SkelMotion: A simple Python implementation of video motion visuzliation for 3D motion capture data

Derek Harter
Texas A&M University - Commerce

October 29, 2022

Abstract

We describe a simple library of Python/NumPy/Matplotlib code for generating animations of simple 3D motion capture data points. The motion capture method is agnostic with respect to this visualizaiton tool, expecting a time series of time stamped coordinates captured in a 3-dimensional space. This tool was developed originally for a simple Kinect system implementing skeleton tracking of 15 joint positions, where each data point consists of an accurate time of capture, and 3 accurate coordinates, (x, y, z) position of each of the 15 joints at each time step. This library can be extended to vizualize and create animations of any such motion capture data that can be reformulated using this basic structure of a time series of N 3D points, given this time series and a matrix describing the relationship of the N points to one another in space for the visuzliation. This library should be useful as is, or with easy modification, for many such visualization requirements of similar motion capture data. This library is available at: https://github.com/DerekHarter/kinect-skeleton-tracking

Keywords: motion tracking, skeleton tracking, scientific visualization

Contents

List of Figures

List of Tables

1 Introduction

Creating animated visualization in the base Python 3.X scientific python stack is certainly much simpler than it has been in the past, but still requires many specialized steps to achieve the task of creating an animation that can be saved as a movie for presentation. The standard Matplotlib animation library documentation (Matplotlib 3.6.0 Stable Release Documentation, 2022b) contains several examples of generated animations, but only 1 of animating a 3D set of data (Matplotlib 3.6.0 Stable Release Documentation, 2022a). The basic technique to use an instance of `matplotlib.animation` to generate a sequence of frames from a set of 3D points is clear from the documentation, but can certainly be difficult to generalize for much more complex data than shownin the basic example.

3D motion capture data are a very useful and widespread type of data that we often need to visualize. The goal of this small library is to simplify the task of creating animated visuzliations of such 3D time series data. The library presented here can be generalized to work with a time series of N 3D points in space to create an animation of subsequent frames of the captured points evolving over time. The only additional general information needed is a matrix of the relationship of the N points in space that should be connected as being related points in the resulting animation.

We used motion capture from a very basic Kinect skeleton tracking experiment, using the NiTE 2.0 library (Nuitrack, 2022; PrimeSense, 2022) which includes skeletron tracking capture in the development of this visualizaiton library. However the general format, described in methods, can be extended to many kinds of motion capture devices where a timestape and a set of 3D point positions can be obtained for the captured object in motion. In the example implementation discussed, we have $N = 15$ joint positions (e.g. head, neck, left shoulder, etc.) captured by the motion tracking device. An example of a rendered frame of the motion capture animation is shown in Figure 1. Sample videos of the resulting animation are available in the repository X.

2 Methods

2.1 Implementation

The version of the SkelMotion library described here was developed on a Python 3.9 scientific tool stack environment with the following versions of the important libraries:

```
from platform import python_version
print(python_version())
> 3.9.7

import numpy as np
print(np.__version__)
> 1.20.3

import matplotlib as mpl
print(mpl.__version__)
> 3.4.3
```

We have 15 joint positions in the example implementation used in developing the library. Each joint position is captured approximately 30 times per second by the Kinect sensor programmed with the NiTE 2.0 skeleton tracking library software. So while the library was generated specifically with 15

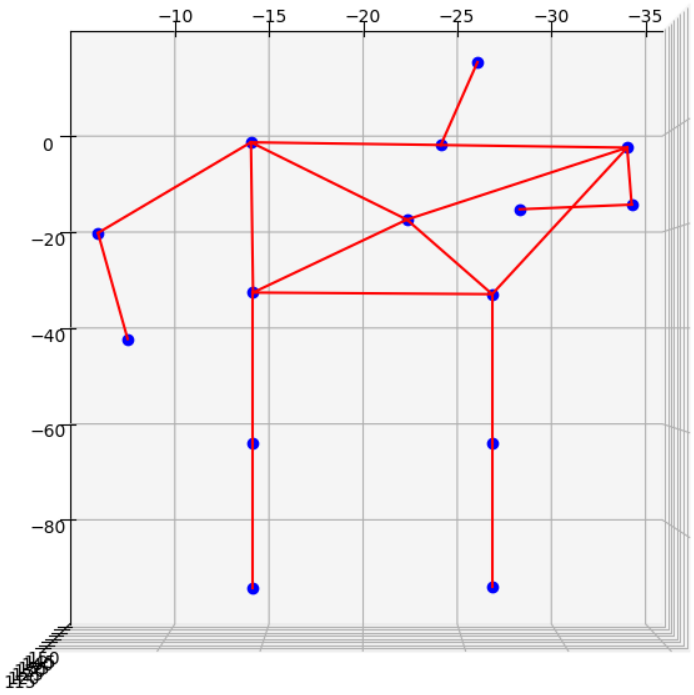

Figure 1: Example of a rendered frame of 3D motion capture joint positions

3D motion capture points, the general format of the input data needed to render a 3D animation is

```
timeStamp, joint0X, joint0Y, joint0Z,
           joint1X, joint1Y, joint1Z,
           ...,
           jointNX, jointNY, jointNZ
```

as a standard comma separated value data file, where N can be specified when rendering a motion capture animation of how many joint/position points are in the input. The only other information needed by the library is a connection graph of the points if we want to visualize the relationship of the rendered points to one another while being animated. Again using the Kinect 15 joint position data, we specify a joint graph for the software like this:

```
# the relationship of the joint points to assigned numeric position.
# the numeric id corresponds to the expected position in the input
file
joint_names = [
    'Head', 'Neck', 'Torso',            # 0 1 2
    'LeftShoulder', 'RightShoulder',    # 3 4
    'LeftElbow', 'RightElbow',          # 5 6
    'LeftHand', 'RightHand',            # 7 8
    'LeftHip', 'RightHip',              # 9 10
    'LeftKnee', 'RightKnee',            # 11 12
    'LeftFoot', 'RightFoot'             # 13 14
]

# joint position graph for visualizing relationship between joints
joint_graph = [
    (0, 1), (1, 3), (1, 4),          # head to neck, neck to shoulders
    (3, 5), (4, 6),                  # shoulders to elbows
    (5, 7), (6, 8),                  # elbows to hands
    (3, 2), (4, 2), (9, 2), (10, 2), # shoulders and hips to torso
    (3, 9), (4, 10), (9, 10),        # shoulders to hips, connect hips
    (9, 11), (10, 12),               # hips to knees
    (11, 13), (12, 14),              # knees to feet
]
```

Figure 2: Example of capture point labeling and position graph

2.2 Operation

Install the SkelMotion library from standard PyPy python library.

```
$ pip install skelmotion
```

The library expects a 2D array of values, where the first column is a time stamp. Supsequent columns are the data for motion capture point 0 (x, y, z), motion capture point 1, etc. So given 15 motion capture points, this library expects and array with 46 columns, where the first column is a time stamp, and the subsequent columns are the 3D positions for all points captured. This could be loaded from a csv file, or generated from some other source.

The second parameter corresponds to the `joint_graph` shown above. This is expected to be a python array of tuples, indicating the point graph relationship between captured 3D points. The

default behavior is to return a created object of type `matplotlib.animation.FuncAnimation()` of the completed animation. This can be rendered or saved to a movie.

```
import skelmotion

# joint_position and joint_graph defined as above
element_animation =
skelmotion.moition_capture_animation(joint_positions, joint_graph)

# save resulting animation as a movie
file_name = "joint_animation.mov"
element_animation.save(file_name)
```

3 Use Case

Use case section goes here.

4 Limitations

Relies on current version of Python and matplotlib animation library. Data is expected in the needed format, which may not always be simple to massage into depending on the motion capture device output.

5 Conclusion

This work demonstrates a simple library to visualize 3D motion capture data movement over time. The library expectes data as a simple time series of N captured 3D point positions. A basic animation is created using the Matlplotlib `animation` library and 3D Axes objects as well. Results can be saved as a movie file animation for analysis and presentation.

6 Softeare Availability

Software should be available for Python 3.9 environments with a basic scientific tool stack from standard PyPy resource using `pip install`. Software can also be viewed and downloaded directly from this repository:

- https://github.com/DerekHarter/kinect-skeleton-tracking

7 Acknowledgements

This research supported by Army Research Lab grants. Thanks to Jonathan Z. Bakdash and Laur R. Marusich for input and feedback on this codebase.

References

Matplotlib 3.6.0 Stable Release Documentation. (2022a). *Animated 3d random walk*. Retrieved October 29, 2022, from https://matplotlib.org/3.6.0/gallery/animation/random_walk.html

Matplotlib 3.6.0 Stable Release Documentation. (2022b). *Matplotlib.animation library*. Retrieved October 29, 2022, from https://matplotlib.org/3.6.0/api/animation_api.html

Nuitrack. (2022). *Nuitrack sdk: 3d body (skeletal) tracking middleware*. Retrieved October 29, 2022, from https://nuitrack.com/

PrimeSense. (2022). *Nite 2.0 library documentation*. Retrieved October 29, 2022, from https://documentation.help/NiTE-2.0/documentation.pdf