ASL DRO

Tom Hampshire, Aaron Oliver-Taylor

Dec 14, 2021

CONTENTS:

CHAPTER

ONE

OVERVIEW

ASL DRO is software that can generate digital reference objects for Arterial Spin Labelling (ASL) MRI. It creates synthetic raw ASL data according to set acquisition and data format parameters, based on input ground truth maps for:

- Perfusion rate
- Transit time
- Intrinsic MRI parameters: M0, T1, T2, T2*
- Tissue segmentation (defined as a single tissue type per voxel)

Synthetic data is generated in Brain Imaging Data Structure format, comprising of a NIFTI image file and accompanying json sidecar containing parameters.

ASLDRO was developed to address the need to test ASL image processing pipelines with data that has a known ground truth. A strong emphasis has been placed on ensuring traceability of the developed code, in particular with respect to testing. The DRO pipelines uses a 'pipe and filter' architecture with 'filters' performing data processing, which provides a common interface between processing blocks.

1.1 How To Cite

If you use ASLDRO in your work, please include the following citation

1.2 How To Contribute

Got a great idea for something to implement in ASLDRO, or maybe you have just found a bug? Create an issue at https://github.com/gold-standard-phantoms/asldro/issues to get in touch with the development team and we'll take it from there.

1.2.1 Installation

ASLDRO can be installed as a module directly from the python package index. Once installed it can simply be run as a command-line tool. For more information how to use a python package in this way please see https://docs.python.org/3/installing/index.html

1.2.1.1 Python Version

We recommend using the latest version of Python. ASL DRO supports Python 3.7 and newer.

1.2.1.2 Dependencies

These distributions will be installed automatically when installing ASL DRO.

- nibabel provides read / write access to some common neuroimaging file formats
- numpy provides efficient calculations with arrays and matrices
- jsonschema provides an implementation of JSON Schema validation for Python
- nilearn provides image manipulation tools and statistical learning for neuroimaging data

1.2.1.3 Virtual environments

Use a virtual environment to manage the dependencies for your project, both in development and in production.

What problem does a virtual environment solve? The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

Python comes bundled with the `venv` module to create virtual environments.

Create an environment

Create a project folder and a `venv` folder within:

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

On Windows:

```
$ py -3 -m venv venv
```

Activate the environment

Before you work on your project, activate the corresponding environment:

```
$ . venv/bin/activate
```

On Windows:

```
> venv\Scripts\activate
```

Your shell prompt will change to show the name of the activated environment.

1.2.1.4 Install ASL DRO

Within the activated environment, use the following command to install ASL DRO:

```
$ pip install asldro
```

ASL DRO is now installed. Check out the *Quickstart* or go to the *Documentation Overview*.

1.2.2 Quickstart

ASLDRO has a Jupyter Notebook that runs in the cloud via mybinder.org. This is certainly the simplest way to try out the DRO, generate some data, and also provides an interactive way to generate customised input parameters. Check it out at https://notebooks.gesis.org/binder/v2/gh/gold-standard-phantoms/asldro/develop?filepath=asldro_example.ipynb

1.2.2.1 Getting started

Eager to get started? This page gives a good introduction to ASL DRO. Follow *Installation* to set up a project and install ASL DRO first.

Running with defaults

After installation the command line tool `asldro` will be made available. You can run:

```
asldro generate path/to/output_file.zip
```

to run the DRO generation as-per the ASL White Paper specification. The output file may be either .zip or .tar.gz.

Specifying a parameter file

Is it also possible to specify a parameter file, which will override any of the default values:

```
asldro generate --params path/to/input_params.json path/to/output_file.zip
```

Output default parameters

It is possible to create an example parameters file containing the model defaults by running:

```
asldro output params /path/to/input_params.json
```

which will create the `/path/to/input_params.json` file. The parameters may be adjusted as necessary and used with the 'generate' command.

For details on input parameters see *Parameters*.

For details on the output DRO data see *DRO Output Data*.

Output built-in ground truths

It is also possible to output the high-resolution ground-truth (HRGT) files. To get a list of the available data, type:

```
asldro output hrgt -h
```

To output the HRGT, type:

```
asldro output hrgt HRGT OUTPUT_DIR
```

where HRGT is the code of the files to download, and OUTPUT_DIR is the directory to output to.

Perform ASL quantification

ASLDRO has a built in ASL quantification module, for testing and verification of the DRO outputs:

```
asldro asl-quantify --params QUANT_PARAMS_PATH ASL_NIFTI_PATH OUTPUT_DIR
```

See *ASL Quantification* for more details.

Generate your own ground truths

There are also two ancillary command line features that assist with the creation of HRGT's. To create a valid HRGT:

```
asldro create-hrgt /path/to/hrgt_params.json /path/to/seg_mask.nii.gz /path/to/output_
↪dir
```

This takes a segmentation mask image, where each voxel is has an integer value that defines a region/tissue type, and a parameter file that describes values for each quantity to assign to each region, then concatenates volumes for each quantity into a 5D NIFTI, which is saved along with a JSON file describing the HRGT to the output directory, and are valid as an input HRGT for ASLDRO.

To create a segmentation mask from individual 'fuzzy' masks for each region type:

```
asldro combine-masks /path/to/combine_masks_params.json /path/to/output_image.nii.gz
```

This combines multiple fuzzy masks, where the voxel values define the fraction of that voxel that is occupied by the particular region/tissue, into a single segmentation mask image, that is valid for the create-hrgt command. The combine_masks_params.json defines the values to assign for each region, and an order of priority and thresholds.

For more information about how to use these features see *Making an input ground truth*.

1.2.2.2 Pipeline details

There are three pipelines available in ASLDRO

- The full ASL pipeline.
- A structural MRI pipeline (generates gradient echo, spin echo or inversion recovery signal).
- A ground truth pipeline that simply resamples the input ground truth to the specified resolution.

In a single instance of ASLDRO, the input parameter file can configure any number and configurations of these pipelines to be run, much in the way that this can be done on an MRI scanner.

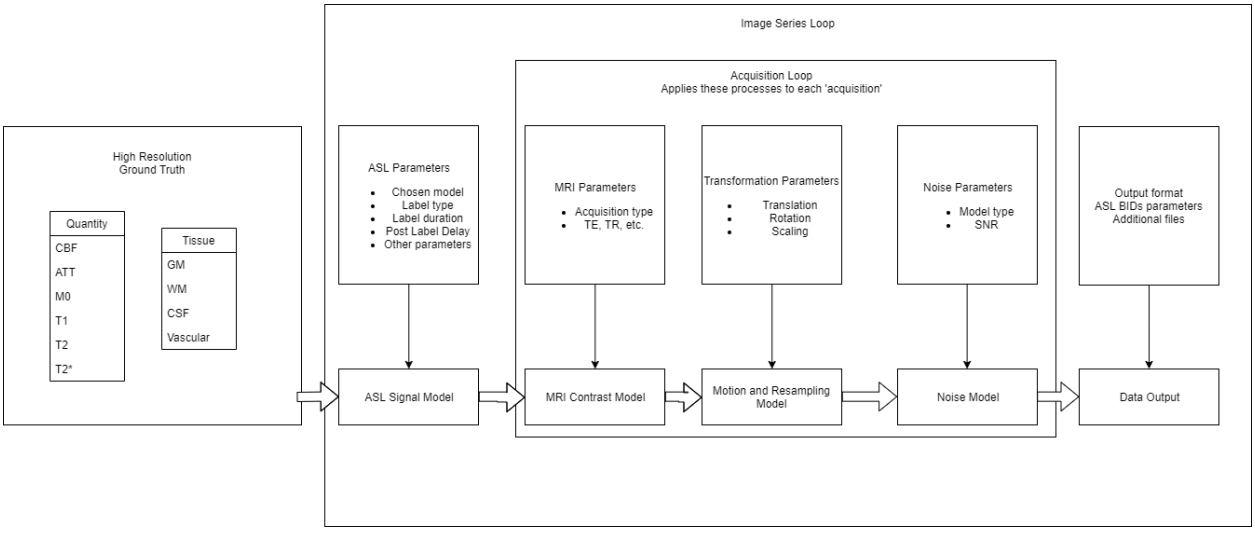
The full ASL pipeline comprises of:

1. Loading in the ground truth volumes.
2. Producing ΔM using the General Kinetic Model for the specified ASL parameters.
3. Generating synthetic M0, Control and Label volumes.
4. Applying motion
5. Sampling at the acquisition resolution
6. Adding instrument and physiological pseudorandom noise.

The structural pipeline excludes the General Kinetic Model, and just generates volumes with synthetic MR contrast. The ground truth pipeline only has the motion model and sampling.

Each pipeline outputs files in BIDS (https://bids.neuroimaging.io/) format, consisting of a NIFTI image file and accompanying json sidecar. In the case of an ASL image an additional '*_aslcontext.tsv' file is also generated which describes the ASL volumes present in the timeseries.

The DRO pipeline is summarised in this schematic (click to view full-size):



1.2.3 Parameters

This page describes the input parameters for ASL DRO. These should all be supplied over the command-line-interface as a json file. This file has two top-level objects. A "global_configuration" object, in which settings that apply to othe running of the instance of ASLDRO are set, and an image_series" object which defines the settings for each set of reference data that is generated.

Below is an example input parameter file, giving its overall structure.

```
{
    "global_configuration": {
        "ground_truth": "hrgt_icbm_2009a_nls_3t",
        "image_override": {},
        "parameter_override": {},
        "subject_label": "001"
    },
    "image_series": [
        {
            "series_type": "asl",
            "series_description": "an asl image series",
            "series_parameters": {
            }
        },
        {
            "series_type": "structural",
            "series_description": "a structural image series",
            "series_parameters": {
            }
        },
        {
            "series_type": "ground_truth",
            "series_description": "a ground truth image series",
            "series_parameters": {
            }
        }
    ]
}
```

1.2.3.1 Global Configuration

The "global_configuration" object is describes key-value pairs that configure how the instance of ASLDRO will run:

ground_truth (string) Determines which ground truth to be used. See *Input Ground Truth* for more information. At present the following are supported:

- "hrgt_icbm_2009a_nls_3t" - based on MNI ICBM 2009a Nonlinear Symmetric template, 3T relaxation times.
- "hrgt_icbm_2009a_nls_1.5t" - based on MNI ICBM 2009a Nonlinear Symmetric template, 1.5T relaxation times.

image_override (object) Contains key/value pairs which will override any of the ground truth images with a given value. For example, if you wish to use a constant "m0", the `image_override` should be set to `{"m0": n}`, where `n` is the floating point or integer value required.

parameter_override (object) Contains key/value pairs which will override any of the ground truth parameters. For example, you might want to change `lambda_blood_brain` to 0.85, in which case you would set `parameter_override` to `{"lambda_blood_brain": 0.85}`.

ground_truth_modulate (object) Contains key/value pairs which will modulate any of the ground truth images with a given scale and/or offset (i.e. $y = sx + c$ where s is the scaling factor, c is the x is the input voxel value, and y is the modulated voxel value and For example, you might want to apply a scale of 0.9 and an offset of 1 to the 'm0' image, therefore, the required dictionary input for 'ground_truth_modulate' would be: `{"m0":{"scale": 0.9, "offset": 1}}`. Note that the scaling is applied before the offset (see *asldro.filters.scale_offset_filter.ScaleOffsetFilter* for more details).

subject_label (string) Corresponds to the BIDS entity `subject`. Files will be saved within a sub-directory named 'sub-<subject_label>', and will be prefixed with 'sub-<subject_label>'. If not supplied will default to "001"

A more complete example is given below:

```
{
    "global_configuration": {
        "ground_truth": "hrgt_icbm_2009a_nls_3t",
        "image_override": {"m0": 5.2},
        "parameter_override": {"lambda_blood_brain": 0.85},
        "ground_truth_modulate": {
            "m0": {"scale": 0.9, "offset": 1},
            "t1": {"offset": 0.5},
            "t2": {"scale": 1.1}
        },
        "subject_label": "asldro01"
    }
}
```

1.2.3.2 Image Series

The "image_series" object is an array, with each entry an object that describes images to generate. This object has three key-value entries:

series_type (string) Specifies which image generation pipeline to run, see below for more details.

series_description (string, optional) A string that can be used to describe the image series. The value here is added to the field "descrip" in the output NIFTI header, and the BIDS field "Description" in the output .json sidecar of any files generated corresponding to the image series.

series_parameters (object, optional) An object containing key-value pairs that configure the image series. Some of these parameters are image series specific, for example the parameters that must be supplied as arrays for `series_type` "asl" are all singleton values for `series_type` "structural". Each `series_type` has its own set of default parameters. This is field is omitted then the pipeline will be run using the complete set of default values.

There are three different image generation pipelines built into ASLDRO:

asl The full ASL generation pipeline, constructing a time-series of m0, control and label volumes with perfusion signal generated by the General Kinetic Model, which is then encoded into gradient or spin echo contrast, motion is optionally applied, and an image is acquired at the specified acquisition resolution.

structural A single volume is generated using gradient echo, spin echo, or inversion recovery contrast, motion optionally applied and an image acquired at the specified acquisition resolution. Typically this is used togenerate the structural images such as T1w or FLAIR that ASL images are registered to.

ground_truth This pipeline simply resamples the input ground truth images to a specified resolution.

Parameters for each `series_type` are described below:

1.2.3.3 Series Type: ASL

The full ASL generation pipeline, comprises of constructing a time-series of m0, control and label volumes with perfusion signal generated by the General Kinetic Model, which is then encoded into gradient or spin echo contrast, motion is optionally applied, and an image is acquired at the specified acquisition resolution.

All strings are case-insensitive, for example "pcasl", "pCASL" and "PCASL" are all valid.

Images generated by the ASL pipeline will be stored in a subdirectory names "perf".

All parameters have defaults if omitted; the default parameters result in a "White Paper" style acquisition using the full General Kinetic Model [2] to generate the perfusion signal:

- pCASL, 1.8s label duration, 1.8s PLD.
- One M0, one Control and one Label volume.
- SNR 1000 (this results in a perfusion signal SNR of ~10).
- Acquisition matrix of [64, 64, 40], gives a voxel size of ~[3.1, 3.6, 4.7] mm
- Background suppression applied to the Control and Label volumes, with pulse times optimised for the T1 values in the ground truth, based on four pulses and a saturation time of 3.9 seconds, however the actual saturation time is 4.0 seconds to ensure all magnetisation is positive.

Single or Multiphase ASL

Single or multiphase ASL data can be generated using the parameter signal_time, depending on the values supplied:

- If `signal_time` is a single value, then single phase ASL data will be generated in accordance with the volumes in asl_context, i.e. a single post labelling delay.
- If `signal_time` is an array of values, then multiphase ASL data will be generated, where a set of volumes according to asl_context will be generated for each value of `signal_time`, i.e. a set of data for each post labelling delay. These images will be

In the ASL series pipeline, `signal_time` is iterated over in the outer loop, and `asl_context` is iterated over in the inner loop. All images are concatenated into a single 4D timeseries in the order of acquisition. For example:

```
{
    "series_type": "asl",
    "series_description": "multiphase asl",
    "series_parameters": {
        "label_duration": 1.0,
        "signal_time": [1.0, 1.25, 1.5],
        "asl_context": "control label"
    }
}
```

Will produce a 4D timeseries with volumes in this order:

1. control, PLD = 0.00s
2. label, PLD = 0.00s
3. control, PLD = 0.25s
4. label, PLD = 0.25s

5. control, PLD = 0.50s
6. label, PLD = 0.50s

signal_time (float or array, optional) The time in seconds after labelling at which the ASL signal should be generated. Note that this is equal to the `label_duration` + Post Label Delay, however the latter is not a defined input parameter. Defaults to `3.6`.

Single value parameters

The following parameters are applied to every volume that is generated by the DRO.

gkm_model (string, optional) Defines which model to use to generate the perfusion signal.

'full' Uses the full "Buxton" General Kinetic Model [2]

'whitepaper' Uses a simplified model derived from the single subtraction quantification equations [1]

Defaults to `"full"`. For implementation details see *GkmFilter*.

label_type (string, optional) Defines the type of ASL labelling: "pcasl", "casl" or "pasl". See *GkmFilter* for implementation details. Defaults to `"pcasl"`.

label_duration (float, optional) The temporal duration in seconds of the labelled bolus of blood. Defaults to `1.8`.

label_efficiency (float, optional): The degree of inversion of the magnetisation by the labelling pulses. Defaults to `0.85`.

acq_matrix (array of integers, optional) A 3-entry array defining the acquisition matrix size [Ni, Nj, Nk]. Defaults to `[64, 64, 20]`.

interpolation (string, optional) The type of interpolation to use when resampling the 'acquired' image. See :class:'.ResampleFilter' for implementation details. Can be:

'continuous' order 3 spline interpolation

'linear' order 1 linear interpolation

'nearest' nearest neighbour interpolation

Default value for an asl image series is `"linear"`.

acq_contrast (string, optional): Defines the MRI signal model to use. "ge" for Gradient Echo, "se" for Spin Echo. See *MriSignalFilter* for implementation details. Defaults to `"se"`.

desired_snr (float, optional) The signal-to-noise ratio of the acquired image. Note this is the base SNR of the image, not the SNR of the perfusion signal. A value of approximately 100 is comparable to the sort of SNR in a single ASL control-label pair. If this has value of 0, then no noise will be added. See *AddComplexNoiseFilter* for implementation details. Defaults to `1000.0`, which provides acceptable SNR for a single subtraction.

random_seed (int, optional) Seed to control the determinism of any pseudo-random behaviour in the pipeline, for example the noise added to images. Defaults to `0`.

output_image_type (string, optional) Specifies the image type of the output image: "magnitude", "complex". Defaults to `"magnitude"`.

background_supppression (object or boolean, optional) Defines whether background suppression is used. Background suppression comprises of a saturation pulse followed by multiple inversion pulses with inversion times chosen so that the magnetisation of tissues with different T1's are nulled at the point that the imaging excitation pulse is played out. Can take the form:

true Background suppression is used, with the default parameters.

false No background suppression is used.

object An object with key/value pairs defining parmaeters for the background suppression.

The default value for `background_suppression` is:

```
{
    "sat_pulse_time": 4.0,
    "sat_pulse_time_opt": 3.98,
    "pulse_efficiency": "ideal",
    "num_inv_pulses": 4,
    "apply_to_asl_context": ["label", "control"]
}
```

Background Suppression Parameters

Below are the valid background suppression parameters for when an object is used as the value for the parameter key `background_suppression`. For more information about the implementation of background suppression see *BackgroundSuppressionFilter*. Note that due to the way these parameters are validated, the values returned if an empty object is provided for the parameter `background_suppression` will differ to those listed in the default value above. Instead they will take the defaults listed below.

sat_pulse_time (float) The time in seconds between the saturation pulse and the imaging excitation pulse. Defaults to `4.0`.

inv_pulse_times (array of floats, optional) Array of the inversion times for each inversion pulse. Defined as the spacing between the inversion pulse and the imaging excitation pulse in seconds. If omitted then optimal inversion times will be calculated.

pulse_efficiency (string or float, optional) Defines the efficiency of the excitation pulse. Can take the values:

- "realistic" Pulse efficiencies are calculated according to a model based on their T1.
- "ideal" Inversion pulses are 100% efficient.
- A numeric value between -1.0 and 0.0, inclusive, explicitly defining the inversion efficiency. -1.0 is full inversion, and 0.0 is no inversion.

The default value is "ideal".

t1_opt (array of floats, optional) The T1 relaxation times, in seconds, to optimise the inversion times for. If omitted then the unique T1 values in the ground truth image for T1 will be used.

sat_pulse_time_opt (float, optional) If present, this value will be used in the pulse timing optimisation. This enables the use of a slightly longer `sat_pulse_time`, to ensure that the magnetisation is positive at the point of excitation. If omitted then `sat_pulse_time` will be used for the optimisation (default)

num_inv_pulses (int) The number of inversion pulses to generate optimised times for. Defaults to 4.

apply_to_asl_context A list defining which asl contexts should have background suppression applied for. The entries can either be `"control"`, `"label"`, or `"m0scan"`. Defaults to `["label", "control]`.

Array Parameters

The following parameters are all supplied as arrays, with each entry corresponding with the volumes defined in `asl_context`.

asl_context (string, optional): A list of the ASL volumes to simulate, any combination of "m0scan", "control" and "label", separated by a space. Defaults to `"m0scan control label"`.

echo_time (array of floats or object, optional): The time in seconds after the excitation pulse that the MRI signal is acquired. This parameter generally affects the T2 or T2* contrast. Default values are given for the value of `asl_context`:

"m0scan" 0.01

"control" 0.01

"label" 0.01

repetition_time (array of floats or object, optional): The time in seconds between successive excitation pulses. This parameter affects the T1 contrast. Default values are given for the value of `asl_context`:

"m0scan" 10.0

"control" 5.0

"label" 5.0

rot_z (array of floats or object, optional): Rotation of the ground truth model in world space about the z-axis in degrees. See *TransformResampleImageFilter* for implementation details. Defaults to `0.0` for every entry in `asl_context`.

rot_y (array of floats or object, optional): Rotation of the ground truth model in world space about the y-axis in degrees. Defaults to `0.0` for every entry in `asl_context`.

rot_x (array of floats or object, optional): Rotation of the ground truth model in world space about the x-axis in degrees. Defaults to `0.0` for every entry in `asl_context`.

transl_x (array of floats or object, optional): Translation of the ground truth model in world space along the x-axis in mm. Defaults to `0.0` for every entry in `asl_context`.

transl_y (array of floats or object, optional): Translation of the ground truth model in world space along the y-axis in mm. Defaults to `0.0` for every entry in `asl_context`.

transl_z (array of floats or object, optional): Translation of the ground truth model in world space along the z-axis in mm. Defaults to `0.0` for every entry in `asl_context`.

Array parameters can also be specified dynamically and generated automatically based on the entries in `asl_context`. This is handled in two different ways:

- `echo_time` and `repetition_time` have values that are defined for the types of entries in `asl_context`, and arrays of these values are constructed accordingly.
- The rotation and translation (`rot_*` and `transl_*`) values are drawn from probability distributions:

  gaussian Values are drawn from a normal/gaussian distribution with defined mean and standard deviation.

  uniform Values are drawn from a uniform distribution with defined minimum and maximum values.

  These are defined by an object with the following entries:

  distribution (string, defaults to `gaussian`): The probability distribution. `gaussian` for normal distribution and `uniform` for a uniform distribution.

mean (float, defaults to 0.0, required if `distribution=='gaussian'`). The mean value of the gaussian distribution.

sd (float, defaults to 0.0, required if `distribution=='gaussian'`). The standard deviation of the gaussian distribution.

min (float, required if `distribution=='uniform'`). The minimum value of the uniform distribution.

max (float, required if `distribution=='uniform'`). The maximum value of the uniform distribution.

seed (int, defaults to 0): The seed for the random number generator. Note this is independent of the `random_seed`. Each parameter will have its own random number generator assigned, which means they will have identical values if the same seed is assigned.

Generated values are rounded to four decimal places so that if the parameter file is saved as a JSON file the results can be reproduced.

For example:

```
{
    "asl_context": "m0scan m0scan control label control label control label",
    "echo_time": {
        "m0scan": 0.012,
        "control": 0.012,
        "label": 0.012
    },
    "repetition_time": {
        "m0scan": 10.0,
        "control": 4.5,
        "label": 4.5
    },
    "rot_x": {
        "distribution": "gaussian",
        "mean": 1.0,
        "sd": 0.1,
        "seed": 12345
    },
    "transl_y": {
        "distribution": "uniform",
        "min": 1.0,
        "max": 0.1,
        "seed": 12345
    }
}
```

Dynamically generates the following array parameters:

```
{
    "asl_context": "m0scan m0scan control label control label control label",
    "echo_time": [0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.12],
    "repetition_time": [10.0, 10.0, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5],
    "rot_x": [0.8576, 1.1264, 0.9129, 0.9741, 0.9925, 0.9259, 0.8632, 1.0649],
    "rot_y": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    "rot_z": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    "transl_x": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
    "transl_y": [0.7954, 0.7149, 0.2824, 0.3914, 0.648, 0.7005, 0.4615, 0.8319],
    "transl_z": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
}
```

1.2.3.4 Series Type: Structural

The structural pipeline simulates the acquisition of a single volume using gradient echo, spin echo, or inversion recoverycontrast, motion optionally applied and an image acquired at the specified acquisition resolution. Typically this is used togenerate the structural images such as T1w or FLAIR that ASL images are registered to.

All parameters have defaults if omitted; the default parameters result in spin echo contrast with TE=5ms and TR=300m and an acquisition voxel size of 1x1x1mm.

All strings are case-insensitive, for example “ge”, “GE” and “Ge” are all valid.

Images generated by the structural pipeline will be stored in a subdirectory names “anat”.

Single value parameters

The following parameters are applied to every volume that is generated by the DRO.

acq_matrix (array of integers, optional) A 3-entry array defining the acquisition matrix size [Ni, Nj, Nk]. Defaults to `[197, 233, 189]`.

acq_contrast (string, optional) Defines the MRI signal model to use. “ge” for Gradient Echo, “se” for Spin Echo. See *`MriSignalFilter`* for implementation details. Defaults to `"se"`.

desired_snr (float, optional) The signal-to-noise ratio. If this has value of 0, then no noise will be added. See *`AddComplexNoiseFilter`* for implementation details. Defaults to `100.0`.

random_seed (int, optional) Seed to control the determinism of any pseudo-random behaviour in the pipeline, for example the noise added to images. Defaults to `0`.

echo_time (float, optional) The time in seconds after the excitation pulse that the MRI signal is acquired. This parameter generally affects the T2 or T2* contrast. Defaults to `0.005`.

repetition_time (float, optional) The time in seconds between successive excitation pulses. This parameter affects the T1 contrast. Defaults to `0.3`.

rot_z (float, optional) Rotation of the ground truth model in world space about the z-axis in degrees. See `TransformResampleImageFilter` for implementation details. Defaults to `0.0`.

rot_y (float, optional) Rotation of the ground truth model in world space about the y-axis in degrees. Defaults to `0.0`.

rot_x (float, optional) Rotation of the ground truth model in world space about the x-axis in degrees. Defaults to `0.0`.

transl_x (float, optional) Translation of the ground truth model in world space along the x-axis in mm. Defaults to `0.0`.

transl_y (float, optional) Translation of the ground truth model in world space along the y-axis in mm. Defaults to `0.0`.

transl_z (float, optional) Translation of the ground truth model in world space along the z-axis in mm. Defaults to `0.0`.

interpolation (string, optional) The type of interpolation to use when resampling the ‘acquired’ image. See :class:’.ResampleFilter’ for implementation details. Can be:

‘continuous’ order 3 spline interpolation

‘linear’ order 1 linear interpolation

‘nearest’ nearest neighbour interpolation

Default value for a structural image series is `"linear"`.

output_image_type (string, optional) Specifies the image type of the output image: “magnitude”, “complex”. Defaults to `"magnitude"`.

modality (string, optional) Describes the intent of the generated image, for example “T1w”, “T2w”, “FLAIR”, “anat”. The actual image contrast may or may not actually be representative of this as it is dependent on other parameters. Output BIDS files are names according to this field. Defaults to `"anat"`.

1.2.3.5 Series Type: Ground Truth

The ground truth pipeline simply resamples the input ground truth images to a specified resolution. Individual NIFTI images are output for each ground truth quantity in the input image.

All parameters have defaults if omitted; the default parameters result no motion and a resampled matrix of [64, 64, 40].

Images generated by the Ground Truth pipeline will be stored in a subdirectory names “ground_truth”.

Single value parameters

The following parameters are applied to every volume that is generated by the DRO.

acq_matrix (array of integers, optional) A 3-entry array defining the acquisition matrix size [Ni, Nj, Nk]. Defaults to `[197, 233, 189]`.

rot_z (float, optional) Rotation of the ground truth model in world space about the z-axis in degrees. See *`TransformResampleImageFilter`* for implementation details. Defaults to `0.0`.

rot_y (float, optional) Rotation of the ground truth model in world space about the y-axis in degrees. Defaults to `0.0`.

rot_x (float, optional) Rotation of the ground truth model in world space about the x-axis in degrees. Defaults to `0.0`.

transl_x (float, optional) Translation of the ground truth model in world space along the x-axis in mm. Defaults to``0.0``.

transl_y (float, optional) Translation of the ground truth model in world space along the y-axis in mm. Defaults to `0.0`.

transl_z (float, optional) Translation of the ground truth model in world space along the z-axis in mm. Defaults to `0.0`.

interpolation (array of strings, optional) The type of interpolation to use when resampling the ‘acquired’ image. The first entry defines the type of interpolation for all ground truth volumes except for the `"seg_label"` volume. The second entry is the interpolation type for the `"seg_label"` volume. See :class:’.ResampleFilter’ for implementation details. Each entry can be:

‘continuous’ order 3 spline interpolation

‘linear’ order 1 linear interpolation

‘nearest’ nearest neighbour interpolation

Default value is `["linear", "nearest"]`

1.2.4 DRO Output Data

When ASLDRO's main pipeline is run using the `generate` command, for example:

```
asldro generate --params path/to/input_params.json path/to/output_archive.zip
```

the resulting DRO image data is saved in the archive, in accordance with the data structure and format as specified by BIDS (https://bids.neuroimaging.io/).

ASLDRO currently supports BIDS version 1.5.0, however because of the nature of DRO data there are some small deviations and non-official fields used. Deviations are described in the section *Deviations from the BIDS Standard*.

Each image series defined in the input parameters will have a NIFTI image and corresponding JSON sidecar generated. For ASL image series an additional *_aslcontext.tsv is generated which indicates the type of ASL volumes in the image.

These will be saved in a subdirectory `sub-<subject_label>` and then in the following subdirectories:

- 'perf' for series type 'asl'
- 'anat' for series type 'structural'
- 'ground_truth' for series type 'ground_truth'

Filenames are given by `sub-<subject_label>_acq-<series_number>_<modality_label>.<ext>`, where:

- `<subject_label>` is given by the global configuration parameter *subject_label* which defaults to "001"
- `<series_number>` is the order the particular image series is in the `"image_series"` array in the input parameters file (1 indexed, zero padded to 3 digits).
- **`<modality_label>` is based on series type:**

  asl determined by the input parameter `asl_context`. If `asl_context` only contains entries with `'m0scan'` then it will be `'m0scan'`, otherwise `'asl'`.

  structural determined by the input parameter `modality`, which can be "T1w" (default), "T2w", "FLAIR", "PDw", "T2starw", "inplaneT1", "PDT2", or "UNIT1".

  ground_truth the concatenation of 'ground-truth' and the name of the 'quantity' for the ground truth image, separated by a hyphen. Any underscores in the quantity name will be converted to hyphens.

For example, running the DRO with parameters to generate the following image series in order:

1. asl, `asl_context = "m0scan, control, label"`
2. asl, `asl_context = "control, label"`
3. asl, `asl_context = "m0scan"`
4. structural, `modality = "FLAIR"`
5. strutural, `modality = "T2w"`
6. structural, modality entry missing.
7. ground truth

Will result in the following files output

```
output_archive.zip
  |-- dataset_description.json
  |-- README
  |-- .bidsignore
  |-- sub-001
      |
      |-- perf
      |    |-- sub-001_acq-001_asl.nii.gz
      |    |-- sub-001_acq-001_asl.json
      |    |-- sub-001_acq-001_aslcontext.tsv
      |    |-- sub-001_acq-002_asl.nii.gz
      |    |-- sub-001_acq-002_asl.json
      |    |-- sub-001_acq-002_aslcontext.tsv
      |    |-- sub-001_acq-003_m0scan.nii.gz
      |
      |-- anat
      |    |-- sub-001_acq-004_FLAIR.nii.gz
      |    |-- sub-001_acq-004_FLAIR.json
      |    |-- sub-001_acq-005_T2w.nii.gz
      |    |-- sub-001_acq-005_T2w.json
      |    |-- sub-001_acq-006_T1w.nii.gz
      |    |-- sub-001_acq-006_T1w.json
      |
      |---ground_truth
          |-- sub-001_acq-007_Perfmap.nii.gz
          |-- sub-001_acq-007_Perfmap.json
          |-- sub-001_acq-007_ATTmap.nii.gz
          |-- sub-001_acq-007_ATTmap.json
          |-- sub-001_acq-007_T1map.nii.gz
          |-- sub-001_acq-007_T1map.json
          |-- sub-001_acq-007_T2map.nii.gz
          |-- sub-001_acq-007_T2map.json
          |-- sub-001_acq-007_T2starmap.nii.gz
          |-- sub-001_acq-007_T2starmap.json
          |-- sub-001_acq-007_M0map.nii.gz
          |-- sub-001_acq-007_M0map.json
          |-- sub-001_acq-007_dseg.nii.gz
          |-- sub-001_acq-007_dseg.json
```

1.2.4.1 ASLDRO version

ASLDRO can be run from released packages available at pypi.org, or directly from source code. When a release is created, the version number is manually created and added to the module version (i.e. setup.py), and a tag added to the git source control. However, this means that subsequent edits and commits based on a release will still have the module version as the derived release. So, there is an additional mechanism to ensure that data that are exported to BIDS are traceable to the actual git commit:

1. If no git repository information is found (e.g. installed from pypi using pip) then the version will be the release version.
2. If the git commit hash of the HEAD matches the most recent commit hash from the master branch then it is considered to be the release version.
3. If the git commit hash doesn't match the most recent commit to the master branch, or the repository is 'dirty' (uncommited changes) then the version will be generated using the command `git describe --tags --dirty` (see https://git-scm.com/docs/git-describe for more details).

For example:

```
v2.2.0-77-gc8678e5-dirty
```

- v2.2.0 is the parent branch (which has the tag v2.2.0)
- 77 is the number of commits that the current branch is ahead of the parent branch.
- gc8678e5 has two parts: the 'g' at the start indicates git, and the remainder is the abbreviated commit hash for the current branch.
- dirty indicates that when the DRO was run there were uncommited changes in the repository.

This requires git to be installed on the host system and for the master branch's git commit information to be available. If it isn't then the version number will be the release version appended with "-unverified", e.g. "v2.2.0-unverified".

This version information is included in:

- The JSON sidecars
- dataset_description.json

1.2.4.2 Deviations from the BIDS Standard

Ground Truth Image Series

BIDS specifies that parameter maps should be saved in the 'anat' folder, however ground truth parameter maps generated using the `ground_truth` image series are saved in 'ground_truth' folder.

Additional suffixes have been devised for non-supported parameter maps:

- Perfmap: Perfusion rate map.
- ATTmap: Transit time map.
- Lambdamap: Blood brain partition coefficient map.

The .bidsignore file has entries to ignore everything in the ground_truth folder and in addition the above non-supported suffixes.

Background Suppression

In the BIDS standard it is assumed that background suppression pulses comprise of:

1. **A saturation pulse of duration 0 occuring at the start of the labelling pulse,** e.g. the time betweeen the saturation pulse and the imaging excitation pulse is equal to `label_duration + post_label_delay`.
2. All inversion pulses occur after the start of the labelling pulse.

To allow for more possibilities for background timings, the following changes have been implemented:

BackgroundSuppressionPulseTime (modified) Negative values are permitted. A negative value indicates that the inversion pulse occurs before the label pulse has started.

BackgroundSuppressionSatPulseTime (new) The time in seconds between the saturation pulse and the imaging excitation pulse.

Multiphase ASL

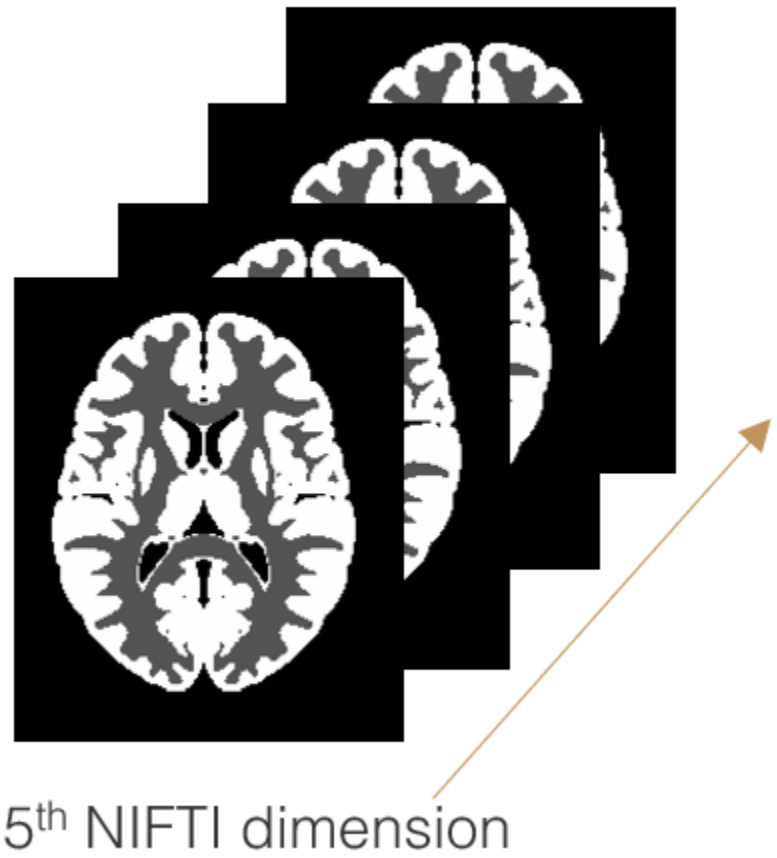
Multiphase ASL data can be generated by supplying the parameter signal_time as an array. For each value in this array the volumes defined in asl_context are generated. ASL BIDS does not currently support multiphase data, so the following has been implemented:

MultiphaseIndex (new) An array of integers, with one entry per volume in the ASL timeseries, indicating the index of the multiphase loop when each volume was generated.

PostLabelingDelay (modified) For multiphase data this is an array of the corresponding post labelling delay times for each multiphase index. For single phase ASL (i.e. only one PLD) then this is a single number.

1.2.5 Input Ground Truth

The input ground truth, known as the High Resolution Ground Truth (or HRGT) comprises of a 5-dimensional NIFTI image, and json parameter file that describes the NIFTI and also supplies any non-image ground truth parameters. 3D Volumes for each ground truth quantity are concatenated along the 5th NIFTI dimension.



1.2.5.1 Selecting a ground truth

Different HRGT's are selected by modifiying the "ground_truth" entry in the input parameter file to the name of the ground truth being used. For example:

```
{
    "global_configuration": {
        "ground_truth": "hrgt_icbm_2009a_nls_3t"
    }
}
```

will use the built-in ground truth "hrgt_icbm_2009a_nls_3t" (see below for more details of these datasets). In addition, it is possible to specify your own ground truth files by using one of the following:

```
{
    "global_configuration": {
        "ground_truth": "/path/to/nifti_file.nii"
    }
}
```

or:

```
{
    "global_configuration": {
        "ground_truth": {
            "nii": "/path/to/nifti_file.nii.gz",
            "json": "/path/to/json_file.json"
    }
}
```

In the two examples above, the first example assumes there is a JSON file at precisely the same path with the same filename, except for a '.json' extension instead of a '.nii'/'.nii.gz' extension. The second example uses an explicitly defined filename for each file type, and may have different paths.

Augmenting values

HRGT parameters and image values can be augmented using the input parameter file. See *Parameters* for more information.

1.2.5.2 Custom ground truth

ASLDRO supports the use of custom ground truths. These must adhere to a specific format to be valid, but are straightforward to make. ASLDRO comes with command-line tools to assist creating custom ground truths. For more information see *Making an input ground truth*

1.2.5.3 Built-in ground truths

ASLDRO comes with built-in ground truths:

hrgt_icbm_2009a_nls_3t

Normal adult 3T ground truth. A 1x1x1mm ground truth based on the MNI ICBM 2009a Nonlinear Symmetric template, obtained from http://www.bic.mni.mcgill.ca/ServicesAtlases/ICBM152NLin2009, with quantitative parameters set based on supplied masks. This ground truth has the following values for each tissue and quantity (corresponding to 3T):

Tissue	Perfusion Rate [ml/100g/min]	Transit Time [s]	T1 [s]	T2 [s]	T2* [s]	label
Grey Matter	60.00	0.80	1.330	0.080	0.066	1
White Matter	20.00	1.20	0.830	0.110	0.053	2
CSF	0.00	1000.0	3.000	0.300	0.200	3

label is the integer value assigned to the tissue in the seg_label volume.

hrgt_icbm_2009a_nls_1.5t

Normal adult 1.5T ground truth. A 1x1x1mm ground truth based on the MNI ICBM 2009a Nonlinear Symmetric template, obtained from http://www.bic.mni.mcgill.ca/ServicesAtlases/ICBM152NLin2009, with quantitative parameters set based on supplied masks. This ground truth has the following values for each tissue and quantity (corresponding to 1.5T):

Tissue	Perfusion Rate [ml/100g/min]	Transit Time [s]	T1 [s]	T2 [s]	T2* [s]	label
Grey Matter	60.00	0.80	1.100	0.092	0.084	1
White Matter	20.00	1.20	0.560	0.082	0.066	2
CSF	0.00	1000.0	3.000	0.400	0.300	3

label is the integer value assigned to the tissue in the seg_label volume.

qasper_3t

A ground truth based on the QASPER perfusion phantom (https://goldstandardphantoms.com/qasper), at 0.5x0.5x0.5mm resolution. Contains the inlet, porous, and outlet regions, and approximates the behaviour of the QASPER phantom by having spatially variable transit times, and perfusion rates that are normalised to an input flow rate of 1mL/min. By using the *ground truth modulate* feature, the scaling factor is then equal to the input flow rate.

Region	Perfusion Rate [ml/100g/min]	Transit Time [s]	T1 [s]	T2 [s]	T2* [s]	M0	λ [g/ml]	label
Inlet	$100\frac{Q}{N_{\text{inlet}}V\lambda_{\text{inlet}}}$	0.0 to 0.25	1.80	1.20	0.90	100	1.00	1
Porous	$100\frac{Q}{N_{\text{porous}}V\lambda_{\text{porous}}}$	0.25 to 10.0	1.80	0.20	0.10	32	0.32	2
Outlet	$100\frac{Q}{N_{\text{outlet}}V\lambda_{\text{porous}}}$	10.0 to 20.0	1.80	1.20	0.90	100	1.00	3

Q is the bulk flow rate, 1mL/min, N is the number of voxels in a given region, and V is the voxel volume in mL. Label is the integer value assigned to the tissue in the seg_label volume.

The following single value parameters are also present:

t1_arterial_blood 1.8 s

magnetic_field_strength 3.0 T

Transit time maps for each region are procedurally generated:

Inlet The transit time is proportional to the z position, between 0.0s at z=-20mm and 0.25s at z=4.75mm, which is at the interface between the first and second porous layers.

Porous The transit time is generated by subtracting 60 gaussian functions located half way along each arteriole at z=4.75mm from a constant value. This is scaled so that the maximum value is 10.0s, and the minimum (at the centre of each gaussian) is 0.25s.

Outlet The transit time is proportional to the z position, between 10.0s at z=0.0mm, and 20.0s at the end of the perfusion chamber at z=45.0mm.

Note: The QASPER ground truth is not an accurate representation of the true QASPER phantom behaviour. It cannot be used for comparing experimental QASPER data. It is intended for developing and testing image processing pipelines for QASPER data.

ASL Data generated by ASLDRO for the QASPER ground truth will not show the 'wash-out' effect that is seen in real data once the labelled bolus has been delivered and fresh, unlabelled perfusate has flowed into the perfusion chamber. This is because the GKM implementation in ASLDRO assumes that once the magnetisation has reached the tissue it remains there and decays at a rate according to T1. However, in the QASPER phantom there is no exchange into tissue and the perfusate keeps moving. The GKM solution needs to be extended to incorporate a term for this.

1.2.6 Making an input ground truth

The input ground truth consists of two separate files:

- A NIFTI image which has each ground truth quantity concatenated across the 5th dimension.
- A json file which contains specified fields that describe the ground truth.

The following quantities are required:

- Perfusion rate: The rate of delivery of arterial blood to an organ.
- Tissue Transit Time: The time following labelling for the perfusion signal to reach a voxel.
- M0: The equilibrium magnetisation within a voxel.
- T1: Spin-lattice (longitudinal) relaxation time constant.
- T2: Spin-spin (transverse) relaxation time constant.
- T2*: Spin-spin (transverse) relaxation time constant, including time-invariant magnetic field inhomogeneities.
- Segmentation Mask: Voxels are assigned a number corresponding with the specific tissue type.

The blood-brain-partition coefficient can be supplied as an image or as a single whole-brain value in the accompanying JSON file.

By definition the HRGT does not have any partial volume effects; only a single tissue type is in each voxel. This is in contrast with the usual 'fuzzy' segmentation masks that are supplied as part of templates.

The json parameter file describes the following:

- The quantities present in the HRGT NIFTI file, in the order they are concatenated along the 5th dimension.
- The units that correspond with the quantities (note that ASLDRO expects all units to be SI)
- The names of the tissues and the corresponding value in the segmentation mask.

- Parameters: the blood-brain partition coefficient, T1 of arterial blood and the magnetic field strength that the HRGT is for. ASLDRO does not use the magnetic field strength parameter, other than to include in BIDS field "MagneticFieldStrength".

Below is the json parameter file for the built in *hrgt_icbm_2009a_nls_3t* ground truth

```
{
    "quantities": [
        "perfusion_rate", "transit_time", "t1", "t2", "t2_star", "m0", "seg_label"],
    "units": ["ml/100g/min", "s", "s", "s", "s", "", ""],
    "segmentation": {
        "grey_matter": 1,
        "white_matter": 2,
        "csf": 3
    },
    "parameters": {
        "lambda_blood_brain": 0.9,
        "t1_arterial_blood": 1.65,
        "magnetic_field_strength": 3
    }
}
```

The background is assumed to have value 0.

1.2.6.1 How to construct a ground truth

The general procedure for constructing your own ground truth is as follows.

1. Obtain the segmentation masks for each of the tissues you wish to represent in the ground truth. For example this could be from an atlas image, or could simply be automated/manual segmentations from a single scan.
2. If these masks are fuzzy they will need to be thresholded to create a binary mask.
3. Combine these individual binary masks into a Label Map: where each tisue is represented by an integer value. This ensures that there is only one tissue type per voxel.
4. Using this label mask create volumes where values are assigned to each tissue. There should be one volume per quantity, all of the required quantities are necessary for the ASLDRO pipelines to run, but more can be added. All the voxels of a particular tissue could be set to the same value, or some variation could be introduced.
5. Concatenate these volumes along the 5th dimension and save as a NIFTI file. The Label Map is the *'seg_label'* volume, this needs to be included.
6. **Construct the json parameter file:**
    1. list the names of the quantities in the order they are concatenated in the 'quantities' array.
    2. list the corresponding units of the quantities in the 'units' array. If the quantity is unitless then use an empty string "".
    3. Add key/value pairs comprising of the tissue name and the integer value it is assigned in the Label Map image to the object 'segmentation'. Valid tissue names are:
        - 'background': identifies background signal
        - 'grey_matter': identifies grey matter
        - 'white_matter': identifies white matter
        - 'csf': identifies cerebrospinal fluid
        - 'vascular': identifies blood vessels

- 'lesion': identifies lesions

4. Create the 'parameters' object, this must have the entries for 't1_arterial_blood', and 'magnetic_field_strength'. If 'lambda_blood_brain' is not an image quantity then it is also required here.

7. Test your ground truth with ASLDRO. Once loaded it will be validated, so you will receive an error message if there are any issues.

1.2.6.2 Command line tools to generate ground truths

ASLDRO comes with two command line functions to assist with generating custom ground truths:

create-hrgt

```
asldro create-hrgt /path/to/hrgt_params.json /path/to/seg_mask.nii.gz /path/to/output_
↪dir
```

This command creates a ground truth that is valid for ASLDRO using the following:

`/path/to/hrgt_params.json` Path to a JSON file (must have extension .json) that describes how to construct the ground truth. It has the following objects:

'label_values' An array of integers, which must be the same as the integer values in the accompanying seg_mask.nii.gz, including zero values. The order in this array defines the order in the objects 'label_names', and the the array for each quantity in 'quantities'.

'label_names' An array of strings, defining the names for the regions.

'quantities' Comprises multiple objects, one for each quantity to be represented in the HRGT. The value for each quantity is an array of floats, the same length as the number of regions, and corresponding to the order in 'label_values'.

'units' An array of strings, defining the units for each quantity. Must match the number of quantities, and corresponds with the order.

'properties' This requires two objects, 't1_arterial_blood' and 'magnetic_field_strength', both of which are floats. Additional properties are allowed and will propagate to the output hrgt JSON file. Note that to be a valid input HRGT to ASLDRO, either one of the quantities, or one of the parameters must be 'lambda_blood_brain'.

`/path/to/seg_mask.nii.gz` A NIFTI file (must have extension .nii or .nii.gz) which defines segmentation regions. It is recommended that this is an integer data type, however floating point data is accepted and will have a ceiling operation applied to it.

`/path/to/output_dir` The directory to output the HRGT files to. hrgt.nii.gz and hrgt.json will be created here, if the files already exist they will be overwritten.

An example of the hrgt_params.json is shown below, the quantity values are used in the built-in hrgt_icbm_2009a_nls_3t ground truth:

```
{
   "label_values": [0, 1, 2, 3],
   "label_names": ["background", "grey_matter", "white_matter", "csf"],
   "quantities": {
      "perfusion_rate": [0.0, 60.0, 20.0, 0.0],
      "transit_time": [0.0, 0.8, 1.2, 1000.0],
      "t1": [0.0, 1.33, 0.83, 3.0],
```

(continues on next page)

(continued from previous page)

```
        "t2": [0.0, 0.08, 0.11, 0.3],
        "t2_star": [0.0, 0.066, 0.053, 0.2],
        "m0": [0.0, 74.62, 64.73, 68.06]
    },
    "units": ["ml/100g/min", "s", "s", "s", "s", ""],
    "parameters": {
        "t1_arterial_blood": 1.80,
        "lambda_blood_brain": 0.9,
        "magnetic_field_strength": 3.0
    }
}
```

combine-masks

```
asldro combine-masks /path/to/combine_masks_params.json /path/to/output_image.nii.gz
```

This command combines multiple 'fuzzy' masks into a single segmentation mask. A fuzzy mask is defined as having voxel values between 0 and 1 that define the fraction of that voxel that is occupied by the particular region/tissue; they can represent partial volumes. ASLDRO defines the HRGT to not have any partial volume effects, i.e. only a single tissue/region type per voxel. The `combine-masks` function combines these fuzzy masks in a defined manner, using a combination of thresholding and a priority order for each mask.

A voxel will be assigned a region value if the following conditions are all met:

1. Its fuzzy_mask value is higher than a defined threshold (default is 0.05)
2. Its fuzzy_mask value is highest out of any other masks that also have non-zero values in that voxel.
3. If it jointly has the same highest value as one or more other masks, it must have a higher priority assigned than the competing regions.

The following arguments are required:

`/path/to/combine_masks_params.json` Path to a JSON file (must have extension .json) which describes how the masks are to be combined. This has the following objects:

'mask_files' An array of paths to the mask files to combine. Each file must be in NIFTI format with extension .nii or .nii.gz. They must all have the same dimensions, and have the same affine matrix (srow_x, srow_y, and srow_z) so that they correspond to the same location in world-space.

'region_values' An array of integers, to assign to each mask region. Their order corresponds with the order of the mask files.

'region_priority' An array of integers defining the priority order for the regions, 1 being the highest priority.

'threshold' (optional) A number between 0 and 1 that defines the threshold that voxel values must be greater than to be considered for array assignment. Default value is 0.05.

`/path/to/output_image.nii.gz` Path to a NIFTI (must have extension .nii or.nii.gz) image that will be created.

If only a single mask file is supplied, only the threshold condition will be applied. However 'region_values' and 'region_priority' must still be present.

An example of the combine_masks_params.json file is shown below:

```
{
   "mask_files": [
      "/path/to/mask_1.nii.gz",
      "/path/to/mask_2.nii.gz",
      "/path/to/mask_3.nii.gz"
   ],
   "region_values": [1, 2, 3],
   "region_priority": [2, 1, 3],
   "threshold": 0.13
}
```

1.2.7 ASL Quantification

ASDRO has a command line function to perform quantification on valid ASL BIDS data, comprising a NIFTI, JSON sidecar and *aslcontext.tsv file

```
asldro asl-quantify --params /path/to/quant_params.json /path/to/asl.nii.gz /path/to/
↪output_dir
```

where:

path/to/quant_params Path to a JSON file (must have extension .json) providing parameters for the quantifiction calculation. If supplied they will override any values that are within the ASL image's BIDS sidecar. This allows assumed values to be overidden for example. It has the following objects:

QuantificationModel (string) defaults to "whitepaper" (see *AslQuantificationFilter* for options)

ArterialSpinLabelingType (string) "PCASL", "CASL" or "PASL"

PostLabelingDelay (float) The post labeling delay in seconds.

LabelingDuration (float) The label duration in seconds (pCASL/CASL only)

BolusCutOffDelayTime (float) The bolus cutoff delay time (PASL only)

LabelingEfficiency (float) The efficiency of the labeling pulse

T1ArterialBlood (float) If not supplied the default value is based on the value of the BIDS field "MagneticFieldStrength":

1.5 Tesla 1.35s

3.0 Tesla 1.65s

/path/to/asl.nii.gz The path to a valid ASL NIFTI file (must have extension .nii or .nii.gz). It is assumed that in the same location there are also corresponding `/path/to/asl.json` and `/path/to/aslcontext.tsv` files.

path/to/output_dir The directory to output the generated perfusion rate/CBF maps to, comprising a NIFTI image and JSON sidecar. The files will be given the same filename as the input NIFTI, with '_cbf' appended before the extension.

For more details on implementation see the `asl_quantification()` pipeline.

An example of the quant_params.json file is given below:

```
{
    "QuantificationModel": "whitepaper",
    "PostLabelingDelay": 1.8,
    "LabelingEfficiency": 0.85,
    "BloodBrainPartitionCoefficient": 0.9,
    "T1ArterialBlood": 1.65,
    "ArterialSpinLabelingType": "PCASL",
    "LabelingDuration": 1.8,
}
```

1.2.8 Development

Development of this software project must comply with a few code styling/quality rules and processes:

- Pylint must be used as the linter for all source code. A linting configuration can be found in `.pylintrc`. There should be no linting errors when checking in code.
- Before committing any files, black must be run with the default settings in order perform autoformatting on the python files, and prettier must be run (.prettierrc in project root) to autoformat JSON files.
- Before pushing any code, make sure the CHANGELOG.md is updated as per the instructions in the CHANGELOG.md file.
- The project's software development processes must be used (found here).

1.2.9 References

1.2.10 API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

1.2.10.1 asldro package

Subpackages

asldro.containers package

Submodules

asldro.containers.image module

Classes for image encapsulation Used to create a standard interface for ND images which can be instantiated with either NIFTI files or using numpy arrays

class asldro.containers.image.**BaseImageContainer**(*data_domain: str = 'SPATIAL_DOMAIN'*, *image_type=None*, *metadata=None*, ***kwargs*)

Bases: `abc.ABC`

An abstract (interface) for an ND image. Defines a set of accessors :param data_domain: defines the data domain as SPATIAL_DOMAIN or INVERSE_DOMAIN (default is SPATIAL_DOMAIN) :param image_type: the image type. Must be one of: - REAL_IMAGE_TYPE - IMAGINARY_IMAGE_TYPE - MAGNITUDE_IMAGE_TYPE - PHASE_IMAGE_TYPE - COMPLEX_IMAGE_TYPE if this is not specified, it will

be set to REAL_IMAGE_TYPE for scalar image dtypes and COMPLEX_IMAGE_TYPE for complex dtypes :param metadata: a metadata dictionary which is associated with the image data. This might contain, for example, timing parameters associated with the image acquisition.

abstract property affine
Return a 4x4 numpy array with the image affine transformation

abstract as_nifti() → *asldro.containers.image.NiftiImageContainer*
Return the image container as a NiftiImageContainer. If the container is already a NiftiImageContainer, return self

abstract as_numpy() → *asldro.containers.image.NumpyImageContainer*
Return the image container as a NumpyImageContainer. If the container is already a NumpyImageContainer, return self

clone() → *asldro.containers.image.BaseImageContainer*
Makes a deep copy of all member variables in a new ImageContainer

abstract property has_nifti
Returns True if the image has an associated nifti container

abstract property image
Return the image data as a numpy array

property metadata
Get the metadata

abstract property shape
Returns the shape of the image [x, y, z, t, etc]

abstract property space_units
Uses the NIFTI header xyzt_units to extract the space units. Returns one of: 'meter' 'mm' 'micron'

abstract property time_step_seconds
Return the time step in seconds

abstract property time_units
Uses the NIFTI header xyzt_units to extract the time units. Returns one of: 'sec' 'msec' 'usec'

abstract property voxel_size_mm
Returns the voxel size in mm

class asldro.containers.image.**NiftiImageContainer**(*nifti_img: Union[nibabel.Nifti1Image, nibabel.Nifti2Image] = None*, ***kwargs*)
Bases: *asldro.containers.image.BaseImageContainer*

A container for an ND image. Must be initialised with a nibabel Nifti1Image or Nifti2Image

property affine
Return a 4x4 numpy array with the image affine transformation

as_nifti() → *asldro.containers.image.NiftiImageContainer*
Returns self

as_numpy() → *asldro.containers.image.NumpyImageContainer*
Return the NiftiImageContainer as a NumpyImageContainer.

property has_nifti
Returns True if the image has an associated nifti container

property header
Returns the NIFTI header if initialised from a NIFTI file, othewise returns None

property image
Return the image data as a numpy array. Returns data in the type it is created (i.e. won't convert to float64 as .get_fdata() will)

property nifti_type
Return the type of NIFTI data contained here (nib.Nifti1Image or nib.Nifti2Image)

property shape
Returns the shape of the image [x, y, z, t, etc]

property space_units
Uses the NIFTI header xyzt_units to extract the space units. Returns one of: 'meter' 'mm' 'micron'

property time_step_seconds
Return the time step in seconds

property time_units
Uses the NIFTI header xyzt_units to extract the time units. Returns one of: 'sec' 'msec' 'usec'

property voxel_size_mm
Returns the voxel size in mm

class asldro.containers.image.**NumpyImageContainer**(*image: numpy.ndarray*, *affine: numpy.ndarray = numpy.eye*, *space_units: str = 'mm'*, *time_units: str = 'sec'*, *voxel_size=numpy.array*, *time_step=1.0*, ***kwargs*)

Bases: *asldro.containers.image.BaseImageContainer*

A container for an ND image. Must be initialised with a a numpy array and some associated metadata.

property affine
Return a 4x4 numpy array with the image affine transformation

as_nifti() → *asldro.containers.image.NiftiImageContainer*
Return the NumpyImageContainer as a NiftiImageContainer.

as_numpy() → *asldro.containers.image.NumpyImageContainer*
Returns self

property has_nifti
Returns True if the image has an associated nifti container

property header
Returns the NIFTI header if initialised from a NIFTI file, othewise returns None

property image
Return the image data as a numpy array

property shape
Returns the shape of the image [x, y, z, t, etc]

property space_units
Uses the NIFTI header xyzt_units to extract the space units. Returns one of: 'meter' 'mm' 'micron'

property time_step_seconds
Return the time step in seconds

property time_units
Uses the NIFTI header xyzt_units to extract the time units. Returns one of: 'sec' 'msec' 'usec'

property voxel_size_mm
Returns the voxel size in mm

Module contents

asldro.data package

Submodules

asldro.data.affine_test_data module

Test data for test_resampling.py

asldro.data.filepaths module

Constants with data file paths

Module contents

asldro.filters package

Submodules

asldro.filters.acquire_mri_image_filter module

AcquireMriImageFilter Class

class asldro.filters.acquire_mri_image_filter.**AcquireMriImageFilter**

Bases: *asldro.filters.filter_block.FilterBlock*

A filter block that simulates the acquisition of an MRI image based on ground truth inputs.

Combines:

1. *MriSignalFilter*
2. *TransformResampleImageFilter*
3. *AddComplexNoiseFilter*

Returns AddComplexNoiseFilter

Inputs

Input Parameters are all keyword arguments for the AcquireMriImageFilter.add_inputs() member function. They are also accessible via class constants, for example *AcquireMriImageFilter.KEY_T1*

Parameters

- **'t1'** (BaseImageContainer) – Longitudinal relaxation time in seconds
- **'t2'** (BaseImageContainer) – Transverse relaxation time in seconds
- **'t2_star'** (BaseImageContainer) – Transverse relaxation time including time-invariant magnetic field inhomogeneities.
- **'m0'** (BaseImageContainer) – Equilibrium magnetisation
- **'mag_eng'** – Added to M0 before relaxation is calculated, provides a means to encode another signal into the MRI signal (non-complex data).

- **'acq_contrast'** (*str*) – Determines which signal model to use: "ge" (case insensitive) for Gradient Echo, "se" (case insensitive) for Spin Echo, "ir" (case insensitive) for Inversion Recovery.
- **'echo_time'** (*float*) – The echo time in seconds
- **'repetition_time'** (*float*) – The repeat time in seconds
- **'excitation_flip_angle'** (*float*) – Excitation pulse flip angle in degrees. Only used when "acq_contrast" is "ge" or "ir".
- **'inversion_flip_angle'** (*float, optional*) – Inversion pulse flip angle in degrees. Only used when acq_contrast is "ir".
- **'inversion_time'** – The inversion time in seconds. Only used when acq_contrast is "ir".
- **'image_flavour'** (*str, optional*) – sets the metadata image_flavour in the output image to this.
- **'translation'** – $[\Delta x, \Delta y, \Delta z]$ amount to translate along the x, y and z axes.
- **'rotation'** (*Tuple[float, float, float], optional*) – $[\theta_x, \theta_y, \theta_z]$ angles to rotate about the x, y and z axes in degrees(-180 to 180 degrees inclusive).
- **'rotation_origin'** (*Tuple[float, float, float], optional*) – $[x_r, y_r, z_r]$ coordinates of the point to perform rotations about.
- **target_shape** (*Tuple[int, int, int], optional*) – $[L_t, M_t, N_t]$ target shape for the acquired image
- **'interpolation'** (*str, optional*) – Defines the interpolation method for the resampling:

  'continuous' order 3 spline interpolation (default method for ResampleFilter)

  'linear' order 1 linear interpolation

  'nearest' nearest neighbour interpolation
- **'snr'** (*float or int*) – the desired signal-to-noise ratio (>= 0). A value of zero means that no noise is added to the input image.
- **'reference_image'** (BaseImageContainer, *optional*) – The reference image that is used to calculate the amplitude of the random noise to add to *'image'*. The shape of this must match the shape of *'image'*. If this is not supplied then *'image'* will be used for calculating the noise amplitude.

Outputs

Parameters **'image'** (BaseImageContainer) – Synthesised MRI image.

KEY_ACQ_CONTRAST = 'acq_contrast'

KEY_ECHO_TIME = 'echo_time'

KEY_EXCITATION_FLIP_ANGLE = 'excitation_flip_angle'

KEY_IMAGE = 'image'

KEY_IMAGE_FLAVOUR = 'image_flavour'

KEY_INTERPOLATION = 'interpolation'

KEY_INVERSION_FLIP_ANGLE = 'inversion_flip_angle'

KEY_INVERSION_TIME = 'inversion_time'

KEY_M0 = 'm0'

KEY_MAG_ENC = 'mag_enc'

KEY_REF_IMAGE = 'reference_image'

KEY_REPETITION_TIME = 'repetition_time'

KEY_ROTATION = 'rotation'

KEY_ROTATION_ORIGIN = 'rotation_origin'

KEY_SNR = 'snr'

KEY_T1 = 't1'

KEY_T2 = 't2'

KEY_T2_STAR = 't2_star'

KEY_TARGET_SHAPE = 'target_shape'

KEY_TRANSLATION = 'translation'

asldro.filters.add_complex_noise_filter module

Add complex noise filter block

class asldro.filters.add_complex_noise_filter.**AddComplexNoiseFilter**
Bases: *asldro.filters.filter_block.FilterBlock*

A filter that adds normally distributed random noise to the real and imaginary parts of the fourier transform of the input image.

Inputs

Input parameters are all keyword arguments for the `AddComplexNoiseFilter.add_inputs()` member function. They are also accessible via class constants, for example *AddComplexNoiseFilter.KEY_SNR*.

Parameters

- **`'image'`** (BaseImageContainer) – An input image which noise will be added to. Can be either scalar or complex. If it is complex, normally distributed random noise will be added to both real and imaginary parts.
- **`'snr'`** (*float or int*) – the desired signal-to-noise ratio (>= 0). A value of zero means that no noise is added to the input image.
- **`'reference_image'`** (BaseImageContainer, *optional*) – The reference image that is used to calculate the amplitude of the random noise to add to *'image'*. The shape of this must match the shape of *'image'*. If this is not supplied then *'image'* will be used for calculating the noise amplitude.

Outputs

Parameters **`'image'`** (BaseImageContainer) – The input image with complex noise added.

The noise is added pseudo-randomly based on the state of numpy.random. This should be appropriately controlled prior to running the filter

KEY_IMAGE = 'image'

KEY_REF_IMAGE = 'reference_image'

KEY_SNR = 'snr'

asldro.filters.add_noise_filter module

Add noise filter

class asldro.filters.add_noise_filter.**AddNoiseFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that adds normally distributed random noise to an input image.

Inputs

Input parameters are all keyword arguments for the `AddNoiseFilter.add_inputs()` member function. They are also accessible via class constants, for example *AddNoiseFilter.KEY_SNR*.

Parameters

- **`'image'`** (`BaseImageContainer`) – An input image which noise will be added to. Can be either scalar or complex. If it is complex, normally distributed random noise will be added to both real and imaginary parts.
- **`'snr'`** (*float or int*) – the desired signal-to-noise ratio (>= 0). A value of zero means that no noise is added to the input image.
- **`'reference_image'`** (`BaseImageContainer`, *optional*) – The reference image that is used to calculate the amplitude of the random noise to add to *'image'*. The shape of this must match the shape of *'image'*. If this is not supplied then *'image'* will be used for calculating the noise amplitude.

Outputs

Parameters **`'image'`** (`BaseImageContainer`) – The input image with noise added.

'reference_image' can be in a different data domain to the *'image'*. For example, *'image'* might be in the inverse domain (i.e. fourier transformed) whereas *'reference_image'* is in the spatial domain. Where data domains differ the following scaling is applied to the noise amplitude:

- *'image'* is *SPATIAL_DOMAIN* and 'reference_image' is *INVERSE_DOMAIN*: 1/N
- *'image'* is *INVERSE_DOMAIN* and 'reference_image' is *SPATIAL_DOMAIN*: N

Where N is *reference_image.image.size*

The noise is added pseudo-randomly based on the state of numpy.random. This should be appropriately controlled prior to running the filter

Note that the actual SNR (as calculated using "A comparison of two methods for measuring the signal to noise ratio on MR images", PMB, vol 44, no. 12, pp.N261-N264 (1999)) will not match the desired SNR under the following circumstances:

- *'image'* is *SPATIAL_DOMAIN* and *'reference_image'* is *INVERSE_DOMAIN*
- *'image'* is *INVERSE_DOMAIN* and *'reference_image'* is *SPATIAL_DOMAIN*

In the second case, performing an inverse fourier transform on the output image with noise results in a spatial domain image where the calculated SNR matches the desired SNR. This is how the *AddNoiseFilter* is used within the `AddComplexNoiseFilter`

`KEY_IMAGE = 'image'`

`KEY_REF_IMAGE = 'reference_image'`

`KEY_SNR = 'snr'`

asldro.filters.affine_matrix_filter module

Affine Matrix Filter

class asldro.filters.affine_matrix_filter.**AffineMatrixFilter**(*name: str = 'Unknown'*)

Bases: *asldro.filters.basefilter.BaseFilter*

A filter that creates an affine transformation matrix based on input parameters for rotation, translation, and scaling.

Conventions are for RAS+ coordinate systems only

Inputs

Input Parameters are all keyword arguments for the `AffineMatrixFilter.add_inputs()` member function. They are also accessible via class constants, for example *AffineMatrixFilter.KEY_ROTATION*

Parameters

- **'rotation'** (*Tuple[float, float, float], optional*) – [θ_x, θ_y, θ_z] angles to rotate about the x, y and z axes in degrees(-180 to 180 degrees inclusive), defaults to (0, 0, 0)
- **'rotation_origin'** (*Tuple[float, float, float], optional*) – [x_r, y_r, z_r] coordinates of the point to perform rotations about, defaults to (0, 0, 0)
- **'translation'** (*Tuple[float, float, float], optional*) – [Δx, Δy, Δz] amount to translate along the x, y and z axes. defaults to (0, 0, 0)
- **'scale'** (*Tuple[float, float, float], optional*) – [s_x, s_y, s_z] scaling factors along each axis, defaults to (1, 1, 1)
- **'affine'** (*np.ndarray(4), optional*) – 4x4 affine matrix to apply transformation to, defaults to *numpy.eye(4)*
- **'affine_last'** (*np.ndarray(4), optional*) – input 4x4 affine matrix that is applied last, defaults to *numpy.eye(4)*

Outputs

Once run, the filter will populate the dictionary `AffineMatrixFilter.outputs` with the following entries

Parameters

- **'affine'** (*np.ndarray(4)*) – 4x4 affine matrix with all transformations combined.
- **'affine_inverse'** (*np.ndarray(4)*) – 4x4 affine matrix that is the inverse of *'affine'*

The output affine matrix is calculated as follows:

$$\mathbf{M} = \mathbf{BSTT_r R_z R_y R_x T_r}^{-1}\mathbf{M_{in}}$$

where,

$\mathbf{A}$ = Existing affine matrix

$\mathbf{B}$ = Affine matrix to combine last

$$\mathbf{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{scaling matrix}$$

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{translation matrix}$$

$$\mathbf{T_r} = \begin{pmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{translation to rotation centre matrix}$$

$$\mathbf{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{rotation about x matrix}$$

$$\mathbf{R_y} = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{rotation about y matrix}$$

$$\mathbf{R_z} = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{rotation about z matrix}$$

```
KEY_AFFINE = 'affine'
KEY_AFFINE_INVERSE = 'affine_inverse'
KEY_AFFINE_LAST = 'affine_last'
KEY_ROTATION = 'rotation'
KEY_ROTATION_ORIGIN = 'rotation_origin'
KEY_SCALE = 'scale'
KEY_TRANSLATION = 'translation'
```

asldro.filters.append_metadata_filter module

AppendMetadataFilter

class asldro.filters.append_metadata_filter.**AppendMetadataFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that can add key-value pairs to the metadata dictionary property of an image container. If the supplied key already exists the old value will be overwritten with the new value. The input image container is modified and a reference passed to the output, i.e. no copy is made.

Inputs

Input Parameters are all keyword arguments for the `AppendMetadataFilter.add_inputs()` member function. They are also accessible via class constants, for example *AppendMetadataFilter.KEY_METADATA*

Parameters

- **`'image'`** (`BaseImageContainer`) – The input image to append the metadata to
- **`'metadata'`** (*dict*) – dictionary of key-value pars to append to the metadata property of the input image.

Outputs

Once run, the filter will populate the dictionary `AppendMetadataFilter.outputs` with the following entries

Parameters **`'image'`** – The input image, with the input metadata merged.

`KEY_IMAGE = 'image'`

`KEY_METADATA = 'metadata'`

asldro.filters.asl_quantification_filter module

ASL quantification filter class

class asldro.filters.asl_quantification_filter.**AslQuantificationFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that calculates the perfusion rate for arterial spin labelling data.

Inputs

Input Parameters are all keyword arguments for the `AslQuantificationFilter.add_input()` member function. They are also accessible via class constants, for example *AslQuantificationFilter.KEY_CONTROL*

Parameters

- **`'control'`** (`BaseImageContainer`) – the control image (3D or 4D timeseries)
- **`'label'`** (`BaseImageContainer`) – the label image (3D or 4D timeseries)
- **`'m0'`** (`BaseImageContainer`) – equilibrium magnetisation image
- **`'label_type'`** (*str*) – the type of labelling used: “pasl” for pulsed ASL “pcasl” or “casl” for for continuous ASL.
- **`'lambda_blood_brain'`** (*float*) – The blood-brain-partition-coefficient (0 to 1 inclusive)

- **'label_duration'** (*float*) – The temporal duration of the labelled bolus, seconds (0 or greater). For PASL this is equivalent to TI_1
- **'post_label_delay'** (*float or List[float]*) – The duration between the end of the labelling pulse and the imaging excitation pulse, seconds (0 or greater). For PASL this is equivalent to TI. If 'model'=='full' then this must be a list and the length of this must match the number of unique entries in 'multiphase_index'.
- **'label_efficiency'** (*float*) – The degree of inversion of the labelling (0 to 1 inclusive)
- **'t1_arterial_blood'** (*float*) – Longitudinal relaxation time of arterial blood, seconds (greater than 0)
- **'t1_tissue'** (*float or BaseImageContainer*) – Longitudinal relaxation time of the tissue, seconds (greater than 0). Required if 'model'=='full'
- **'model'** (*str*) – defines which model to use
  - 'whitepaper' uses the single-subtraction white paper equation
  - 'full' least square fitting to the full GKM.
- **'multiphase_index'** – A list the same length as the fourth dimension of the label image that defines which phase each image belongs to, and is also the corresponding index in the 'post_label_delay' list. Required if 'model'=='full'.

Outputs

Parameters **'perfusion_rate'** (BaseImageContainer) – map of the calculated perfusion rate

If 'model'=='full' the following are also output:

Parameters

- **'transit_time'** (BaseImageContainer) – The estimated transit time in seconds.
- **'std_error'** (BaseImageContainer) – The standard error of the estimate of the fit.
- **'perfusion_rate_err'** – One standard deviation error in the fitted

perfusion rate. :type 'perfusion_rate_err': BaseImageContainer :param 'transit_time_err': One standard deviation error in the fitted

transit time.

Quantification Model

The following equations are used to calculate the perfusion rate, depending on the input model:

'whitepaper' simplified single subtraction equations [1].

- for pCASL/CASL see *AslQuantificationFilter.asl_quant_wp_casl*
- for PASL see *AslQuantificationFilter.asl_quant_wp_pasl*.

'full' Lease squares fitting to the full General Kinetic Model [2]. See *AslQuantificationFilter.asl_quant_lsq_gkm*.

```
ESTIMATION_ALGORITHM = {'full':  'Least Squares fit to the General Kinetic Model for\n
```

FIT_IMAGE_NAME = {'perfusion_rate_err': 'RCBFErr', 'std_error': 'FITErr', 'transit_t

FIT_IMAGE_UNITS = {'perfusion_rate_err': 'ml/100g/min', 'std_error': 'a.u.', 'transi

FULL = 'full'

KEY_CONTROL = 'control'

KEY_LABEL = 'label'

KEY_LABEL_DURATION = 'label_duration'

KEY_LABEL_EFFICIENCY = 'label_efficiency'

KEY_LABEL_TYPE = 'label_type'

KEY_LAMBDA_BLOOD_BRAIN = 'lambda_blood_brain'

KEY_M0 = 'm0'

KEY_MODEL = 'model'

KEY_MULTIPHASE_INDEX = 'multiphase_index'

KEY_PERFUSION_RATE = 'perfusion_rate'

KEY_PERFUSION_RATE_ERR = 'perfusion_rate_err'

KEY_POST_LABEL_DELAY = 'post_label_delay'

KEY_STD_ERROR = 'std_error'

KEY_T1_ARTERIAL_BLOOD = 't1_arterial_blood'

KEY_T1_TISSUE = 't1_tissue'

KEY_TRANSIT_TIME = 'transit_time'

KEY_TRANSIT_TIME_ERR = 'transit_time_err'

M0_TOL = 1e-06

WHITEPAPER = 'whitepaper'

static asl_quant_lsq_gkm(*control: numpy.ndarray*, *label: numpy.ndarray*, *m0_tissue: numpy.ndarray*, *lambda_blood_brain: numpy.ndarray*, *label_duration: float*, *post_label_delay: List[float]*, *label_efficiency: float*, *t1_arterial_blood: float*, *t1_tissue: numpy.ndarray*, *label_type: str*) → dict

Calculates the perfusion and transit time by least-squares fitting to the ASL General Kinetic Model [2].

Fitting is performed using `scipy.optimize.curve_fit`.

See *GkmFilter* and `GkmFilter.calculate_delta_m_gkm` for implementation details of the GKM function.

Parameters

- **control** (*np.ndarray*) – control signal, must be 4D with signal for each post labelling delay on the 4th axis. Must have same dimensions as `label`.
- **label** (*np.ndarray*) – label signal, must be 4D with signal for each post labelling delay on the 4th axis. Must have same dimensions as `control`.
- **m0_tissue** (*np.ndarray*) – equilibrium magnetisation of the tissue.
- **lambda_blood_brain** (*np.ndarray*) – tissue partition coefficient in g/ml.
- **label_duration** (*float*) – duration of the labelling pulse in seconds.

- **post_label_delay** (*np.ndarray*) – array of post label delays, must be equal in length to the number of 3D volumes in `control` and `label`.
- **label_efficiency** (*float*) – The degree of inversion of the labelling pulse.
- **t1_arterial_blood** (*float*) – Longitudinal relaxation time of the arterial blood in seconds.
- **t1_tissue** (*np.ndarray*) – Longitudinal relaxation time of the tissue in seconds.
- **label_type** (*str*) – The type of labelling: pulsed ('pasl') or continuous ('casl' or 'pcasl').

Returns

A dictionary containing the following np.ndarrays:

'perfusion_rate' The estimated perfusion rate in ml/100g/min.

'transit_time' The estimated transit time in seconds.

'std_error' The standard error of the estimate of the fit.

'perfusion_rate_err' One standard deviation error in the fitted perfusion rate.

'transit_time_err' One standard deviation error in the fitted transit time.

Return type dict

`control`, `label`, `m0_tissue`, `t1_tissue` and `lambda_blood_brain` must all have the same dimensions for the first 3 dimensions.

static asl_quant_wp_casl(*control: numpy.ndarray*, *label: numpy.ndarray*, *m0: numpy.ndarray*, *lambda_blood_brain: float*, *label_duration: float*, *post_label_delay: float*, *label_efficiency: float*, *t1_arterial_blood: float*) → numpy.ndarray

Performs ASL quantification using the White Paper equation for (pseudo)continuous ASL [1].

$$f = \frac{6000 \cdot \lambda \cdot (\mathrm{SI}_{\mathrm{control}} - \mathrm{SI}_{\mathrm{label}}) \cdot e^{\frac{\mathrm{PLD}}{T_{1,b}}}}{2 \cdot \alpha \cdot T_{1,b} \cdot \mathrm{SI}_{\mathrm{M0}} \cdot (1 - e^{-\frac{\tau}{T_{1,b}}})}$$

where,

$f =$ perfusion rate in ml/100g/min

$\mathrm{SI}_{\mathrm{control}} =$ control image signal

$\mathrm{SI}_{\mathrm{label}} =$ label image signal

$\mathrm{SI}_{\mathrm{M0}} =$ equilibrium magnetision signal

$\tau =$ label duration

PLD = Post Label Delay

$T_{1,b} =$ longitudinal relaxation time of arterial blood

$\alpha =$ labelling efficiency

$\lambda =$ blood-brain partition coefficient

Parameters

- **control** (*np.ndarray*) – control image, $\mathrm{SI}_{\mathrm{control}}$
- **label** (*np.ndarray*) – label image $\mathrm{SI}_{\mathrm{label}}$
- **m0** (*np.ndarray*) – equilibrium magnetisation image, $\mathrm{SI}_{\mathrm{M0}}$
- **lambda_blood_brain** (*float*) – blood-brain partition coefficient in ml/g, λ

- **label_duration** (*float*) – label duration in seconds, τ
- **post_label_delay** (*float*) – duration between the end of the label pulse and the start of the image acquisition in seconds, PLD
- **label_efficiency** (*float*) – labelling efficiency, α
- **t1_arterial_blood** (*float*) – longitudinal relaxation time of arterial blood in seconds, $T_{1,b}$

Returns the perfusion rate in ml/100g/min, f

Return type np.ndarray

static asl_quant_wp_pasl(*control: numpy.ndarray*, *label: numpy.ndarray*, *m0: numpy.ndarray*, *lambda_blood_brain: float*, *bolus_duration: float*, *inversion_time: float*, *label_efficiency: float*, *t1_arterial_blood: float*) → numpy.ndarray

Performs ASL quantification using the White Paper equation for pulsed ASL [1].

$$f = \frac{6000 \cdot \ \lambda \cdot (\mathrm{SI}_{\mathrm{control}} - \mathrm{SI}_{\mathrm{label}}) \cdot e^{\frac{\mathrm{TI}}{T_{1,b}}}}{2 \cdot \alpha \cdot \mathrm{TI}_1 \cdot \mathrm{SI}_{\mathrm{M0}}}$$

where,

$f =$ perfusion rate in ml/100g/min

$\mathrm{SI}_{\mathrm{control}} =$ control image signal

$\mathrm{SI}_{\mathrm{label}} =$ label image signal

$\mathrm{SI}_{\mathrm{M0}} =$ equilibrium magnetisation signal

$\mathrm{TI} =$ inversion time

$\mathrm{TI}_1 =$ bolus duration

$T_{1,b} =$ longitudinal relaxation time of arterial blood

$\alpha =$ labelling efficiency

$\lambda =$ blood-brain partition coefficient

Parameters

- **control** (*np.ndarray*) – control image, $\mathrm{SI}_{\mathrm{control}}$
- **label** (*np.ndarray*) – label image, $\mathrm{SI}_{\mathrm{label}}$
- **m0** (*np.ndarray*) – equilibrium magnetisation image, $\mathrm{SI}_{\mathrm{M0}}$
- **lambda_blood_brain** (*float*) – blood-brain partition coefficient in ml/g, λ
- **inversion_time** (*float*) – time between the inversion pulse and the start of the image acquisition in seconds, TI
- **bolus_duration** (*float*) – temporal duration of the labelled bolus in seconds, defined as the duration between the inversion pulse and the start of the bolus cutoff pulses (QUIPPSS, Q2-TIPS etc), TI_1
- **label_efficiency** (*float*) – labelling efficiency, α
- **t1_arterial_blood** (*float*) – longitudinal relaxation time of arterial blood in seconds, $T_{1,b}$

Returns the perfusion rate in ml/100g/min, f

Return type np.ndarray

asldro.filters.background_suppression_filter module

Background Suppression Filter

class asldro.filters.background_suppression_filter.**BackgroundSuppressionFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that simulates a background suppression pulse sequence on longitudinal magnetisation. It can either use explicitly supplied pulse timings, or calculate optimised pulse timings for specified T1s.

Inputs

Input Parameters are all keyword arguments for the `BackgroundSuppressionFilter.add_inputs()` member function. They are also accessible via class constants, for example `CombineTimeSeriesFilter.KEY_T1`.

Parameters

- **`'mag_z'`** (`BaseImageContainer`) – Image of the initial longitudinal magnetisation. Image data must not be a complex data type.
- **`'t1'`** (`BaseImageContainer`) – Image of the longitudinal relaxation time. Image data must be greater than 0 and non-compex. Also its shape should match the shape of `'mag_z'`.
- **`'sat_pulse_time'`** (*float*) – The time, in seconds between the saturation pulse and the imaging excitation pulse. Must be greater than 0.
- **`'inv_pulse_times'`** (*list[float], optional*) – The inversion times for each inversion pulse, defined as the spacing between the inversion pulse and the imaging excitation pulse. Must be greater than 0. If omitted then optimal inversion times will be calculated for `'num_inv_pulses'` number of pulses, and the T1 times given by `'t1_opt'`.
- **`'t1_opt'`** (*list[float]*) – T1 times, in seconds to optimise the pulse inversion times for. Each must be greater than 0, and if omitted then the unique values in the input `t1` will be used.
- **`'mag_time'`** (*float*) – The time, in seconds after the saturation pulse to sample the longitudinal magnetisation. The output magnetisation will only reflect the pulses that will have run by this time. Must be greater than 0. If omitted, defaults to the same value as `'sat_pulse_time'`. If `'mag_time'` is longer than `'sat_pulse_time'`, then this difference will be added to both `'sat_pulse_time'` and also `'inv_pulse_times'` (regardless of whether this has been supplied as an input or optimised values calculated). If the pulse timings already include an added delay to ensure the magnetisation is positive then this parameter should be omitted.
- **`'num_inv_pulses'`** – The number of inversion pulses to calculate optimised timings for. Must be greater than 0, and this parameter must be present if `'inv_pulse_times'` is omitted.
- **`'pulse_efficiency'`** (*str or float*) – Defines the efficiency of the inversion pulses. Can take the values:

  'realistic' Pulse efficiencies are calculated according to a model based on the T1. See *BackgroundSuppressionFilter.calculate_pulse_efficiency* for details on implementation.

  'ideal' Inversion pulses are 100% efficient.

-1 to 0 The efficiency is defined explicitly, with -1 being full inversion and 0 no inversion.

Outputs

Once run, the filter will populate the dictionary `BackgroundSuppressionFilter.outputs` with the following entries:

Parameters

- **`'mag_z'`** (`BaseImageContainer`) – The longitudinal magnetisation at t=``mag_time``.
- **`'inv_pulse_times'`** (*`list[float]`*) – The inversion pulse timings.

Metadata

The output `'mag_z'` inherits metadata from the input `'mag_z'`, and then has the following entries appended:

background_suppression `True`

background_suppression_inv_pulse_timing `'inv_pulse_times'`

background_suppression_sat_pulse_timing `'sat_pulse_time'`

background_suppression_num_pulses The number of inversion pulses.

Background Suppression Model

Details on the model implemented can be found in *`BackgroundSuppressionFilter.calculate_mz`*

Details on how the pulse timings are optimised can be found in *`BackgroundSuppressionFilter.optimise_inv_pulse_times`*

`EFF_IDEAL = 'ideal'`

`EFF_REALISTIC = 'realistic'`

`KEY_INV_PULSE_TIMES = 'inv_pulse_times'`

`KEY_MAG_TIME = 'mag_time'`

`KEY_MAG_Z = 'mag_z'`

`KEY_NUM_INV_PULSES = 'num_inv_pulses'`

`KEY_PULSE_EFFICIENCY = 'pulse_efficiency'`

`KEY_SAT_PULSE_TIME = 'sat_pulse_time'`

`KEY_T1 = 't1'`

`KEY_T1_OPT = 't1_opt'`

`M_BACKGROUND_SUPPRESSION = 'background_suppression'`

`M_BSUP_INV_PULSE_TIMING = 'background_suppression_inv_pulse_timing'`

`M_BSUP_NUM_PULSES = 'background_suppression_num_pulses'`

`M_BSUP_SAT_PULSE_TIMING = 'background_suppression_sat_pulse_timing'`

`static calculate_mz`(*initial_mz: numpy.ndarray*, *t1: numpy.ndarray*, *inv_pulse_times: list*, *sat_pulse_time: float*, *mag_time: float*, *inv_eff: numpy.ndarray*, *sat_eff: numpy.ndarray = 1.0*) → numpy.ndarray

Calculates the longitudinal magnetisation after a sequence of background suppression pulses [4]

Parameters

- **`initial_mz`** (*`np.ndarray`*) – The initial longitudinal magnetisation, $M_z(t = 0)$

- **t1** (*np.ndarray*) – The longitudinal relaxation time, T_1
- **inv_pulse_times** (*list[float]*) – Inversion pulse times, with respect to the imaging excitation pulse, $\{\tau_i, \tau_{i+1}...\tau_{M-1}, \tau_M\}$
- **mag_time** (*float*) – The time at which to calculate the longitudinal magnetisation, t, cannot be greater than sat_pulse_time
- **sat_pulse_time** (*float*) – The time between the saturation pulse and the imaging excitation pulse, Q.
- **inv_eff** (*np.ndarray*) – The efficiency of the inversion pulses, χ .-1 is complete inversion.
- **sat_eff** (*np.ndarray*) – The efficiency of the saturation pulses, ψ. 1 is full saturation.

Returns The longitudinal magnetisation after the background suppression sequence

Return type np.ndarray

Equation

The longitudinal magnetisation at time t after the start of a background suppression sequence has started is calculated using the equation below. Only pulses that have run at time t contribute to the calculated magnetisation.

$$M_z(t) = M_z(t=0) \cdot (1 + ((1-\psi)-1)\chi^n e^{-\frac{t}{T_1}} + \sum_{m=1}^{n} (\chi^m - \chi^{m-1}) e^{-\frac{\tau_m}{T_1}}) \tag{1.1}$$

where

$M_z(t)$ = longitudinal magnetisation at time t

Q = the delay between the saturation pulse and imaging excitation pulse

ψ = saturation pulse efficiency, $0 \leq \psi \leq 1$

χ = inversion pulse efficiency, $-1 \leq \chi \leq 0$

τ_m = inversion time of the m^{th} pulse

T_1 = longitudinal relaxation time

(1.9)

static calculate_pulse_efficiency(*t1: numpy.ndarray*) → numpy.ndarray

Calculates the pulse efficiency per T1 based on a polynomial fit [3].

Parameters **t1** (*np.ndarray*) – t1 times to calculate the pulse efficiencies for, seconds.

Returns The pulse efficiencies, χ

Return type np.ndarray

Equation

$$\chi = \begin{cases} -0.998 \\ -\left(\begin{array}{r} -2.245\times10^{-15} T_1^4 \\ +2.378\times10^{-11} T_1^3 \\ -8.987\times10^{-8} T_1^2 \\ +1.442\times10^{-4} T_1 \\ +9.1555\times10^{-1} \end{array} \right) \\ -0.998 \end{cases} \tag{1.10}$$

static optimise_inv_pulse_times(*sat_time: float*, *t1: numpy.ndarray*, *pulse_eff: numpy.ndarray*, *num_pulses: int*, *method: str = 'Nelder-Mead'*) → scipy.optimize.OptimizeResult

Calculates optimised inversion pulse times for a background suppression pulse sequence.

Parameters

- **sat_time** (*float*) – The time, in seconds between the saturation pulse and the imaging excitation pulse, Q.
- **t1** (*np.ndarray*) – The longitudinal relaxation times to optimise the pulses for, T_1.
- **pulse_eff** (*np.ndarray*) – The inversion pulse efficiency, χ. corresponding to each `t1` entry.
- **num_pulses** (*int*) – The number of inversion pulses to optimise times for, N. Must be greater than 0.
- **method** (*str, optional*) – The optimisation method to use, see `scipy.optimize.minimize` for more details. Defaults to "Nelder-Mead".

Raises **ValueError** – If the number of pulses is less than 1.

Returns The result from the optimisation

Return type OptimizeResult

Equation

A set of optimimal inversion times, $\{\tau_i, \tau_{i+1} ... \tau_{M-1}, \tau_M\}$ are calculated by minimising the sum-of-squares of the magnetisation of all the T1 species in `t1_opt`:

$$\min \left(\sum_i^N M_z^2(t = Q, T_{1,i}, \chi, \psi, \tau) + \sum_i^N \begin{cases} 1 & M_z(t = Q, T_{1,i}, \chi, \psi, \tau) < 0 \\ 0 & M_z(t = Q, T_{1,i}, \chi, \psi, \tau) \geq 0 \end{cases} \right) \tag{1.15}$$

(1.16)

where

$N =$ The number of T_1 species to optimise for (1.17)

(1.18)

asldro.filters.basefilter module

BaseFilter classes and exception handling

class asldro.filters.basefilter.**BaseFilter**(*name: str = 'Unknown'*)

Bases: abc.ABC

An abstract base class for filters. All filters should inherit this class

add_child_filter(*child:* asldro.filters.basefilter.BaseFilter, *io_map: Mapping[str, str] = None*)

See documentation for *add_parent_filter*

add_input(*key: str*, *value*)

Adds an input with a given key and value. If the key is already in the inputs, an RuntimeError is raised

add_inputs(*input_dict: Mapping[str, Any]*, *io_map: Mapping[str, str] = None*, *io_map_optional: bool = False*)

Adds multiple inputs via a dictionary. Optionally, maps the dictionary onto different input keys using an io_map. :param input_dict: The input dictionary :param io_map: The dictionary used to perform the mapping. All keys and values must be strings. For example: As an example: {

"one": "two", "three": "four"

} will map inputs keys of “one” to “two” AND “three” to “four”. If io_map is None, no mapping with be performed. :param io_map_optional: If this is False, a KeyError will be raised if the keys in the io_map are not found in the input_dict. :raises KeyError: if keys required in the mapping are not found in the input_dict

add_parent_filter(*parent:* asldro.filters.basefilter.BaseFilter, *io_map: Mapping[str, str] = None*)
Add a parent filter (the inputs of this filter will be connected to the output of the parents). By default, the ALL outputs of the parent will be directly mapped to the inputs of this filter using the same KEY. This can be overridden by supplying io_map. e.g. io_map = {

“output_key1”:”input_key1”, “output_key2”:”input_key2”, . . . }

will map the output of the parent filter with a key of “output_key1” to the input of this filter with a key of “input_key1” etc. If *io_map* is defined ONLY those keys which are explicitly listed are mapped (the others are ignored)

run(*history=None*)
Calls the _run class on all parents (recursively) to make sure they are up-to-date. Then maps the parents’ outputs to inputs for this filter. Then calls the _run method on this filter.

exception asldro.filters.basefilter.**BaseFilterException**(*msg: str*)
Bases: Exception

Exceptions for this modules

exception asldro.filters.basefilter.**FilterInputKeyError**
Bases: Exception

Used to show an error with a filter’s input keys e.g. multiple values have been assigned to the same input

exception asldro.filters.basefilter.**FilterInputValidationError**
Bases: Exception

Used to show an error when running the validation on the filter’s inputs i.e. when running _validate_inputs()

exception asldro.filters.basefilter.**FilterLoopError**
Bases: Exception

Used when a loop is detected in the filter chain

asldro.filters.bids_output_filter module

asldro.filters.combine_fuzzy_masks_filter module

Combined Fuzzy Masks Filter

class asldro.filters.combine_fuzzy_masks_filter.**CombineFuzzyMasksFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter for creating a segmentation mask based on one or more ‘fuzzy’ masks.

Inputs

Input Parameters are all keyword arguments for the CombineFuzzyMasksFilter.add_input() member function. They are also accessible via class constants, for example *CombineFuzzyMasksFilter.KEY_THRESHOLD*

Parameters

- **'fuzzy_mask'** (BaseImageContainer *or list[*BaseImageContainer*]*) – Fuzzy mask images to combine. Each mask should have voxel values between 0 and 1, defining the fraction of that region in each voxel.

- **'region_values'** (*int or list[int]*) – A list of values to assign to each region in the output 'seg_mask'. The order corresponds with the order of the masks in 'fuzzy_mask'.
- **'region_priority'** (*list[int] or int*) – A list of priority order for the regions, 1 being the highest priority. The order corresponds with the order of the masks in 'fuzzy_mask', and all values must be unique. If 'fuzzy_mask' is a single image then this input can be omitted.
- **'threshold'** (*float, optional*) – The threshold value, below which a region's contributions to a voxel are ignored. Must be between 0 and 1.0. Defaults to 0.05.

Outputs

Once run, the filter will populate the dictionary `CombineFuzzyMasksFilter.outputs` with the following entries

Parameters **'seg_label'** (BaseImageContainer) – A segmentation mask image constructed from the inputs, defining exclusive regions (one region per voxel). The image data type is numpy.int16.

KEY_FUZZY_MASK = 'fuzzy_mask'

KEY_REGION_PRIORITY = 'region_priority'

KEY_REGION_VALUES = 'region_values'

KEY_SEG_MASK = 'seg_mask'

KEY_THRESHOLD = 'threshold'

asldro.filters.combine_time_series_filter module

Combine Time Series Filter

class asldro.filters.combine_time_series_filter.**CombineTimeSeriesFilter**

Bases: *asldro.filters.basefilter.BaseFilter*

A filter that takes, as input, as set of ImageContainers. These should each represent a single time point in a time series acquisition. As an output, these ImageContainers will be concatenated across the 4th (time) dimension and their metadata combined with the following rules:

- if all values of a given field are the same, for all time-series, use that value in the output metadata; else,
- concatenate the values in a list.

Instance variables of the BaseImageContainer such as `image_flavour`, will all be checked for consistency and copied across to the output image.

Inputs

Input Parameters are all keyword arguments for the `CombineTimeSeriesFilter.add_inputs()` member function. They are also accessible via class constants, for example `CombineTimeSeriesFilter.KEY_T1`.

Parameters **'image_NNNNN'** – A time-series image. The order of these time series will be determined by the NNNNN component, which shall be a positive integer. Any number of digits can be used in combination in NNNNN. For example, as sequence, *image_0000*, *image_1*, *image_002*, *image_03* is valid.

Note: the indices MUST start from 0 and increment by 1, and have no missing or duplicate indices. This is to help prevent accidentally missing/adding an index value.

Note: If the image data type is complex, it is likely that most NIFTI viewers will problems displaying 4D complex data correctly.

Outputs

Once run, the filter will populate the dictionary `MriSignalFilter.outputs` with the following entries

Parameters **`'image'`** (`BaseImageContainer`) – A 4D image of the combined time series.

`INPUT_IMAGE_REGEX_OBJ = re.compile('^image_(?P<index>[0-9]+)$')`

`KEY_IMAGE = 'image'`

asldro.filters.create_volumes_from_seg_mask module

Create volumes from segmentation mask filter

class asldro.filters.create_volumes_from_seg_mask.**CreateVolumesFromSegMask**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter for assigning values to regions defined by a segmentation mask, then concatenating these images into a 5D image

Inputs

Input Parameters are all keyword arguments for the `CreateVolumesFromSegMask.add_input()` member function. They are also accessible via class constants, for example *CreateVolumesFromSegMask.KEY_IMAGE*

Parameters

- **`'seg_mask'`** (`BaseImageContainer`) – segmentation mask image comprising integer values for each region. dtype of the image data must be an unsigned or signed integer type.
- **`'label_values'`** (*list[int]*) – List of the integer values in `'seg_mask'`, must not have any duplicate values. The order of the integers in this list must be matched by the input `'label_names'`, and the lists with quantity values for each quantity in the input `'quantitites'`.
- **`'label_names'`** (*list[str]*) – List of strings defining the names of each region defined in `'seg_mask'`. The order matches the order given in `'label_values'`.
- **`'quantities'`** – Dictionary, containing key/value pairs where the key name defines a a quantity, and the value is an array of floats that define the value to assign to each region. The order of these floats matches the order given in `'label_values'`.
- **`'units'`** (*list[str]*) – List of strings defining the units that correspond with each quantity given in the dictionary `'quantities'`, as given by the order defined in that dictionary.

Outputs

Once run, the filter will populate the dictionary `CreateVolumesFromSegMask.outputs` with the following entries

Parameters

- **`'image'`** (`BaseImageContainer`) – The combined 5D image, with volumes where the values for each quantity have been assigned to the regions defined in `'seg_mask'`. The final entry in the 5th dimension is a copy of the image `'seg_mask'`.
- **`'image_info'`** (*dict*) – A dictionary describing the regions, quantities and units in the outpu `'image'`. This is of the same format as the ground truth JSON file, however there is no 'parameters' object. See *Making an input ground truth* for more information on this format.

KEY_IMAGE = 'image'

KEY_IMAGE_INFO = 'image_info'

KEY_LABEL_NAMES = 'label_names'

KEY_LABEL_VALUES = 'label_values'

KEY_QUANTITIES = 'quantities'

KEY_SEG_MASK = 'seg_mask'

KEY_UNITS = 'units'

asldro.filters.filter_block module

FilterBlock class

class asldro.filters.filter_block.**FilterBlock**(*name: str = 'Unknown'*)
Bases: *asldro.filters.basefilter.BaseFilter*

A filter made from multiple, chained filters. Used for when the same configuration of filters is used multiple times, or needs to be tested as a whole

run(*history=None*)
Calls the BaseFilter's run method to make sure all of the inputs of this FilterBlock are up-to-date and valid. Then runs this FilterBlock's output filter, and populates the outputs to this FilterBlock.

asldro.filters.fourier_filter module

Fourier Transform filter

class asldro.filters.fourier_filter.**FftFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter for performing a n-dimensional fast fourier transform of input. Input is either a NumpyImageContainer or NiftiImageContainer. Output is a complex numpy array of the discrete fourier transform named 'kdata'

KEY_IMAGE = 'image'

class asldro.filters.fourier_filter.**IfftFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter for performing a n-dimensional inverse fast fourier transform of input. Input is a numpy array named 'kdata'. Output is a complex numpy array of the inverse discrete fourier transform named 'image'

KEY_IMAGE = 'image'

asldro.filters.gkm_filter module

General Kinetic Model Filter

class asldro.filters.gkm_filter.**GkmFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that generates the ASL signal using the General Kinetic Model.

Inputs

Input Parameters are all keyword arguments for the `GkmFilter.add_inputs()` member function. They are also accessible via class constants, for example *GkmFilter.KEY_PERFUSION_RATE*

Parameters **'perfusion_rate'** (BaseImageContainer) – Map of perfusion rate, in ml/100g/min (>=0)

:param 'transit_time' Map of the time taken for the labelled bolus to reach the voxel, seconds (>=0).

Parameters

- **'m0'** – The tissue equilibrium magnetisation, can be a map or single value (>=0).
- **'label_type'** (*str*) – Determines which GKM equations to use:
  - "casl" OR "pcasl" (case insensitive) for the continuous model
  - "pasl" (case insensitive) for the pulsed model
- **'label_duration'** (*float*) – The length of the labelling pulse, seconds (0 to 100 inclusive)
- **'signal_time'** (*float*) – The time after labelling commences to generate signal, seconds (0 to 100 inclusive)
- **'label_efficiency'** (*float*) – The degree of inversion of the labelling (0 to 1 inclusive)
- **'lambda_blood_brain'** (*float or* BaseImageContainer) – The blood-brain-partition-coefficient (0 to 1 inclusive)
- **'t1_arterial_blood'** (*float*) – Longitudinal relaxation time of arterial blood, seconds (0 exclusive to 100 inclusive)
- **'t1_tissue'** (BaseImageContainer) – Longitudinal relaxation time of the tissue, seconds (0 to 100 inclusive, however voxels with `t1 = 0` will have `delta_m = 0`)
- **'model'** (*str*) – The model to use to generate the perfusion signal:
  - "full" for the full "Buxton" General Kinetic Model [2]
  - "whitepaper" for the simplified model, derived from the quantification equations the ASL Whitepaper consensus paper [1].

  Defaults to "full".

Outputs

Once run, the filter will populate the dictionary `GkmFilter.outputs` with the following entries

Parameters **'delta_m'** (BaseImageContainer) – An image with synthetic ASL perfusion contrast. This will be the same class as the input 'perfusion_rate'

Metadata

The following parameters are added to `GkmFilter.outputs["delta_m"].metadata`:

- `label_type`
- `label_duration` (pcasl/casl only)
- `post_label_delay`
- `bolus_cut_off_flag` (pasl only)
- `bolus_cut_off_delay_time` (pasl only)
- `label_efficiency`
- `lambda_blood_brain` (only if a single value is supplied)
- `t1_arterial_blood`
- `m0` (only if a single value is supplied)
- `gkm_model` = `model`

`post_label_delay` is calculated as `signal_time - label_duration`

`bolus_cut_off_delay_time` takes the value of the input `label_duration`, this field is used for pasl in line with the BIDS specification.

Equations

The general kinetic model [2] is the standard signal model for ASL perfusion measurements. It considers the difference between the control and label conditions to be a deliverable tracer, referred to as $\Delta M(t)$.

The amount of $\Delta M(t)$ within a voxel at time t depends on the history of:

- delivery of magnetisation by arterial flow
- clearance by venous flow
- longitudinal relaxation

These processes can be described by defining three functions of time:

1. The delivery function $c(t)$ - the normalised arterial concentration of magnetisation arriving at the voxel at time t.
2. The residue function $r(t, t')$ - the fraction of tagged water molecules that arrive at time t' and are still in the voxel at time t.
3. The magnetisation relaxation function $m(t, t')$ is the fraction of the original longitudinal magnetisation tag carried by the water molecules that arrived at time t' that remains at time t.

Using these definitions $\Delta M(t)$ can be constructed as the sum over history of delivery of magnetisation to the

tissue weighted with the fraction of that magnetisation that remains in the voxel:

$\Delta M(t) = 2 \cdot M_{0,b} \cdot f \cdot \{c(t) * [r(t) \cdot m(t)]\}$

where

$*$ = convolution operator

$r(t)$ = residue function = $e^{-\frac{ft}{\lambda}}$

$m(t) = e^{-\frac{t}{T_1}}$

$c(t)$ = delivery function, defined as plug flow = $\begin{cases} 0 & 0 < t < \Delta t \\ \alpha e^{-\frac{t}{T_{1,b}}} \text{ (PASL)} & \Delta t < t < \Delta t + \tau \\ \alpha e^{-\frac{\Delta t}{T_{1,b}}} \text{ (CASL/pCASL)} & \\ 0 & t > \Delta t + \tau \end{cases}$

α = labelling efficiency

τ = label duration

Δt = initial transit delay, ATT

$M_{0,b}$ = equilibrium magnetisation of arterial blood = $\frac{M_{0,\text{tissue}}}{\lambda}$

f = the perfusion rate, CBF

λ = blood brain partition coefficient

$T_{1,b}$ = longitudinal relaxation time of arterial blood

T_1 = longitudinal relaxation time of tissue

Note that all units are in SI, with f having units s^{-1}. Multiplying by 6000 gives units of $ml/100g/min$.

Full Model

The full solutions to the GKM [2] are used to calculate $\Delta M(t)$ when `model=="full"`:

- (p)CASL:

$$\Delta M(t) = \begin{cases} 0 & 0 < t \leq \Delta t \\ 2M_{0,b} f T_1' \alpha e^{-\frac{\Delta t}{T_{1,b}}} q_{ss}(t) & \Delta t < t < \Delta t + \tau \\ 2M_{0,b} f T_1' \alpha e^{-\frac{\Delta t}{T_{1,b}}} e^{-\frac{t-\tau-\Delta t}{T_1'}} q_{ss}(t) & t \geq \Delta t + \tau \end{cases}$$

where

$$q_{ss}(t) = \begin{cases} 1 - e^{-\frac{t-\Delta t}{T_1'}} & \Delta t < t < \Delta t + \tau \\ 1 - e^{-\frac{\tau}{T_1'}} & t \geq \Delta t + \tau \end{cases}$$

$$\frac{1}{T_1'} = \frac{1}{T_1} + \frac{f}{\lambda}$$

- PASL:

$$\Delta M(t) = \begin{cases} 0 & 0 < t \leq \Delta t \\ 2M_{0,b} f (t - \Delta t) \alpha e^{-\frac{t}{T_{1,b}}} q_p(t) & \Delta t < t < t\Delta t + \tau \\ 2M_{0,b} f \alpha \tau e^{-\frac{t}{T_{1,b}}} q_p(t) & t \geq \Delta t + \tau \end{cases}$$

where

$$q_p(t) = \begin{cases} \frac{e^{kt}(e^{-k\Delta t} - e^{-kt})}{k(t-\Delta t)} & \Delta t < t < \Delta t + \tau \\ \frac{e^{kt}(e^{-k\Delta t} - e^{k(\tau+\Delta t)}}{k\tau} & t \geq \Delta t + \tau \end{cases}$$

$$\frac{1}{T_1'} = \frac{1}{T_1} + \frac{f}{\lambda}$$

$$k = \frac{1}{T_{1,b}} - \frac{1}{T_1'}$$

*Simplified Model"

The simplified model, derived from the single subtraction quantification equations (see *AslQuantificationFilter*) are used when `model=="whitepaper"`:

- (p)CASL:

$$\Delta M(t) = \begin{cases} 0 & 0 < t \leq \Delta t + \tau \\ 2M_{0,b} f T_{1,b} \alpha (1 - e^{-\frac{\tau}{T_{1,b}}}) e^{-\frac{t-\tau}{T_{1,b}}} & t > \Delta t + \tau \end{cases}$$

- PASL

$$\Delta M(t) = \begin{cases} 0 & 0 < t \leq \Delta t + \tau \\ 2M_{0,b} f \tau \alpha e^{-\frac{t}{T_{1,b}}} & t > \Delta t + \tau \end{cases}$$

```
CASL = 'casl'
KEY_DELTA_M = 'delta_m'
KEY_LABEL_DURATION = 'label_duration'
KEY_LABEL_EFFICIENCY = 'label_efficiency'
KEY_LABEL_TYPE = 'label_type'
KEY_LAMBDA_BLOOD_BRAIN = 'lambda_blood_brain'
KEY_M0 = 'm0'
KEY_MODEL = 'model'
KEY_PERFUSION_RATE = 'perfusion_rate'
KEY_SIGNAL_TIME = 'signal_time'
KEY_T1_ARTERIAL_BLOOD = 't1_arterial_blood'
KEY_T1_TISSUE = 't1_tissue'
KEY_TRANSIT_TIME = 'transit_time'
MODEL_FULL = 'full'
MODEL_WP = 'whitepaper'
M_BOLUS_CUT_OFF_DELAY_TIME = 'bolus_cut_off_delay_time'
```

M_BOLUS_CUT_OFF_FLAG = 'bolus_cut_off_flag'

M_GKM_MODEL = 'gkm_model'

M_POST_LABEL_DELAY = 'post_label_delay'

PASL = 'pasl'

PCASL = 'pcasl'

static calculate_delta_m_gkm(*perfusion_rate: numpy.ndarray*, *transit_time: numpy.ndarray*, *m0_tissue: numpy.ndarray*, *label_duration: float*, *signal_time: float*, *label_efficiency: float*, *partition_coefficient: numpy.ndarray*, *t1_arterial_blood: float*, *t1_tissue: numpy.ndarray*, *label_type: str*) → numpy.ndarray

Calculates the difference in magnetisation between the control and label condition (ΔM) using the full solutions to the General Kinetic Model [2].

Parameters

- **perfusion_rate** (*np.ndarray*) – Map of perfusion rate
- **transit_time** (*np.ndarray*) – Map of transit time
- **m0_tissue** (*np.ndarray*) – The tissue equilibrium magnetisation
- **label_duration** (*float*) – The length of the labelling pulse
- **signal_time** (*float*) – The time after the labelling pulse commences to generate signal.
- **label_efficiency** (*float*) – The degree of inversion of the labelling pulse.
- **partition_coefficient** (*np.ndarray*) – The tissue-blood partition coefficient
- **t1_arterial_blood** (*float*) – Longitudinal relaxation time of the arterial blood.
- **t1_tissue** (*np.ndarray*) – Longitudinal relaxation time of the tissue
- **label_type** (*str*) – Determines the specific model to use: Pulsed ("pasl") or (pseudo)Continuous ("pcasl" or "casl") labelling

Returns the difference magnetisation, ΔM

Return type np.ndarray

static calculate_delta_m_whitepaper(*perfusion_rate: numpy.ndarray*, *transit_time: numpy.ndarray*, *m0_tissue: numpy.ndarray*, *label_duration: float*, *signal_time: float*, *label_efficiency: float*, *partition_coefficient: numpy.ndarray*, *t1_arterial_blood: float*, *label_type: str*) → numpy.ndarray

Calculates the difference in magnetisation between the control and label condition (ΔM) using the single subtraction simplification from the ASL Whitepaper consensus paper [1].

Parameters

- **perfusion_rate** (*np.ndarray*) – Map of perfusion rate
- **transit_time** (*np.ndarray*) – Map of transit time
- **m0_tissue** (*np.ndarray*) – The tissue equilibrium magnetisation
- **label_duration** (*float*) – The length of the labelling pulse

- **signal_time** (*float*) – The time after the labelling pulse commences to generate signal.
- **label_efficiency** (*float*) – The degree of inversion of the labelling pulse.
- **partition_coefficient** (*np.ndarray*) – The tissue-blood partition coefficient
- **t1_arterial_blood** (*float*) – Longitudinal relaxation time of the arterial blood.
- **t1_tissue** (*np.ndarray*) – Longitudinal relaxation time of the tissue
- **label_type** (*str*) – Determines the specific model to use: Pulsed ("pasl") or (pseudo)Continuous ("pcasl" or "casl") labelling

Returns the difference magnetisation, ΔM

Return type np.ndarray

static check_and_make_image_from_value(*arg: float*, *shape: tuple*, *metadata: dict*, *metadata_key: str*) → numpy.ndarray

Checks the type of the input parameter to see if it is a float or a BaseImageContainer. If it is an image:

- return the image ndarray
- check if it has the same value everywhere (i.e. an image override), if it does then place the value into the *metadata* dict under the *metadata_key*

If it is a float: * make a ndarray with the same value * place the value into the *metadata* dict under the *metadata_key*

This makes calculations more straightforward as a ndarray can always be expected.

Arguments

Parameters

- **arg** (*float or BaseImageContainer*) – The input parameter to check
- **shape** (*tuple*) – The shape of the image to create
- **metadata** (*dict*) – metadata dict, which is updated by this function
- **metadata_key** (*str*) – key to assign the value of arg (if a float or single value image) to

Returns image of the parameter

Rype np.ndarray

static compute_arrival_state_masks(*transit_time: numpy.ndarray*, *signal_time: float*, *label_duration: float*) → dict

Creates boolean masks for each of the states of the delivery curve

Parameters

- **transit_time** (*np.ndarray*) – map of the transit time
- **signal_time** (*float*) – the time to generate signal at
- **label_duration** (*float*) – The duration of the labelling pulse

Returns

a dictionary with three entries, each a ndarray with shape the same as *transit_time*:

"not_arrived" voxels where the bolus has not reached yet

"arriving" voxels where the bolus has reached but not been completely delivered.

"arrived" voxels where the bolus has been completely delivered

Return type dict

asldro.filters.ground_truth_loader module

Ground truth loader filter

class asldro.filters.ground_truth_loader.**GroundTruthLoaderFilter**

Bases: *asldro.filters.basefilter.BaseFilter*

A filter for loading ground truth NIFTI/JSON file pairs.

Inputs

Input Parameters are all keyword arguments for the `GroundTruthLoaderFilter.add_input()` member function. They are also accessible via class constants, for example *GroundTruthLoaderFilter.KEY_IMAGE*

Parameters

- **`'image'`** (NiftiImageContainer) – ground truth image, must be 5D and the 5th dimension have the same length as the number of quantities.
- **`'quantities'`** (*list[str]*) – list of quantity names
- **`'units'`** (*list[str]*) – list of units corresponding to the quantities, must be the same length as quantities
- **`'parameters'`** (*dict*) – dictionary containing keys `'t1_arterial_blood'`, `'lambda_blood_brain'` and `'magnetic_field_strength'`.
- **`'segmentation'`** – dictionary containing key-value pairs corresponding to tissue type and label value in the `'seg_label'` volume.
- **`'image_override'`** (*dict*) – (optional) dictionary containing single-value override values for any of the `'image'` that are loaded. The keys must match the quantity name defined in `'quantities'`.
- **`'parameter_override'`** (*dict*) – (optional) dictionary containing single-value override values for any of the `'parameters'` that are loaded. The keys must match the key defined in `'parameters'`.
- **`'ground_truth_modulate'`** (*dict*) – dictionary with keys corresponding with quantity names. The possible dictionary values (both optional) are:

```
{
    "scale": N,
    "offset": M,
}
```

Any corresponding images will have the corresponding scale and offset applied before being output. See `ScaleOffsetFilter` for more details.

Outputs

Once run, the filter will populate the dictionary `GroundTruthLoaderFilter.outputs` with output fields based on the input `'quantities'`.

Each key in `'quantities'` will result in a NiftiImageContainer corresponding to a 3D/4D subset of the nifti input (split along the 5th dimension). The data types of images will be the same as those input EXCEPT for a quantity labelled `'seg_label'` which will be converted to a uint16 data type.

If 'override_image' is defined, the corresponding 'image' will be set to the overriding value before being output.

If 'override_parameters' is defined, the corresponding parameter will be set to the overriding value before being output.

If 'ground_truth_modulate' is defined, the corresponding 'image'(s) will be scaled and/or offset by the corresponding values.

The keys-value pairs in the input `'parameters'` will also be destructured and piped through to the output, for example:

Parameters

- **`'t1'`** (NiftiImageContainer) – volume of T1 relaxation times
- **`'seg_label'`** (NiftiImageContainer *(uint16 data type)*) – segmentation label mask corresponding to different tissue types.
- **`'magnetic_field_strength'`** (*float*) – the magnetic field strenght in Tesla.
- **`'t1_arterial_blood'`** (*float*) – the T1 relaxation time of arterial blood
- **`'lambda_blood_brain'`** (*float*) – the blood-brain-partition-coefficient

A field metadata will be created in each image container, with the following fields:

Magnetic_field_strength corresponds to the value in the *'parameters'* object.

Quantity corresponds to the entry in the `'quantities'` array.

Units corresponds with the entry in the `'units'` array.

The `'segmentation'` object from the JSON file will also be piped through to the metadata entry of the `'seg_label'` image container.

KEY_GROUND_TRUTH_MODULATE = 'ground_truth_modulate'

KEY_IMAGE = 'image'

KEY_IMAGE_OVERRIDE = 'image_override'

KEY_MAG_STRENGTH = 'magnetic_field_strength'

KEY_PARAMETERS = 'parameters'

KEY_PARAMETER_OVERRIDE = 'parameter_override'

KEY_QUANTITIES = 'quantities'

KEY_QUANTITY = 'quantity'

KEY_SEGMENTATION = 'segmentation'

KEY_UNITS = 'units'

asldro.filters.image_tools module

Filters for basic image container manipulation and maths

class asldro.filters.image_tools.**FloatToIntImageFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter which converts image data from float to integer.

Inputs

Input Parameters are all keyword arguments for the `FloatToIntImageFilter.add_input()` member function. They are also accessible via class constants, for example *FloatToIntImageFilter.KEY_IMAGE*

Parameters

- **`'image'`** (BaseImageContainer) – Image to convert from float to integer. The dtype of the image data must be float.
- **`'method'`** – Defines which method to use for conversion:
  - ”round”: returns the nearest integer
  - ”floor”: returns the largest integer that is less than the input value.
  - ”ceil”: returns the smallest integer that is greater than the input value.
  - ”truncate”: Removes the decimal portion of the number. This will round down for positive numbers and up for negative.

Outputs

Once run, the filter will populate the dictionary `FloatToIntImageFilter.outputs` with the following entries

Parameters **`'image'`** (BaseImageContainer) – The input image, with the image data as integer type.

`CEIL = 'ceil'`

`FLOOR = 'floor'`

`KEY_IMAGE = 'image'`

`KEY_METHOD = 'method'`

`METHODS = ['round', 'floor', 'ceil', 'truncate']`

`ROUND = 'round'`

`TRUNCATE = 'truncate'`

asldro.filters.invert_image_filter module

Invert image filter

class asldro.filters.invert_image_filter.**InvertImageFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter which simply inverts the input image.

Must have one input named ‘image’. These correspond with a derivative of BaseImageContainer.

Creates a single output named ‘image’.

asldro.filters.json_loader module

JSON file loader filter

class asldro.filters.json_loader.**JsonLoaderFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter for loading a JSON file.

Inputs

Input parameters are all keyword arguments for the `JsonLoaderFilter.add_inputs()` member function. They are also accessible via class constants, for example *JsonLoaderFilter.KEY_FILENAME*.

Parameters

- **`'filename'`** (*str*) – The path to the JSON file to load
- **`'schema'`** (*dict*) – (optional) The schema to validate against (in python dict format). Some schemas can be found in asldro.validators.schemas, or one can just in input here.
- **`'root_object_name'`** (*str*) – Optionally place all of the key-value pairs inside this object

Outputs

Creates a multiple outputs, based on the root key,value pairs in the JSON filter. For example: { "foo": 1, "bar": "test"} will create two outputs named "foo" and "bar" with integer and string values respectively. The outputs may also be nested i.e. object or arrays.

If the input parameter `'root_object_name'` is supplied then these outputs will be nested within an object taking the name of the value of `'root_object_name'`

KEY_FILENAME = 'filename'

KEY_ROOT_OBJECT_NAME = 'root_object_name'

KEY_SCHEMA = 'schema'

asldro.filters.load_asl_bids_filter module

Load ASL BIDS filter class

class asldro.filters.load_asl_bids_filter.**LoadAslBidsFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that loads in ASL data in BIDS format, comprising of a NIFTI image file, json sidear and tsv aslcontext file. After loading in the data, image containers are created using the volumes described in aslcontext. For each of these containers, the data in sidecar is added to the metadata object. In addition a metadata 'asl_context' is created which is a list of the corresponding volumes contained in each container. Any metadata entries that are an array and specific to each volume have only the corresponding values copied.

Inputs

Input Parameters are all keyword arguments for the `LoadAslBidsFilter.add_inputs()` member function. They are also accessible via class constants, for example *LoadAslBidsFilter.KEY_SIDECAR*

Parameters

- **`'image_filename'`** (*str*) – path and filename to the ASL NIFTI image (must end in .nii or.nii.gz)
- **`'sidecar_filename'`** – path and filename to the json sidecar (must end in .json)

- **'aslcontext_filename'** (*str*) – path and filename to the aslcontext file (must end in .tsv). This must be a tab separated values file, with heading 'volume_type' and then entries which are either 'control', 'label', or 'm0scan'.

Outputs

Once run, the filter will populate the dictionary `LoadAslBidsFilter.outputs` with the following entries

Parameters

- **'source'** (BaseImageContainer) – the full ASL NIFTI image
- **'control'** (BaseImageContainer) – control volumes (as defined by aslcontext)
- **'label'** (BaseImageContainer) – label volumes (as defined by aslcontext)
- **'m0'** (BaseImageContainer) – m0 volumes (as defined by aslcontext)

ASL_CONTEXT_MAPPING = {'control': 'control', 'label': 'label', 'm0': 'm0scan'}

KEY_ASLCONTEXT_FILENAME = 'aslcontext_filename'

KEY_CONTROL = 'control'

KEY_IMAGE_FILENAME = 'image_filename'

KEY_LABEL = 'label'

KEY_M0 = 'm0'

KEY_SIDECAR = 'sidecar'

KEY_SIDECAR_FILENAME = 'sidecar_filename'

KEY_SOURCE = 'source'

LIST_FIELDS_TO_EXCLUDE = ['ScanningSequence', 'ComplexImageComponent', 'ImageType', 'A

asldro.filters.mri_signal_filter module

MRI Signal Filter

class asldro.filters.mri_signal_filter.**MriSignalFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that generates either the Gradient Echo, Spin Echo or Inversion Recovery MRI signal.

- Gradient echo is with arbitrary excitation flip angle.
- Spin echo assumes perfect 90° excitation and 180° refocusing pulses.
- Inversion recovery can have arbitrary inversion pulse and excitation pulse flip angles.

Inputs

Input Parameters are all keyword arguments for the `MriSignalFilter.add_inputs()` member function. They are also accessible via class constants, for example *MriSignalFilter.KEY_T1*

Parameters

- **'t1'** (BaseImageContainer) – Longitudinal relaxation time in seconds (>=0, non-complex data)
- **'t2'** (BaseImageContainer) – Transverse relaxation time in seconds (>=0, non-complex data)

- **'t2_star'** (BaseImageContainer) – Transverse relaxation time including time-invariant magnetic field inhomogeneities, only required for gradient echo (>=0, non-complex data)
- **'m0'** (BaseImageContainer) – Equilibrium magnetisation (non-complex data)
- **'mag_eng'** – Added to M0 before relaxation is calculated, provides a means to encode another signal into the MRI signal (non-complex data)
- **'acq_contrast'** (*str*) – Determines which signal model to use: "ge" (case insensitive) for Gradient Echo, "se" (case insensitive) for Spin Echo, "ir" (case insensitive) for Inversion Recovery.
- **'echo_time'** (*float*) – The echo time in seconds (>=0)
- **'repetition_time'** (*float*) – The repeat time in seconds (>=0)
- **'excitation_flip_angle'** (*float, optional*) – Excitation pulse flip angle in degrees. Only used when 'acq_contrast' is "ge" or "ir". Defaults to 90.0
- **'inversion_flip_angle'** (*float, optional*) – Inversion pulse flip angle in degrees. Only used when acq_contrast is "ir". Defaults to 180.0
- **'inversion_time'** – The inversion time in seconds. Only used when 'acq_contrast' is "ir". Defaults to 1.0.
- **'image_flavour'** (*str*) – sets the metadata 'image_flavour' in the output image to this.

Outputs

Once run, the filter will populate the dictionary MriSignalFilter.outputs with the following entries

Parameters **'image'** (BaseImageContainer) – An image of the generated MRI signal. Will be of the same class as the input 'm0'

Output Image Metadata

The metadata in the output image MriSignalFilter.outputs["image"] is derived from the input 'm0'. If the input 'mag_enc' is present, its metadata is merged with precedence. In addition, following parameters are added:

- 'acq_contrast'
- 'echo time'
- 'excitation_flip_angle'
- 'image_flavour'
- 'inversion_time'
- 'inversion_flip_angle'
- ``'mr_acq_type`` = "3D"

Metadata entries for 'units' and 'quantity' will be removed.

'image_flavour' is obtained (in order of precedence):

1. If present, from the input 'image_flavour'
2. If present, derived from the metadata in the input 'mag_enc'
3. "OTHER"

Signal Equations

The following equations are used to compute the MRI signal:

Gradient Echo

$$S(\mathrm{TE}, \mathrm{TR}, \theta_1) = \sin\theta_1 \cdot \left(\frac{M_0 \cdot (1 - e^{-\frac{TR}{T_1}})}{1 - \cos\theta_1 e^{-\frac{TR}{T_1}} - e^{-\frac{TR}{T_2}} \cdot \left(e^{-\frac{TR}{T_1}} - \cos\theta_1\right)} + M_{\text{enc}}\right) \cdot e^{-\frac{\mathrm{TE}}{T_2^*}}$$

Spin Echo (assuming 90° and 180° pulses)

$$S(\mathrm{TE}, \mathrm{TR}) = (M_0 \cdot (1 - e^{-\frac{\mathrm{TR}}{T_1}}) + M_{\text{enc}}) \cdot e^{-\frac{\mathrm{TE}}{T_2}}$$

Inversion Recovery

$$S(\mathrm{TE}, \mathrm{TR}, \mathrm{TI}, \theta_1, \theta_2) = \sin\theta_1 \cdot \left(\frac{M_0(1 - (1 - \cos\theta_2)\, e^{-\frac{TI}{T_1}} - \cos\theta_2 e^{-\frac{TR}{T_1}})}{1 - \cos\theta_1 \cos\theta_2 e^{-\frac{TR}{T_1}}} + M_{\text{enc}}\right) \cdot e^{-\frac{TE}{T_2}}$$

$\theta_1 =$ excitation pulse flip angle

$\theta_2 =$ inversion pulse flip angle

TI = inversion time

TR = repetition time

TE = echo time

```
CONTRAST_GE = 'ge'

CONTRAST_IR = 'ir'

CONTRAST_SE = 'se'

KEY_ACQ_CONTRAST = 'acq_contrast'

KEY_ACQ_TYPE = 'mr_acq_type'

KEY_BACKGROUND_SUPPRESSION = 'background_suppression'

KEY_ECHO_TIME = 'echo_time'

KEY_EXCITATION_FLIP_ANGLE = 'excitation_flip_angle'

KEY_IMAGE = 'image'

KEY_IMAGE_FLAVOUR = 'image_flavour'

KEY_INVERSION_FLIP_ANGLE = 'inversion_flip_angle'

KEY_INVERSION_TIME = 'inversion_time'

KEY_M0 = 'm0'

KEY_MAG_ENC = 'mag_enc'

KEY_REPETITION_TIME = 'repetition_time'

KEY_T1 = 't1'

KEY_T2 = 't2'

KEY_T2_STAR = 't2_star'
```

asldro.filters.nifti_loader module

NIFTI file loader filter

class asldro.filters.nifti_loader.**NiftiLoaderFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter for loading a NIFTI image from a file.

Must have a single string input named 'filename'.

Creates a single Image container as an output named 'image'

asldro.filters.phase_magnitude_filter module

PhaseMagnitudeFilter Class

class asldro.filters.phase_magnitude_filter.**PhaseMagnitudeFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter block that will take image data and convert it into its Phase and Magnitude components. Typically, this will be used after a *AcquireMriImageFilter* which contains real and imaginary components, however it may also be used with image data that is of type:

- `REAL_IMAGE_TYPE`: in which case the phase is 0° where the image value is positive, and 180° where it is negative.
- `IMAGINARY_IMAGE_TYPE`: in which case the phase is 90° where the image value is positive, and 270° where it is negative.
- `MAGNITUDE_IMAGE_TYPE`: in which case the phase cannot be defined and so the output phase image is set to `None`.

Inputs

Input Parameters are all keyword arguments for the `PhaseMagnitudeFilter.add_inputs()` member function. They are also accessible via class constants, for example *PhaseMagnitudeFilter.KEY_IMAGE*

Parameters **'image'** (BaseImageContainer) – The input data image, cannot be a phase image

Outputs

Parameters

- **'phase'** (BaseImageContainer) – Phase image (will have `image_type==PHASE_IMAGE_TYPE`)
- **'magnitude'** (BaseImageContainer) – Magnitude image (will have `image_type==MAGNITUDE_IMAGE_TYPE`)

KEY_IMAGE = 'image'

KEY_MAGNITUDE = 'magnitude'

KEY_PHASE = 'phase'

asldro.filters.resample_filter module

Resample Filter

class asldro.filters.resample_filter.**ResampleFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that can resample an image based on a target shape and affine. Note that nilearn actually applies the inverse of the target affine.

Inputs

Input Parameters are all keyword arguments for the `ResampleFilter.add_inputs()` member function. They are also accessible via class constants, for example *ResampleFilter.KEY_AFFINE*

Parameters

- **`'image'`** (BaseImageContainer) – Image to resample
- **`'affine'`** (*np.ndarray(4)*) – Image is resampled according to this 4x4 affine matrix
- **`'shape'`** (*Tuple[int, int, int]*) – Image is resampled according to this new shape.
- **`'interpolation'`** (*str, optional*) – Defines the interpolation method:

  'continuous' order 3 spline interpolation (default)

  'linear' order 1 linear interpolation

  'nearest' nearest neighbour interpolation

Outputs

Once run, the filter will populate the dictionary `ResampleFilter.outputs` with the following entries:

Parameters **`'image'`** (BaseImageContainer) – The input image, resampled in accordance with the input shape and affine.

The metadata property of the `ResampleFilter.outputs["image"]` is updated with the field `voxel_size`, corresponding to the size of each voxel.

CONTINUOUS = 'continuous'

INTERPOLATION_LIST = ['continuous', 'linear', 'nearest']

KEY_AFFINE = 'affine'

KEY_IMAGE = 'image'

KEY_INTERPOLATION = 'interpolation'

KEY_SHAPE = 'shape'

LINEAR = 'linear'

NEAREST = 'nearest'

asldro.filters.scale_offset_filter module

ScaleOffsetFilter Class

class asldro.filters.scale_offset_filter.**ScaleOffsetFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that will take image data and apply a scale and/or offset according to the equation:

$$I_{output} = I_{input} * m + b$$

where 'm' is the scale and 'b' is the offset (scale first then offset)

Inputs

Input Parameters are all keyword arguments for the `ScaleOffsetFilter.add_inputs()` member function. They are also accessible via class constants, for example *ScaleOffsetFilter.KEY_IMAGE*

Parameters

- **'image'** (BaseImageContainer) – The input image
- **'scale'** (*float / int*) – (optional) a scale to apply
- **'offset'** (*float / int*) – (optional) an offset to apply

Outputs

Parameters **'image'** (BaseImageContainer) – The output image

KEY_IMAGE = 'image'

KEY_OFFSET = 'offset'

KEY_SCALE = 'scale'

asldro.filters.transform_resample_image_filter module

Transform resample image filter

class asldro.filters.transform_resample_image_filter.**TransformResampleImageFilter**
Bases: *asldro.filters.basefilter.BaseFilter*

A filter that transforms and resamples an image in world space. The field of view (FOV) of the resampled image is the same as the FOV of the input image.

Conventions are for RAS+ coordinate systems only

Inputs

Input Parameters are all keyword arguments for the `TransformResampleImageFilter.add_inputs()` member function. They are also accessible via class constants, for example *TransformResampleImageFilter.KEY_ROTATION*

Parameters

- **'image'** (BaseImageContainer) – The input image
- **'translation'** – $[\Delta r_x, \Delta r_y, \Delta r_z]$ amount to translate along the x, y and z axes. defaults to (0, 0, 0)
- **'rotation'** (*Tuple[float, float, float], optional*) – $[\theta_x, \theta_y, \theta_z]$ angles to rotate about the x, y and z axes in degrees(-180 to 180 degrees inclusive), defaults to (0, 0, 0)

- **'rotation_origin'** (*Tuple[float, float, float], optional*) – $[x_r, y_r, z_r]$ coordinates of the point to perform rotations about, defaults to (0, 0, 0)
- **target_shape** (*Tuple[int, int, int]*) – $[L_t, M_t, N_t]$ target shape for the resampled image
- **'interpolation'** (*str, optional*) – Defines the interpolation method for the resampling:

  'continuous' order 3 spline interpolation (default method for ResampleFilter)

  'linear' order 1 linear interpolation

  'nearest' nearest neighbour interpolation

Outputs

Once run, the filter will populate the dictionary `TransformResampleImageFilter.outputs` with the following entries

Parameters **'image'** (`BaseImageContainer`) – The input image, resampled in accordance with the specified shape and applied world-space transformation.

The metadata property of the `TransformResampleImageFilter.outputs["image"]` is updated with the field `voxel_size`, corresponding to the size of each voxel.

The output image is resampled according to the target affine:

$$\mathbf{A} = (\mathbf{T}(\mathbf{\Delta r}_{\text{im}})\mathbf{ST}(\mathbf{\Delta r})\mathbf{T}(\mathbf{r_0})\mathbf{RT}(\mathbf{r_0})^{-1})^{-1}$$

where,

$\mathbf{T}(\mathbf{r_0}) = \mathbf{T}(x_r, y_r, z_r) =$ Affine for translation to rotation centre

$\mathbf{T}(\mathbf{\Delta r}) = \mathbf{T}(\Delta r_x, \Delta r_y, \Delta r_z) =$ Affine for translation of image in world space

$\mathbf{T}(\mathbf{\Delta r}_{\text{im}}) = \mathbf{T}(x_0/s_x, y_0/s_y, z_0/s_z)^{-1} =$ Affine for translation to the input image origin

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{translation matrix}$$

$$\mathbf{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ & 0 & 0 & 1 \end{pmatrix} = \text{scaling matrix}$$

$$[s_x, s_y, s_z] = \frac{[L_t, M_t, N_t]}{[v_x, v_y, v_z] \cdot [L_i, M_i, N_i]}$$

divisions and multiplications are element-wise (Hadamard)

$[L_i, M_i, N_i] =$ shape of the input image

$[v_x, v_y, v_z] =$ voxel dimensions of the input image

$[x_0, y_0, z_0] =$ input image origin coordinates (vector part of input image's affine)

$\mathbf{R} = \mathbf{R_z}\mathbf{R_y}\mathbf{R_x} =$ Affine for rotation of image in world space

$$\mathbf{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{rotation about x matrix}$$

$$\mathbf{R_y} = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{rotation about y matrix}$$

$$\mathbf{R_z} = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \text{rotation about z matrix}$$

After resampling the output image's affine is modified to only contain the scaling:

$$\mathbf{A}_{\text{new}} = (\mathbf{T}(\mathbf{\Delta r}_{\text{im}})\mathbf{S})^{-1}$$

```
INTERPOLATION_LIST = ['continuous', 'linear', 'nearest']
KEY_IMAGE = 'image'
KEY_INTERPOLATION = 'interpolation'
KEY_ROTATION = 'rotation'
KEY_ROTATION_ORIGIN = 'rotation_origin'
KEY_TARGET_SHAPE = 'target_shape'
KEY_TRANSLATION = 'translation'
```

VOXEL_SIZE = 'voxel_size'

Module contents

asldro.pipelines package

Submodules

asldro.pipelines.asl_quantification module

asldro.pipelines.check_nifti_units module

asldro.pipelines.check_nifti_units.**check_nifti_units**()

asldro.pipelines.combine_masks module

Pipeline to combine fuzzy masks into a single segmentation mask

asldro.pipelines.combine_masks.**combine_fuzzy_masks**(*params_filename: str*, *output_filename: str = None*) → *asldro.containers.image.BaseImageContainer*

Combines fuzzy masks into a single segmentation mask image.

Parameters

- **params_filename** (*str*) – Path to the combining masks parameter JSON file.
- **output_filename** (*str*) – Path to the output combined mask NIFTI image, defaults to None

Returns The combined mask, as an image container.

Return type *BaseImageContainer*

asldro.pipelines.create_qasper_ground_truth module

asldro.pipelines.generate_ground_truth module

Pipeline to generate a ground truth image and save

asldro.pipelines.generate_ground_truth.**generate_hrgt**(*hrgt_params_filename: str*, *seg_mask_filename: str*, *output_dir: str = None*) → dict

Generates a high-resolution ground truth (hrgt) based on:

- A segmentation mask image
- A file describing what values to assign to each region.

The hrgt is saved in the folder `output_dir`

Parameters

- **hrgt_params_filename** (*str*) – Path to the hrgt parameter JSON file

- **seg_mask_filename** (*str*) – Path to the segmentation mask NIFTI image
- **output_dir** (*str, optional*) – Directory to save files to, defaults to None

Returns dictionary containing the ground truth image, and the ground truth parameter file

Return type dict

Module contents

asldro.utils package

Submodules

asldro.utils.example_resampling module

asldro.utils.filter_validation module

asldro.utils.general module

General utilities

asldro.utils.general.**generate_random_numbers**(*specification: dict*, *shape=None*, *rng=None*) → numpy.ndarray

Generates a set of numbers according to the prescribed distributions, returning them as a list

Parameters

- **specification** (*dict*) – The distribution to use to generate the parameters:
  - 'gaussian' - normal distribution with mean and standard deviation
  - 'uniform' - rectangular distribution with min and max values
  - 'mean' - mean value of the distribution (gaussian only)
  - 'sd' - standard deviation of the distribution (gaussian only)
  - 'min' - minimum value of the distribution (uniform only)
  - 'max' - maximum value of the distribution (uniform only)
- **shape** (*int or tuple of ints*) – length of the list to return
- **rng** (*[type], optional*) – The random number generator to use, defaults to None

Returns List of the generated numbers

Return type list

asldro.utils.general.**map_dict**(*input_dict: Mapping[str, Any]*, *io_map: Mapping[str, str]*, *io_map_optional: bool = False*) → Dict[str, str]

Maps a dictionary onto a new dictionary by changing some/all of the keys.

Parameters

- **input_dict** – The input dictionary
- **io_map** – The dictionary used to perform the mapping. All keys and values must be strings. For example:

```
{
    "one": "two",
    "three": "four"
}
```

will map inputs keys of “one” to “two” AND “three” to “four”.

- **io_map_optional** – If this is False, a KeyError will be raised if the keys in the io_map are not found in the input_dict.

Raises **KeyError** – if keys required in the mapping are not found in the input_dict

Returns the remapped dictionary

asldro.utils.general.**splitext**(*path*)

The normal os.path.splitext treats path/example.tar.gz as having a filepath of path/example.tar with a .gz extension - this fixes it

asldro.utils.generate_numeric_function module

Functions to generate arrays of mathematical function values

asldro.utils.generate_numeric_function.**generate_circular_function_array**(*func*, *xx: numpy.ndarray*, *yy: numpy.ndarray*, *zz: numpy.ndarray*, *array_size: int*, *array_angular_increment: Union[float, int]*, *func_params: dict*, *array_origin: Union[numpy.ndarray, list, tuple]* = *0.0, 0.0, 0.0*) → numpy.ndarray

Produces a superposition of the supplied function in a circular array. The circular array’s axis is about the z axis.

Parameters

- **func** (*function*) – The function to use to generate the circular array. This should accept arrays of x, y and z values of the entire domain to generate the function (i.e.

3d), as generated by np.meshgrid. It must also accept the argument ''loc", which is the origin of the function, and `theta`, which is the angle in degrees to rotate the function (mathematically) about an axis parallel to z that runs through `loc`. An example function is *generate_point_function()*.

- **xx** (*np.ndarray*) – Array of x-coordinates, generate by meshgrid. Can be sparse.
- **yy** (*np.ndarray*) – Array of y-coordinates, generate by meshgrid. Can be sparse.
- **zz** (*np.ndarray*) – Array of z-coordinates, generate by meshgrid. Can be sparse.
- **array_size** (*int*) – The number of instances of `func` in the array.
- **array_angular_increment** (*Union[float, int]*) – The amount, in degrees, to increment the function each step.
- **func_params** (*dict*) – A dictionary of function parameters. Must have entries:
  - 'loc': np.ndarray, list or tuple length 3, $[x_0, y_0, z_0]$ values.
- **array_origin** (*Union[np.ndarray, list, tuple], optional*) – The origin of the circular array, $[x_{a,0}, y_{a,0}, z_{a,0}]$, defaults to (0.0, 0.0, 0.0)

Raises

- **KeyError** – If argument *func_params* does not have an entry 'loc'.
- **ValueError** – If value of *func_params['loc']* is not a np.ndarray, list or tuple.
- **ValueError** – If value of *func_params['loc']* is not 1D and of lenght 3

Returns An array, comprising the function output arrayed at each position, normalised so that its maximum value is 1.0

Return type np.ndarray

asldro.utils.generate_numeric_function.**generate_gaussian_function**(*xx: numpy.ndarray*, *yy: numpy.ndarray*, *zz: numpy.ndarray*, *loc: Union[tuple, list, numpy.ndarray] = numpy.zeros*, *fwhm: Union[tuple, list, numpy.ndarray] = numpy.ones*, *theta: float = 0.0*) → numpy.ndarray

Generates a 3-dimensional gaussian function. Can be used with *generate_circular_function_array()*.

Parameters

- **xx** (*np.ndarray*) – Array of x-coordinates, generate by meshgrid. Can be sparse.

- **yy** (*np.ndarray*) – Array of y-coordinates, generate by meshgrid. Can be sparse.
- **zz** (*np.ndarray*) – Array of z-coordinates, generate by meshgrid. Can be sparse.
- **loc** (*Union[tuple, list, np.ndarray], optional*) – origin of the gaussian, $[x_0, y_0, z_0]$, defaults to np.zeros((3,))
- **fwhm** (*Union[tuple, list, np.ndarray], optional*) – full-width-half-maximum of the gaussian, $[\text{fwhm}_x, \text{fwhm}_y, \text{fwhm}_z]$, defaults to np.ones((3,))
- **theta** (*float, optional*) – polar rotation of the gaussian (about an axis parallel to z at the gaussian origin), degrees, defaults to 0.0

Raises

- **ValueError** – If xx, yy, and zz do not all have shapes that permit broadcasting
- **ValueError** – If loc is not 1D and of length 3
- **ValueError** – If fwhm is not 1D and of length 3
- **ValueError** – If any entry in fwhm is < 0.0

Returns An array with shape the result of broadcasting `xx`, `yy`, and `zz`, values given by the 3D gaussian function.

Return type np.ndarray

Equation

The gaussian is generated according to:

$$f(x, y, z) = e^{-(a(x-x_0)^2+2b(x-x_0)(y-y_0)+c(y-y_0)^2+d(z-z_0)^2)}$$

where,

$$a = \frac{\cos^2\theta}{2\sigma_x^2} + \frac{\sin^2\theta}{2\sigma_y^2}$$

$$b = -\frac{\sin 2\theta}{4\sigma_x^2} + \frac{\sin 2\theta}{4\sigma_y^2}$$

$$c = \frac{\sin^2\theta}{2\sigma_x^2} + \frac{\cos^2\theta}{2\sigma_y^2}$$

$$d = \frac{1}{2\sigma_z^2}$$

$$\sigma_a = \frac{\text{fwhm}_a}{2\sqrt{2\ln 2}}$$

asldro.utils.generate_numeric_function.**generate_point_function**(*xx: numpy.ndarray*, *yy: numpy.ndarray*, *zz: numpy.ndarray*, *loc: Union[tuple, list, numpy.ndarray] = numpy.zeros*) → numpy.ndarray

Generates an array where the element closest to the location defined by `loc` is 1.0. If `loc` is out of bounds then no point is created. Can be used with *generate_circular_function_array()*.

Parameters

- **xx** (*np.ndarray*) – Array of x-coordinates, generate by meshgrid. Can be sparse.
- **yy** (*np.ndarray*) – Array of y-coordinates, generate by meshgrid. Can be sparse.
- **zz** (*np.ndarray*) – Array of z-coordinates, generate by meshgrid. Can be sparse.
- **loc** (*Union[tuple, list, np.ndarray], optional*) – location of the point, $[x_0, y_0, z_0]$, defaults to np.zeros((3,))

Returns An array with shape the result of broadcasting `xx`, `yy`, and `zz`, where the element closes to the location defined by `loc` is 1.0, other elements are 0.0. If `loc` is out of bounds all elements are 0.0.

Return type np.ndarray

asldro.utils.resampling module

Utility Functions for the ASL DRO

asldro.utils.resampling.**rot_x_mat**(*theta: float*) → numpy.ndarray
creates a 4x4 affine performing rotations about x

Parameters **theta** (*float*) – angle to rotate about x in degrees

Returns 4x4 affine for rotating about x

Return type np.array

asldro.utils.resampling.**rot_y_mat**(*theta: float*) → numpy.ndarray
creates a 4x4 affine performing rotations about y

Parameters **theta** (*float*) – angle to rotate about y in degrees

Returns 4x4 affine for rotating about y

Return type np.array

asldro.utils.resampling.**rot_z_mat**(*theta: float*) → numpy.ndarray
creates a 4x4 affine performing rotations about z

Parameters **theta** (*float*) – angle to rotate about z in degrees

Returns 4x4 affine for rotating about z

Return type np.array

asldro.utils.resampling.**scale_mat**(*scale: Union[Tuple[float, float, float], numpy.ndarray]*) → numpy.ndarray
Creates a 4x4 affine performing scaling

Parameters **vector** (*Tuple[float, float, float]*) – describes (sx, sy, sz) scaling factors

Returns 4x4 affine for scaling

Return type np.array

asldro.utils.resampling.**transform_resample_affine**(*image: Union[nibabel.Nifti1Image, nibabel.Nifti2Image, asldro.containers.image.BaseImageContainer]*, *translation: Tuple[float, float, float]*, *rotation: Tuple[float, float, float]*, *rotation_origin: Tuple[float, float, float]*, *target_shape: Tuple[int, int, int]*) → Tuple[numpy.ndarray, numpy.ndarray]

Calculates the affine matrices that transform and resample an image in world-space. Note that while an image (NIFTI or BaseImageContainer derived) is accepted an an argument, the image is not acutally resampled.

Parameters

- **image** (*Union[nib.Nifti1Image, nib.Nifti2Image, BaseImageContainer]*) – The input image
- **translation** (*Tuple[float, float, float]*) – vector to translate in the image by in world space
- **rotation** (*Tuple[float, float, float]*) – angles to rotate the object by in world space
- **rotation_origin** (*Tuple[float, float, float]*) – coordinates for the centre point of rotation in world space
- **target_shape** (*Tuple[int, int, int]*) – target shape for the resampled image

Returns [*target_affine_with_motion*, *target_affine_no_motion*]. *target_affine_with_motion* is the affine to supply to a resampling function. After resampling theresampled image's affine should be set to to *target_affine_no_motion* so that it only has the image formation (scaling) operation performed (not the motion)

Return type Tuple[np.array, np.array]

asldro.utils.resampling.**transform_resample_image**(*image: Union[nibabel.Nifti1Image, nibabel.Nifti2Image, asldro.containers.image.BaseImageContainer]*, *translation: Tuple[float, float, float]*, *rotation: Tuple[float, float, float]*, *rotation_origin: Tuple[float, float, float]*, *target_shape: Tuple[int, int, int]*) → Tuple[Union[nibabel.Nifti2Image, nibabel.Nifti2Image], numpy.ndarray]

Transforms and resamples a nibabel NIFTI image in world-space

Parameters

- **image** (*Union[nib.Nifti1Image, nib.Nifti2Image]*) – The input image
- **translation** (*Tuple[float, float, float]*) – vector to translate in the image by in world space
- **rotation** (*Tuple[float, float, float]*) – angles to rotate the object by in world space
- **rotation_origin** (*Tuple[float, float, float]*) – coordinates for the centre point of rotation in world space
- **target_shape** (*Tuple[int, int, int]*) – target shape for the resampled image

Returns [*resampled_image*, *target_affine*]. *resampled_image* is the input image with the transformation and resampling applied. *target_affine* is the affine that was used to resample the image. Note *resampled_image* has an affine that only corresponds to voxel scaling and not motion, i.e. the image FOV is the same as the input image FOV.

Return type Tuple[Union[nib.Nifti2Image, nib.Nifti2Image], np.array]

asldro.utils.resampling.**translate_mat**(*translation: Union[Tuple[float, float, float], numpy.ndarray]*) → numpy.ndarray

Creates a 4x4 affine performing translations

Parameters **vector** (*Tuple[float, float, float]*) – describes (x, y, z) to translate along respective axes

Returns 4x4 affine for translation

Return type np.array

Module contents

asldro.validators package

Subpackages

asldro.validators.schemas package

Submodules

asldro.validators.schemas.index module

Index of the JSON schema files

asldro.validators.schemas.index.**load_schemas**()

Return all of the schemas in this directory in a dictionary where the keys are the filename (without the .json extension) and the values are the JSON schemas (in dictionary format) :raises jsonschema.exceptions.SchemaError if any of the JSON files in this directory are not valid (Draft 7) JSON schemas

Module contents

Submodules

asldro.validators.ground_truth_json module

A validator for the JSON file used in the ground truth input

asldro.validators.ground_truth_json.**validate_input**(*input_dict: dict*)

Validates the provided dictionary against the ground truth schema. Raises a jsonschema.exceptions.ValidationError on error

asldro.validators.parameters module

Used to perform parameter validation. The most useful documentation can be found on the class 'ParameterValidator'

class asldro.validators.parameters.**Parameter**(*validators: Union[Callable[[...], bool], List[Callable[[...], bool]]], default_value=None, optional=False*)

Bases: object

A description of a parameter which is to be validated against

class asldro.validators.parameters.**ParameterValidator**(*parameters: Dict[str, asldro.validators.parameters.Parameter], post_validators: List[asldro.validators.parameters.Validator] = None*)

Bases: object

Used to validate a dictionary of parameters specified with the Parameter class against an input dictionary. Will also insert any default values that are missing from the input dictionary.

get_defaults()

Return a dictionary of default values for each of the parameters in the ParameterValidator. If a parameter does not have a default value, it is excluded from the dictionary :return: a dictionary of default parameter values

validate(*d: dict, error_type: Type[Exception] = <class 'asldro.validators.parameters.ValidationError'>*) → dict

Validate an input dictionary, replacing missing dictionary entries with default values. If any of the dictionary entries are invalid w.r.t. any of the validators, a ValidationError will be raised (unless error_type is defined, see parameter docs). :param d: the input dictionary. e.g.: {"foo": "bar foo bar"} :param error_type: the type of Exception to be raised. :return: the dictionary with any defaults filled. e.g. {

"foo": "bar foo bar", "bar": [1, 2, 3],

}

exception asldro.validators.parameters.**ValidationError**

Bases: Exception

Used to indicate that a dictionary is invalid

class asldro.validators.parameters.**Validator**(*func: Callable[[Any], bool], criteria_message: str*)

Bases: object

All _validator functions return an object of this class. The object can be called with a value to check whether the value is valid. A string method is also available to display the validator's criteria message.

asldro.validators.parameters.**and_validator**(*validators: List[asldro.validators.parameters.Validator]*) → *asldro.validators.parameters.Validator*

Boolean AND between supplied validators. If all of the supplied validators evaluate as True, this validator evaluates as True.

Parameters **validators** (*Union[List[Validator], Tuple[Validator, ..]]*) – A list (or tuple) of validators

asldro.validators.parameters.**for_each_validator**(*item_validator=<class 'asldro.validators.parameters.Validator'>*) → *asldro.validators.parameters.Validator*

Validates that the value must be iteratable and each of its items are valid based on the item_validator. e.g. validator=for_each_validator(greater_than_validator(0.7)) would create a validator that check all items in a last are > 0.7 :param item_validator: a validator to apply to each item in a list

asldro.validators.parameters.**from_list_validator**(*options: list*, *case_insensitive: bool = False*) → *asldro.validators.parameters.Validator*

Validates that a given value is from a list. :param option: the list of options, one of which the value must match :param case_insensitive: perform a case-insensitive matching

asldro.validators.parameters.**greater_than_equal_to_validator**(*start*) → *asldro.validators.parameters.Validator*

Validate that a given value is greater or equal to a number. Can be used with int, float or BaseImageContainer. :param start: the value to be greater than or equal to

asldro.validators.parameters.**greater_than_validator**(*start*) → *asldro.validators.parameters.Validator*

Validate that a given value is greater than a number. Can be used with int, float or BaseImageContainer. :param start: the value to be greater than

asldro.validators.parameters.**has_attribute_value_validator**(*"a_property"*, *500.0*)

would create a validators that check that an object *(obj') has a property `a_property* that matches 500.0. i.e. *obj.a_property == 500.0* :param attribute_name: the attribute name to compare :param attribute_value: the value of the attribute to compare against

asldro.validators.parameters.**isinstance_validator**(*a_type: Union[type, Tuple[type, ...]]*) → *asldro.validators.parameters.Validator*

Validates that a given value is an instance of the given type(s) (or or derived from). a_type: a type e.g. str, or a tuple of types e.g. (int, str)

asldro.validators.parameters.**list_of_type_validator**(*a_type: Union[type, Tuple[type, ...]]*) → *asldro.validators.parameters.Validator*

Validates that a given value is a list of the given type(s). a_type: a type e.g. str, or a tuple of types e.g. (int, str)

asldro.validators.parameters.**non_empty_list_validator**() → *asldro.validators.parameters.Validator*

Validates that a value is a list and is non-empty

asldro.validators.parameters.**of_length_validator**(*length: int*) → *asldro.validators.parameters.Validator*

Validates that a given value has a given length. Might be, for example, a list or a string. :param length: the required length of the value

asldro.validators.parameters.**or_validator**(*validators: List[*asldro.validators.parameters.Validator*]*) → *asldro.validators.parameters.Validator*

Boolean OR between supplied validators. If any of the supplied validators evaluate as True, this validator evaluates as True.

Parameters **validators** (*Union[List[Validator], Tuple[Validator, ..]]*) – A list (or tuple) of validators

asldro.validators.parameters.**range_exclusive_validator**(*start*, *end*) → *asldro.validators.parameters.Validator*

Validate that a given value is between a given range (excluding the start and end values). Can be used with int, float or BaseImageContainer. :param start: the start value :param end: the end value

asldro.validators.parameters.**range_inclusive_validator**(*start*, *end*) → *asldro.validators.parameters.Validator*

Validate that a given value is between a given range (including the start and end values) Can be used with int, float or BaseImageContainer. :param start: the start value :param end: the end value

asldro.validators.parameters.**regex_validator**(*pattern: str, case_insensitive: bool = False*) → asldro.validators.parameters.Validator

Validates that a value matches the given regex pattern :param pattern: the regex pattern to match :param case_insensitive: perform a case-insensitive matching

asldro.validators.parameters.**reserved_string_list_validator**(*strings: List[str], delimiter=' ', case_insensitive: bool = False*) → asldro.validators.parameters.Validator

Validates that the value is a string which is comprised only of the list of given strings, separated by the delimiter. The strings may be repeated multiple times and in any order, although the value must not be the empty string. e.g. with strings=['foo','bar'] and delimiter='_', this would match: "foo_bar_foo", "foo", "bar", "bar_bar_bar_bar_foo" but would not match: "", "FOO", "anythingelse", "foo__bar", "bar foo" :param strings: a list of strings :param delimiter: a delimiter (defaults to space) :param case_insensitive: perform a case-insensitive matching

asldro.validators.parameters.**shape_validator**(*keys: List[*asldro.containers.image.BaseImageContainer*], maxdim: int = None*) → asldro.validators.parameters.Validator

Checks that all of the keys have the same shape If all the supplied inputs have matching shapes, this validator evaluates as True.

Note this is intended to be used as a post validator as the argument for the validator is a dictionary.

Parameters **images** (*Union[List[str], Tuple[str]]*) – A list (or tuple) of strings

asldro.validators.user_parameter_input module

A user input validator. Used to initialise the model. All of the validation rules are contained within this file. The validator may be used with: *d = USER_INPUT_VALIDATOR(some_input_dictionary) d* will now contain the input dictionary with any defaults values added. A ValidationError will be raised if any validation rules fail.

asldro.validators.user_parameter_input.**asl_context_length_validator_generator**(*other*)

asldro.validators.user_parameter_input.**generate_parameter_distribution**(*param: dict, length=1*) → list

Generates a list of values based on the supplied distribution specification. If the argument param is not a dictionary, its value will be returned. Values will be returned rounded to 4 decimal places.

Parameters

- **param** (*dict*) – Parameter distribution
- **length** (*int, optional*) – number of values to generate, defaults to 1

Raises ***ValidationError*** – [description]

Returns [description]

Return type list

asldro.validators.user_parameter_input.**get_example_input_params**() → dict

Generate and validate an example input parameter dictionary. Will contain one of each supported image type containing the default parameters for each. :return: the validated input parameter dictionary :raises asldro.validators.parameters.ValidationError: if the input

validation does not pass

asldro.validators.user_parameter_input.**validate_input_params**(*input_params: dict*) → dict

Validate the input parameters :param input_params: The input parameters asa Python dict :returns: The parsed input parameter dictionary, with any defaults added :raises asldro.validators.parameters.ValidationError: if the input

validation does not pass

Module contents

Submodules

asldro.cli module

asldro.definitions module

Global module definitions

asldro.examples module

Module contents

asldro init

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Aaron Oliver-Taylor, Thomas Hampshire, Nadia A. S. Smith, Michael Stritt, Jan Petr, Johannes Gregori, Matthias Günther, Henk J. Mutsaerts, and Xavier Golay. ASLDRO: Digital reference object software for Arterial Spin Labelling. In *Proc. Intl. Soc. Mag. Reson. Med. 29*, number 2731. 2021. URL: https://submissions2.mirasmart.com/ISMRM2021/ViewSubmission.aspx?sbmID=1108&validate=false.

[1] David C. Alsop, John A. Detre, Xavier Golay, Matthias Günther, Jeroen Hendrikse, Luis Hernandez-Garcia, Hanzhang Lu, Bradley J. MacIntosh, Laura M. Parkes, Marion Smits, Matthias J. P. van Osch, Danny J. J. Wang, Eric C. Wong, and Greg Zaharchuk. Recommended implementation of arterial spin-labeled perfusion MRI for clinical applications: a consensus of the ISMRM perfusion study group and the european consortium for ASL in dementia. *Magnetic Resonance in Medicine*, 73(1):102–116, apr 2014. doi:10.1002/mrm.25197.

[2] Richard B. Buxton, Lawrence R. Frank, Eric C. Wong, Bettina Siewert, Steven Warach, and Robert R. Edelman. A general kinetic model for quantitative perfusion imaging with arterial spin labeling. *Magnetic Resonance in Medicine*, 40(3):383–396, sep 1998. doi:10.1002/mrm.1910400308.

[3] Nasim Maleki, Weiying Dai, and David C. Alsop. Optimization of background suppression for arterial spin labeling perfusion imaging. *Magnetic Resonance Materials in Physics, Biology and Medicine*, 25(2):127–133, oct 2011. doi:10.1007/s10334-011-0286-3.

[4] Sanjay Mani, John Pauly, Steven Conolly, Craig Meyer, and Dwight Nishimura. Background suppression with multiple inversion recovery nulling: applications to projective angiography. *Magnetic Resonance in Medicine*, 37(6):898–905, jun 1997. doi:10.1002/mrm.1910370615.

PYTHON MODULE INDEX

a

INDEX

A

B

C

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

V

W