The Digital Orchestra Toolbox for Max

Joseph Malloch
GEM Lab
Faculty of Computer Science
Dalhousie University
Halifax, Canada
joseph.malloch@dal.ca

Marlon Schumacher
IMWI
Hochschule für Musik
Karlsruhe
Karlsruhe, Germany
marlon.schumacher@hfm-karlsruhe.de

Stephen Sinclair
Inria Chile
Santiago, Chile
stephen.sinclair@inria.cl

Marcelo M. Wanderley
IDMIL, CIRMMT
McGill University
Montreal, Canada
marcelo.wanderley@mcgill.ca

ABSTRACT

The Digital Orchestra Toolbox for Max is an open-source collection of small modular software tools for aiding the development of Digital Musical Instruments. Each tool takes the form of an "abstraction" for the visual programming environment Max, meaning it can be opened and understood by users within the Max environment, as well as copied, modified, and appropriated as desired. This paper describes the origins and motivations for creating the Toolbox, broadly outlines the types of tools included, and follows the development of the project over the last twelve years. We also present examples of several digital musical instruments built using the Toolbox.

Author Keywords

digital musical instruments, mapping, Max, toolbox, creativity support

CCS Concepts

•**Applied computing → Sound and music computing;** Performing arts;

1. INTRODUCTION

The McGill Digital Orchestra was a large research-creation project supported by the Appui à la recherche-création program of the Fonds de recherche sur la société et la culture (FQRSC) of the Quebec government [7, 1]. The project brought together research-creators and researchers in performance, composition and music technology to work collaboratively in creating tools for live performance with digital technology. A large part of this research focused on developing new musical interfaces. The grant had a duration of three years and culminated with a performance of new works during the 2008 MusiMars/MusiMarch Festival in Montreal. In the context of the project, student research assistants were hired from the three contributing academic areas: four doctoral students in performance, two from composition, and four graduate students from music technology.

A large part of the performers' collaboration focused on extending and refining the mapping relationships used to translate performer gesture into sound. To this end, weekly meeting/mapping sessions were held in which group members collaboratively developed instrument voices and playing techniques. It became apparent at an early stage in the project that contributors were dealing with redundant tasks for gesture acquisition and conditioning, and that a set of higher-level building block tools would be useful to aid composers and musicians in the development of mappings. Therefore, in a collaborative research effort we started collecting abstractions into a software repository to which individual developers would contribute with the goal of developing a toolbox allowing to focus on higher-level tasks. Both the Digital Orchestra Toolbox and the mapping system described in [4] were developed to make these meetings more productive and enjoyable.

2. RELATED WORK

The hid library for Pd is a collection of approximately thirty objects for the Pure Data graphical programming environment, created to facilitate the use of standard Human Interface Devices (HIDs) for controlling software[8]. The library provides an object `hid` for interfacing with generic HIDs, as well as specific objects built on top of it for using joysticks, computer mice, and computer keyboards as input. In addition, Steiner includes objects for processing the data from the HID, including cartesian-polar transformation, filtering, and logarithmic and exponential curves.

One year later, Steiner released and published the Mapping Library for Pd, building on the experience of the hid library [9, 10]. Steiner states that the larger goal of the library (beyond its use to the Pd and DMI communities) is to start a dialogue on the subject of standard *primitives* for mapping, analogous to standard unit generators for audio. In this way the library serves to explore the sort of basic functions that are necessary for mapping. There are approximately 130 objects in the mapping library, coded by Steiner and by Cyrille Henry, including a great number of transfer functions, curves, break-point functions, and control-rate IIR and FIR filters. Interpolation objects and some windowed statistics (mean, median, minima and maxima) are also represented.

Since the Mapping Library was developed with the goal of defining and creating a complete set of mapping primitives,

NIME'18, June 3-6, 2018, Blacksburg, Virginia, USA.

it is not surprising that it is so exhaustive. In contrast to our approach, the library uses normalized values (0–1) for nearly all input and output, including for angles and musical pitches. Both the hid and mapping libraries are included in the Pd-extended distribution[1].

3. THE DIGITAL ORCHESTRA TOOLS

The Digital Orchestra Toolbox (DOT) is a collection of modular tools we developed for creating digital musical instruments for the McGill Digital Orchestra Project. Providing 130 tools, each with corresponding help/documentation, the intention was to solve specific needs during project development rather than to define and complete a set of mapping primitives. An additional difference from the Mapping Toolbox is that, wherever feasible, the DOT employs physical units (e.g., radians, millimeters) rather than normalizing in order to avoid assumptions about the context of use.

The Digital Orchestra Toolbox has been under active development since 2006, and available publicly from the IDMIL website since 2008. We have tracked nearly 4000 direct downloads of the toolbox since 2011, and it is also distributed as part of various DMI software driver packages and compositions. It is currently available from the package manager built into Max.

3.1 Abstractions vs. External Objects

Unlike many packages available for extending Max, the digital orchestra toolbox consists only of *abstractions* – Max patchers loadable as objects – rather than precompiled *external objects*. Thus, in addition to ensuring cross-platform functionality, the internal structure and function of the tools is viewable, understandable, and editable by users of the environment (Max) – *within the context of that environment*, and thus provide not only functional, but pedagogical value, encouraging reuse, adaptation, and appropriation.

3.2 Contents of the Toolbox

In this section we briefly outline the organization and current contents of the Digital Orchestra Toolbox. This outline is not an exhaustive list of the available tools, but is intended to simply highlight some of the more interesting features.

3.2.1 *Stream Processing*

Since the toolbox was developed primarily for supporting DMI development, the largest category of tools it contains relate to processing streams of real-time sensor data. This includes an array of smoothing filters (averaging filters, median filters), calculation of standard deviation or exponential moving deviation, detection of local minima and maxima, etc.

One of the most popular and useful abstractions added to the toolbox ended up being *leaky integration*. As an estimator of recent "energy" expended by a performer over the course of a movement, gesture, or sequence, the 'leaky bucket' metaphor was both powerful and easily understandable by composers and performers. For this purpose, the toolbox offers the abstraction `dot.aggregate.leaky` which includes a simple linear leak by default but can be configured to adapt leak size as a function of the internal aggregated value – enabling for example a system with multiple resting states that can be perturbed by the actions of a performer.

3.2.2 *Open Sound Control (OSC)*

The toolbox also contains a number of abstractions related to sending and receiving Open Sound Control (OSC) messages. The Max internal objects `udpsend` and `udpreceive` already handle the low-level aspects of packaging OSC messages correctly, however it can be difficult to construct and route OSC address strings (message identifiers) without resorting to external objects such as CNMAT's `oscroute` or the Jamoma project's `j.oscroute`. The DOT provides utilities for constructing address strings, and the abstraction `dot.osc.route` for routing.

3.2.3 *Serial Communications*

Many of the digital musical instruments developed by our research group rely on serial data transport from an embedded platform such as Arduino[2] to a laptop computer. To support simple and trouble-free interfaces of this sort, the DOT includes a wrapper for the Max internal `serial` object that supports reading and/or polling the serial port at a given rate. In practice, we have usually found that pushing the serial data rather than polling for each sample results in higher update rates and lower jitter, but can result in overflow and system crashes; to avoid this problem the `dot.io.serial` abstraction is used to send a heartbeat message to the Arduino firmware, without which the device switches to standby mode until the next heartbeat is received.

Another method for increasing the communication rate of our DMIs is to send binary data rather than ascii text (i.e. in Arduino use the method `Serial.write()` rather than `Serial.print()`). This method requires packetization of messages which we usually accomplish using the low-overhead Serial Line Internet Protocol (SLIP coding). The abstractions `dot.slip.encode` and `dot.slip.decode` are provided, along with examples for packetizing messages from Arduino firmware.

3.2.4 *Orientation*

One of the more complex tasks in the creation of DMIs that include motion or orientation sensing is to process data in polar, spherical or quaternion representations, since naïve application of operations such as averaging filters or interpolation cannot be used in those contexts. The DOT includes a few abstractions for converting between Cartesian and polar/spherical coordinate systems (to complement the existing internal objects available in Max), and a utility for "unwrapping" a polar representation so that averaging filters can be applied. Basic quaternion operations are also provided, including multiplication, inverse, conjugate, and spherical interpolation. These have been employed for processing and fusing acceleration, angular velocity, and magnetic field data from IMU/MARG systems in several DMIs.

3.2.5 *Gesture*

The *gesture* category currently only includes one abstraction: `dot.gesture.jab`. This tool was developed for characterizing jabbing gestures from acceleration data for the T-Stick DMI, and uses somewhat more complex signal processing than a simple derivative (see figure 1). We hope to expand the selection of gesture-related tools in the future.

3.2.6 *Audio*

The development of audio tools for DOT was motivated by the notion that many musical gestures can be seen as continuous functions of time, much like audio signals. Today's standard computer's audio processing capabilities satisfy requirements for *control intimacy* (e.g. $< 10ms$ latency,

[1] `http://puredata.info/`

[2] `https://www.arduino.cc/`

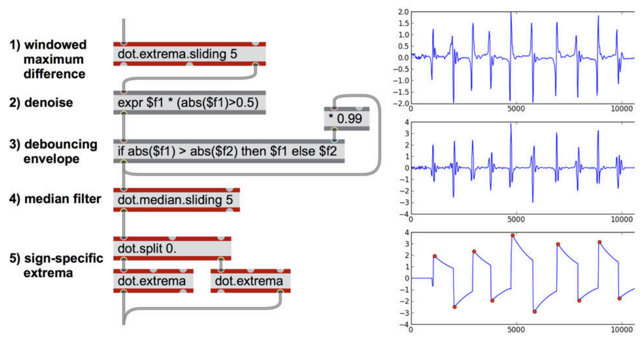

Figure 1: Left: Max patch for detecting *jab* gestures for the T-Stick DMI. Right: graphs showing the process of gesture extraction as the instrument is jabbed in alternating directions every second; top: incoming acceleration measurements; middle: windowed maximum difference; bottom: debouncing envelopes with identified jabs located at the peaks.

$< 1ms$ jitter, $> 1kHz$ bandwidth) and allow for sample-accurate synchronization with media synthesis I/O streams, which event-based transmission schemes often lack [11].

Besides signal conditioning tools (e.g. auto-scaling, mixing), the DOT includes audio abstractions for following envelopes and detecting discrete events, as well as interpolation of magnitudes and phases of multiple input signals to a single output, a form of convergent mapping. The DMI described in section 4.3 required sampling the mechanical oscillation of elasto-resistive sensors at audio-rates; as a consequence, we extended the DOT with tools for Frequency Domain Multiplexing (FDM) via double-sideband-suppressed-carrier (DSBSC) amplitude modulation.

This allows for the transmission of multiple gesture signals in a single audio channel which is often favoured over Time-Domain Multiplexing (TDM) due to the relative simplicity and ease of implementation. DOT includes tools for extracting the real parts of multiplexed signals via heterodyne filtering (`dot.am.demodulate.asynchronous~`) and extraction of instantaneous magnitude, phase, and frequency from analytical signals (`dot.sinusoid.properties~`).

4. EXAMPLES OF USE

Here we present several DMI projects that were supported by the toolbox.

4.1 The T-Stick DMI

The T-Sticks are a family of gestural musical controllers in development since 2006 [5]. The hardware is presently in its third revision and approximately twenty more T-Sticks have been built, including prototypes integrating haptic feedback and additional sensing modalities. The T-Stick has been performed and demonstrated many times internationally, including appearances in North and South America, Europe, and Asia. The repertoire of compositions for the T-Stick includes works by eight different composers.

An effort was made to sense all of the affordances [6] of the interface. A combination of inertial, pressure, deformation and multitouch capacitive touch sensing is used, with all of the sensors and circuitry embedded inside the interface body. From these sensors a large number of variables are measured or derived: touching, squeezing, twisting, brushing, hitting, shaking, swinging, tilting, and rolling.

In software, the T-Stick uses 55 instances of the DOT tools, in addition to the objects used for communication (the T-Stick uses the Max bindings for *libmapper* [4] to dynamically create mapping connections). Serial input and SLIP decoding abstractions are used as described above, and smoothing, denoising, debouncing, and polar or spherical conversions are performed to condition the incoming data for further processing or analysis. For some models of T-Stick, gyroscope data (angular velocity) is decoded from hacked WiiMotionPlus hardware and bias is removed with adaptive filters prior to sensor fusion and processing using the DOT's selection of quaternion abstractions.

The tool `dot.aggregate.leaky` is used seven times in the driver software, for calculating measures of brushing, brushing "energy", shaking, and a helical "eggbeater" gesture.

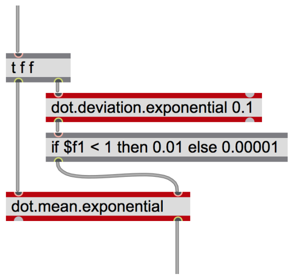
t f f
dot.deviation.exponential 0.1
if $f1 < 1 then 0.01 else 0.00001
dot.mean.exponential

Figure 2: Efficient adaptive bias removal used for processing rate gyroscope data. An inexpensive exponential approximation of signal deviation is used to estimate activity; the bias estimate will converge quickly when the sensor is at rest, but resist disturbance from noisy movement.

4.2 The Spine DMI

The Spine is a "prosthetic" digital musical instrument developed for the collaborative project *Les Gestes*, in which we endeavoured to design new instruments for dancers [3]. The new instruments extrapolate from the T-Stick, which we had already used in the performance *Duo pour un violoncelle et un danseur* with the same collaborators. Starting with foam prototypes, the Spine and its companion instruments the Rib and the Visor were developed iteratively using participatory design through frequent workshops, parallel problem solving, and digital fabrication methods. The current models are fabricated from laser-cut transparent acrylic, transparent PVC tubing, and PETG rods. The entire structure is assembled using interference fitting rather than any glues or fasteners.

In terms of sensing, the Spine tracks and reports its orientation and shape using inertial and magnetic-field sensing at each end of its body. Unlike the T-Stick, sensor-fusion algorithms run on-board each sensing node in the instrument in order to support scaling to large numbers of nodes. While porting quaternion tools to C for the embedded sensor fusion eliminated the need for some DOT tools, a companion visualization of the Spine was created for debugging and presentation which required the addition of matrix versions of the quaternion tools (figure 3).

Work on the Spine project also highlighted the need for careful treatment of floating-point data when using IIR filters downstream, since even one `NaN` or `INF` value will persist in the output until the patch is restarted. Subsequently, "sanity checks" on floating-point data were added to our regular process, and the `dot.float.sanitize` abstraction is now embedded in several filter tools by default.

In rehearsal and on tour for the piece *Les Gestes*, two Spine DMIs shared a driver application with a number of Rib and Visor DMIs; in all, the application included 109 instances of DOT tools.

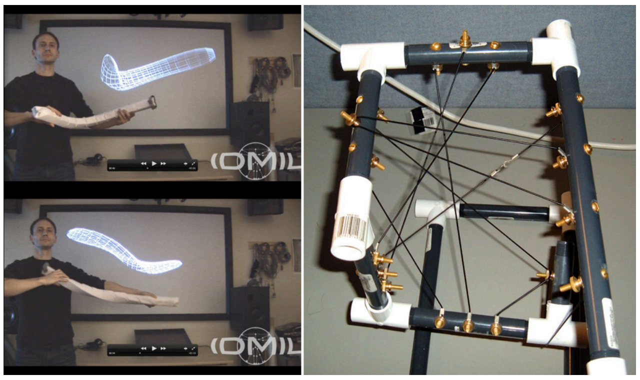

Figure 3: Left: live visualization of the Spine shape and orientation; right: the prototype sensor-harp.

4.3 Sensor Harp DMI

This project was developed by Graham Boyes and our second author within a course on DMIs at the SSMU of McGill University in 2009. The aim was to build a physically-informed gestural controller whose primary means of interaction involves the manipulation of a vibrating mechanism and therefore may be considered a hybrid between a controller and a sound generator. The controller consists of thin PVC tubes into which regularly-spaced copper screws are fixed. Inside the tubes individual wires are attached to the copper screws on one end, and to inputs and outputs of an audio interface on the other, transmitting sinusoidal carrier signals produced by the computer. The sensing system consists of carbon-coated, elastic sensors which change their electrical resistance as a function of stretch (Images SI Inc). The sensors can then be used as connectors between the copper screws closing the loop for returning the produced carrier signals back to the computer. Manipulation of the stretch sensors thus produces amplitude modulations which can be extracted and used as control signals, see also [2] for a similar approach to gesture acquisition from a 2D surface. This acquisition scheme can be useful for detection of performance gestures: the discontinuity in the signal when plucked (in contrast to fast pulling or stretching) causes a spike in the instantaneous frequency estimation, which we found to be more accurate compared to amplitude-based detection, such as envelope followers or Schmitt-triggers, for example (cf. section 3.2.6). A major design goal was to reinforce the physical interaction between performer and instrument, making use of the controller's inherent visual, tactile and kinesthetic feedback. Rather than carrying the gestures of the human performer, the modulated audio signals carry gestures of the performer-system-interaction. Fig. 3 shows a prototype build of the controller.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have described an open-source "toolbox" consisting of more than 130 abstractions for the Max programming environment, focused on supporting the task of developing new digital musical instruments. An overview of the toolbox structure and scope was provided, along with several examples of use from the development of digital musical instruments.

This toolbox has continued to grow and evolve since the project in which it originated, and we invite any interested programmers, instrument designers, composers or performers to use, modify, or contribute to all aspects of the project[3].

Future work will focus primarily on improving the documentation for the toolbox. Each tool is already accompanied by a working example in the form of a Max help patcher; we plan to add more in-depth descriptions and a series of tutorials focused on cleaning and processing sensor data streams and extracting "gestural" information.

[3]https://github.com/IDMIL/digital-orchestra-toolbox

6. ACKNOWLEDGMENTS

The creation of the Digital Orchestra Toolbox was supported by funding from the Fonds de recherche du Québec - Société et culture and the Natural Sciences and Engineering Research Council of Canada. Development tool place primarily at the Input Devices and Music Interaction Laboratory, McGill University.

7. REFERENCES

[1] S. Ferguson and M. M. Wanderley. The McGill Digital Orchestra: An interdisciplinary project on digital musical instruments. *Journal of Interdisciplinary Music Studies*, 4(2):17–35, 2010.

[2] R. Jones, P. Driessen, A. Schloss, and G. Tzanetakis. A force-sensitive surface for intimate control. In *In Proceedings of the International Conference on New Interfaces for Musical Expression (NIME*, pages 236–241, 2009.

[3] J. Malloch. *A Framework and Tools for Mapping of Digital Musical Instruments*. PhD thesis, McGill University, December 2013.

[4] J. Malloch, S. Sinclair, and M. M. Wanderley. Distributed tools for interactive design of heterogeneous signal networks. *Multimedia Tools and Applications*, 74(15):5683–5707, February 2014.

[5] J. Malloch and M. M. Wanderley. The T-Stick: From musical interface to musical instrument. In *Proceedings of the 2007 International Conference on New Interfaces for Musical Expression (NIME07)*, New York City, USA, 2007.

[6] D. A. Norman. *The design of everyday things.* Doubleday, 1990.

[7] X. Pestova, E. Donald, H. Hindman, J. Malloch, M. T. Marshall, F. Rocha, S. Sinclair, D. A. Stewart, M. M. Wanderley, and S. Ferguson. The CIRMMT/McGill Digital Orchestra project. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 295–298, 2009.

[8] H.-C. Steiner. [hid] toolkit: a unified framework for instrument design. In *Proceedings of the 2005 International Conference on New Interfaces for Musical Expression*, pages 140–143, Vancouver, Canada, 2005.

[9] H.-C. Steiner. Towards a catalog and software library of mapping methods. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 106–109, Paris, France, 2006. IRCAM – Centre Pompidou.

[10] H.-C. Steiner and C. Henry. Progress report on the mapping library for Pd. In *Proceedings of the PureData Convention*, Montreal, Canada, 2007.

[11] D. Wessel and M. Wright. Problems and prospects for intimate control of computers. *Computer Music Journal*, 26:3:11–22, Fall 2002.