Modosc: A Library of Real-Time Movement Descriptors for Marker-Based Motion Capture

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

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

ABSTRACT

Marker-based motion capture systems that stream precise movement data in real-time afford interaction scenarios that can be subtle, detailed, and immediate. However, challenges to effectively utilizing this data include having to build bespoke processing systems which may not scale well, and a need for higher-level representations of movement and movement qualities. We present *modosc*, a set of Max abstractions for computing motion descriptors from raw motion capture data in real time. *Modosc* is designed to address the data handling and synchronization issues that arise when working with complex marker sets, and to structure data streams in a meaningful and easily accessible manner. This is achieved by adopting a multiparadigm programming approach using o.dot and Open Sound Control. We describe an initial set of motion descriptors, the addressing system employed, and design decisions and challenges.

CCS CONCEPTS

• **Computing methodologies** → **Motion capture**; *Motion processing*; • **Human-centered computing** → *Interaction design*; • **Applied computing** → Performing arts;

KEYWORDS

Motion capture, motion descriptors, motion analysis, expressive movement, interaction design, Max, Open Sound Control, modosc.

ACM Reference Format:
Luke Dahl and Federico Visi. 2018. Modosc: A Library of Real-Time Movement Descriptors for Marker-Based Motion Capture. In *Proceedings of ACM MOCO conference (MOCO'18).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 MOTION CAPTURE

Methods for recording human and animal movement have relied on optical "capture" of movement since the work of Muybridge and Marey in the late 19th century [11]. Marey and Bernstein developed techniques for tracking the position of specific points or line segments on the mover's body through the use of retro-reflective tape or light bulbs, thereby reducing the complexity of representation in order to facilitate analysis. Interestingly, marker-based infra-red motion capture (MoCap) systems, considered the "gold standard" for measuring complex movement, still rely on optical measurement of a small number of points on the body.

Motion tracking technologies are used in a variety of applications. Researchers study human body movement in the context of medicine, sports science, dance, robotics, embodied cognition, music, and more. Animators and game designers frequently use motion data as source material. Newer MoCap systems that are capable of streaming data in real-time are used in immersive virtual and augmented reality systems, movement-based interaction design, and in scientific studies.

1.1 Motion Descriptors

The main feature of marker-based MoCap, i.e. its ability to track the precise location in 3D space of a number of points on a mover's body, can also prove to be an impediment or challenge to the effective use of the data. Depending on the application, the user of the data may be interested in aspects of the movement that are not easily visible from 3D position data. For example, body-centric metrics such as joint angles must be computed from the positions of multiple points.

Many such motion descriptors have been proposed. These are quantities which can be computed from the displacement data of single or aggregate markers. So-called "low-level" descriptors might represent the velocity or acceleration of a single point, whereas"high-level" descriptors are intended to represent qualities of the body's shape or of the history of recent movements. For example, researchers have described algorithms for detecting the affective state of the mover [7] or for calculating "expressive" aspects of a movement [8].

Most commercial MoCap systems provide native computation of some low-level descriptors, such as position derivatives or joint angles. To compute other descriptors researchers must rely on libraries such as the MoCap Toolbox [2], or they must code their own. For many of these systems motion descriptors are calculated "off-line", taking as input the recording of an entire motion. This has benefits, such as having access to future position values to calculate movement derivatives for a specific moment in time. However, relying on off-line processing precludes the use of motion descriptors in situations that require real-time responses or feedback.

1.2 The Need for Real-time Motion Descriptors

In this paper we present *modosc* (for "MOtion Descriptors OSC"), a library of Max abstractions for providing easy access to motion descriptors calculated from MoCap data in real-time.

The creation of modosc is motivated by a number of use cases. Studies on the perception of sound and movement may require real-time feedback from a participant's movement. Real-time feedback from a high-fidelity MoCap system can be used to prototype

MOCO'18, 28-30 June 2018, Genoa, Italy

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

movement-based interactions that may then be implemented and deployed on lower cost systems. Similarly, when designing auditory displays that sonify motion [1], different sonification models can be quickly evaluated and synthesis parameters dynamically adjusted.

Real-time motion descriptors are especially useful in the performing arts, where, for example, a performer's movements can be used to affect aspects of accompanying sound or visuals. The authors are motivated in particular by wanting to explore interaction scenarios involving movement and musical sound.

Other programming frameworks address these needs in various ways. Most notably, Eyesweb [3] offers tools for real-time human movement analysis, while PiPo [12] is an API that provides tools to extract descriptors from audio and motion data. Whereas Eyesweb is standalone and runs on Windows only, Modosc takes advantage of Max as a popular environment for implementing real-time interactions. Compared to PiPo, the multiparadigm approach of modosc enables users to more easily create or modify descriptors.

2 THE MODOSC LIBRARY

Modosc is a set of Max abstractions that receives data in real-time from a marker-based MoCap system, computes various motion descriptors from this data, and makes the resulting values easily accessible for use within Max or other platforms for real-time interaction.

2.1 OSC and o.dot

Modosc uses Open Sound Control (OSC) [15] to receive, represent, and transmit motion data. OSC was chosen due to its flexibility, cross-platform compatibility, ease of use, and popularity among media arts researchers and practitioners [6]. Qualisys MoCap systems can stream OSC data, and other vendors have similar ways of streaming motion data over IP networks.

The implementation of modosc relies heavily on the multiparadigm programming approach offered by o.dot. O.dot is a framework for writing dynamic programs using C-like language inside a host environment such as Max [9]. While Max offers many useful tools for rapidly prototyping multimedia interactions, standard Max programming approaches can become unwieldy or completely ineffective when dealing with data from multiple data streams (i.e. MoCap markers) which need to be processed synchronously.

Compared to more conventional Max objects, o.dot allows advanced formatting and parsing of OSC data bundles. The ability to process an entire OSC bundle at once, instead of individual OSC messages, simplifies synchronization. And o.dot allows the evaluation of functions that would be difficult or cumbersome to implement using standard Max objects.

Within each modosc abstraction, data that is already in the stream of OSC bundles is used as input to expressions written and evaluated inside an `o.expr.codebox`. The results are 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 this process within the `mo.centroid` abstraction.

2.2 Data Types

Modosc stores both incoming MoCap data and the output of motion descriptor calculations as data bound to specific addresses within a stream of OSC bundles. Since OSC messages consist of an address and some data bound to that address, data within the modosc library is structured according to an address hierarchy. The two main data types used in modosc are *points* and *groups*.

2.2.1 Points. Points represent a single entity in space. The most common use of a point is to store the position of a specific MoCap marker. Points can also store orientation. Since many MoCap systems can identify small sets of markers in a unique static configuration and transmit both their position and orientation, points are also used to represent these "6DoF rigid bodies".

As an example, a 3D point named "handR" is defined as a 3D position vector bound to an OSC address and the resulting OSC message would be: `/modosc/points/handR/pos:[11,22,33]`. Similarly, a 6DoF body named "forearm" has its 3D position data bound as: `/modosc/points/forearm/pos:[11,22,33]`. And its orientation euler angles (roll, pitch, and yaw) are bound to an additional OSC address as: `/modosc/points/forearm/rot:[44,55,66]`.

New points may also be generated as the output of a motion descriptor. For example, the descriptor `mo.centroid` calculates a new point that represents the 3D position corresponding to the centroid of a group of points.

Points can also have other data associated with them. For example, if the point "handR" is processed by the `mo.velocity` descriptor, the magnitude velocity output would be stored in the stream as: `/modosc/points/handR/vel_mag:[1.234]`

2.2.2 Groups. A group is a named list of points bound to an OSC address in the `/modosc/groups` sub-domain. A group does not directly contain data for the points in the group. Rather it refers to the group's points which are already in the data stream.

A common use for groups is to define a region of the body for which you wish to calculate some descriptor. For example the group "hands", containing "handL" and "handR" would be represented as: `/modosc/groups/hands/points:["/handL","/handR"]`.

Groups are defined using the `mo.group` abstraction. The group name can then be used as the argument for modosc abstractions that compute descriptors on multiple points, and these results are stored with the group. For example, if the group "hands" is processed by `mo.contractionIndex` the output would be represented as: `/modosc/groups/hands/ci:[0.34]`.

Groups also store a weight value for each point in the list. Weights are used as parameters for some descriptors such as `mo.QoM` (quantity of motion). When a new group is created all weights are set to 1 by default and can then be changed using `mo.setWeights`.

2.3 Descriptors

The following motion descriptors are implemented in the initial release of modosc.

2.3.1 Point Descriptors. The first step in processing data streamed from a MoCap system is to format it into modosc points. The initial release of modosc can accept data from the Qualisys Track Manager (QTM) software. Data from a marker or a rigid body tracked by QTM can be used to define new modosc points using `mo.qtm3D` and `mo.qtm6Deuler`.

Velocity and acceleration of points are calculated using `mo.velocity` and `mo.acceleration`. These are based on Skogstad's low-pass

differentiators, which are designed specifically for dealing with possibly noisy MoCap position data [13]. Our implementation is coded in o.dot and can process all three dimensions for multiple points within a single descriptor object. These descriptors have a parameter for setting the amount of noise-reduction (i.e. low-pass filtering).

Jerk, the third derivative, is calculated by `mo.jerk` using a simple first-order difference of the already-filtered acceleration data. For each of these derivative descriptors, both the 3D vector and magnitudes are calculated. For example, when `mo.acceleration` processes the point "handR", the results will be stored as: `/modosc/points/handR/acc:[1.2,3.4,5.6]` and `/modosc/points/handR/acc_mag:[7.89]`.

Noisy position data can be filtered using `mo.positionLPF` which is also based on Skogstad's filters. This descriptor should be placed downstream of `mo.velocity` and `mo.acceleration` since these use position data as input and perform their own noise-reduction.

Inspired by theoretical work on human motion by Flash and Hogan [5], Piana et al. proposed fluidity index as a motion descriptor. Our implementation, `mo.fluidity`, takes a single point as an argument and outputs fluidity index to `/modosc/points/point_name/fluidity`.

2.3.2 Group Descriptors. The initial release of modosc includes the following descriptors for processing groups of points. The geometric center of the points in the group is calculated by `mo.centroid`. The coordinates of the centroid are bound to a new address in the `/modosc/points` sub-domain. `mo.centerOfMass` works in a similar way, except it takes into account the weight of each point.

Fenza et al. defined quantity of motion (QoM) as the sum of the speeds of a set of points multiplied by their mass [4]. Glowinski et al. [7] included a similar measure in their expressive feature set, denoted as overall motion energy. The modosc implementation, `mo.QoM`, computes QoM taking into consideration the weight of each point.

Contraction index is calculated by summing the Euclidean distances of each point in a group from the group's centroid [4]. 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 [10]. In modosc, contraction index is implemented in `mo.contractionIndex`.

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.

2.4 Using modosc

The modosc library allows clean and simple patches to compute motion descriptors on large amounts of MoCap data. Rather than sending many messages in parallel as in a typical Max patch, a typical modosc data flow consists of abstractions connected in series as shown in figure 1. This example patch receives data for two points from a Qualisys system. The data is formatted for modosc, velocity is calculated for each point, and the position data is smoothed. A group is created called `hands`, and then a number of descriptors are calculated on the group.

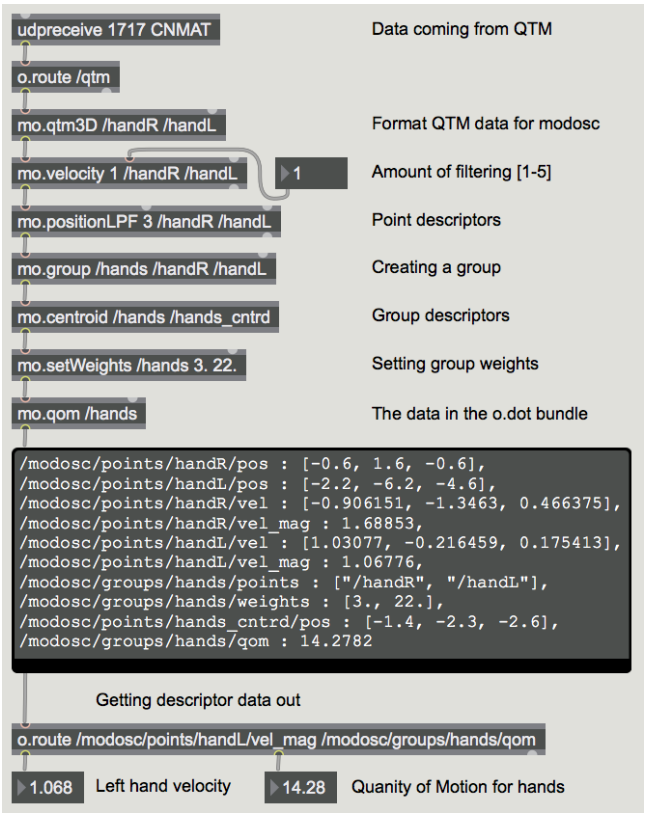

Figure 1: A chain of modosc abstractions for processing a stream of motion data and their outputs shown in an o.display. Abstractions are connected in series, resulting in patches that are easy to read and maintain.

The main consideration that a user needs to be aware of is that some descriptors rely on the output of other descriptors. For example `mo.qom` needs the velocity for each point in the group, and so `mo.velocity` for these points must be upstream.

Figure 1 also shows how the outputs of the descriptors are represented in the o.dot bundle. This data can easily be retrieved from the bundle for use in Max by using `o.route`, as shown in the figure.

3 DISCUSSION

3.1 Design Issues

Our design goals were for modosc to provide easy-to-use real-time motion descriptors, and to use a simple modular design that is easily extensible. We also chose to avoid repeated computation of low-level descriptors. For example, `mo.qom` requires that velocity already be computed upstream. This allows velocity to be computed once, and then used by subsequent descriptors. This goal is slightly at odds with the ease-of-use goal, as it requires the user to be aware of a descriptor's input requirements.

One challenge of implementing modosc has to do with missing data. When using MoCap a marker may become temporarily occluded, or a 6DoF rigid body may not be recognized, resulting in points dropping out of the data stream. If some o.dot code attempts

Figure 2: An example modosc abstraction coded using o.dot in Max which computes the centroid of a group of points.

to process data at an address that is not present in the bundle or has no data bound to it, o.dot will stop processing all data in that stream until it reappears. This would be disastrous for real-time scenarios, and would also be confusing for users of the library.

To address this, each descriptor checks that its required inputs are present. If they are not, the data stream is simply routed around the subsequent code. This is more complicated in the case of group descriptors, which must first check that the group itself has been defined, and then check that data for each point in the group is also present. Figure 2 shows an example of this process in `mo.centroid`.

3.2 Future Work

The alpha release of modosc[1] includes the components described in this paper. We plan to add new descriptors and are confident that most prospective descriptors can be implemented using o.dot. The library includes templates to help users create their own custom descriptors.

Modosc currently relies on QTM's native OSC support. Other MoCap systems can be configured to stream data via OSC by taking advantage of their respective SDKs, and new abstractions will format this data to be used in modosc.

Rotations are currently expressed in Euler angles. We plan to add support for quaternion representation, which would enable the implementation of descriptors based on rotational data [14] and also avoid issues such as gimbal lock.

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

Modosc makes extracting descriptors from multiple sources relatively easy. The same design principles can be used to include descriptors for other devices such as IMUs and EMG sensors, since both technologies are commonly used in movement analysis and interaction design.

ACKNOWLEDGMENTS

Thanks to John MacCallum and Rama Gottfriend for their invaluable assistance with using o.dot. This research was partially supported by the European Research Council (grant agreement: 725319, PI: Clemens Wöllner) for the project "Slow motion: Transformations of musical time in perception and performance".

REFERENCES

[1] Éric O Boyer, Frédéric Bevilacqua, Patrick Susini, and Sylvain Hanneton. 2017. Investigating three types of continuous auditory feedback in visuo-manual tracking. *Experimental Brain Research* 235, 3 (2017), 691–701. https://doi.org/10.1007/s00221-016-4827-x

[2] Birgitta Burger and P Toiviainen. 2013. MoCap Toolbox-A Matlab toolbox for computational analysis of movement data. *Proceedings of the Sound and Music Computing …* (2013), 172–178. https://jyx.jyu.fi/dspace/handle/123456789/42837

[3] Antonio Camurri, Barbara Mazzarino, and Gualtiero Volpe. 2004. Analysis of Expressive Gesture: The EyesWeb Expressive Gesture Processing Library. In *Gesture-based Communication in Human-Computer Interaction, LNAI 2915*. 460–467. https://doi.org/10.1007/978-3-540-24598-8_42

[4] D Fenza, L Mion, S Canazza, and A Rodà. 2005. Physical movement and musical gestures: A multilevel mapping strategy. In *Proceedings of Sound and Music Computing Conference, SMC 2005*. Salerno, Italy. http://www.scopus.com/inward/record.url?eid=2-s2.0-84905187477{&}partnerID=40{&}md5=8ecf57292b9c0a47f55b32bfb4acc4a4

[5] T Flash and N Hogan. 1985. The coordination of arm movements: an experimentally confirmed mathematical model. *The Journal of Neuroscience* 5, 7 (jul 1985), 1688–1703. https://doi.org/4020415

[6] Adrian Freed and Andrew Schmeder. 2009. Features and Future of Open Sound Control version 1.1 for NIME. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. Pittsburgh, PA, United States, 116–120. http://www.nime.org/proceedings/2009/nime2009_116.pdf

[7] D. Glowinski, N. Dael, A. Camurri, G. Volpe, M. Mortillaro, and K. Scherer. 2011. Toward a Minimal Representation of Affective Gestures. *IEEE Transactions on Affective Computing* 2, 2 (apr 2011), 106–118. https://doi.org/10.1109/T-AFFC.2011.7

[8] Caroline Larboulette and Sylvie Gibet. 2015. A review of computable expressive descriptors of human motion. In *Proceedings of the 2nd International Workshop on Movement and Computing - MOCO '15*. ACM Press, New York, New York, USA, 21–28. https://doi.org/10.1145/2790994.2790998

[9] John Maccallum, Rama Gottfried, Ilya Rostovtsev, Jean Bresson, and Adrian Freed. 2015. Dynamic Message-Oriented Middleware with Open Sound Control and Odot. In *International Computer Music Conference*. ICMA, Denton, United States. https://hal.archives-ouvertes.fr/hal-01165775/document

[10] Stefano Piana, Alessandra Staglianò, Francesca Odone, and Antonio Camurri. 2016. Adaptive Body Gesture Representation for Automatic Emotion Recognition. *ACM Transactions on Interactive Intelligent Systems* 6, 1 (mar 2016), 1–31. https://doi.org/10.1145/2818740

[11] Nicolás Salazar Sutil. 2015. *Motion and Representation: The Language of Human Movement*. MIT Press, Cambridge, MA, USA. 288 pages. https://books.google.co.uk/books/about/Motion{_}and{_}Representation.html?id=C{_}sAogEACAAJ{&}pgis=1

[12] Norbert Schnell, Diemo Schwarz, Joseph Larralde, and Riccardo Borghesi. 2017. PiPo, A Plugin Interface for Afferent Data Stream Processing Modules. In *International Symposium on Music Information Retrieval (ISMIR)*. Suzhou, China. https://hal.archives-ouvertes.fr/hal-01575288/document

[13] Ståle A Skogstad. 2013. Filtering Motion Capture Data for Real-Time Applications. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, W Yeo, K Lee, A Sigman, Ji H., and G Wakefield (Eds.). Graduate School of Culture Technology, KAIST, Daejeon, Republic of Korea, 142–147. http://www.nime.org/proceedings/2013/

[14] Federico Visi, Esther Coorevits, Rodrigo Schramm, and Eduardo Reck Miranda. 2017. Musical Instruments, Body Movement, Space, and Motion Data: Music as an Emergent Multimodal Choreography. *Human Technology* 13, 1 (may 2017), 58–81. https://doi.org/10.17011/ht/urn.201705272518

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