our basic approach is to allow users to explicitly specify any of these choices, but to also provide automated procedures for making these choices in a reasonable manner.

Generating Random Mappings from Incomplete Specifications

Our system can instantiate a fully-functional mapping given only the number, types (continuous, discrete, or binary), and value ranges of the input features and output parameters. The user may optionally specify which of the input features will be used to compute each output, or he may leave this to be randomly chosen by the system. The user can also choose whether to provide labeled, unlabeled, or no training examples to constrain or guide the instantiation of each function comprising the mapping. If labeled data is supplied, the function is learned from the data just as in Wekinator. For unlabeled data, the instantiation of the function will be guided by the data (e.g. a *k*-class classifier can be created by randomly assigning class labels to clusters learned using k-means). If no data is supplied, a function is instantiated based on the input and output types and ranges.

If the user has not chosen a function type for a parameter, the system currently chooses one at random from the set of all feasible types (e.g., a classification function may be chosen for binary or discrete—but not continuous—outputs). Future work will explore smarter methods for choosing functions.

By giving users the option to automate the choice of functions and the selection of inputs to use in computing each function, we allow users to create a new mapping without specifying any information about how the inputs and outputs should be related. Fiebrink et al. [3] found that composers often enjoyed being surprised by the mapping functions produced by Wekinator and "discovering" new instruments through their use of the system. Our new approach makes it even easier to discover new mapping ideas: if a user chooses to generate mappings using only functions without training examples, he or she can create a working instrument with the click of a button. Since the automated configurations are pseudo-random, the user can generate different mappings simply with successive clicks of the same button. Thus, users can create and play with many mappings quickly when they do not have a good idea of how instruments should behave.

GUI Support for Mapping Exploration and Comparison

We now describe each section of our new GUI to support instrument creation and comparison (Figure 2):

1. The Workspace: The interface is a workspace that contains multiple instruments. At any time, users can switch between instruments via a combo box (top left) encouraging them to explore side-by-side creations of instruments, described later.

2. Stash: The "stash" is a global space where functions and datasets can be saved and copied into other instruments. Knowing that they can save valuable parts of an instrument, users will be able to go on to (1) modify that instrument without worrying how the valuable parts were affected, and (2) create "composite" instruments, where some parts of the instrument came from other instruments.

3. History Tree: The history tree will save all previous revisions of an instrument, each at a different node in the tree. When the user makes any changes to an instrument, a new node in the tree appears. The user

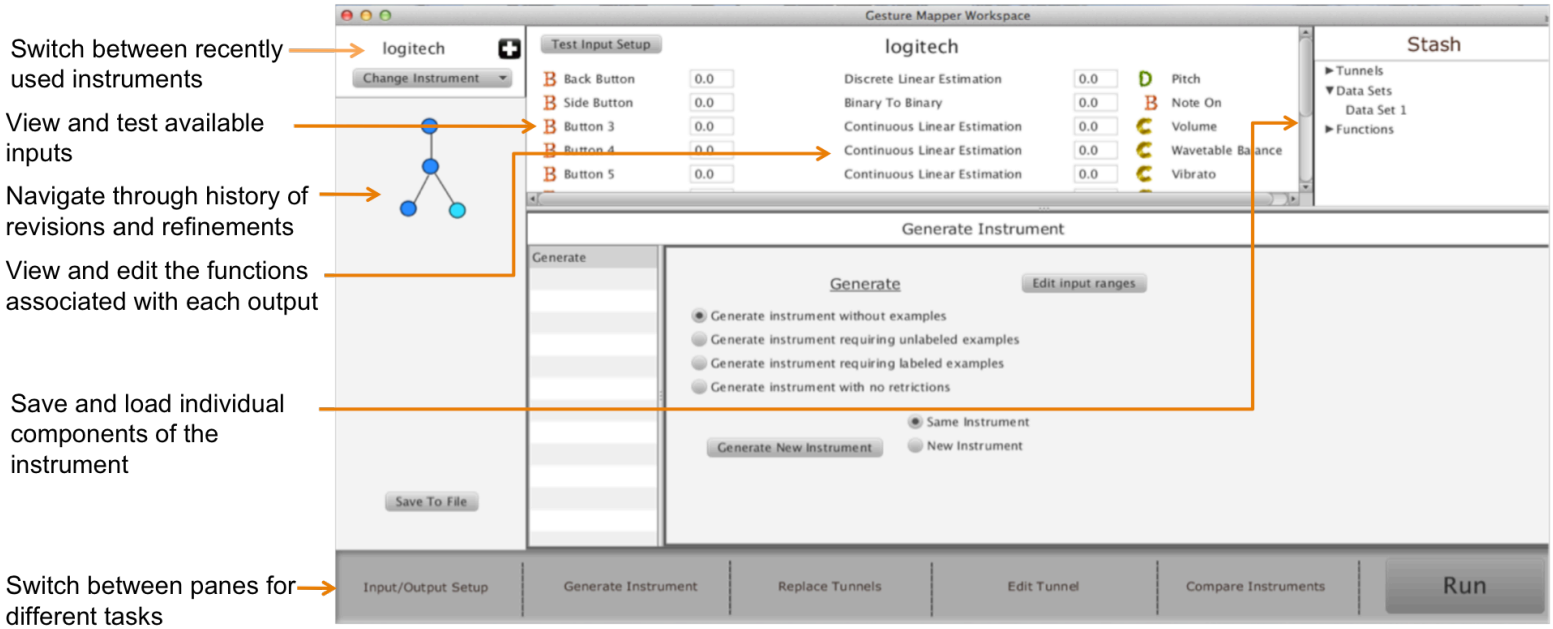

Figure 2: Our GUI workspace, as seen when generating a new instrument.

can click on the tree to revert to any node. This allows users to explore new ideas without worrying about losing their current progress. Also, if users have many paths they want to explore from the same node, they can simply explore one path, travel back to that node, and then explore the new path. The result will be two leaf nodes that the user will be able to compare side by side in the Compare Instruments interface.

4. Compare Instruments: This interface (implemented but not shown) allows users to quickly compare multiple instruments by playing them in quick succession. The interface also allows users to play instruments simultaneously. This enables users to design for ensembles and encourages "composite" instruments, where the true instrument is made up of many "instruments" within the system.

Training Data Manipulation

Each function is assigned its own training data set, so users can remove an output or edit a function's dataset without affecting the functions that drive other outputs. Users can also add inputs and outputs without affecting existing functions. Future work will develop pre- and post-processing mechanisms to maintain data sets past changes to input and output data types, enabling users to change data types without the overhead of creating a new training data set.

Current Progress

Our system can currently generate mappings and automatically choose functions and their inputs. Functions requiring no examples and labeled examples exist in the system, but functions requiring unlabeled examples do not yet exist in the system. Users can create labeled training sets, but cannot yet create unlabeled training sets. Mappings can be run in real-time, and multiple mappings can be run simultaneously. Functions can be manually edited. Mappings can be saved beyond the current session. The stash and history tree are not yet implemented.

Conclusion

We have presented a new set of methods to support digital musical instrument builders creating mappings from musicians' gestures to sound. Our methods explicitly accommodate designers' needs to quickly discover and evaluate alternative designs, by supporting creation of fully-functional mappings from incomplete user specifications, and by using stochastic algorithms to instantaneously create multiple candidate mappings to satisfy any user-specified constraints. Compared to past work in [2], we give users more options for communicating constraints and goals to the mapping system, including the creation of mappings from labeled or unlabeled gesture examples, as well as randomly-generated mappings from no examples at all. We thus aim to support rapid exploration even at early stages of the design process when the design goals have not yet solidified.

Our next steps include the implementation of the history tree and stash elements of the interface, and exploration of the utility of the system in studies with instrument builders. The most significant intended contributions of our ongoing work include a usable tool for instrument designers, as well as data-driven algorithms and interfaces that can be applied or adapted to improve the design process for other real-time interactive systems in music, dance, gaming, physiotherapy, and other domains.

Acknowledgements

The work of Chris Laguna was supported in part by the Distributed Research Experiences for Undergraduates (DREU) program, a joint project of the CRA Committee on the Status of Women in Computing Research and the Coalition to Diversify Computing, which is funded in part by the NSF Broadening Participation in Computing program (NSF CNS-0540631).

References

[1] Dow, S. P. et al. "Parallel prototyping leads to better design results, more divergence, and increased self-efficacy." ACM TOCHI 17, 4, Article 18, 2010.

[2] Fiebrink, R., D. Trueman, and P.R. Cook. "A meta-instrument for interactive, on-the-fly machine learning." *Proc. New Interfaces for Musical Expression*, 2009.

[3] Fiebrink, R., et al. "Toward understanding human-computer interactions in composing the instrument." *Proc. International Computer Music Conference*, 2010.

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

[5] Resnick, M., et al. "Design principles for tools to support creative thinking." *Report of Workshop on Creativity Support Tools*, 2005.

[6] Rittel, H. "On the planning crisis: Systems analysis of the 'first and second generations.'" Bedriftsøkonomen, no. 8: 390–96.