audio_synch_tool Documentation

Release 1.3.0

Andres FR

Jun 12, 2019

CONTENTS:

CHAPTER

ONE

AUDIO_SYNCH_TOOL PACKAGE

1.1 Submodules

1.2 audio_synch_tool.mmc module

Warning: Since BSON can be bigger than JSON (see https://stackoverflow.com/a/24116464) this module is abandoned until a better solution comes up.

The goal of this module is to provide an interface between MVNX and MMC, both in JSON and BSON format.

1. Have a proper definition of MMC
2. load MVNX into an XML tree
3. convert logical data into MMC
4. export/import MMC both as JSON and BSON
5. convert logical data back into XML tree

class audio_synch_tool.mmc.**ObjectifiedMvnToJsonEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

This encoder handles the conversion from an MVNX object that follows our modified schema and has been imported via lxml.objectify, into a JSON object. It can be passed to json.dumps to create a JSON string that can be then exported as plain text or BSON.

ALL_ARRAYS_AS_FLOAT = True

EXPECTED_TYPES = [<class 'lxml.etree._Attrib'>, <class 'lxml.objectify.StringElement'>

default(*o*)

This function doesn't need to be recursive, that is handled by the encoder itself.

classmethod process_attrib(*atts*)

classmethod process_string(*s*)

audio_synch_tool.mmc.**mvn_to_json**(*mvn_in_path*, *json_out_path*, *validate_mvn=False*, *write_as_binary=True*)

audio_synch_tool.mmc.**test**(*mvn_path*, *mvn_schema_path*, *out_path='/tmp/test.json'*, *as_bson=False*)

1.3 audio_synch_tool.mvn module

This module contains functionality concerning the adaption of the XSENS MVN-XML format into our Python setup.

The official explanation can be found in section 14.4 of this document:: https://usermanual.wiki/Document/MVNUserManual.1147412416.pdf

A copy is stored in this repository.

The following section introduces more informally the contents of the imported MVN file and the way they can be accessed from Python:

```
# load mvn schema https://www.xsens.com/mvn/mvnx/schema.xsd
mvn_path = "XXX"
mmvn = Mvn(mvn_path, validate=True)

# These elements contain some small metadata:
mmvn.mvn.attrib
mmvn.mvn.comment.attrib
mmvn.mvn.securityCode.attrib["code"]
mmvn.mvn.subject.attrib

# subject.segments contain 3D pos_b labels:
for ch in mmvn.mvn.subject.segments.iterchildren():
    ch.attrib, [p.attrib for p in ch.points.iterchildren()]

# Segments can look as follows: ``['Pelvis', 'L5', 'L3', 'T12', 'T8', 'Neck',
'Head', 'RightShoulder', 'RightUpperArm', 'RightForeArm', 'RightHand',
'LeftShoulder', 'LeftUpperArm', 'LeftForeArm', 'LeftHand', 'RightUpperLeg',
'RightLowerLeg', 'RightFoot', 'RightToe', 'LeftUpperLeg', 'LeftLowerLeg',
'LeftFoot', 'LeftToe']``

# sensors is basically a list of names
for s in mmvn.mvn.subject.sensors.iterchildren():
    s.attrib

#  Joints is a list that connects segment points:
for j in mmvn.mvn.subject.joints.iterchildren():
    j.attrib["label"], j.getchildren()

# miscellaneous:
for j in mmvn.mvn.subject.ergonomicJointAngles.iterchildren():
    j.attrib, j.getchildren()

for f in mmvn.mvn.subject.footContactDefinition.iterchildren():
    f.attrib, f.getchildren()

# The bulk of the data is in the frames.
print("frames_metadata:", mmvn.frames_metadata)
print("first_frame_type:", mmvn.config_frames[0]["type"])
print("normal_frames_type:", mmvn.normal_frames[0]["type"])
```

(continues on next page)

(continued from previous page)

```
print("num_normal_frames:", len(mmvn.normal_frames))

# Metadata looks like this:
{'segmentCount': '23', 'sensorCount': '17', 'jointCount': '22'}

# config frames have the following fields:
['orientation', 'position', 'time', 'tc', 'ms', 'type']

# normal frames have the following fields:
['orientation', 'position', 'velocity', 'acceleration',
 'angularVelocity', 'angularAcceleration', 'footContacts',
 'sensorFreeAcceleration', 'sensorMagneticField', 'sensorOrientation',
 'jointAngle', 'jointAngleXZY', 'jointAngleErgo', 'centerOfMass', 'time',
 'index', 'tc', 'ms', 'type']
```

The following fields contain metadata about the frame:

time ms since start (integer). It is close to `int(1000.0 * index / samplerate)`, being equal most of the times and at most 1 milisecond away. It is neither truncated nor rounded, maybe it is given by the hardware.

index starts with 0, +1 each normal frame

tc string like '02:23:28:164'

ms unix timestamp like 1515983008686 (used to compute time)

type one of "identity", "tpose", "tpose-isb", "normal"

The following fields are float vectors of the following dimensionality:

orientation `segmentCount*4 = 92` Quaternion vector

position, velocity, acceleration, angularVelocity, angularAcceleration `segmentCount*3 = 69` 3D vectors in `(x,y,z)` format

footContacts `4` 4D boolean vector

sensorFreeAcceleration, sensorMagneticField `sensorCount*3 = 51`

sensorOrientation `sensorCount*4 = 68`

jointAngle, jointAngleXZY `jointCount*3 = 66`

jointAngleErgo `12`

centerOfMass `3`

The units are SI for position, velocity and acceleration. Angular magnitudes are in radians except the `jointAngle...` ones that are in degrees. All 3D vectors are in `(x,y,z)` format, but the `jointAngle...` ones differ in the Euler-rotation order by which they are computed (ZXY, standard or XZY, for shoulders usually).

class audio_synch_tool.mvn.**Mvn**(*mvn_path*, *validate=False*)

Bases: object

This class imports and adapts an XML file (expected to be in MVN format) to a Python-friendly representation. See this module's docstring for usage examples and more information.

MVNX_SCHEMA_PATH = '/home/a9fb1e/github-work/audio-synch-tool/audio_synch_tool/data/mv

export(*filepath*, *pretty_print=True*, *extra_comment=''*)

Saves the current `mvn` attribute to the given file path as XML and adds the `self.mvn.attrib["pythonComment"]` attribute with a timestamp.

extract_frame_info()

Returns The tuple `(frames_metadata, config_frames, normal_frames)`

static extract_frames(*mvn*)

The bulk of the MVN file is the `mvn->subject->frames` section. This function parses it and returns its information in a python-friendly format.

Parameters **mvn** – An XML tree, expected to be in MVN format

Returns a tuple `(frames_metadata, config_frames, normal_frames)` where the metadata is a dict in the form `{'segmentCount': '23', 'sensorCount': '17', 'jointCount': '22'}`, the config frames are the first 3 frame entries (expected to contain special config info) and the normal_frames are all frames starting from the 4th. Both frame outputs are relational collections of dictionaries that can be formatted into tabular form.

static extract_normalframe_magnitudes(*normal_frames*)

Returns A list of the magnitude names in each of the `self.normal_frames`

extract_normalframe_sequences(*frames_metadata*, *normal_frames*, *device='cpu'*)

Returns a dict with torch tensors of shape `(num_normalframes, num_channels)` where the number of channels is e.g. 1 for scalar magnitudes, 3 for 3D vectors (in xyz format)...

extract_segments()

Returns A list of the segment names in `self.mvn.subject.segments`, ordered by id (starting at 1 and incrementing +1).

get_audio_synch()

Returns a list with the `audio_sample` attributes for the normal frames, or None if there is at least 1 normal frame without the `audio_sample` attribute. Note that this function assumes that the entries are integers in the form '123', so they are retrieved int(x). If that is not the case, they may be truncated or even throw an exception.

set_audio_synch(*stretch*, *shift*)

Given the list of normal frames in this Mvn, each one with an "index" field, this function adds an `audio_sample` attribute to each frame, where `audio_sample = round(index * stretch + shift)`. See `utils.convert_anchors` for converting anchor points into stretch and shift.

1.4 audio_synch_tool.plotters module

class audio_synch_tool.plotters.**AudioMvnSynchToolChecker**(*audio_array*, *audio_samplerate*, *mvn*, *max_datapoints=10000*)

Bases: *audio_synch_tool.plotters.MultipleDownsampledPlotter1D*

This doesn't modify the synch, just displays it

make_fig()

Parameters **toolbar_widgets** – A list of class names of type SignalTransformButtons. Must include a `fig` attribute, which will be automatically set to be the Figure returned by this method.

Returns A matplotlib Figure containing the array given at construction. The interactive plot of the figure will react to the user's zooming by showing approximately `self.max_datapoints` number of samples.

class audio_synch_tool.plotters.**AudioMvnSynchToolEditor**(*wav_path*, *mvnx_path*, *validate_mvnx=False*, *max_datapoints=10000*)

Bases: *audio_synch_tool.plotters.MultipleDownsampledPlotter1D*

TEXTBOX_WIDGET_CLASSES = [<class 'audio_synch_tool.plotters.NumberPromptOri1'>, <class

TOOLBAR_BUTTON_CLASSES = [<class 'audio_synch_tool.plotters.SynchAndSaveMvnButton'>]

make_fig()

Parameters **toolbar_widgets** – A list of class names of type SignalTransformButtons. Must include a `fig` attribute, which will be automatically set to be the Figure returned by this method.

Returns A matplotlib Figure containing the array given at construction. The interactive plot of the figure will react to the user's zooming by showing approximately `self.max_datapoints` number of samples.

class audio_synch_tool.plotters.**MultipleDownsampledPlotter1D**(*y_arrays*, *samplerates=None*, *max_datapoints=10000*, *shared_plots=None*, *x_arrays=None*, *xtick_formatters=None*)

Bases: `object`

This class generates a matplotlib figure that plots several 1-dimensional arrays. Since some arrays can be quite long, it features a built-in downsampling mechanism that plots a (configurable) number of points at most. The downsampling mechanism is based on this code:

https://matplotlib.org/3.1.0/gallery/event_handling/resample.html

---

Note: The downsampling mechanism only affects the display, not the data, and is refreshed every time the user changes the perspective. If the user zooms close enough the data will be displayed in its original form. Therefore downsampling helps to provide a quicker interaction when scrolling large files, while allowing for sample-precise inspection.

FIG_ASPECT_RATIO = (10, 8)

FIG_MARGINS = {'bottom': 0.1, 'hspace': 0.5, 'left': 0.1, 'right': 0.95, 'top': 0

NUM_DECIMALS = 3

NUM_XTICKS = 15

NUM_YTICKS = 10

SHOW_IDX = True

TEXTBOX_HEIGHT_RATIO = 0.22

TICK_FONTSIZE = 7

TICK_ROT_DEG = 15

make_fig(*textbox_widgets=[]*, *toolbar_widgets=[]*)

Parameters **toolbar_widgets** – A list of class names of type SignalTransformButtons. Must include a `fig` attribute, which will be automatically set to be the Figure returned by this method.

Returns A matplotlib Figure containing the array given at construction. The interactive plot of the figure will react to the user's zooming by showing approximately `self.max_datapoints` number of samples.

class audio_synch_tool.plotters.**NumberPrompt**(*axis*)
Bases: matplotlib.widgets.TextBox

AX_TITLE = ''

INITIAL_VAL = ''

LABEL = ''

NAME = ''

PADDING = 0.1

SIZE_PERCENT = '200%'

class audio_synch_tool.plotters.**NumberPromptDest1**(*axis*)
Bases: *audio_synch_tool.plotters.NumberPrompt*

AX_TITLE = 'Destiny 1'

NAME = 'Dest 1'

class audio_synch_tool.plotters.**NumberPromptDest2**(*axis*)
Bases: *audio_synch_tool.plotters.NumberPrompt*

AX_TITLE = 'Destiny 2'

NAME = 'Dest 2'

class audio_synch_tool.plotters.**NumberPromptOri1**(*axis*)
Bases: *audio_synch_tool.plotters.NumberPrompt*

AX_TITLE = 'Origin 1'

NAME = 'Ori 1'

class audio_synch_tool.plotters.**NumberPromptOri2**(*axis*)
Bases: *audio_synch_tool.plotters.NumberPrompt*

AX_TITLE = 'Origin 2'

NAME = 'Ori 2'

class audio_synch_tool.plotters.**SignalTransformButtons**(*toolmanager*, *name*)
Bases: matplotlib.backend_tools.ToolBase

The class attribute `plotter` has to be set to point to an instance of `plt.Figure` at some point before usage.

fig = None

tool_group = 'signal_transforms'

class audio_synch_tool.plotters.**SynchAndSaveMvnButton**(*toolmanager*, *name*)
Bases: *audio_synch_tool.plotters.SignalTransformButtons*

Given the current anchors, set the audio_synch info into the MVN and save it to the given path. ..note:

```
This assumes the following textboxes ordering!:
``[ori1, dest1, ori2, dest2, outpath]``
```

default_keymap = ''

description = 'Given the current anchors, set the audio_synch info into the MVN and sa

image = '/home/a9fb1e/github-work/audio-synch-tool/audio_synch_tool/data/synch_and_sav

name = 'synch_and_save_mvn'

trigger(**args*, ***kwargs*)

class audio_synch_tool.plotters.**TextPromptOutPath**(*axis*)

Bases: matplotlib.widgets.TextBox

AX_TITLE = 'Synched MVNX Out Path:'

INITIAL_VAL = '/home/a9fb1e'

LABEL = ''

NAME = 'OutPath'

PADDING = 0.5

SIZE_PERCENT = '500%'

audio_synch_tool.plotters.**add_to_toolbar**(*fig*, **widgets*)

Adds a set of button-like widgets to the toolbar of a given figure.

Parameters

- **fig** – a plt.Figure
- **widgets** – A number of class names that extend mpl.backend_tools.ToolBase. They must have at least defind the name and tool_group fields.

audio_synch_tool.plotters.**redefine_plt_shortcuts**(*fig*, *help_keys=['f1']*, *save_keys=['ctrl+s']*)

Some of them collide, some of them are unwanted. This function redefines most of the matplotlib.rcParams entries.

1.5 audio_synch_tool.utils module

class audio_synch_tool.utils.**DownsamplableFunction**(*y_arr*, *max_datapoints*, *x_arr=None*)

Bases: object

Encapsulates the downsampling functionality to prevent side effects, and reduce code verbosity in plotter.

downsampled(*xstart*, *xend*, *verbose=False*)

This function performs downsampling by reading one sample every (xend-xstart)// max_datapoints from x_vals and y_vals.

class audio_synch_tool.utils.**IdentityFormatter**

Bases: object

This functor can be passed to plt.FuncFormatter to generate custom tick labels. It fulfills the interface (val, pos)->str.

Specifically, for a given val returns str(val). In most cases this is the default matplotlib behaviour, but using this formatter forces it to behave ALWAYS like this and avoid some other smart conversions like scientific notation for big numbers.

class audio_synch_tool.utils.**ProportionalFormatter**(*ratio*, *num_decimals=3*)
Bases: object

This functor can be passed to `plt.FuncFormatter` to generate custom tick labels. It fulfills the interface (val, pos)->str.

Specifically, converts `val` number representing a sample into a string in the form `val [val2]` where `val2 = val * ratio`. This can be useful e.g. to show the original values if the signal was resampled.

class audio_synch_tool.utils.**SampleToTimestampFormatter**(*samplerate*, *num_decimals=3*, *show_idx=True*)
Bases: object

This functor can be passed to `plt.FuncFormatter` to generate custom tick labels. It fulfills the interface (val, pos)->str.

Specifically, converts `val` number representing a sample into a string showing the corresponding elapsed time since sample 0, asuming the given samplerate. Usage example:

```
ax.xaxis.set_major_formatter(plt.FuncFormatter(
                             SampleToTimestampFormatter(sr)))
```

class audio_synch_tool.utils.**SharedXlimCallbackFunctor**(*functors*)
Bases: object

If multiple axes share the x-limits, calling the functor from one of them triggers a call for every one of them.

class audio_synch_tool.utils.**SynchedMvnFormatter**(*mvn*, *num_decimals=3*)
Bases: object

This functor can be passed to `plt.FuncFormatter` to generate custom tick labels. It fulfills the interface (val, pos)->str.

Specifically, it looks in the MVN file for the frame index with the given `val` and, if found, returns the string `val [frame_idx]` (otherwise just val). For that, it expects that the frame has defined the `audio_sample` attribute.

class audio_synch_tool.utils.**Timedelta**(*sample_nr*, *samplerate*)
Bases: object

as_tuple()

Returns the tuple of integers (days, hours, mins, secs, microsecs)

days

hours

microsecs

mins

sample_nr

samplerate

secs

total_seconds

class audio_synch_tool.utils.**XlimCallbackFunctor**(*axis*, *lines*, *arrays*, *shared_axes*, *verbose=False*)
Bases: object

Encapsulates the downsampling callback functionality to prevent side effects. Instances of this functor can be passed to an axis like this:

```
``ax.callbacks.connect('xlim_changed',
                DownsamplingCallbackFunctor(ax, [line1..], [arr1...]))``
```

audio_synch_tool.utils.**convert_anchors**(*ori1*, *dest1*, *ori2*, *dest2*)

Given a signal that we want to shift and stretch on the x axis, this affine operation can be defined by picking 2 points of the signal (ori1 and ori2, called here "anchors"), and mapping them to other 2 points (dest1 and dest2). For the given anchors, this function returns the corresponding stretching ratio and shifting amount (after stretching) needed to match the given anchors. This is given by solving the formula:

```
``[dest1, dest2] = shift + stretch * [ori1, ori2]``
```

Parameters **ori1** (*number*) – any real-valued number. Same for ori2, dest1, dest2.

Returns a tuple (stretch, shift) with 2 real-valued numbers.

audio_synch_tool.utils.**make_timestamp**(*timezone='Europe/Berlin'*)

Output example: day, month, year, hour, min, sec, milisecs: 10_Feb_2018_20:10:16.151

audio_synch_tool.utils.**resolve_path**(**path_elements*)

A convenience path wrapper to find elements in this package. Retrieves the absolute path, given the OS-agnostic path relative to the package root path (by bysically joining the path elements via `os.path.join`). E.g., the following call retrieves the absolute path for `<PACKAGE_ROOT>/a/b/test.txt`:

```
resolve_path("a", "b", "test.txt")
```

Params strings path_elements From left to right, the path nodes, the last one being the filename.

Return type str

audio_synch_tool.utils.**tensor_has_all_integers**(*tnsr*)

Given a PyTorch tensor (usually floats), returns True if all entries are integers, false otherwise

1.6 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

I

L

M

N

O

P

R

S

T

X