aertb

Release 0.4.0

Rafael Mosca

Feb 14, 2021

OVERVIEW:

CHAPTER

ONE

THE LIBRARY

1.1 Description

This library intends to be a minimal tool for loading and manipulating events in Python from files with common event-camera file extensions .

See the project on PyPI or do `pip3 install aertb`

1.2 Usage

```
from aertb.core.loaders import PolarityEventFile
file = PolarityEventFile('../myFile.myext')
# handle every supported file extension
# then file.header, file.get_events() ...
```

Supported extensions:

- `.dat`: N-Cars / Prophesee Cameras
- `.bin`: N-MNIST, N-Caltech101
- `.aedat`: PokerDVS, PostureDVS
- `.mat`: DVS-Barrel

It also make the process of loading and iterating HDF5 files easier.

```
from aertb.core import HDF5File
dataset_train = HDF5File('TRAIN.h5')
train_iterator = dataset_train.iterator(n_samples_group=10, rand=23)
for sample in tqdm(train_iterator):
    # do something with sample.events, sample.label or sample.name
```

Example: making a GIF

```
from aertb.core import HDF5File, make_gif
file = HDF5File('../DVS_Barrel.hdf5')
sample = file.load_events(group='moving', name='11')
make_gif(sample, filename='sample_moving.gif', camera_size=(128, 128), n_frames=480,␣
↪gtype='std')
```

The library also includes a command line interface for converting files from a given extension to hdf5, as well as gif making capabilities for easy visualisation of the files.

1.3 Notebook Examples

CHAPTER

TWO

AERTB.CORE

2.1 aertb.core.loaders

Suggested use is:

```
from aertb.core.loaders import PolarityEventFile
file = PolarityEventFile('../myFile.myext')
# handle every supported file extension
# then file.header, file.get_events() ...
```

class aertb.core.loaders.**AedatLoader**(**args*, ***kwargs*)

Bases: object

Finds the appropriate version to load the file ...

get_header(*filename*)

classmethod get_version(*filename*)

Gets the version of the AEDAT file in format (DIGIT.DIGIT)

instance = None

load_events(*filename*, *polarities*, *to_secs*)

load_events_v2(*filename*, *polarities*, *to_secs*)

Returns events from an aedat file. Each dat file is a binary file in which events are encoded using 4 bytes (unsigned int32) for the timestamps and 4 bytes (unsigned int32) for the data. The data is composed of 7 bits for the x position, 7 bits for the y position and 1 bit for the polarity.

Parameters

- **filename** (*str,*) – the name of the file to load
- **polarities** (*list,*) – the polarity encoding, can be [0,1] or [-1,1], by default [-1, 1]
- **to_secs** (*bool,*) – determines whether to keep in microseconds (False) or convert to seconds (True), by default True

Returns a numpy structured array with (x, y, ts, p) fields

Return type np.array

parse_header(*f*)

class aertb.core.loaders.**BinLoader**(**args*, ***kwargs*)

Bases: aertb.core.loaders.interface.LoaderInterface

get_header(*filename*)

instance = None

load_events (*filename*, *polarities*, *to_secs*)
Reads a binary file containing events. To use with the N-MNIST or N-CALTECH101 dataset

Each event occupies 40 bits as described below: bit 39 - 32: Xaddress (in pixels) bit 31 - 24: Yaddress (in pixels) bit 23: Polarity (0 for OFF, 1 for ON) bit 22 - 0: Timestamp (in microseconds)

class aertb.core.loaders.**DatLoader**(**args*, ***kwargs*)
Bases: aertb.core.loaders.interface.LoaderInterface

dat_events (*f*, *ev_type*, *polarities*, *to_secs*)
Load the events with the appropriate file extension

Parameters

- **f** – the file pointer
- **ev_type** – the DAT event type
- **polarities** – how the polarities hould be encoded
- **to_secs** – if we should encode TS in secs

get_header (*filename*)

instance = None

load_cd_events (*f*, *polarities*, *to_secs*)

load_events (*filename*, *polarities=[- 1, 1]*, *to_secs=False*)
Returns events from a dat file. Each dat file is a binary file in which events are encoded using 4 bytes (unsigned int32) for the timestamps and 4 bytes (unsigned int32) for the data, encoding is little-endian ordering. The data is composed of 14 bits for the x position, 14 bits for the y position and 1 bit for the polarity (encoded as -1/1).

Parameters

- **filename** – the filename/path to the .dat file
- **polarities** – the polarity encoding, can be [0,1] or [-1,1] (default)

Returns

Return type returns: a recarray with (x, y, ts, p)

load_stereo_cd_events (*f*, *polarities*, *to_secs*)

parse_header (*f*)

class aertb.core.loaders.**LoaderInterface**
Bases: abc.ABC

abstract get_header (*filename*)

abstract load_events (*filename*, *polarities*, *to_secs*)

class aertb.core.loaders.**MatLoader**(**args*, ***kwargs*)
Bases: aertb.core.loaders.interface.LoaderInterface

get_header (*filename*)

instance = None

load_events (*filename*, *polarities=[- 1, 1]*, *to_secs=False*)
Returns events from a mat file. Each mat file is a MATLAB file containing an object with the TD events.

Works on DVS-BARREL 'Original.mat'

Parameters

- **filename** – the filename/path to the .mat file
- **polarities** – the polarity encoding, can be [0,1] or [-1,1] (default)

Returns

Return type returns: a recarray with (x, y, ts, p)

load_sample_events(*filename*, *object_name='ROI'*, *sample=0*, *polarities=[- 1, 1]*, *to_secs=False*)

Returns events from a mat file. Each mat file is a MATLAB file containing an object with the TD events.

Works on DVS-BARREL 'Stabilized.mat / Moving.mat'

Parameters

- **filename** – the filename/path to the .mat file
- **object_name** – the name of the MATLAB object saved in the file
- **sample** – an integer specifying the sample to load
- **polarities** – the polarity encoding, can be [0,1] or [-1,1] (default)

Returns

Return type returns: a recarray with (x, y, ts, p)

class aertb.core.loaders.**PolarityEventFile**(*filename*)

Bases: object

A top level class redirecting instructions to the correct loader class for the given file

get_header()

Gets the header specified in the file

property header

load_events(*polarities=[- 1, 1]*, *to_secs=True*)

Returns a structured event array from a supported event file

Parameters

- **polarities** (*list, optional*) – the polarity encoding, can be [0,1] or [-1,1], by default [-1, 1]
- **to_secs** (*bool, optional*) – determines whether to keep in microseconds (False) or convert to seconds (True), by default True

Returns a numpy structured array with (x, y, ts, p) fields

Return type np.array

aertb.core.loaders.**get_loader**(*extension*)

Acts as a factory method for File loaders

2.2 aertb.core.processing

aertb.core.processing.**clean**(*events*, *tau=0.5*, *val=10*, *R=3*, *allow_first=200*)
Removes some of the noise in an event based scenario

Parameters

- **events** (*the event structured numpy array*) –
- **tau** (*float, optional*) – regulates the decay of the memory, by default 0.5 important to set it properly!
- **val** (*int, optional*) – Mask value threshold, by default 10
- **R** (*int, optional*) – Dimension of the neighborhood, by default 3
- **allow_first** (*int, optional*) – how many events are let through at the beginning without seeing the mask value, by default first 200

Returns the input structured array without the noise events

Return type np.array

aertb.core.processing.**downscale**(*events*, *factor=2*, *camera_size=None*)

Downscales the image by the given factor.

Parameters

- **events** (*np.array*) – structured array with fields (x,y)
- **factor** (*int, optional*) – the downscale factor must perfectly divide the height and with of the image, otherwise it will raise an error., by default 2
- **camera_size** (*tuple, optional*) – Camera size will be assumed from events, if different, it must be specified with this parameter as (width, height), by default None

Returns the input structured array modified so that the resulting image from events is downscaled by the given factor.

Return type np.array

aertb.core.processing.**flip_diagonal**(*events*)
Modifes a set of events so that the resulting image is flipped along the main diagonal

aertb.core.processing.**flip_horizontal**(*events*)
Modifies a set of events so that the resulting image is flipped horizontally

aertb.core.processing.**flip_vertical**(*events*)
Modifies a set of events so that the resulting image is flipped vertically

aertb.core.processing.**rotate**(*events*, *angle*, *direction='ccw'*)
Modifes a set of events so that the resulting image is rotated

Parameters

- **events** (*np.array,*) – the input event structured array
- **angle** (*int,*) – degrees to rotate: 90, 180, or 270
- **direction** (*str, optional*) – 'ccw' counterclockwise or 'cw' clockwise, by default 'ccw'

Returns the input array modified so the rotation takes place

Return type np.array

2.3 aertb.core.const

aertb.core.const.**HDF5_ALIAS = {'.H5', '.h5', '.hdf', '.hdf5'}**
contains all the known aliases for an HDF5 file

aertb.core.const.**SUPPORTED_EXT = {'.Aedat', '.Bin', '.Dat', '.H5', '.aedat', '.bin', '.dat**
contains all the supported file extensions by the library

2.4 aertb.core.hdf5tools

class aertb.core.hdf5tools.**HDF5File**(*filename*, *groups='all'*)
Bases: object

A wrapper offering useful methods over an HDF5 file, to access the original file use the .file attribute

fixed_train_test_split(*n_train*, *n_test*, *rand=23*)

Parameters

- **n_train** – number of train samples per group
- **n_test** – number of test samples per group

get_file_stats()
Returns a dictionary with key: group and value: sample count

get_sample_names(*n_samples_group='all'*, *rand=- 1*)
Returns the samples contained in the file

Parameters **rand** – if greater than zero it specifies the seed for the random selection, if negative it is sequential

iterator(*n_samples_group='all'*, *rand=23*)
returns an iterator over the file samples

Parameters

- **n_samples_group** (*str, optional*) – the samples to consider for each label group, by default 'all'
- **rand** (*int, optional*) – a seed for shuffling, by default 23

Returns it can be iterated with next() or a for loop

Return type iterator

load_events(*group*, *name*)

Parameters

- **group** – the group/label of the sample to load
- **name** – the name of the sample to load

Returns a structured array of events

Return type np.array

train_test_split(*test_percentage*, *stratify=True*, *rand=23*)
creates a train/test split from a single HDF5 file,

Parameters

- **test_percentage** – specifies in a float [0.0, 1) how big should be the test set
- **groups** – specify the groups as a list of strings from where the samples should be taken, all other groups will be ignored
- **statify** – if stratify=True the percentages will be relative to the class count and therefore the test set will have the same distribution as the class count, otherwise the test samples are taken randomly regardless of their class, in some scenarios this may cause that some classes may not be in the test set
- **rand** – specifies the random seed for shuffling the samples, use negative numbers or None to return samples in a sequential order

class aertb.core.hdf5tools.**HDF5FileIterator**(*file*, *samples*)

Bases: object

Returns an iterator over an HDF5 file, suggested usage is:

iterator = HDF5FileIterator(..) for elem in iterator:

do something ...

reset()

Resets the iterator

aertb.core.hdf5tools.**create_hdf5_dataset**(*dataset_name*, *file_or_dir*, *ext*, *polarities=[0, 1]*, *to_secs=True*)

Creates an HDF5 file with the specified name, for a parent directory containing .dat files. It will create a different group for each subdirectory

Parameters

- **dataset_name** – the name of the HDF5 file with file extension
- **parent_dir** – the path pointing to the parent directory where the dat files reside
- **polarities** – indicates the polarity encoding for the data, it can be [0,1] or [-1,1]

2.5 aertb.core.types

class aertb.core.types.**EvSample**(*label*, *name*, *events*)

Bases: tuple

property events

Alias for field number 2

property label

Alias for field number 0

property name

Alias for field number 1

class aertb.core.types.**Sample**(*group*, *name*)

Bases: tuple

property group

Alias for field number 0

property name

Alias for field number 1

aertb.core.types.**event_dtype = dtype([('x', '<u2'), ('y', '<u2'), ('ts', '<f4'), ('p', 'i1**
default event type for structured np.arrays containing polarity events for sensor with size up to 65536x65536

aertb.core.types.**stereo_event_dtype = dtype([('x', '<u2'), ('y', '<u2'), ('ts', '<f4'), ('p**
default event type for structured np.arrays containing stereo events

2.6 aertb.core.viz

aertb.core.viz.**make_gif**(*events*, *filename='my_gif.gif'*, *n_frames=8*, *f_type='decay'*, *axis=False*, ***kwargs*)

Creates a GIF from the passed events with the specified number of frames. additional paramters such as 'tau' for the exponential decay 'duration' for the length of the GIF (default 200), etc. are specified as keyword arguments (**kwargs)

Parameters

- **events** (*np.array*) – the events to be used for the GIF
- **filename** (*str, optional*) – the path+name for the gif file, by default 'my_gif.gif'
- **n_frames** (*int, optional*) – the number of frames that will be computed, by default 8
- **f_type** (*str, optional*) – the type of frame used: 'nop' - no polarity, 'decay' - nop+exponential decay, 'std' - pos/neg polarity, by default 'decay'
- **axis** (*bool, optional*) – determines whether the GIF should show axis labels, by default False

Raises **ValueError** – When an invalid frame type is selected

CHAPTER

THREE

SHELL

The library also includes a command line interface and shell for converting files from a given extension to hdf5, as well as gif making capabilities for easy visualisation of the files.

3.1 Opening the Shell

- If the install with pip worked perfectly, you can now type *aertb* in a terminal window and the Shell will open.
- If you are installing it from Github: download you should download the project from github and follow the following

instructions:

- a) `git clone ...`
- b) Create a virual environment, if venv is not installed run `pip install virtualenv`, then `python3 -m venv aertb_env`
- c) Run `source aertb_env/bin/activate`
- d) Run the following command: `pip install -r requirements.txt`
- e) Open the shell with `python3 .` or with the `__main__.py` file

3.2 Using the Shell

Once the Shell is open you get a a similar output on your terminal:

type *help* to see supported commands and *help* <*topic*> to get more info of the command

3.3 Examples

Creating an HDF5 out of a directory:

```
tohdf5 -f 'example_data/' -e 'dat' -o 'mytest.h5'
```

The recommended directory shape is

```
|--Parent
    |-- LabelClass1
        |-- SampleName1
        |-- SampleName2
        |-- ....
    |-- LabelClass2
        |-- SampleName1
        |-- SampleName2
        |-- ....
    |-- ...
```

And we suggest that train and test are kept as separate folders so they translate to two different files

Creating an HDF5 out of a single file:

```
tohdf5 -f 'example_data/bin/one/03263.bin' -o 'mytest2.h5'
```

Creating a gif out of a given file:

```
makegif -f 'example_data/prophesee_dat/test_23l_td.dat' -o 'myGif.gif' -nfr 240 -g
↪'std'
```

Exiting the Shell

1. type `quit`
2. Exit virtual environment: `$ deactivate`

PYTHON MODULE INDEX

a

INDEX

M

N

P

R

S

T