PySight

Release 0.8.1

May 10, 2018

Contents

CHAPTER 1

Installation

1.1 Users without Python installed:

Download and install Anaconda for Python 3.6.

It's usually good habit to create a new environment for new projects. At the command line:

```
conda create --name py36 python=3.6 -c conda-forge
source activate py36
pip install pysight
```

1.2 Users who already installed Python

In a virtual environment simply install PySight:

```
pip install pysight
```

In some environments you may be required to install `numpy` before installing PySight.

The "Usage" page provides more details on the operation of PySight.

CHAPTER 2

Usage

To use PySight do one of the following:

1. write a Python script containing:

```
from pysight import main_multiscaler_readout


df, movie = main_multiscaler_readout.run()
```

2. From the command line:

```
python /path/to/pysight/dir/main_multiscaler_readout.py
```

3. For batch processing of multiple list files:

```
from pysight import main_multiscaler_readout

foldername: str = r'/path/to/folder/with/list/files'
globstr: str = ''  # a glob-pattern string to filter files to parse. Default is
↪'*.lst'
recursive: bool = False  # Boolean whether to iterate over nested folders in the␣
↪main folder
cfg_fname: str = r'/path/to/existing/cfg.json'

data_record = main_multiscaler_readout.run_batch(foldername=foldername,␣
↪globstr=globstr, recursive=recursive)
```

4. For parallel execution of multiple list files:

```
from pysight import main_multiscaler_readout

foldername: str = r'/path/to/folder/with/list/files'
globstr: str = ''  # a glob-pattern string to filter files to parse. Default is
↪'*.lst'
recursive: bool = False  # boolean whether to iterate over nested folders in the␣
↪main folder
```

(continues on next page)

(continued from previous page)

```
n_process: int = None  # number of cores to uses. None means all available.
cfg_fname: str = r'/path/to/existing/cfg.json'

data_record = main_multiscaler_readout.mp_batch(foldername=foldername,␣
↪globstr=globstr,
                                                recursive=recursive, n_proc=n_
↪process)
```

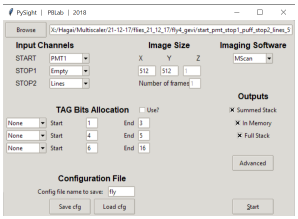
These command will open a GUI in which you'll have to choose either a `.lst` or a pickle (`.p`) file to parse. Alternatively, if you input a configuration file, generated by PySight, you skip the GUI.

If you use option #1, The algorithm will create the pandas DataFrame `df` containing all data, and a `movie` object with allocated data. Outputs come in the form of in-memory numpy arrays and `.hdf5` compressed multidimensional files. The output options are:

- **In Memory** - The returned objects contain all photons and generated histograms. Use it if you wish to further process the data in Python. Access the data with the `.hist` field of the movie object.
- **Full Stack** - PySight will save to disk a multidimensional histogram, one per spectral channel, with all of the events recorded. Dimensions are [t, x, y, z, tau].
- **Summed Stack** - PySight will sum the t dimension of the Full Stack to received a summed projection over time of the entire experiment.

Obviously, if you run the script from the command line (option #2) or in batch mode you can only interact with the final `.hdf5` files.

2.1 GUI Options



Running **PySight** will open the GUI seen above.

Choosing a `.lst` file for analysis is done with the *Browse* button, located at the top-left corner of the GUI.

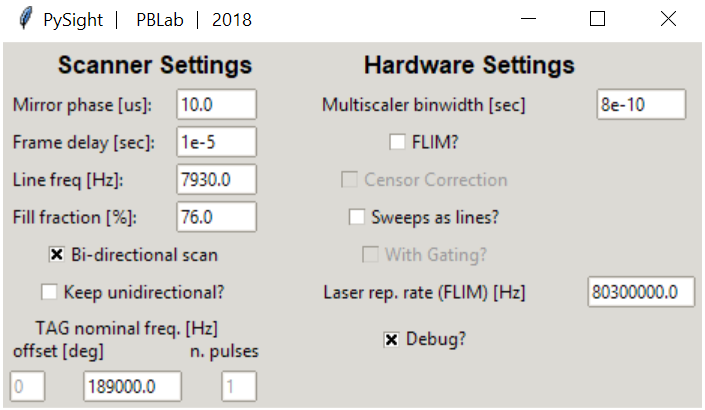
The *Input Channels* option allows you to specify what was the input device to all three (supported) of the analog inputs of the multiscaler. Note that at least one of these fields must be a `PMT1` entry.

To the right, *Image Size* determines the shape of the output matrix from the algorithm. The X dimension corresponds to the line signal, and should correspond to the original signal fed into the multiscaler. Below it you can specify the number of frames if a linesframes signal wasn't captured.

Imaging Software should be specified when working with either ScanImage or MScan. PySight is tested on these two acquisition applications, and thus might fail with data acquired using different software. Please page the package author in GitHub if you wish to integrate your acquisition into PySight.

TAG Bit Allocation specifies each bit's responsibility during the acquisition.

Lastly, At the bottom part of the GUI you can load a specific `.json` file to be used as a config file. A default `default.json` file is supplied with the package, but other files can be saved and loaded easily, to allow easier use of the GUI. The full filename of the data is also saved.



In the *Advanced* pop-up window, you can define more system parameters, including:

- *Mirror phase*: Corrects pixel-shift artifacts in bidirectional scanning.
- *Frame delay*: Time between subsequent frames. Used in *MScan* mode.
- *Line freq*: Resonant mirrors nominal frequency.
- *Fill fraction*: Percentage of time that the mirror spends inside the field of view. High values can cause distortions in the image - a typical value is in the low 70's.
- *Bidirectional scan*: Whether data was acquired in both forward and back phases of the resonant mirror.
- *Keep unidirectional?*: When in bidirectional mode, back phase photons are usually discarded. This option allows you to gain them back into the image. Make sure to set a proper value to the mirror phase parameter.
- *TAG nominal frequency?*: For acquisitions with the TAG lens. Leftmost entry - offset of the pulse. Middle - nominal frequency of the lens. Rightmost - number of pulses per period (currently only one is supported).
- *Binwidth*: The binwidth, in seconds, of your model of the MCS6A.
- *FLIM?*: Check if you wish to add another dimension in the output stack for the time-since-pulse of each photon. Multiscaler must have either a frequency divider connected to its "REF CLK" input (recommended) or a laser-clock signal in one of the analog outputs.
- *Sweeps as lines?*: You can omit the line signal if each sweep of the multiscaler corresponds to a line in the image. Usually it is not recommended to do so.
- *Laser repetition rate*: For FLIM.
- *Debug?*: Reads a relatively small portion of a file, allows for quick code-checking.

2.2 Advanced

You should also specify, in case data was acquired uni-directionally, whether the algorithm should keep the photons arriving during that returning phase. Below, specify the phase delay of the scanners and fill fraction, to cancel pixel-shift and remove the non-linear area of the image, located in the edges. Phase delay is only used in bidirectional mode.

2.3 Limitations

- List (`.lst`) files have to be saved in `ASCII` format, and not binary.
- Only three input channels are currently supported.

- Timepatch `3` is currently unsupported (and can usually be avoided completely).

CHAPTER 3

pysight

3.1 pysight package

3.1.1 Subpackages

3.1.1.1 pysight.ascii_list_file_parser package

Submodules

pysight.ascii_list_file_parser.apply_df_funcs module

pysight.ascii_list_file_parser.distribute_data module

class pysight.ascii_list_file_parser.distribute_data.**DistributeData**(*df*, *dict_of_inputs*, *use_tag_bits=False*) → None

Bases: object

Separates the channel-specific data to their own channels. Inputs:

param df pd.DataFrame with data

param dict_of_inputs Mapping of inputs to data they contain

param use_tag_bits Whether TAG bits are needed

data_to_grab

df

dict_of_data

dict_of_inputs

run() → None
Runs the allocation function, populating self.dict_of_data

use_tag_bits

pysight.ascii_list_file_parser.fileIO_tools module

pysight.ascii_list_file_parser.tabulation_tools module

pysight.ascii_list_file_parser.timepatch_switch module

Module contents

3.1.1.2 pysight.nd_hist_generator package

Subpackages

pysight.nd_hist_generator.line_signal_validators package

Submodules

pysight.nd_hist_generator.line_signal_validators.mscan module

__author__ = Hagai Hargil

class pysight.nd_hist_generator.line_signal_validators.mscan.**MScanLineValidator**(*sig_val*) → None

Bases: object

run() → Tuple[Dict, <Mock name='mock.uint64' id='140411483324936'>]
Interpolate MScan-specific line signals :return: Dictionary containing the data and the mean difference between subsequent lines

sig_val

pysight.nd_hist_generator.line_signal_validators.rectify_lines module

pysight.nd_hist_generator.line_signal_validators.scanimage module

__author__ = Hagai Hargil

class pysight.nd_hist_generator.line_signal_validators.scanimage.**ScanImageLineValidator**(*sig_* → Non

Bases: object

run() → Tuple[Dict, <Mock name='mock.uint64' id='140411483324936'>]
Interpolate SI-specific line signals :return: Dictionary containing the data and the mean difference between subsequent lines

sig_val

pysight.nd_hist_generator.line_signal_validators.validation_tools module

Module contents

__author__ = Hagai Hargil

Submodules

pysight.nd_hist_generator.allocation_tools module

pysight.nd_hist_generator.censor_tools module

pysight.nd_hist_generator.gating_tools module

__author__ = Hagai Hargil

class pysight.nd_hist_generator.gating_tools.**GatedDetection**(*raw*, *reprate=80300000.0*, *binwidth=8e-10*) → None

Bases: object

Gating the unneeded photons

bins_bet_pulses

binwidth

data

range_length

raw

reprate

run()
Main pipeline of class

pysight.nd_hist_generator.movie_tools module

pysight.nd_hist_generator.output_tools module

pysight.nd_hist_generator.photon_df_tools module

class pysight.nd_hist_generator.photon_df_tools.**PhotonDF**(*dict_of_data*, *num_of_channels=1*) → None

Bases: object

Create initial photon dataframe and set the channel as its index

dict_of_data

gen_df()
If a single PMT channel exists, create a df_photons object. Else, concatenate the two data channels into a single dataframe.

Return pd.DataFrame Photon data

num_of_channels

pysight.nd_hist_generator.tag_bits_tools module

pysight.nd_hist_generator.tag_tools_v2 module

pysight.nd_hist_generator.volume_gen module

class pysight.nd_hist_generator.volume_gen.**VolumeGenerator**(*frames*, *data_shape*, *MAX_BYTES_ALLOWED=300000000*) → None

Bases: object

Generate the list of volume chunks to be processed. Main method is "create_frame_slices", which returns a generator containing slice objects that signify the chunks of volumes to be processed simultaneously. Inputs: :param frames pd.DataFrame: Frames for the entire dataset. Should not contain a closing, right-edge, frame. :param data_shape tuple: Shape of the final n-dimensional array (from the Output object) :param MAX_BYTES_ALLOWED int: Number of bytes that can be held in RAM ("magic number")

MAX_BYTES_ALLOWED

bytes_per_frames

create_frame_slices(*create_slices=True*) → Generator
Main method for the pipeline. Returns a generator with slices that signify the start time and end time of all frames.

Parameters **bool** (*create_slices*) – Used for testing, always keep true.

data_shape

frame_per_chunk

frame_slices

frames

full_frame_chunks

num_of_chunks

num_of_frames

Module contents

__author__ = Hagai Hargil

3.1.2 Submodules

3.1.3 pysight.main module

3.1.4 pysight.tkinter_gui_multiscaler module

3.1.5 Module contents

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

4.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.2 Documentation improvements

PySight could always use more documentation, whether as part of the official PySight docs, in docstrings, or even on the web in blog posts, articles, and such.

4.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/PBLab/python-pysight/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

4.4 Development

To set up PySight for local development:

1. Fork python-pysight (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-pysight.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

5. Submit a pull request through the GitHub website.

4.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests[1].
2. Update documentation when there's new API, functionality, etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

[1] If you don't have all the necessary Python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 5

Authors

- Hagai Har-Gil - http://pblab.tau.ac.il/en/

Under the supervision of Dr. Pablo Blinder, Tel Aviv University.

CHAPTER 6

Changelog

6.1 0.1.0 (2017-02-27)

- First release on PyPI.

6.2 0.1.1 (2017-02-27)

- Bug fixes during installation of Numba.
- Added the `run()` method for `main_multiscaler_readout`.

6.3 0.1.2 (2017-02-27)

- Includes `tifffile` and minor improvements.

6.4 0.1.3 (2017-02-28)

- Changed IO from `.read()` to `.readlines()` for better Linux compatibility.
- `.tif` is now saved frame-by-frame to save memory, and the method was renamed to `create_tif()`.

6.5 0.1.4 (2017-02-28)

- Frames are now generated with a generator.
- Fix to installation problems of previous version.

6.6 0.1.5 (2017-02-28)

- Single-lined frames are now supported.

6.7 0.1.6 (2017-02-28)

- More tests coverage.
- Enforced a few types checks.

6.8 0.1.7 (2017-03-01)

- Potential fix to `pip install` issues.
- Start of TAG lens interpolation support.

6.9 0.2.0 (2017-03-05)

- Support for TAG lens added - phase interpolation and image display. Note: The algorithm currently assumes that the pulse is triggered at the zero-phase of the TAG lens.
- `pip` installation fixed by requiring Numba as a prerequisite.
- Number of pixels in the "Frame" direction (x) supersedes the number of frames as listed by the user.
- Due to massive changes, one test is currently broken.

6.10 0.3.0 (2017-03-07)

- Added method `create_array` to Movie() that returns a deque containing the raw data generated by the `np.histogram` function, for visualization and analysis purposes.
- Added method `create_single_volume` to Movie() that sums all stacks into a single array.
- Fixed bugs in `tag_tools`, mainly in `verify_periodicity()`.
- Allows for more elaborate user inputs, requiring to choose which type of output you wish for.
- Basic FLIM support.

6.11 0.3.1 (2017-03-07)

- Tiffs are now saved untiled. Depth axis is x-axis.
- Installation should run smoothly if following the instructions.

6.12 0.3.2 (2017-03-07)

- Added verifications on the FLIM input.
- Bug fixes in FLIM implementation.

6.13 0.3.3 (2017-03-08)

- Code can take care of the the infamous [-1, . . . , -1] index list.
- Added `debug` mode in which the algorithm reads only a limited amount of lines from a file.
- Fixed minor bug in `__create_hist`.
- Decreased size of package by removing excess lines of data for tests.

6.14 0.3.4 (2017-03-09)

- More fixes to the [-1] vector problem.
- Added a `sort` function before handling the data, because of irregularities.

6.15 0.3.5 (2017-03-11)

- Added sinusoidal interpolation to TAG phase.
- Sorting is now only done for TAG lens input.
- Added `fileIO_tools.py` module for increased simplicity.
- Added more verifications to user inputs from GUI that pop up sooner, before heavy computation is made.
- Increased file IO speed with a new `np.fromfile` method.

6.16 0.3.6 (2017-03-14)

- Basic support for TAG bits - no actual interpolation yet.
- GUI additions and changes.
- Minor performance upgrades.

6.17 0.4.0 (2017-03-16)

- Changes file IO completely. Performance should be higher.
- TAG lens bug fixes.
- Updated docs.
- Updated tests.

6.18 0.4.1 (2017-03-30)

- Updates to setup.py to allow docs to build successfully.
- Small updates to docs.
- GUI improvements.

6.19 0.4.2 (2017-03-30)

- Added Dask `delayed` interface.

6.20 0.4.3 (2017-04-02)

- Removed Dask.
- Refactored class structure, remove the `Frame` class.
- Refactored GUI code.

6.21 0.4.4 (2017-04-08)

- Changes to file IO.
- Number of requested frames should actually matter now.
- GUI improvements.

6.22 0.4.5 (2017-04-17)

- Bug fixes and improvements to TAG lens interpolation.

6.23 0.4.6 (2017-05-16)

- Use `Debug?` to read a small portion of an `.lst` file.
- Changed defaults in GUI.
- Allows acquisition in bi-directional scanning mode. This is enabled with the `Mirror phase` and `Flyback` parameters in the GUI.
- Backend changes for possible future support of binary files.
- **The code allows to dismiss unwanted input channels by specifying them as "Empty".**
  - If you mark a channel as containing data while it's inactive, an error will terminate execution.
- Massive refactoring of pipeline.

6.24 0.4.7 (2017-05-25)

- Fixed some of the tests.
- Added option to save or discard photons arriving during the returning phase of a unidirectional scan. This is the default option now.
- Introduced `Fill Fraction` parameter that determines the amount of *time* the mirrors spend "inside" the image.
- Some tests are working again.
- Many other bugfixes.

6.25 0.4.8 (2017-05-31)

- Added type hinting. As a result, disabled support for Python version 3.5. Code is now entirely 3.6-dependent.
- Added `.json` configuration files to the GUI. It also automatically loads the last modified configuration file.
- Updated docs.

6.26 0.5.0 (2017-06-04)

- **Added the `CensorCorrection` class for processing generated data using the censor correction method. Current availab**
  - `censored.gen_bincount_deque()`: Bin the photons into their relative laser pulses, and count how many photons arrived due to each pulse.
  - `censored.find_temp_structure_deque()`: Generate a summed histogram of the temporal structure of detected photons.
- Fixed linux bug with `Deque` import.
- Added tests.

6.27 0.5.1 (2017-06-04)

- Another go at Linux namespace conflicts.

6.28 0.5.2 (2017-06-06)

- Added basic support for "Censor Correction".

6.29 0.5.3 (2017-06-06)

- Bug fixes, including support for single-pixel images.
- Script should require less memory while running.

6.30 0.5.4 (2017-06-06)

- Fixed untested typo.

6.31 0.5.5 (2017-06-07)

- Bug fixes for single-pixel frames
- Bug fixes for defining amount of frames manually in script.
- Censor correction shouldn't require as much memory as it did. It's still not as fast as it can be.
- Loading a configuration file will make it the "last used" file, reloading it when re-running PySight.

6.32 0.5.6 (2017-06-08)

- ML classification is functional.

6.33 0.5.7 (2017-06-08)

- More robust data generation.
- Added `scikit-learn` to `requirements.txt` and `setup.py`.
- `CensorCorrection().learn_histograms()` now receives the power label as its input - must be an integer.
- Return of `CensorCorrection().learn_histograms()` is now `data, labels`.
- Made `__get_bincount_deque()` private. To be accessed using `learn_histograms()` only.

6.34 0.5.8 (2017-06-08)

- Robustness upgrades.
- QOL changes to GUI.
- A "power" number is needed for `learn_histograms()` - the percent of power given to the Qubig. It's just for saving, labeling is done with the `label` keyword.
- A `foldername` to which the data will be saved to has also been added.

6.35 0.5.9 (2017-06-11)

- Much (MUCH) faster concatenation of the data.
- Fixed a bug with the number of empty histograms that were added to the learning dataset.

6.36 0.5.10 (2017-06-12)

- Changes and optimizations for the file IO process.
- Fixed a bug with laser pulses verification.
- Added offset parameter for laser input.

6.37 0.5.11 (2017-06-22)

- Added FLIM functionality with laser clock in the Multiscaler's clock.

6.38 0.5.12 (2017-06-22)

- Fixed small bug with GUI.
- Possible fix to TAG lens interpolation.

6.39 0.5.13 (2017-06-26)

- Added SciPy dependency.
- Added nanoFLIM histogramming.

6.40 0.5.14 (2017-06-26)

- Bug fixes and performance improvements.

6.41 0.5.15 (2017-06-27)

- Refactoring of output:
  - Start of censor correction is integrated into the generation of the outputs.
  - More efficient when required to output several types of data.

6.42 0.5.16 (2017-06-27)

- Fixed small bug with censor correction.

6.43 0.5.17 (2017-06-29)

- Fixed `.tif` generation.
- Refactoring of `FileIO` (tests still pass).

6.44 0.5.18 (2017-06-29)

- **Added metadata from `.lst` file to the saved `.tif` file. Variables saved:**
  - "fstchan"
  - "holdafter"
  - "periods"
  - "rtpreset"
  - "cycles"
  - "sequences"
  - "range"
  - "sweepmode"
  - "fdac"

6.45 0.5.19 (2017-06-29)

- Fixed small bug with censoring.
- Added checks to see whether we need censor correction.
- Added tests for `lst_tools` - they should pass, much like `file_io`'s tests.

6.46 0.5.20 (2017-07-01)

- **Refactored the output-generating script, while changing the possible outputs of PySight:**
  - Summed tif.
  - Full stack as tif.
  - In memory - both stack and tif accessed through the `movie` object.

6.47 0.5.21 (2017-07-07)

- Added the acquisition delay and "hold-after" times to the calculation of the the absolute time of each event.
- Decreased package size dramatically by deleting unneeded test data.
- All 34 tests of code pass. I'll try to keep it that way :)
- Added an extrapolation method to create fake lines when the line data is too corrupt to work with. This is done using the new "line frequency" and "frame delay" parmaeters in the GUI.

6.48 0.5.22 (2017-07-17)

- Added an optional line frequency entry to the GUI.
- Refactoring of some parts of the validation tools.
- Small performance upgrade.
- Added an option to treat sweeps as lines.

6.49 0.5.23 (2017-07-20)

- Supports generating images from pure sweeps, without a line signal.
- Supports generating images from combined sweep and line signals.

6.50 0.5.24 (2017-07-30)

- Bugfixes for line validations.
- Added methods `show_summed(channel)` and `show_stack(channel, iterable)`.
- More refactoring to decrease class absolute size.
- Small bug fix in the sweeps-as-lines implementation.

6.51 0.5.25 (2017-08-26)

- Added the `photons_per_pulse` property to `Movie()`.
- Introduced the `run_batch(foldername)` function to the `main` module, to run PySight with the same configs on multiple `.lst` files in a folder.
- Added the `num_of_vols` property to `Movie()`.

6.52 0.6.0 (2017-08-27)

- Changed output file format to `.hdf5` due to compatibility issues of ‘‘.tif‘‘s.

6.53 0.6.1 (2017-08-28)

- Added gating to photons that arrive too early (or too late) after a laser pulse.

6.54 0.6.2 (2017-08-29)

- Fixed bug with `movie.show_stack()`.

6.55 0.6.3 (2017-09-11)

- Better support for "early" photons.
- Allow for no outputs from PySight.

6.56 0.6.4 (2017-09-18)

- Fixed bug with two-channel output.
- Fixed bug with "early" photons.

6.57 0.6.5 (2017-09-18)

- Writing output `.hdf5` to disk is much faster now.

6.58 0.6.6 (2017-09-27)

- Now compressing HDF5 files.
- Fixed small bug in TAG implementation.

6.59 0.6.7 (2017-09-28)

- Faster I/O.
- Datasets are now `uint8` (full stack) and `uint16` (summed stack).
- Allowing outputs without the "In Memory" requirement.
- Added a progress bar.
- `show_summed()` works, `show_stack()` might not.

6.60 0.6.8 (2017-09-28)

- Small bug fix in progress bar.

6.61 0.6.9 (2017-09-29)

- Stacking the final array is now an order-of-magnitude faster - the first dimension is now considered *time*.
- Fixed a bug with singleton dimensions.
- Fixed a bug with no "In Memory" output.

6.62 0.6.10 (2017-10-03)

- Fixed a bug occurring when TAG lens interpolation fails.
- Discovered another bug with the interpolation process which is currently unresolved.
- Fixed small issue with a TAG test function.

6.63 0.6.11 (2017-10-06)

- Complete re-write of TAG lens processing module.

6.64 0.6.12 (2017-10-08)

- Removed the experimental `parallel` feature from the Numba implementation.
- Fixed bidirectional image generation.
- Default fill fraction is now 75% to better suit ScanImage's defaults.

6.65 0.6.13 (2017-10-08)

- The TAG phase is now between 0 and 1, generating non-cyclic volumes.

6.66 0.6.14 (2017-10-30)

- Added a `glob_str` and `recursive` parameters to `run_batch()`.
- Added a `DEBUG` suffix to files generated when debugging.
- Changed license to creative commons.
- Small bug fixes, somewhat decreased memory usage.
- Improved bidirectional scanning performance and robustness by reworking its mechanism.

6.67 0.6.15 (2017-11-05)

- Better bidirectional support.

6.68 0.6.16 (2017-11-20)

- Support for non-phase allocation of TAG pulses.
- Removal of old TAG module.
- `run_batch()` works without choosing a mock list file.

6.69 0.6.17 (2017-12-04)

- Fixed a bug with the filename of the ‘‘DEBUG‘‘ged version.

6.70 0.6.18 (2017-12-05)

- Fixed a bug with bidirectional scanning.
- Possible fix for data that don’t have lines since the beginning of the experiment.

6.71 0.6.19 (2017-12-06)

- Bug with lines allocation in the `Volume` object following an API change in pandas.
- Allows for single frame experiments.

6.72 0.6.20 (2017-12-17)

- Fixed a bug with TAG lens interpolation.

6.73 0.6.21 (2017-12-19)

- More work on TAG interpolation.

6.74 0.6.22 (2017-12-19)

- Added interpolation for missing line signals.

6.75 0.6.23 (2017-12-20)

- Fixed bugs with interpolations and TAG signals.

6.76 0.6.24 (2017-12-20)

- Deals with more edge-cases in missing line signals.

6.77 0.6.25 (2017-12-20)

- Even more edge-case handling.

6.78 0.6.26 (2017-12-21)

- Missing line signals take mirror phase into account.
- Fixes for MScan system.

6.79 0.6.27 (2017-12-24)

- Separated handling of unidir and bidir corrupt line signals.
- Refactored line signal handling module.
- Added multiple tests to line signal handling.

6.80 0.6.28 (2017-12-25)

- Better handling of line signal.

6.81 0.6.29 (2017-12-25)

- Code cleanups.
- More adjustments to line handling in bidirectional mode.

6.82 0.6.30 (2017-12-25)

- Fixed another bug with the line handling.
- Fixed a bug with a missing PMT channel.

6.83 0.6.31 (2017-12-26)

- Bug fix for empty volumes with multichannel support.

6.84 0.6.32 (2017-12-26)

- Trial with Numba and setuptools.
- Type annotations.
- Documentation update.
- Fixes for single-photon bug.

6.85 0.6.33 (2017-12-26)

- Hotfix to `attrs` problem in `setup.py`.

6.86 0.6.34 (2017-12-27)

- Changed output of `run_batch` to a DataFrame.
- Refactored `tabulation_tools`.

6.87 0.7.0 (2018-01-01)

- Refactoring and additions to GUI, including new choices between imaging systems.
- Better UI and UX.
- Not all tests pass.

6.88 0.7.1 (2018-01-01)

- GUI is now startable with "S" key and or "Enter".
- More tests to new SignalValidator class.
- Bug fix for the validation process.
- Making progress on multiprocessing support.

6.89 0.7.2 (2018-01-01)

- Minor bug fixes.

6.90 0.7.3 (2018-01-01)

- Bug fixes to `run_batch`.
- New function `mp_batch(foldername, glob_str)` for parallel processing of a folder of list files.

6.91 0.8.0 (2018-03-19)

- Added `recursive` and `n_proc` keywords to `mp_batch`, and changed return type to `None`.
- Changed source tree structure for better clarity.
- Renamed `run_batch` to `run_batch_lst`.
- More internal improvements.
- Z-axis bins range is equal, i.e. each bin spans the same axial distance in microns.

- Travis CI is back on.
- Added option to run PySight with a predetermined config file: `main.run(cfg_file='/path/to/file.json')`.
- New integration tests.

6.92 0.8.1 (2018-03-19)

- Bug fix in `setup.py`.

6.93 0.8.2 (2018-04-XX)

- Unified configuration file keyword to be `cfg_file`.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

U

V