REFLECTIONS ON THE DEVELOPMENT OF THE MUSICAL GESTURES TOOLBOX FOR PYTHON

Bálint LACZKÓ[1] and **Alexander Refsum JENSENIUS**[1]

[1]**fourMs Lab, RITMO Centre for Interdisciplinary Studies in Rhythm, Time, and Motion, Department of Musicology, University of Oslo**,

ABSTRACT

The paper presents the Musical Gestures Toolbox (MGT) for Python, a collection of modules targeted at researchers working with video recordings. The toolbox includes video visualization techniques such as creating motion videos, motion history images, and motiongrams. These visualizations allow for studying video recordings from different temporal and spatial perspectives. The toolbox also includes basic computer vision methods, and it is designed to integrate well with audio analysis toolboxes. The MGT was initially developed to analyze music-related body motion (of musicians, dancers, and perceivers) but is equally helpful for other disciplines working with video recordings of humans, such as linguistics, pedagogy, psychology, and medicine.

1. INTRODUCTION

Over the years, we have developed various software tools and toolboxes in the fourMs Lab at the University of Oslo to analyze and visualize data from motion capture equipment or video recordings. One aim has been to create video visualization methods that can create spatial and temporal representations similar to what is possible with motion capture data. These video analysis tools started as standalone applications developed in Max [1], which were later modularized into the first Musical Gestures Toolbox (MGT) for Max [2]. These modules were again merged into the Jamoma framework [3], allowing for more complex video analysis in real-time applications [4]. This collection was later 'ported' to the Musical Gestures Toolbox for Matlab [5], which has now been reimplemented in the Musical Gestures Toolbox for Python. [1]

All the MGT versions aim to provide researchers and practitioners with a simple workflow for analyzing and visualizing human body motion from video recordings. The toolbox can be used through traditional scripts, with interactive scripting in iPython (Figure 1), or with Jupyter Notebooks. We have also been particularly interested in supporting workflows that integrate relevant audio analysis and visualization tools in the new MGT for Python.

[1] https://github.com/fourMs/MGT-python

Figure 1. Example of how MGT can be used with iPython.

The paper starts with an overview of existing Python packages for video analysis. Then we present the content of MGT for Python, discuss some use cases, and present some thoughts about future development directions.

2. BACKGROUND

There is already a plethora of Python projects related to video analysis. [2] Most of these projects focus on implementing a single or a few related functions, and they often make extensive use of deep learning architectures. Their typical functions are single or multiple object tracking, object counting, anomaly detection, video classification, and segmentation.

Our target group is music researchers with limited programming experience, in which case, pre-packaged toolboxes are preferable. Reviewing the Python Package Index (PyPi), [3] we find primarily toolboxes that implement basic video editing features, conversion between formats, or embedding them in documents. Almost all of these use Python bindings and wrappers for the versatile FFmpeg library [4] to maintain fast rendering. A few packages offer some form of video analysis, such as Video-facenet [5] or general-purpose solutions like scikit-video. [6]

Of existing toolboxes, MGT for Python most resembles scikit-video, with a collection of high-level tools for media reading and storing, motion estimation tools, and some utility functions. Scikit-video also aims to create an easy-to-use toolbox for 'students, engineers, instructors, and researchers.' However, it does not include visualization tools or workflows to organize the analyzed data. Their motion

[2] https://github.com/topics/video-analysis?l=python
[3] https://pypi.org/
[4] https://ffmpeg.org/
[5] https://pypi.org/project/video-facenet/
[6] http://www.scikit-video.org/stable/

estimation tools focus more on functions typically used in video compression algorithms, such as block motion compensation. Their measurement tools focus on video quality assessment and scene detection. These tools can be helpful for some tasks but less for our primary analysis object: human (music-related) body motion.

3. THE TOOLBOX

MGT for Python offers a suite of tools based on well-established video and audio analysis techniques, as well as an array of utilities for basic video manipulation. Additionally, it supports data visualization and data alignment.

3.1 Aims and Priorities

The design goals of MGT for Python are primarily based on the needs of our ongoing research on music-related body motion. This includes studies of both performers and perceivers. Second, we are interested in providing tools for others that use video recordings in their research. The design goals can be summarized as follows:

High-level, easy-to-use interface: Since we aim the toolbox at students and researchers, our tools should be comprehensible to someone with limited programming experience. The toolbox structure should be easy to grasp, and the nomenclature should signify the corresponding functionality.

Thorough documentation: In addition to providing complete documentation for all classes, methods and functions in the toolbox, the package should have a comprehensive set of examples and tutorials that can ease the learning curve.

Consistency: Classes and functions should work consistently. The nomenclature should follow the same logic across modules, and the toolbox should be consistent with earlier implementations.

Speed: Every tool that renders an output file (video, image, or text) should perform its job as fast as possible, given the underlying limitations of Python and the backend processes used.

Support for different workflows: Since MGT for Python is intended both for education and research, the tools should work reliably both in Jupyter notebooks and in terminal programs. Additionally, it should offer various coding styles for fast prototyping, in-depth analysis, and batch execution.

Integration of video and audio analysis: The toolbox should offer ways to align and interleave extracted time-based data with time-based audio analysis tools integrated in the package.

Compatibility: The toolbox needs to support a variety of file formats, compression standards, and durations.

Cross-platform: The toolbox should work reliably on different platforms.

Scalability: The toolbox should support multiprocessing to scale to the available resources on servers and virtual desktop infrastructures.

3.2 Backend

MGT for Python builds on four backend libraries. Most tools use FFmpeg for rendering. In order to reduce dependencies, we have implemented our suite of wrapper functions to FFplay (for windowed playback), FFprobe (for getting file information), and FFmpeg (for rendering videos and images).

While we use FFmpeg for basic video manipulation, format conversion, and rendering visualizations, OpenCV [7] is used to perform frame-by-frame-based analyses. The Matplotlib [8] library is used to visualize data and Librosa [9] is used for audio analysis. Finally, the *pose* module uses pretrained models from OpenPose. [10]

3.3 MgObject, loading and viewing videos

To work with video files in MGT for Python, we use the *MgObject* class. It is responsible for pointing to the file location, performing all preprocessing steps, or keeping tabs on alternate versions of the same file (such as .avi and .mp4). All the more advanced processing functions are methods of the *MgObject*.

To view the video of the *MgObject*, we use the *show* method. It supports both a windowed playback (via FFplay) or embedded in a Jupyter Notebook (for which it auto-converts the video to .mp4 if necessary).

The show method also has an optional *key* parameter to reference the result of a process called earlier on a source video (perhaps in a code segment that is not exposed) or to make sure we show the latest render of a process. The rest of the *MgObject* attributes are there to apply some preprocesses to the source video.

3.4 Preprocesses

When a video is loaded into an *MgObject*, it is also possible to apply some basic processes that prepare it for analysis. There are six types of preprocessing steps (listed in order of execution):

trim: Trim the start and stop of the source video.

skip: Skip every *n* frames.

rotate: Rotate the video by an angle.

cb: Adjust the contrast and brightness of the video.

crop: Crop the spatial dimensions of the video either with automatic detection of the area of motion or manually through a graphical user interface.

grayscale: Convert the video to grayscale. This will cause all processes called on the MgObject to function in grayscale mode, improving processing speed.

[7] https://github.com/opencv/opencv-python
[8] https://matplotlib.org/
[9] https://librosa.org/
[10] https://github.com/CMU-Perceptual-Computing-Lab/openpose

All of these processes use the FFmpeg backend. Specifying all attributes for the preprocesses results in a single video file by default. It is also possible to keep the results of all steps as separate videos using the *keep_all* attribute.

3.5 Video-based processes

The tools to analyze and visualize videos include:

motion: The most frequently used function that generates a motion video, horizontal and vertical motiongrams (Figure 2) and plots of the centroid and quantity of motion found in the video.

motionvideo: A shortcut to only render the motion video.

motiongrams: A shortcut to only output the motiongrams.

motiondata: A shortcut to only output the motion data (time, centroid and quantity of motion for each video frame) as a CSV file.

motionplots: A shortcut to only output the motion plots (centroid and quantity of motion).

videograms: A visualization that resembles motiongrams, but based on the original video source.

history: Renders a history video by layering the last *n* frames on the current frame for each frame in the video (Figure 3).

average: Renders an average image of all video frames.

flow.sparse: Renders a sparse optical flow video (using the OpenCV implementation) (Figure 4).

flow.dense: Renders a dense optical flow video (using the OpenCV implementation).

pose: Renders a video with human pose estimation (using pre-trained models from OpenPose) and optionally outputs the pose data as a CSV file.

The above tools are, in fact, all methods of the *MgObject* class. The usual workflow with MGT for Python is to (1) load a video file into an *MgObject*, (2) optionally apply some preprocessing, (3) apply an analysis/visualization process on the video by calling some method on the *MgObject*, (4) use the process results, such as viewing the rendered video or image, plotting the analysis, or reusing the result in another process.

3.6 Audio-based processes

MGT for Python offers several tools to analyze the audio track of video files. These are implemented both as class methods for *MgObject* and as standalone functions. These tools are based on the *librosa* audio analysis package and the *matplotlib* package for showing composite figures. To make working with these figures simpler and more flexible, we use the *MgFigure* class as a data structure.

The list of audio-based processes is the following:

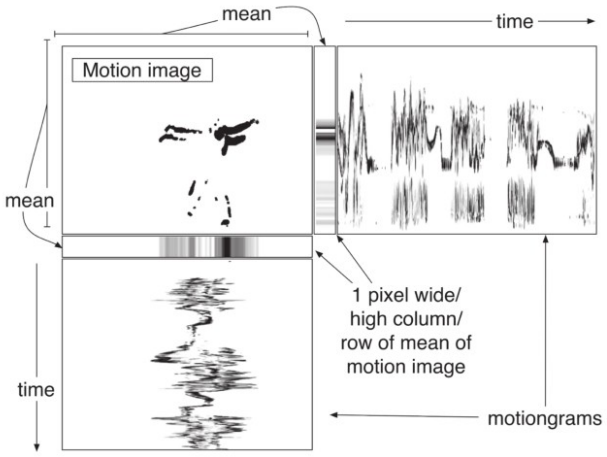

Figure 2. Sketch of the calculation of motiongrams from a motion image.

Figure 3. motion visualization with video delay.

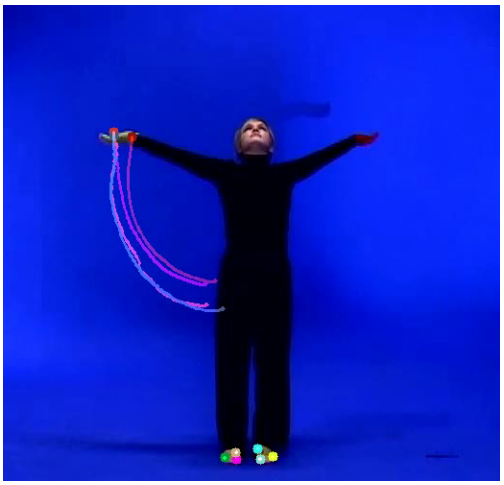

Figure 4. motion visualization with sparse optical flow.

audio.waveform (or **mg_audio_waveform**): Renders a waveform plot of the audio track/file.

audio.spectrogram (or **mg_audio_spectrogram**): Renders a spectrogram of the audio track/file.

audio.tempogram (or **mg_audio_tempogram**): Renders plots of onset strength and tempogram (based on the former).

audio.descriptors (or **mg_audio_descriptors**): Renders plots of: RMS energy, spectral flatness, spectral centroid, spectral bandwidth, and spectral rolloff of the audio track/file.

3.7 Figures, images, lists

When working with video files in MGT for Python, we almost always use *MgObjects* to preprocess the videos via the objects' attributes and apply other processes via class methods. There are similar helper classes for working with matplotlib figures and image files. We use the *MgFigure* class to make matplotlib figures reusable and modular, and *MgImage* serves a similar purpose for images. We also have our extended list implementation with *MgList* that replicates the same functionality as standard Python lists, with an additional method *as_figure*. It allows us to compose a stack of time-aligned plots in a specific order.

The workflow will typically to start by creating the *MgFigures* and *MgImages* (or *MgLists* of these) to stack together. Next, we gather them into an *MgList* before calling *as_figure*. The first element in the *Mglist* will correspond to the top figure in the stack and the last element to the bottom figure. See Figure 5 for an example.

4. USE CASES

In this section, we will describe typical use cases of MGT for Python. The toolbox currently focuses on three types of tasks: (1) extracting motion data, (2) motion visualization, (3) aligning audio and video data and visualizations.

4.1 Extracting and analyzing motion data from videos

There are many methods for analyzing human music-related body motion [6]. Of all these, regular video recordings may arguably be considered the cheapest and most versatile. Most smartphones can provide a video quality that is sufficient for motion extraction. The challenge is how to visualize and analyze motion from the video files. That is the core functionality of MGT. We often work with motion capture systems in our labs. Even though marker-based or sensor-based motion capture systems provide high spatial and temporal accuracy and precision, they do not capture the context in the same way as a video recording. MGT for Python gives researchers a tool for analyzing motion data from video files with just a few lines of code.

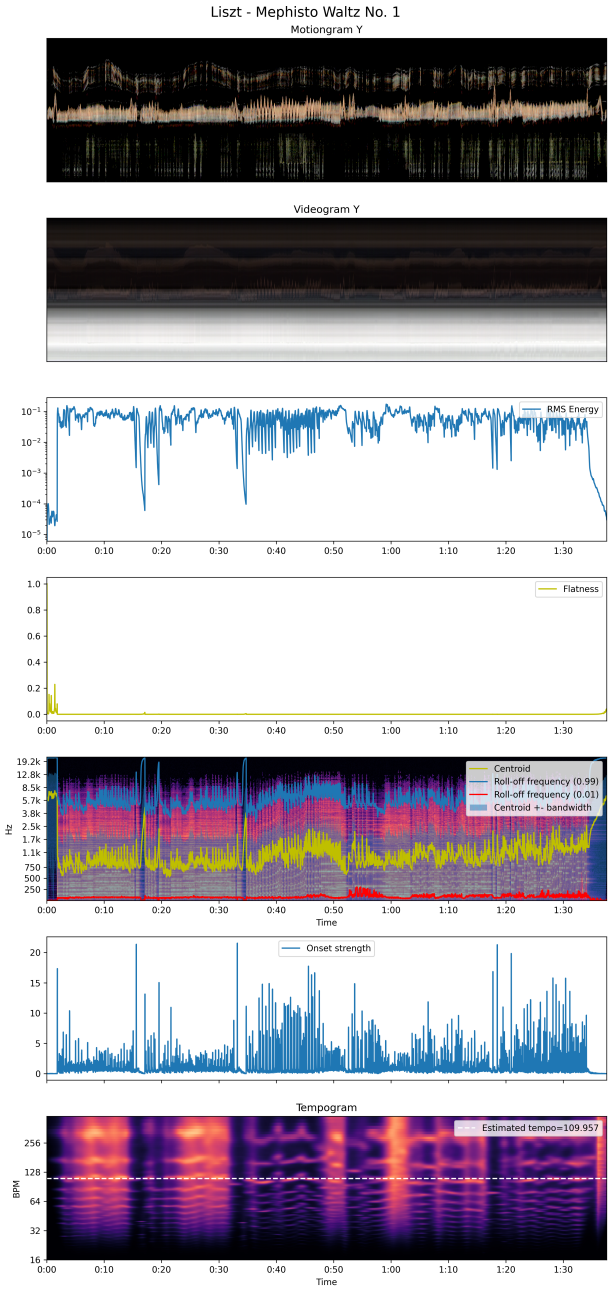

Figure 5. Time-aligned audiovisual analysis. From top to bottom: Vertical motiongram, vertical videogram, RMS energy, spectral descriptors, onset strength, tempogram.

4.2 Visualizing motion capture data with images

MGT offers a range of motion visualization tools. We can isolate motion from a static background, visualize the direction each body part is moving, emphasize its speed and trajectory, or summarize the area or evolution in a single image. Such visualizations often need lengthy boilerplate code to set up. MGT for Python aims to offer fast-rendering visualization tools as simple as a function call.

4.3 Aligning video and audio analysis

In addition to the video-based analysis and visualizations, we also want to offer simple ways to align motion and audio data. Currently, we support this in the form of stacking plots and images in a time-aligned fashion. These composite plots can inform the correlations between onsets, rhythm, loudness, or other spectral characteristics and motion speed, area, or trajectory.

5. CONCLUSIONS AND FUTURE WORK

The Musical Gestures Toolbox for Python offers simple ways to extract, analyze and visualize motion, and its relation to musical sound. Our ultimate goal with MGT is to offer an object-oriented way to integrate audio, video, and motion capture data.

MGT is designed to offer a simple, flexible, and fast workflow for researchers, teachers, or students. So far, we have focused on creating a set of video analysis tools with a handful of audio-related functions. At present, we have worked on integrations with librosa [7]. In the future, it would also be relevant to include support for other audio libraries, including Madmom [8] and Essentia [9]. Another direction is to include analysis and visualization functions aimed at motion capture data. Here we will build on some of the solutions developed in the MGT for Matlab, which includes support for combining audio analysis with MIRToolbox [10] and MoCap Toolbox [11].

Another future goal of the Python implementation of MGT is to make the toolbox scalable so that it can run on servers and distributed computing systems. FFmpeg (our main rendering backend) already supports multi-core execution. However, the OpenCV backend functions are looping inside Python, which is inherently single-threaded and thus inefficient. We have already made an experimental multithreaded version of the *motion* function in the latest release. In the future, we want to redesign all OpenCV-based rendering loops to support multithreaded workflows.

As the toolbox grows, we also want to build a test suite with complete code coverage to maintain stability throughout development. Finally, since MGT for Python aims towards research and education, we also plan to continuously extend the toolbox's feature set based on user feedback.

Acknowledgments

Thanks to Frida Furmyr and Marcus Widmer, who developed the first version of MGT for Python, and Bo Zhou, who co-developed MGT for Matlab, which the Python toolbox builds on. This work was partially supported by the Research Council of Norway through its Centres of Excellence scheme, project number 262762 and by NordForsk's Nordic Sound and Music Computing Network (NordicSMC), project number 86892

6. REFERENCES

[1] A. R. Jensenius, R. I. Godøy, and M. M. Wanderley, "Developing tools for studying musical gestures within the Max/MSP/Jitter environment," in *Proceedings of the International Computer Music Conference*, 2005, pp. 282–285.

[2] A. R. Jensenius, "Action–Sound: Developing Methods and Tools to Study Music-Related Body Movement," PhD Thesis, University of Oslo, 2007.

[3] T. Place and T. Lossius, "Jamoma - A modular standard for structuring patches in Max," in *Proceedings of the International Computer Music Conference*, 2006, pp. 143–146.

[4] T. Place, T. Lossius, A. R. Jensenius, and N. Peters, "Flexible Control of Composite Parameters in Max/Msp," in *Proceedings of the International Computer Music Conference*, Belfast, 2008, pp. 233–236.

[5] B. Zhou, "Video Analysis of Music Related Body Motion in Matlab," Master's Thesis, University of Oslo, 2016.

[6] A. R. Jensenius, "Methods for studying music-related body motion," in *Handbook of Systematic Musicology*, R. Bader, Ed. Berlin Heidelberg: Springer-Verlag, 2018, pp. 567–580.

[7] B. McFee, C. Raffel, D. Liang, D. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "Librosa: Audio and Music Signal Analysis in Python," in *Proceedings of the International Python in Science Conference*, Austin, Texas, 2015, pp. 18–24.

[8] S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer, "Madmom: A new python audio and music signal processing library," in *Proceedings of the 24th ACM International Conference on Multimedia*, 2016, pp. 1174–1178.

[9] D. Bogdanov, N. Wack, E. Gómez Gutiérrez, S. Gulati, H. Boyer, O. Mayor, G. Roma Trepat, J. Salamon, J. R. Zapata González, and X. Serra, "Essentia: An open source library for audio analysis," *ACM SIGMM Records. 2014; 6 (1): 18-21.*, 2014.

[10] O. Lartillot and P. Toiviainen, "A Matlab toolbox for musical feature extraction from audio," in *International Conference on Digital Audio Effects*, 2007, pp. 237–244.

[11] B. Burger and P. Toiviainen, "MoCap Toolbox - A Matlab toolbox for computational analysis of movement data," in *Proceedings of the Sound and Music Computing Conference*, 2013, pp. 172–178.