io_anim_mvnx Documentation

Release 0.1.0

Andres FR

Aug 04, 2019

CONTENTS:

1 io_anim_mvnx package **1**
- 1.1 Submodules . . . 1
- 1.2 io_anim_mvnx.mvnx module . . . 1
- 1.3 io_anim_mvnx.mvnx_import module . . . 3
- 1.4 io_anim_mvnx.operators module . . . 5
- 1.5 io_anim_mvnx.utils module . . . 6
- 1.6 Module contents . . . 8

2 Indices and tables **10**

Python Module Index **11**

Index **12**

CHAPTER

ONE

IO_ANIM_MVNX PACKAGE

1.1 Submodules

1.2 io_anim_mvnx.mvnx module

This module contains functionality concerning the adaption of the XSENS MVN-XML format into our Python setup. The adaption tries to be as MVN-version-agnostc as possible. Still, it is possible to validate the file against a given schema.

The official explanation can be found in section 14.4 of the *XSENS MVN User Manual*:

https://usermanual.wiki/Document/MVNUserManual.1147412416.pdf

A copy is stored in this package's repository:

https://github.com/andres-fr/blender-mvnx-io

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

```
# MVNX schemata can be found in this package or in
# https://www.xsens.com/mvn/mvnx/schema.xsd
mvn_path = "XXX"
mmvn = Mvnx(mvn_path)

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

# subject.segments contain 3D pos_b labels:
for ch in mmvn.mvnx.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.mvnx.subject.sensors.iterchildren():
    s.attrib
```

(continues on next page)

(continued from previous page)

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

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

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

# The bulk of the data is in the frames.
frames_metadata, config_frames, normal_frames = mmvn.extract_frame_info()
```

When calling `extract_frame_info`, we expect specific fields to have specific datatypes. This is reflected In the globals:

```
KNOWN_STR_FIELDS, KNOWN_INT_FIELDS, KNOWN_FLOAT_VEC_FIELDS
```

These are passed as default parameters to the Mvnx constructor, but can be changed at will. The following exemplifies how the metadata could look like:

Metadata:

```
{'segmentCount': 23, 'sensorCount': 17, 'jointCount': 22}
```

And this what fields would the non-normal frames have:

```
['orientation', 'position', 'time', 'tc', 'ms', 'type']
```

As for the normal frames:

```
['orientation', 'position', 'velocity', 'acceleration',
 'angularVelocity', 'angularAcceleration', 'footContacts',
 'sensorFreeAcceleration', 'sensorMagneticField', 'sensorOrientation',
 'jointAngle', 'jointAngleXZY', 'jointAngleErgo', 'centerOfMass', 'time',
 'index', 'tc', 'ms', 'type']
```

More information about the MVNX format can be found in section 14.4 of the already mentioned document:

https://usermanual.wiki/Document/MVNUserManual.1147412416.pdf

class io_anim_mvnx.mvnx.**Mvnx**(*mvnx_path*, *mvnx_schema_path=None*, *str_fields={'tc', 'type'}*, *int_fields={'index', 'jointCount', 'ms', 'segmentCount', 'sensorCount', 'time'}*, *float_vec_fields={'acceleration', 'angularAcceleration', 'angularVelocity', 'centerOfMass', 'jointAngle', 'jointAngleErgo', 'jointAngleXZY', 'orientation', 'position', 'sensorFreeAcceleration', 'sensorMagneticField', 'sensorOrientation', 'velocity'}*)

Bases: `object`

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

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

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

extract_frame_info()

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

static extract_frames(*mvnx*, *str_fields*, *int_fields*, *fvec_fields*)

The bulk of the MVNX file is the `mvnx->subject->frames` section. This function parses it and returns its information in a python-friendly format, mainly via the `process_dict` function.

Parameters

- **mvnx** – An XML tree, expected to be in MVNX format
- **fields** (*collection*) – Collection of strings with field names that are converted to the specified type (fvec is a vector of floats).

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. Fields found in the given int and vec field lists will be converted and the rest will remain as XML nodes.

extract_joints()

Returns A tuple (X, Y). The element X is a list of the joint names ordered as they appear in the MVNX file. The element Y is a list in the original MVNX ordering, in the form [((seg_ori, point_ori), (seg_dest, point_dest)), ...], where each element contains 4 strings summarizing the origin->destiny of a connection.

extract_segments()

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

io_anim_mvnx.mvnx.**process_dict**(*d*, *str_fields*, *int_fields*, *fvec_fields*)

Returns a copy of the given dict where the values (expected strings) whose keys are in the specified fields are converted to the specified type. E.g. If `int_fields` contains the `index` string and the given dict contains the `index` key, the corresponding value will be converted via `int()`.

io_anim_mvnx.mvnx.**str_to_vec**(*x*)

Converts a node with a text like '1.23, 2.34 ...' into a list like [1.23, 2.34, ...]

1.3 io_anim_mvnx.mvnx_import module

This module contains the required functionality to import an MVNX file as a moving set of bones into Blender.

It allows for different options regarding their connectivity, scale...

The general workflow is documented in section 5 of the *Moven User Manual*:

http://www.cs.unc.edu/Research/stc/FAQs/Xsens/Moven/Moven%20User%20Manual.pdf

A copy is stored in this package's repository:

https://github.com/andres-fr/blender-mvnx-io

See the specific docstrings and code commentary for more details.

io_anim_mvnx.mvnx_import.**global_to_inherited_quats**(*quaternions*, *joints*, *name_to_idx_map*)

Parameters

- **quaternions** (*list*) – [q1, q2, ...]
- **joints** – A list of segment connections in the form [(parent_name, child_name), ...]
- **name_to_idx_map** (*dict*) – A dict in the form {seg_name: idx, ...} where The quaternion for the segment seg_name can be found in quaternions[idx].

Given the list of MVNX quaternions and their tree relations, return a list with same shape, but each quaternion is expressed relative to its parent. For that, it suffices the following calculation:

```
q_child_relative = q_parent_glob.conjugated() * q_child_glob
```

Warning: This function assumes that all the input quaternion orientations are given with respect to the same global reference.

io_anim_mvnx.mvnx_import.**load_mvnx_into_blender**(*context*, *filepath*, *mvnx_schema_path=None*, *connectivity='CONNECTED'*, *scale=1.0*, *report=<built-in function print>*, *frame_start=0.0*, *inherit_rotations=True*, *add_identity_pose=True*, *add_t_pose=True*, *verbose=True*)

Parameters

- **context** – A Blender context like bpy.context
- **filepath** – Path expected to point to an MVNX file (XML)
- **mvnx_schema_path** – Optional path to a validation schema for the MVNX
- **connectivity** (*str*) – One of `['CONNECTED', 'INDIVIDUAL']`. If individual, all bones will have no parent and no children, and their positions and rotations will be loaded independently from the others. If connected, the relations defined in the MVNX will be regarded to form a tree of bones, where only the tree roots will have a position. All bones will in any case have angles: see `inherit_rotations` for more information.
- **inherit_rotations** (*bool*) – If true, rotating a bone will propagate the same rotation to all its children, so the rotations are expressed with respect to the parent. Otherwise the rotations are absolute and rotating a bone will displace the children but their orientation won't change. Note that in both modes the imported animation will look identical.
- **add_identity_pose** (*bool*) – If true, the 'identity' frame (zero rotations) is added at the beginning of the sequence.
- **add_t_pose** (*bool*) – If true, the 'tpose' frame is added at the beginning of the sequence (but after the identity if given).
- **verbose** (*bool*) – If true, prints some information about the process to the terminal.

The main routine in this module: given the MVNX information and some other configurations, it creates a set of bones in Blender (so-called *Armature*) that will have the shape and perform the sequence specified in the MVNX. The function returns a pointer to the armature, whose name will be the same as the MVNX file imported (plus potentially extra '.XYZ' digits if a file is imported multiple times).

io_anim_mvnx.mvnx_import.**set_bone_head_and_tail**(*bone, segment_points, joints, root_points=['pHipOrigin'], leaf_points=['pTopOfHead', 'pRightTopOfHand', 'pLeftTopOfHand', 'pRightToe', 'pLeftToe'], scale=1.0*)

Parameters

- **bone** – an EditBone
- **segment_points** – A dict of dicts that allows to find an offset given a joint connector. Expected: {seg_name: {p_name: 3d_vector, ...}, ...}.
- **joints** – A list in the original MVNX ordering, in the form [((seg_ori, point_ori), (seg_dest, point_dest)), ...], where each element contains 4 strings summarizing the origin->destiny of a connection.
- **root_points** (*list*) – If a given bone/segment is root (has no parent), the first match in this list will be taken as bone.head.
- **leaf_points** (*list*) – If a given bone/segment is a leaf (has no children), the first match in this list will be taken as bone.tail.
- **scale** (*float*) – A positive float by which head and tail vectors will be multiplied.

For a given bone, this function reads the information of its parent and, given some MVNX data and assumptions, calculates its head and tail positions. It is also responsible of rescaling the armature.

Warning:

This function assumes the following:

1. If the bone has a parent, it can be found under bone.parent.
2. The head and tail of the parent have been already properly set.
3. If this bone is a root, its segment will have a point with a label in root_points.
4. If this bone is a leaf, its segment will have a point with a label in leaf_points.
5. For a given bone, there aren't any possible collisions between the different root or head_points, i.e., the first match in the list will always be a good match (this happens e.g. if each leaf has a uniquely named leaf_point, which is usually the case).

1.4 io_anim_mvnx.operators module

This module contains subclasses from bpy.types.Operator defining user-callable functors. Operators can also be embedded in Panels and other UI elements.

class io_anim_mvnx.operators.**ImportMVNX**

Bases: bpy_types.Operator, bpy_extras.io_utils.ImportHelper

Load an MVNX motion capture file. This Operator is heavily inspired in the oficially supported ImportBVH.

VERBOSE_IMPORT = True

bl_idname = 'import_anim.mvnx'

bl_label = 'Import MVNX'

bl_options = {'REGISTER', 'UNDO'}

bl_rna = <bpy_struct, Struct("IMPORT_ANIM_OT_mvnx")>

execute(*context*)

Passes the properties captured by the UI to the load_mvnx_into_blender function.

Returns {'FINISHED'} if everything went OK, {'CANCELLED'} otherwise.

1.5 io_anim_mvnx.utils module

Utilities for interaction with Blender

class io_anim_mvnx.utils.**ArgumentParserForBlender**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)

Bases: argparse.ArgumentParser

This class is identical to its superclass, except for the parse_args method (see docstring). It resolves the ambiguity generated when calling Blender from the CLI with a python script, and both Blender and the script have arguments. E.g., the following call will make Blender crash because it will try to process the script's -a and -b flags:

```
blender --python my_script.py -a 1 -b 2
```

To bypass this issue this class uses the fact that Blender will ignore all arguments given after a double-dash ('–'). The approach is that all arguments before '–' go to Blender, arguments after go to the script. The following CLI calls work fine:

```
blender --python my_script.py -- -a 1 -b 2
blender --python my_script.py --
```

get_argv_after_doubledash(*argv*)

Parameters **argv** (*list<str>*) – Expected to be sys.argv (or alike).

Returns The argv sublist after the first '--' element (if present, otherwise returns an empty list).

Return type list of str

Note: Works with any *ordered* collection of strings (e.g. list, tuple).

parse_args()

This method is expected to behave identically as in the superclass, except that the sys.argv list will be pre-processed using get_argv_after_doubledash before. See the docstring of the class for usage examples and details.

Note: By default, *argparse.ArgumentParser* will call *sys.exit()* when encountering an error. Blender will

react to that shutting down, making it look like a crash. Make sure the arguments are correct!

class io_anim_mvnx.utils.**ImportFilesCollection**
Bases: bpy_types.PropertyGroup

This property group allows to load multiple files from the UI file browser menu, by selecting them with shift pressed. Source and usage example:

```
https://www.blender.org/forum/viewtopic.php?t=26470
```

bl_rna = <bpy_struct, Struct("ImportFilesCollection")>

class io_anim_mvnx.utils.**KeymapManager**
Bases: list

This class implements functionality for registering/deregistering keymaps into Blender. It also behaves like a regular list, holding the keymaps currently registered. To inspect the registered keymaps simply iterate the instance.

KEYMAP_NAME = 'Object Mode'

KEYMAP_REGION_TYPE = 'WINDOW'

KEYMAP_SPACE_TYPE = 'EMPTY'

register(*context*, *key*, *stroke_mode*, *op_name*, *ctrl=True*, *shift=True*, *alt=False*)
Adds a new keymap to this collection, and to the config in context.window_manager.keyconfigs.addon. See the API for details:

https://docs.blender.org/api/blender2.8/bpy.types.KeyMap.html
https://docs.blender.org/api/blender2.8/bpy.types.KeyMapItem.html
https://docs.blender.org/manual/de/dev/advanced/keymap_editing.html

Usage example:

```
kmm = KeymapManager()
kmm.register(bpy.context, "D", "PRESS", MyOperator.bl_idname)
```

Parameters

- **context** (*bpy.types.Context*) – The Blender context to work in.
- **key** (*str*) – See bpy.types.KeyMapItem.key_modifier
- **stroke_mode** (*str*) – See bpy.types.KeyMapItem.value
- **op_name** (*str*) – Name of a valid operation in bpy.ops (usually the bl_idname)
- **ctrl, shift, alt** (*booleans*) – Modifiers of the key

Returns None

unregister()
Removes every mapped item from every keymap in this collection, and then empties the collection.

class io_anim_mvnx.utils.**OperatorToMenuManager**
Bases: list

This class implements functionality for adding/removing operators into Blender UI menus. It also behaves like a regular list, holding the currently registered items. Usage example:

```
omm = OperatorToMenuManager()
# In register():
omm.register(MyOperator, bpy.types.VIEW3D_MT_object)
# ... in unregister():
omm.unregister
```

register(*op_class*, *menu_class*)

Parameters

- **op_class** (*bpy.types.Operator*) – (Sub)class handle with desired functionality.
- **menu_class** (*bpy.types.{Header, Panel, ..}*) – Class handle for the Blender GUI where the functionality can be triggered.

Note: `op_class` must define the `bl_idname` and `bl_label` fields.

unregister()
Removes every mapped operator from every menu class in this collection, then empties the collection.

io_anim_mvnx.utils.**is_number**(*s*)

Returns True iff s is a number.

io_anim_mvnx.utils.**make_timestamp**(*timezone='Europe/Berlin'*)
Output example: day, month, year, hour, min, sec, milisecs: 10_Feb_2018_20:10:16.151

io_anim_mvnx.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

io_anim_mvnx.utils.**rot_euler_degrees**(*rot_x*, *rot_y*, *rot_z*, *order='XYZ'*)

Parameters **rot** (*float*) – Rotation angle in degrees.

Returns An Euler rotation object with the given rotations (converted to gradians) and rotation order.

io_anim_mvnx.utils.**str_to_vec**(*s*)
Converts a string like '1.23, 2.34 ...' into a list like [1.23, 2.34, ...]

1.6 Module contents

This add-on allows you to import motion capture data in MVNX format into Blender.

After activating it, it features an operator that can be found in `[File > Import/Export]`. Clicking on it will open a file navigator with a set of options to customize how the MVNX will be imported into a Blender armature.

Position the mouse over the different options or read the corresponding docstrings to get more info about what do they do.

To install this add-on, make sure Blender's Python is able to find it under `addon_utils.paths()`, and that the Blender version matches to make it installable. Alternatively, run this init file as a script from Blender.

`io_anim_mvnx.`**`register`**`()`
Main register function, called on startup by Blender

`io_anim_mvnx.`**`unregister`**`()`
Main unregister function, called on shutdown by Blender

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

io_anim_mvnx, 8
io_anim_mvnx.mvnx, 1
io_anim_mvnx.mvnx_import, 3
io_anim_mvnx.operators, 5
io_anim_mvnx.utils, 6

INDEX

A

ArgumentParserForBlender (*class in io_anim_mvnx.utils*), 6

B

bl_idname (*io_anim_mvnx.operators.ImportMVNX attribute*), 5
bl_label (*io_anim_mvnx.operators.ImportMVNX attribute*), 5
bl_options (*io_anim_mvnx.operators.ImportMVNX attribute*), 5
bl_rna (*io_anim_mvnx.operators.ImportMVNX attribute*), 5
bl_rna (*io_anim_mvnx.utils.ImportFilesCollection attribute*), 7

E

execute() (*io_anim_mvnx.operators.ImportMVNX method*), 6
export() (*io_anim_mvnx.mvnx.Mvnx method*), 2
extract_frame_info() (*io_anim_mvnx.mvnx.Mvnx method*), 2
extract_frames() (*io_anim_mvnx.mvnx.Mvnx static method*), 3
extract_joints() (*io_anim_mvnx.mvnx.Mvnx method*), 3
extract_segments() (*io_anim_mvnx.mvnx.Mvnx method*), 3

G

get_argv_after_doubledash() (*io_anim_mvnx.utils.ArgumentParserForBlender method*), 6
global_to_inherited_quats() (*in module io_anim_mvnx.mvnx_import*), 3

I

ImportFilesCollection (*class in io_anim_mvnx.utils*), 7
ImportMVNX (*class in io_anim_mvnx.operators*), 5
io_anim_mvnx (*module*), 8
io_anim_mvnx.mvnx (*module*), 1
io_anim_mvnx.mvnx_import (*module*), 3
io_anim_mvnx.operators (*module*), 5
io_anim_mvnx.utils (*module*), 6
is_number() (*in module io_anim_mvnx.utils*), 8

K

KEYMAP_NAME (*io_anim_mvnx.utils.KeymapManager attribute*), 7
KEYMAP_REGION_TYPE (*io_anim_mvnx.utils.KeymapManager attribute*), 7
KEYMAP_SPACE_TYPE (*io_anim_mvnx.utils.KeymapManager attribute*), 7
KeymapManager (*class in io_anim_mvnx.utils*), 7

L

load_mvnx_into_blender() (*in module io_anim_mvnx.mvnx_import*), 4

M

make_timestamp() (*in module io_anim_mvnx.utils*), 8
Mvnx (*class in io_anim_mvnx.mvnx*), 2

O

OperatorToMenuManager (*class in io_anim_mvnx.utils*), 7

P

parse_args() (*io_anim_mvnx.utils.ArgumentParserForBlender method*), 6
process_dict() (*in module io_anim_mvnx.mvnx*), 3

R

register() (*in module io_anim_mvnx*), 9
register() (*io_anim_mvnx.utils.KeymapManager method*), 7
register() (*io_anim_mvnx.utils.OperatorToMenuManager method*), 8
resolve_path() (*in module io_anim_mvnx.utils*), 8

rot_euler_degrees() (*in module io_anim_mvnx.utils*), 8

S

set_bone_head_and_tail() (*in module io_anim_mvnx.mvnx_import*), 4
str_to_vec() (*in module io_anim_mvnx.mvnx*), 3
str_to_vec() (*in module io_anim_mvnx.utils*), 8

U

unregister() (*in module io_anim_mvnx*), 9
unregister() (*io_anim_mvnx.utils.KeymapManager method*), 7
unregister() (*io_anim_mvnx.utils.OperatorToMenuManager method*), 8

V

VERBOSE_IMPORT (*io_anim_mvnx.operators.ImportMVNX attribute*), 5