Real-Time Motion Capture Analysis and Music Interaction with the Modosc Descriptor Library

Federico Visi
Institute for Systematic Musicology
Universität Hamburg
Hamburg, Germany
mail@federicovisi.com

Luke Dahl
Department of Music
University of Virginia
Charlottesville, VA, United States
lukedahl@virginia.edu

ABSTRACT

We present *modosc*, a set of Max abstractions designed for computing motion descriptors from raw motion capture data in real time. The library contains methods for extracting descriptors useful for expressive movement analysis and sonic interaction design. Moreover, *modosc* is designed to address the data handling and synchronization issues that often arise when working with complex marker sets. This is achieved by adopting a multiparadigm approach facilitated by odot and Open Sound Control to overcome some of the limitations of conventional Max programming, and structure incoming and outgoing data streams in a meaningful and easily accessible manner. After describing the contents of the library and how data streams are structured and processed, we report on a sonic interaction design use case involving motion feature extraction and machine learning.

Author Keywords

Expressive movement, motion capture, motion descriptors, motion features, sonic interaction design, musical interaction, HCI, Max, Open Sound Control, modosc, NIME.

CCS Concepts

•**Applied computing** → **Sound and music computing;** Performing arts; •**Information systems** → *Music retrieval;*

1. INTRODUCTION

Motion capture technologies have been employed in several research fields. As precision and accessibility of motion tracking improved, an increasing number researchers took advantage of such technologies to study human body movement in multiple contexts. Motion data has been at the center of many research works dealing with music cognition, gait analysis, dance studies, behavioral sciences, user studies, robotics, and more. In addition to its more conventional usage in digital animation and game design, motion capture has been employed to develop immersive experiences involving virtual and augmented reality, for scientific research as well as for entertainment purposes.

Despite the technology being a few decades old, marker-based infra-red motion capture (MoCap) can be considered the gold standard for measuring complex movement in a three-dimensional space. Tracking precision and temporal resolution have progressively improved, allowing accurate tracking of finger movements and facial expressions. Recent MoCap systems are also capable of streaming motion data live, thus making real-time applications possible. Some programming environments such as Eyesweb [1] offer tools for real-time human movement analysis.

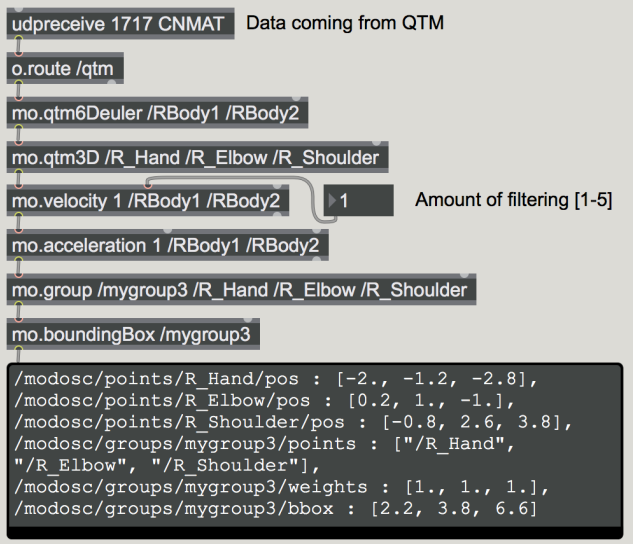

Figure 1: A chain of modosc abstractions processing a stream of motion data and their OSC output printed in an o.display. Abstractions are connected in series, resulting in patches that are easy to read and maintain.

2. MOTIVATION AND DESIGN

Having access to real-time motion data allows one to experiment with musical interactions based on body movement. However, extracting motion descriptors from raw motion data is often a necessary step in the design of effective motion-to-sound mappings. In addition, motion descriptors are frequently employed in expressive movement analysis, motion recognition, and music performance analysis. Providing easy access to motion descriptors computed in real-time from motion capture data is the main purpose of *modosc*[1] (MOtion Descriptors OSC).

Modosc is a set of Max abstractions for handling motion data in real-time using Open Sound Control (OSC) [14] and extracting various motion descriptors from raw motion data streams using the multiparadigm programming approach of-

[1] `https://github.com/motiondescriptors/modosc`

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

fered by odot [8]. OSC was chosen as the main networking protocol due to its flexibility, cross-platform compatibility, ease of use, and popularity among music and media arts researchers and practitioners [6]. While Max offers many useful tools for rapidly prototyping musical interactions, standard Max programming approaches can make tasks such as synchronizing and naming multiple real-time data streams often challenging. To address these challenges, modosc methods are implemented using odot objects. Odot is a framework for writing dynamic programs using C-like language inside a host environment such as Max [8]. Compared to more conventional Max objects, it provides access to advanced formatting and parsing of OSC data bundles, allowing for greater control over timing and synchronization of multiple data streams. In addition, it allows the evaluation of functions that would be difficult or cumbersome to implement using standard objects.

The typical modosc data flow consists of abstractions connected in series as shown in figure 1. Rather than sending many messages in parallel, motion data is bundled into OSC packets. In each abstraction, data that is already in the stream may be used as inputs to expressions written and evaluated inside a `o.expr.codebox`. These may generate new data bound to OSC addresses, which are then blended with the rest of the stream before being sent out of the outlet. Figure 2 shows an example of how this process works in `mo.centroid`, an abstraction for calculating the geometric center of a groups of points. This design approach results in more streamlined patches that are easier to maintain and re-purpose. Moreover, this avoids many timing and synchronization issues that would arise using a more conventional patching approach.

For the development of the alpha release of modosc, we took advantage of the native support for OSC real-time streaming of Qualisys Track Manager (QTM), Qualisys' tracking software used for recording data using their Oqus and Miqus cameras.

3. HANDLING OF AFFERENT STREAMS AND DATA TYPES

Before motion descriptors can be extracted, MoCap data streams from QTM need to be parsed and assigned to OSC addresses in the `modosc/` domain. Data from a marker or a rigid body tracked by QTM can be used to define a new modosc point using `mo.qtm3D` and `mo.qtm6Deuler` methods. Points can be then collected in groups. Points and groups are the two main data types on which modosc abstractions operate.

3.1 Points

A point consists in 3D or 6DoF data bound to an OSC address. The data is in the form of a vector describing the position (3D) or the position and rotation (6DoF) of a single point in the coordinate system. As an example, a 3D point named "hand" is defined by a 3D vector bound to the corresponding OSC address:
`modosc/points/hand/pos : [11, 22, 33]`.

Similarly, a 6DoF point named "forearm" has its position data (x, y, and z coordinates) bound to a specific OSC address: `modosc/points/forearm/pos : [11, 22, 33]`. Euler angles (roll, pitch, and yaw) are instead bound to an additional OSC address:
`modosc/points/forearm/rot: [44, 55, 66]`.

New points can be defined from incoming MoCap data as mentioned above, or by processing the data of previously defined points or groups. For example, `mo.centroid` creates a new point corresponding to the geometric centre of an input group (see figure 2).

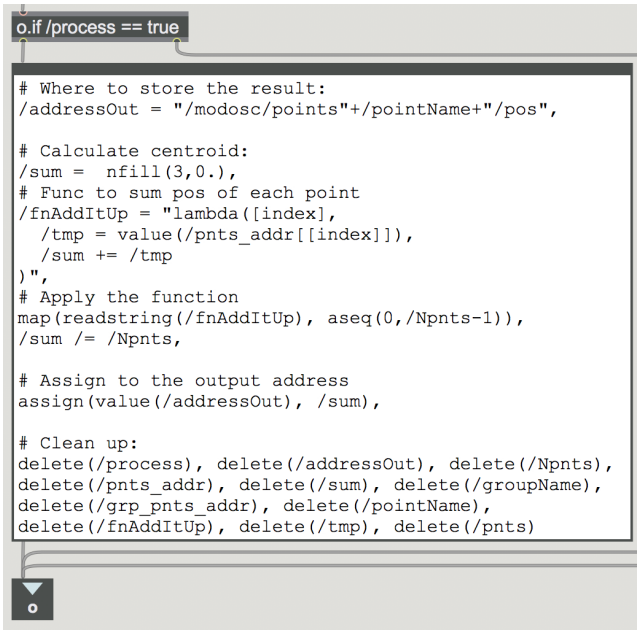

Figure 2: An example of how modosc abstractions are coded using odot in Max: the code in this `o.expr.codebox` is used to compute the centroid of an input group (see section 4.2.1).

3.2 Groups

In modosc, a group is a list of points bound to an OSC address. Groups are defined using `mo.group`. To avoid confusion with points, new groups are bound to a new OSC address in the `/modosc/groups` sub-domain. Groups do not directly contain MoCap data. Rather, they are used to refer to multiple points that define a specific area of a marker set. For example, the group `/modosc/groups/arms` might be used to list the points placed on the arms of a performer. The group address can then be used as the argument of modosc abstractions that compute descriptors using multiple points, such as `mo.contractionIndex`. In addition, groups also store the weight values of each point in the list. Weights are used as parameters for the computation of some motion descriptors such as `mo.QoM` (quantity of motion, see section 4.2.2). When a new group is created, all weights are set to 1 by default and can then be changed using `mo.setWeights`.

4. DESCRIPTORS

Modosc abstractions typically consist of odot code wrapped in a Max patcher (see figure 2). Some abstractions may use local variables bound to temporary OSC addresses that are deleted before the new data stream is sent to the outlet.

There are two main types of descriptors:

- single point descriptors: they take one or more points as their main arguments and output a descriptor for each input point;
- group descriptors: they take a group as their main argument and output descriptors that refer to the motion of the whole group.

Generally, using modosc descriptors do not require changing the odot code, and – as it is idiomatic in Max – all the inputs can be specified in the arguments of the abstraction or by sending messages to secondary inlets.

4.1 Single Point Descriptors

4.1.1 Velocity, Acceleration, and Jerk

`mo.velocity` and `mo.acceleration` take a single point as their argument and output 3D vector and magnitude of velocity and acceleration respectively bound to new OSC addresses. For example, `mo.acceleration /Hand` computes the acceleration of the point `/Hand` and outputs the acceleration vector to `/modosc/points/Hand/acc` and the magnitude to `/modosc/points/Hand/acc_mag`. Velocity and acceleration are calculated using an odot implementation of Skogstad's IIR low-pass differentiators, which are designed to reduce noise in real-time motion capture applications [11]. Jerk is calculated as the first difference of acceleration, thus `mo.acceleration` must precede `mo.jerk`.

4.1.2 Fluidity

Inspired by the theoretical work on human motion by Flash and Hogan [5], Piana et al. [9] defined Fluidity Index as the inverse of the integral of jerk. Fluidity Index is implemented in `mo.fluidity`, which takes a single point as an argument and outputs fluidity index to `/modosc/points/point_name/fluidity`.

4.2 Group Descriptors

4.2.1 Centroid and Center of Mass

`mo.centroid` calculates the geometric center of the points listed in the input group. The coordinates of the centroid are bound to a new OSC address in the `/modosc/points` sub-domain. The name of the output point is specified in the second argument. `mo.centerOfMass` works in a similar way, except it additionally takes into consideration the weight of each point in order to compute the center of mass (or barycenter) of the input group. Weights can be changed using `mo.setWeights`, as described in section 3.2.

4.2.2 Quantity of Motion

Fenza et al. defined quantity of motion (QoM) as the sum of the speeds of a set of points multiplied by their mass [3]. Glowinski et al. [7] included a similar measure in their expressive feature set, denoted as overall motion energy. Modosc implementation (`mo.QoM`) computes QoM of a group of points, taking into consideration the weight of each point.

4.2.3 Contraction Index

Contraction index is calculated by summing the Euclidean distances of each point in a group from the group's centroid [3]. It is an indicator of the overall contraction or expansion of a group of points and it has been used for emotion recognition applications based on body movements [9]. In modosc, contraction index is implemented in `mo.contractionIndex`.

4.2.4 Bounding Box

Bounding shapes have been used in the analysis of affective gestures [7]. `mo.boundingBox` computes the height, width, and depth of the rectangular parallelepiped enclosing the points listed in the input group.

5. USE CASE: INTERACTION DESIGN WITH MODOSC AND WEKINATOR

To provide a practical example of how the library can be used alongside other tools for sonic interaction design, we describe a basic use case in which three modosc descriptors are sent as input features to a set of machine learning models in Wekinator [4] to control the parameters of a physical modelling algorithm in Max. We chose Wekinator since it is a well-documented, easy to use machine learning tool, popular among music and media arts researchers and practitioners. However, any other environment and framework that can send and receive OSC messages can be used with modosc. To make this example easier to replicate, we will use some of the Max patches available in Wekinator's example package[2]. The general data flow is schematized in figure 3.

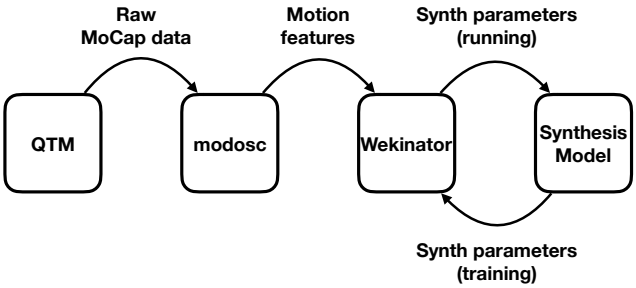

Figure 3: Data flow of the example use case with modosc, Wekinator, and physical modelling.

We used a simple marker set with three rigid bodies, one placed on the chest of the performer and the remaining two on the hands. In QTM, we named the rigid bodies `chest`, `L_hand`, and `R_Hand` respectively and activated real-time data streaming via OSC. Modosc is employed to extract the fluidity index of each hand and the contraction index of a group containing all three rigid bodies. To do so, in Max the data from QTM is used to define three new 6DoF points using `mo.qtm6Deuler`. The new points are then used to form a new group named `upper_body` using `mo.group`. To extract the descriptors, `mo.fluidity` is used with `/L_hand` and `/R_Hand` as arguments, while `mo.contractionIndex` is instead called on `/upper_body`. After these simple steps, the three descriptors are accessible at their respective OSC addresses:

```
/modosc/points/L_hand/fluidity
/modosc/points/R_hand/fluidity
/modosc/groups/upper_body/contractionIndex
```

The descriptors were then streamed to the default input port of Wekinator[3]. Wekinator was set to get 3 inputs and send 9 outputs, all continuous controls. The outputs were sent to the 'BlotarSynth_9ContinuousOutputs.maxpat' patch from the Wekinator's example package. The patch includes an object from the PeRColate package[4] that implements a hybrid physical model, and is already configured to get the outputs of the machine learning models and send its control parameters back to Wekinator using the default OSC ports. To train Wekinator's models, we simply selected a preset or adjusted parameters manually in the synthesis patch to obtain the desired sound, hit 'Start Recording' in Wekinator, performed some example arm movements, and stopped data recording before training the models. The same procedure can be repeated with a number of different synthesis presets. After the training procedure is completed, Wekinator can be set to 'Run' in order to get live motion descriptor data from modosc and control the physical model in real time.

[2] `https://github.com/fiebrink1/wekinator_examples`
[3] To do this more quickly, one can simply modify the patch 'SimpleMax_3Inputs.maxpat' found in Wekinator's examples package.
[4] `https://github.com/Cycling74/percolate`

6. DISCUSSION AND FUTURE WORK

The design criteria we adopted were aimed at keeping modosc simple and intuitive to use. The number of available data types is purposely kept low, and rotation angles of 6DoF data are bound to dedicated addresses. This was done in order to avoid potential issues or confusion when using descriptors operating only on positional data. Weight values are stored in groups as this appeared to be the best design choice in terms of usability. In the future, we aim at using weight values to implement more descriptors based on mass.

Currently, one of the main limitations of modosc is that it relies on QTM's native OSC support, and thus it works only with Qualisys systems. However, other motion capture systems can be configured to stream data via OSC by taking advantage of their respective SDKs, and new modosc abstractions can be made to create points and groups from different MoCap data streams. For example, Optitrack motion tracking data can be streamed to various programming environments using their proprietary NatNet SDK. Then, it is relatively simple to implement an OSC bridge[5] and stream the data to Max, where it will be parsed to be used with modosc abstractions.

The way motion data is structured in modosc makes extracting descriptors from multiple sources relatively easy. Thus, we do not exclude the possibility of extending the same design principles and include descriptors for other devices such as IMUs and EMG sensors, since both technologies have been used extensively for music interaction design. [13, 12].

The alpha release of modosc will include the descriptors mentioned in this paper. We are planning to add other abstractions to implement periodic quantity of motion [13] and strike detection [2] in the near future. In addition, the library includes templates to help users with implementing new or custom point and group descriptors.

Rotations are currently expressed in Euler angles. We are planning to add support for quaternion representation, which would enable the implementation of various descriptors based on rotational data [13] and also avoid issues such as gimbal lock.

7. CONCLUSIONS

We presented a set of Max abstractions that make use of odot and OSC to overcome some of the data structuring and synchronization challenges that a more conventional patching approach may have led to. Moreover, we described how modosc can be used in a sonic interaction design scenario involving tools and processes familiar to the NIME community. Thanks to its flexibility and ease of use, we believe modosc can be a useful tool for rapidly prototyping music interactions based on body movement. We also envision using the library in research projects involving virtual and augmented reality, especially for the development of musical instruments and interactions in virtual environments [10].

8. ACKNOWLEDGMENTS

The authors would like to thank John Maccallum and Rama Gottfried for their very kind support on using odot.

This research was partially supported by the European Research Council (grant agreement: 725319, PI: Clemens Wöllner) for the five-year project "Slow motion: Transformations of musical time in perception and performance" (SloMo).

[5]Users have made available several OSC streaming tools, such as OscStreamer for C++ (`https://github.com/hezhao/OscStreamer`, not tested by the authors).

9. REFERENCES

[1] A. Camurri, B. Mazzarino, and G. Volpe. Analysis of Expressive Gesture: The EyesWeb Expressive Gesture Processing Library. In *Gesture-based Communication in Human-Computer Interaction, LNAI 2915*, pages 460–467. 2004.

[2] L. Dahl. Studying the Timing of Discrete Musical Air Gestures. *Computer Music Journal*, 39(2):47–66, jun 2015.

[3] D. Fenza, L. Mion, S. Canazza, and A. Rodà. Physical movement and musical gestures: A multilevel mapping strategy. In *Proceedings of Sound and Music Computing Conference, SMC 2005*, Salerno, Italy, 2005.

[4] R. Fiebrink, D. Trueman, and P. Cook. A metainstrument for interactive, on-the-fly machine learning. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 280–285, Pittsburgh, PA, USA, 2009.

[5] T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *The Journal of Neuroscience*, 5(7):1688–1703, jul 1985.

[6] A. Freed and A. Schmeder. Features and future of open sound control version 1.1 for nime. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 116–120, Pittsburgh, PA, United States, 2009.

[7] D. Glowinski, N. Dael, A. Camurri, G. Volpe, M. Mortillaro, and K. Scherer. Toward a Minimal Representation of Affective Gestures. *IEEE Transactions on Affective Computing*, 2(2):106–118, apr 2011.

[8] J. Maccallum, R. Gottfried, I. Rostovtsev, J. Bresson, and A. Freed. Dynamic Message-Oriented Middleware with Open Sound Control and Odot. In *International Computer Music Conference*, Denton, United States, 2015. ICMA.

[9] S. Piana, A. Staglianò, F. Odone, and A. Camurri. Adaptive Body Gesture Representation for Automatic Emotion Recognition. *ACM Transactions on Interactive Intelligent Systems*, 6(1):1–31, mar 2016.

[10] S. Serafin, C. Erkut, J. Kojs, N. C. Nilsson, and R. Nordahl. Virtual Reality Musical Instruments: State of the Art, Design Principles, and Future Directions. *Computer Music Journal*, 40(3):22–40, sep 2016.

[11] S. A. Skogstad. Filtering Motion Capture Data for Real-Time Applications. In W. Yeo, K. Lee, A. Sigman, J. H., and G. Wakefield, editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 142–147, Daejeon, Republic of Korea, may 2013. Graduate School of Culture Technology, KAIST.

[12] S. A. Skogstad, K. Nymoen, and M. Hovin. Comparing Inertial and Optical Mocap Technologies for Synthesis Control. *Smc'11*, pages 421–426, 2011.

[13] F. Visi, E. Coorevits, R. Schramm, and E. R. Miranda. Musical Instruments, Body Movement, Space, and Motion Data: Music as an Emergent Multimodal Choreography. *Human Technology*, 13(1):58–81, may 2017.

[14] M. Wright. Open Sound Control: An enabling technology for musical networking. *Organised Sound*, 10(3):193–200, 2005.