tristan Documentation

Release 0.2.3

Diamond Light Source — Data Analysis

Jan 19, 2024

CONTENTS:

CHAPTER

ONE

INSTALLATION

```
pip install tristan
```

CHAPTER

TWO

IMAGE BINNING

Tristan is a python package that provides a set of prototype tools for processing data collected on Tristan, the experimental timepix3-based event-mode detector in use at Diamond Light Source.

Intead of images, this detector collects an event stream recording the pixel where the photon hit the detector, its timestamp (time of arrival) and energy (time over threshold). The processing consists in binning these events into one or more images.

2.1 Single image tool

To bin all the events into a single image, for powder processing or similar, use the *images single`* command, alias *images 1*.

This accepts as input either the <file-name-stem>.nxs file, the <file-name-stem>_meta.h5 file or just the collection directory, if only a single data set has been saved there.

```
images single /path/to/file
```

2.2 Multiple image tool

To bin the events into a chronological image sequence, for example a rotation scan, use *images multi*.

As input, this command also accepts the <file-name-stem>.nxs file, the <file-name-stem>_meta.h5 file or the collection parent directory if unique. Additionally, it is also necessary to specify either the number of images, with *-n*, or the exposure time, with *-e*, to know how many images the events should be binned into.

```
images multi /path/to/file -n 1750
```

Alternatively,

```
images multi /path/to/file -e .1ms
```

Note: *-e* accepts most human-readable specifications of units, eg. *-e 100us*, *-e 100µs*, *-e .1ms*, etc…

Another availabe option for this tool is the *-a* flag, alias *–align-trigger*, which aligns the image start time with the first specified trigger signal. This is useful for examining changes in the sample after a trigger signal.

```
images multi -n 400 -a TTL-rising /path/to/file
```

2.3 Static sequence pump-probe tool

The tool *images pump-probe*, alias *images pp*, aggregates all the events from a pump-probe measurement, divides the pump rep period into bins of equal 'width' in time and creates an image for each bin. The resulting sequence of images describes the evolution of the system following a pump pulse, averaged over all pump pulses.

Similarily to *images multi*, this tool requires the trigger type to be specified with *-t*, and the bin 'width' with *-e* ` or the number of bins with *-n*.

```
images pp -n 20 -t TTL-rising /path/to/file
```

For example, this tool could be used to create a 'waterfall plot' of the intensity of a single reflection from a static sample, as it evolves in response to pump pulses.

2.4 Multiple sequence pump-probe tool

To bin events into images representing different pump-probe delays, use *images sequences*, alias *images sweeps*. This tool first divides the pump rep period into bins of equal duration and then creates a sweep of images for each bin, using only the events that fall into that bin. The result is a sequence, or sweep, of images for each pump-probe delay bin.

In the same manner as *images multi*, it is required to set either the exposure time of the images with *-e*, or the number of images per sweep with *-n*. As for the triggers, the trigger signal is specified with *-t*, as in *images pp*. It is also necessary to provide the pump-probe delay intervals either by duration, with *-i*, or by number, with *-x*.

```
images sequences -x 20 -n 180 -t TTL-rising /path/to/file
```

For example, this could be used to deconstruct a rotation data collection into several rotation datasets, each corresponding to a different pump-probe delay window.

2.5 Serial crystallography tool (gated access)

To bin events into images gated by trigger signals, use *images serial*, which will write one image per gate signal. Each 'gate-open' signal is taken as the start of an exposure and the next 'gate-close' signal is taken as the end of the exposure.

This tool requires at least the rising edge of the trigger signal, specified with *-g*, to be passed as *gate open* and will then look for the corresponding falling edge to be used as *gate close*.

```
images serial -g SYNC-rising /path/to/file
```

In some cases, it might be more useful to look at the events collected between different kinds of trigger signals, by specifying the *gate open* signal with *-g* and the *gate close* using the *-c* flag as in the example below.

```
images serial -g TTL-rising -c SYNC-falling /path/to/file
```

CHAPTER

THREE

APPLY THE FLATFIELD CORRECTION

A tool to apply the flat-field correction to the binned images if needed. It is possible to choose whether to multiply or divide the images by the flat-field.

```
apply-flat-field /path/to/binned_img_file /path/to/flatfield_file {multiply, divide}
```

CHAPTER

FOUR

DIAGNOSTIC TOOLS

4.1 Cues inspection tool

This tool inspects all the cue messages in a Tristan dataset and prints out a summary of how many instances are found, whether they have the same timestamp across the modules and the time interval between TTL rising and falling edge.

```
cues /path/to/collection/directory
```

4.2 Trigger inspection tool

This tool runs a quick check on the trigger signals - recorded as cue messages - in a Tristan dataset:

- Looks for shutter opening and closing cues and their timestamps
- Calculates the number of TTL rising edges and LVDS rising and falling edges and looks for their timestamps
- Looks for SYNC triggers and timestamps if running a serial crystallography experiment (to run: add the -e/–expt ssx option to the command line)
- Calculates the time interval between triggers

This check is run on every module of the detector to be sure that all are correctly saving the cue messages. If one module doesn't show some or all of the triggers/timestamps, it's a sign that something might be wrong with the setup.

A copy of the results is saved as a *.log* file in the working directory, unless otherwise specified using the *-o* option.

```
find-trigger-intervals /path/to/collection/directory filename_root -o .
```

4.3 Valid events check

This tool checks that all modules contain at least some valid events ie. events whose timestamp falls in the interval between the shutter open and close signals. It is useful to diagnose synchronization problems during a collection.

```
valid-events /path/to/collection/directory filename_root -o . -s 101.43 801.43
```

As this process has to look through the full dataset, it might take some time to run. Thus, it should only be used when synchronization issues are suspected, for example in case the image binning returns datasets full of 0.

4.4 Modules inspection tool

This tool checks that all files from all detector modules contain valid data, and assigns each file to the correct module. If everything is as it should be for a 10M detector, there will be 10 consecutive files listed for each module.

A copy of the results is saved as a *.log* file in the working directory, unless otherwise specified using the *-o* option.

```
check-tristan-files /path/to/collection/directory filename_root -o .
```

CHAPTER

FIVE

GETTING HELP

Every command in the tristan package has a help message that explains its usage and shows a list of accepted positional and optional arguments. The help message is printed by passing the option –help, alias -h, to any of the commands.

```
images multi -h
```

Or,

```
find-trigger-intervals --help
```

CHAPTER

SIX

API

6.1 General

Utilities for processing data from the Large Area Time-Resolved Detector

This module provides tools to interpret NeXus-like data in HDF5 format from the experimental Timepix-based event-mode detector, codenamed Tristan, at Diamond Light Source.

tristan.**compute_with_progress**(*collection*)

Compute a Dask collection, showing the progress of the top layer of the task graph.

Parameters

collection – A single Dask collection.

6.2 Binning

Tools for binning events to images.

tristan.binning.**align_bins**(*start: int*, *align: int*, *end: int*, *n_bins: int*)

Divide an interval into a specified number of bins, aligning with a given value.

Take three integers, `start align end`, and find a way to span the largest possible interval between `start` and `end` with a specified number of bins, while ensuring that one of the bin edges is aligned with a specified value.

Parameters

- **start** – The start of the interval.
- **align** – The value to which a bin edge should be aligned.
- **end** – The end of the interval.
- **n_bins** – The number of bins.

Returns

The first bin edge and the bin width, from which all the bin edges can be derived.

tristan.binning.**create_cache**(*output_file: Path | str*, *num_images: int*, *image_size: tuple[int, int]*) → Array

Make a Zarr array of zeros, suitable for using as an image binning cache.

The array will have shape (num_images, *image_size) and will be chunked by image, i.e. the chunk shape will be (1, *image_size).

Parameters

- **output_file** – Output file name. Any file extension will be replaced with .zarr.

- **num_images** – The number of images in the array.
- **image_size** – The size of an image in the array.

Returns:

tristan.binning.**events_to_images**(*data: DataFrame*, *bins: Sequence[int]*, *image_size: tuple[int, int]*, *cache: _SupportsArray[dtype] | _NestedSequence[_SupportsArray[dtype]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → DataFrame

Construct a stack of images from events data.

From a sequence of LATRD events data, bin the events to images and store the binned images in a cache array. The cache may be backed with on-disk storage, as in the case of a Zarr array, or may be a simple in-memory object, like a NumPy array.

Parameters

- **data** – LATRD events data. Must have an `event_time_offset` column and an `event_id` column.
- **bins** – The time bin edges of the images (in clock cycles, to match the event timestamps).
- **image_size** – The size of each image.
- **cache** – An array representing the eventual image stack, having shape `(len(bins) - 1, *image_size)`, to which the pixel counts from this binning operation will be added.

Returns

A Dask collection representing the lazy image binning computation.

tristan.binning.**find_start_end**(*data: dd.DataFrame*)

Find the shutter open and shutter close timestamps.

Parameters

data – LATRD data. Must contain one 'cue_id' entry and one 'cue_timestamp_zero' entry. The two arrays are assumed to have the same length.

Returns

The shutter open and shutter close timestamps, in clock cycles.

tristan.binning.**find_time_bins**(*data: DataFrame*, *bins: Sequence[int]*)

Convert the event timestamps in LATRD data to time bin indices.

For each event, determine the index of the bin into which the event will fall.

Parameters

- **data** – LATRD events data. Must have an `event_time_offset` column.
- **bins** – The time bin edges of the images (in clock cycles, to match the event timestamps).

Returns

A DataFrame which matches the input data except that the `event_time_offset` column is replaced with a column of `time_bin` indices.

tristan.binning.**make_images**(*data: DataFrame*, *image_size: tuple[int, int]*, *cache: _SupportsArray[dtype] | _NestedSequence[_SupportsArray[dtype]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*)

Bin LATRD events data into images of event counts.

Given a collection of events data, a known image shape and an array of the desired time bin edges, make an image for each time bin, representing the number of events recorded at each pixel. Add the binned images to an array representing the full image stack.

Parameters

- **data** – LATRD data. Must have an `event_id` column and an `image_index` column.
- **image_size** – The (y, x), i.e. (slow, fast) dimensions (number of pixels) of the image.
- **cache** – Array representing the image stack, to which the binned events should be added. This might be a Zarr array, in which case it functions as an on-disk cache of the binned images.

6.3 Data

Tools for extracting data on cues and events from Tristan data files.

`tristan.data.`**cue_times**(*data: DataFrame*, *message: int*, *after: int | None = None*, *before: int | None = None*) → Array

Find the timestamps of all instances of a cue message in a Tristan data set.

The found timestamps are de-duplicated.

Parameters

- **data** – A DataFrame of LATRD data. Must contain one column for cue id messages and one for cue timestamps.
- **message** – The message code, as defined in the Tristan standard.
- **after** – Ignore instances of the specified message before this timestamp.

Returns

The timestamps, measured in clock cycles from the global synchronisation signal, de-duplicated.

`tristan.data.`**first_cue_time**(*data: DataFrame*, *message: int*, *after: int | None = None*) → DataFrame | None

Find the timestamp of the first instance of a cue message in a Tristan data set.

Parameters

- **data** – LATRD data. Must contain one 'cue_id' column and one 'cue_timestamp_zero' column. The two arrays are assumed to have the same length.
- **message** – The message code, as defined in the Tristan standard.
- **after** – Ignore instances of the specified message before this timestamp.

Returns

The timestamp, measured in clock cycles from the global synchronisation signal. If the message doesn't exist in the data set, this returns None.

`tristan.data.`**latrd_data**(*raw_file_paths: Iterable[str | Path]*, *keys: Iterable[str] = ('cue_id', 'cue_timestamp_zero', 'event_id', 'event_time_offset', 'event_energy')*) → DataFrame | dict[str, dask.array.core.Array]

A context manager to read LATRD data sets from multiple files.

The yielded DataFrame has a column for each of the specified LATRD data keys. Each key must be a valid LATRD data key and the chosen data sets must all have the same length. The data will be rechunked into partitions approximately the size of the default Dask array chunk size, but with chunk boundaries aligned with HDF5 file boundaries.

Parameters

- **raw_file_paths** – The paths of the raw LATRD data files.

- **keys** – The set of LATRD data keys to be read.

Yields
The data from all the files.

tristan.data.**pixel_index**(*location: _SupportsArray[dtype] | _NestedSequence[_SupportsArray[dtype]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *image_size: tuple[int, int]*) → _SupportsArray[dtype] | _NestedSequence[_SupportsArray[dtype]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Extract pixel coordinate information from an event location (event_id) message.

Translate a Tristan event location message to the index of the corresponding pixel in the flattened image array (i.e. numbered from zero, in row-major order).

The pixel coordinates of an event on a Tristan detector are encoded in a 32-bit integer location message (the event_id) with 26 bits of useful information. Extract the y coordinate (the 13 least significant bits) and the x coordinate (the 13 next least significant bits). Find the corresponding pixel index in the flattened image array by multiplying the y value by the size of the array in x, and adding the x value.

This function calls the Python built-in divmod and so can be broadcast over array-like data structures.

Parameters

- **location** – Event location message (an integer).
- **image_size** – Shape of the image array in (y, x), i.e. (slow, fast).

Returns
Index in the flattened image array of the pixel where the event occurred.

tristan.data.**seconds**(*timestamp: int*, *reference: int = 0*) → Quantity

Convert a Tristan timestamp to seconds, measured from a given reference timestamp.

The time between the provided timestamp and a reference timestamp, both provided as a number of clock cycles from the same time origin, is converted to units of seconds. By default, the reference timestamp is zero clock cycles, the beginning of the detector epoch.

Parameters

- **timestamp** – A timestamp in number of clock cycles, to be converted to seconds.
- **reference** – A reference time stamp in clock cycles.

Returns
The difference between the two timestamps in seconds.

tristan.data.**valid_events**(*data: DataFrame*, *start: int*, *end: int*) → DataFrame

Return those events that have a timestamp in the specified range.

Parameters

- **data** – LATRD data, containing an 'event_time_offset' column and optional 'event_id' and 'event_energy' columns.
- **start** – The start time of the accepted range, in clock cycles.
- **end** – The end time of the accepted range, in clock cycles.

Returns
The valid events.

CHAPTER

SEVEN

DIAGNOSTICS API

7.1 General

Diagnostic tools for Tristan detector. To be run before collection.

tristan.diagnostics.utils.**define_modules**(*det_config: Literal['1M', '2M', '10M'] = '10M'*) → dict[str, tuple]

Define the start and end pixel of each module in the Tristan detector.

Parameters
det_config (*TConfig, optional*) – Specify how many physical modules make up the Tristan detector currently in use. Available configurations: 1M, 2M, 10M. Defaults to "10M".

Returns
Start and end pixel value of each module - which are defined by a (x,y) tuple. For example a Tristan 1M will return {"0": ([0, 515], [0, 2069])}

Return type
dict[str, tuple]

tristan.diagnostics.utils.**module_cooordinates**(*det_config: Literal['1M', '2M', '10M'] = '10M'*) → dict[str, tuple]

Create a conversion table between module number and its location on the detector.

Parameters
det_config (*TConfig, optional*) – Specify how many physical modules make up the Tristan detector currently in use. Available configurations: 1M, 2M, 10M. Defaults to "10M".

Returns
effectively a conversion table mapping the module number to its location on the detector. For example a Trisstan 1M will return {"0": (0, 0)}

Return type
dict[str, tuple]

tristan.diagnostics.utils.**assign_files_to_modules**(*filelist: list[pathlib.Path | str]*, *det_config: Literal['1M', '2M', '10M'] = '10M'*)

tristan.diagnostics.utils.**get_full_file_list**(*filename_template: str | Path*)

Given a template filename, including directory, get a list of all the files using that template.

Parameters
filename_template (*str | Path*) – Template to look up in the directory.

Returns
A list of all the files found matching the template.

Return type
file_list(list[Path])

tristan.diagnostics.utils.**find_shutter_times**(*filelist*)

7.2 Logging configuration

Logging configuration for Tristan diagnostics.

tristan.diagnostics.diagnostics_log.**config**(*logfile: str | None = None*, *write_mode: str = 'a'*)

Configure the logger.

Parameters

- **logfile** (*str, optional*) – If passed, create a file handle for the logger to write a logfile output. Defaults to None.
- **write_mode** (*str, optional*) – Writing mode for the logfile output. Defaults to "a".

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX