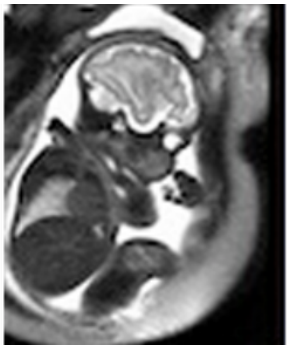
MIALSRTK Documentation

Release 2.1.0

Medical Image Analysis Laboratory and Contributors

Feb 02, 2023

Getting started



mialSRTK

Medical Image Analysis Laboratory Super-Resolution TooKit

Latest released version: v2.1.0

This neuroimaging processing pipeline software is developed by the Medical Image Analysis Laboratory (MIAL) at the University Hospital of Lausanne (CHUV) for use within the lab, as well as for open-source software distribution.

3 4 5 6 7 8 9 10

> **Warning:** THIS SOFTWARE IS FOR RESEARCH PURPOSES ONLY AND SHALL NOT BE USED FOR ANY CLINICAL USE. THIS SOFTWARE HAS NOT BEEN REVIEWED OR APPROVED BY THE FOOD AND DRUG ADMINISTRATION OR EQUIVALENT AUTHORITY, AND IS FOR NON-CLINICAL, IRB-APPROVED RESEARCH USE ONLY. IN NO EVENT SHALL DATA OR IMAGES GENERATED THROUGH THE USE OF THE SOFTWARE BE USED IN THE PROVISION OF PATIENT CARE.

[3] https://doi.org/10.5281/zenodo.4290209
[4] https://hub.docker.com/repository/docker/sebastientourbier/mialsuperresolutiontoolkit
[5] https://travis-ci.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit
[6] https://app.circleci.com/pipelines/github/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit
[7] https://www.codacy.com/gh/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit?utm_source=github.com&utm_medium=referral&utm_content=Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit&utm_campaign=Badge_Coverage
[8] https://mialsrtk.readthedocs.io/en/latest/?badge=latest
[9] https://www.codacy.com/gh/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit?utm_source=github.com&utm_medium=referral&utm_content=Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit&utm_campaign=Badge_Grade
[10] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit#credits

1 Introduction

The `Medical Image Analysis Laboratory Super-Resolution ToolKit (MIALSRTK)` consists of a set of C++ and Python3 image processing and worflow tools necessary to perform motion-robust super-resolution fetal MRI reconstruction.

The *original* `C++ MIALSRTK` library includes all algorithms and methods for brain extraction, intensity standardization, motion estimation and super-resolution. It uses the CMake build system and depends on the open-source image processing Insight ToolKit (ITK) library, the command line parser TCLAP library and OpenMP for multi-threading.

`MIALSRTK` has been extended with the `pymialsrtk` Python3 library following recent advances in standardization of neuroimaging data organization and processing workflows (See *BIDS and BIDS App standards* (page 7)). This library has a modular architecture built on top of the Nipype dataflow library which consists of (1) processing nodes that interface with each of the MIALSRTK C++ tools and (2) a processing pipeline that links the interfaces in a common workflow.

The processing pipeline with all dependencies including the C++ MIALSRTK tools are encapsulated in a Docker image container, which is now distributed as a `BIDS App` which handles datasets organized following the BIDS standard. See *BIDS App usage* (page 9) for more details.

All these design considerations allow us not only to (1) represent the entire processing pipeline as an *execution graph, where each MIALSRTK C++ tools are connected*, but also to (2) provide a *mecanism to record data provenance and execution details*, and to (3) easily customize the BIDS App to suit specific needs as interfaces with *new tools can be added with relatively little effort* to account for additional algorithms.

New

You can now be aware about the adverse impact of your processing on the environment !

With the new `--track_carbon_footprint` option of the `mialsuperresolution-toolkit_docker` and `mialsuperresolutiontoolkit_singularity` BIDS App python wrappers, you can use codecarbon[11] to estimate the amount of carbon dioxide (CO2) produced to execute the code by the computing resources and save the results in `<bids_dir>/code/emissions.csv`.

Then, to visualize, interpret and track the evolution of the CO2 emissions incurred, you can use the visualization tool of `codecarbon` aka `carbonboard` that takes as input the `csv` created:

```
carbonboard --filepath="<bids_dir>/code/emissions.csv" --port=xxxx
```

[11] https://codecarbon.io/

1.1 Aknowledgment

If your are using `MIALSRTK` in your work, please acknowledge this software and its dependencies. See *Citing* (page 64) for more details.

1.2 License information

This software is distributed under the open-source license Modified BSD. See *license* (page 63) for more details.

All trademarks referenced herein are property of their respective holders.

1.3 Help/Questions

If you run into any problems or have any code bugs or questions, please create a new GitHub Issue[12].

1.4 Eager to contribute?

See *Contributing* (page 69) for more details.

1.5 Funding

Originally supported by the Swiss National Science Foundation (grant SNSF-141283).

[12] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/issues

2 Contents

2.1 Installation Instructions for Users

Warning: This software is for research purposes only and shall not be used for any clinical use. This software has not been reviewed or approved by the Food and Drug Administration or equivalent authority, and is for non-clinical, IRB-approved Research Use Only. In no event shall data or images generated through the use of the Software be used in the provision of patient care.

Installation of the `MIALSRTK` processing tools and pipelines has been facilitated through the distribution of a BIDSApp relying on the Docker[13] and Singularity[14] software container technologies, so in order to run `MIALSRTK`, Docker or Singularity must be installed (see instructions in *Prerequisites* (page 5)).

Once Docker or Singularity is installed, the recommended way to run `MIALSRTK` is to use the corresponding `mialsuperresolutiontoolkit` wrapper. Installation instructions for the wrappers can be found in *Wrappers Installation* (page 7), which requires as prerequisites having Python3 (see *Prerequisites* (page 7)) installed and an Internet connection.

If you need a finer control over the Docker/Singularity container execution, or you feel comfortable with the Docker/Singularity Engine, download instructions for the `MIALSRTK BIDS App` can be found in *MIALSRTK Container Image Download* (page 6).

2.1.1 Prerequisites

To run `MIALSRTK` you will need to have either Docker or Singularity containerization engine installed.

While Docker enables `MIALSRTK` to be run on all major operating systems where you have root privileges, Singularity allows you to run `MIALSRTK` on Linux systems where you might not have root privileges such as a High Performance Computing cluster.

Please check https://docs.docker.com/get-started/overview/ and https://sylabs.io/guides/3.7/user-guide/introduction.html If you want to learn more about Docker and Singularity.

[13] https://www.docker.com/
[14] https://sylabs.io/

Installation of Docker Engine

- Install Docker Engine corresponding to your system:
  - For Ubuntu 14.04/16.04/18.04, follow the instructions from https://docs.docker.com/install/linux/docker-ce/ubuntu/
  - For Mac OSX (>=10.10.3), get the .dmg installer from https://store.docker.com/editions/community/docker-ce-desktop-mac
  - For Windows (>=10), get the installer from https://store.docker.com/editions/community/docker-ce-desktop-windows

---

Note: The MIALSRTK BIDSApp has been tested only on Ubuntu and MacOSX. For Windows users, it might be required to make few patches in the Dockerfile.

- Set Docker to be managed as a non-root user
  - Open a terminal
  - Create the docker group:

    ```
    $ sudo groupadd docker
    ```

  - Add the current user to the docker group:

    ```
    $ sudo usermod -G docker -a $USER
    ```

  - Reboot

    After reboot, test if docker is managed as non-root:

    ```
    $ docker run hello-world
    ```

Installation of Singularity Engine

- Install singularity following instructions from the official documentation webpage at https://sylabs.io/guides/3.7/user-guide/quick_start.html#quick-installation-steps

---

Note: If you need to make the request to install Singularity on your HPC, Singularity provides a nice template at https://singularity.lbl.gov/install-request#installation-request to facilitate it.

2.1.2 MIALSRTK Container Image Download

Running Docker?

- Open a terminal
- Get the latest release (v2.1.0) of the BIDS App:

  ```
  $ docker pull sebastientourbier/mialsuperresolutiontoolkit:v2.1.0
  ```

- To display all docker images available:

  ```
  $ docker images
  ```

  You should see the docker image “mialsuperresolutiontoolkit” with tag “v2.1.0” is now available.

- You are ready to use the Docker image of `MIALSRTK` from the terminal. See its commandline usage.

Running Singularity?

- Open a terminal
- Get the latest release (v2.1.0) of the BIDS App:

  ```
  $ singularity pull library://tourbier/default/mialsuperresolutiontoolkit:
  ↪v2.1.0
  ```

- You are ready to use the Singularity image of `MIALSRTK`. See its commandline usage.

2.1.3 The lightweight MIALSRTK wrappers

Prerequisites

The wrappers requires a Python3 environment. We recommend you tu use miniconda3 for which the installer corresponding to your 32/64bits MacOSX/Linux/Win system can be downloaded from https://conda.io/miniconda.html.

Wrappers Installation

Once Python3 is installed, the `mialsuperresolutiontoolkit_docker` and `mialsuperresolutiontoolkit_singularity` wrappers can be installed via `pip` as follows:

- Open a terminal
- Installation with `pip`:

  ```
  $ pip install pymialsrtk==2.1.0
  ```

- You are ready to use the `mialsuperresolutiontoolkit_docker` and `mialsuperresolutiontoolkit_singularity` wrappers. See their commandline usages.

Important: On Mac and Windows, if you want to track the carbon emission incurred by the processing with the `--track_carbon_footprint` option flag, you will need to install the `Intel Power Gadget` tool available `here`.

Help/Questions

Code bugs can be reported by creating a new GitHub Issue[15].

2.2 BIDS and BIDS App standards

`MIALSRTK BIDS App` adopts the BIDS (Brain Imaging Data Structure) standard for data organization and is developed following the BIDS App standard. This means that `MIALSRTK BIDS App` handles dataset formatted following the BIDS App standard and provides a processing workflow containerized in Docker container image (promoting portability and reproduciblity) that can be run with a set of arguments defined by the BIDS App standard directly from the terminal or a script (See *Commandline Usage* (page 9) section for more details).

[15] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/issues

For more information about BIDS and BIDS-Apps, please consult the BIDS Website[16], the Online BIDS Specifications[17], and the BIDSApps Website[18]. HeuDiConv[19] can assist you in converting DICOM brain imaging data to BIDS. A nice tutorial can be found @ BIDS Tutorial Series: HeuDiConv Walkthrough[20] .

2.2.1 BIDS dataset schema

The BIDS App accepts BIDS datasets that adopt the following organization, naming, and file formats:

```
ds-example/

    README
    CHANGES
    participants.tsv
    dataset_description.json

    sub-01/
        anat/
            sub-01_run-1_T2w.nii.gz
            sub-01_run-1_T2w.json
            sub-01_run-2_T2w.nii.gz
            sub-01_run-2_T2w.json
            ...

    ...

    sub-<subject_label>/
        anat/
            sub-<subject_label>_run-1_T2w.nii.gz
            sub-<subject_label>_run-1_T2w.json
            sub-<subject_label>_run-2_T2w.nii.gz
            sub-<subject_label>_run-2_T2w.json
            ...
        ...
    ...

    code/
        participants_params.json
```

where `participants_params.json` is the MIALSRTK BIDS App configuration file, which follows a specific schema (See *config schema* (page 11)), and which defines multiple processing parameters (such as the ordered list of scans or the weight of regularization).

Important: Before using any BIDS App, we highly recommend you to validate your BIDS structured dataset with the free, online BIDS Validator[21].

[16] https://bids.neuroimaging.io/
[17] https://bids-specification.readthedocs.io/en/stable/
[18] https://bids-apps.neuroimaging.io/
[19] https://github.com/nipy/heudiconv
[20] http://reproducibility.stanford.edu/bids-tutorial-series-part-2a/
[21] http://bids-standard.github.io/bids-validator/

2.3 Commandline Usage

MIALSRTK adopts the BIDS standard for data organization and takes as principal input the path of the dataset that is to be processed. The input dataset is required to be in *valid BIDS format*, and it must include *at least one T2w scan with anisotropic resolution per anatomical direction*. See *BIDS and BIDS App standards* (page 7) page that provides links for more information about BIDS and BIDS-Apps as well as an example for dataset organization and naming.

2.3.1 Commandline Arguments

The command to run the MIALSRTK follows the BIDS-Apps[22] definition standard with an additional option for loading the pipeline configuration file.

Argument parser of the MIALSRTK BIDS App

```
usage: mialsuperresolutiontoolkit-bidsapp [-h] [--run_type {sr,
↪preprocessing}]
                                          [--participant_label PARTICIPANT_
↪LABEL [PARTICIPANT_LABEL ...]]
                                          [--param_file PARAM_FILE]
                                          [--openmp_nb_of_cores OPENMP_NB_
↪OF_CORES]
                                          [--nipype_nb_of_cores NIPYPE_NB_
↪OF_CORES]
                                          [--memory MEMORY]
                                          [--masks_derivatives_dir MASKS_
↪DERIVATIVES_DIR]
                                          [--labels_derivatives_dir LABELS_
↪DERIVATIVES_DIR]
                                          [--all_outputs] [-v] [--verbose]
                                         bids_dir output_dir {participant}
```

Positional Arguments

bids_dir The directory with the input dataset formatted according to the BIDS standard.

output_dir The directory where the output files should be stored. If you are running group level analysis this folder should be pre-populated with the results of the participant level analysis.

analysis_level Possible choices: participant

Level of the analysis that will be performed. Only participant is available

[22] https://github.com/BIDS-Apps

Named Arguments

--run_type Possible choices: sr, preprocessing

Type of pipeline that is run. Can choose between running the super-resolution pipeline (*sr*) or only preprocessing (*preprocessing*).

Default: "sr"

--participant_label The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided all subjects should be analyzed. Multiple participants can be specified with a space separated list.

--param_file Path to a JSON file containing subjects' exams information and super-resolution total variation parameters.

Default: "/bids_dir/code/participants_params.json"

--openmp_nb_of_cores Specify number of cores used by OpenMP threads Especially useful for NLM denoising and slice-to-volume registration. (Default: 0, meaning it will be determined automatically)

Default: 0

--nipype_nb_of_cores Specify number of cores used by the Niype workflow library to distribute the execution of independent processing workflow nodes (i.e. interfaces) (Especially useful in the case of slice-by-slice bias field correction and intensity standardization steps for example). (Default: 0, meaning it will be determined automatically)

Default: 0

--memory Limit the workflow to using the amount of specified memory [in gb] (Default: 0, the workflow memory consumption is not limited)

Default: 0

--masks_derivatives_dir Use manual brain masks found in `<output_dir>/<masks_derivatives_dir>/` directory

--labels_derivatives_dir Use low-resolution labelmaps found in `<output_dir>/<labels_derivatives_dir>/` directory.

--all_outputs Whether or not all outputs should be kept(e.g. preprocessed LR images)

Default: False

-v, --version show program's version number and exit

--verbose Verbose mode

Default: False

BIDS App configuration file

The BIDS App configuration file specified by the input flag *–param_file* adopts the following JSON schema:

```
{
  "01": [
    { "sr-id": 1,
      ("session": 01,)
      "stacks": [1, 3, 5, 2, 4, 6],
      "paramTV": {
        "lambdaTV": 0.75,
        "deltatTV": 0.01 }
    },
    { "sr-id": 2,
      ("session": 01,)
      "stacks": [2, 3, 5, 4],
      "ga": 25,
      "paramTV": {
        "lambdaTV": 0.75,
        "deltatTV": 0.01 },
      "custom_interfaces":
        {
        "skip_svr": true,
        "do_refine_hr_mask": false,
        "skip_stacks_ordering": false,
        "do_anat_orientation": true
        }
    }]
  "02": [
    { "sr-id": 1,
      ("session": 01,)
      "stacks": [3, 1, 2, 4],
      "paramTV": {
        "lambdaTV": 0.7,
        "deltatTV": 0.01 }
    }]
  ...
}
```

where:

- `"sr-id"` (mandatory) allows to distinguish between runs with different configurations of the same acquisition set.
- `"stacks"` (optional) defines the list of scans to be used in the reconstruction. The specified order is considered if `"skip_stacks_ordering"` is False
- `"paramTV"` (optional): `"lambdaTV"` (regularization), `"deltaTV"` (optimization time step),

`"num_iterations"`, `"num_primal_dual_loops"`, `"num_bregman_loops"`, `"step_scale"`, `"gamma"` are parameters of the TV super-resolution algorithm.

- `"session"` (optional) It MUST be specified if you have a BIDS dataset composed of multiple sessions with the *sub-XX/ses-YY* structure.
- `"ga"` (optional but mandatory when `do_anat_orientation` is true) subject's gestational age in weeks.
- `"run_type"` (optional): defines the type of run that should be done. It can be set

between *sr* (super-resolution) and *preprocessing* (preprocessing-only). (default is `"sr"`)

- `"custom_interfaces"` (optional): indicates whether optional interfaces of the pipeline should be performed.
  - `"skip_svr"` (optional) the Slice-to-Volume Registration should be skipped in the image reconstruction. (default is False)
  - `"do_refine_hr_mask"` (optional) indicates whether a refinement of the HR mask should be performed. (default is False)
  - `"skip_preprocessing"` (optional) indicates whether the preprocessing stage should be skipped. A minimal preprocessing is still computed: the field-of-view is reduced based on the brain masks and the LR series are masked on the ROI. (default is False)

    Note: This option requires input images to be normalised in the range [0,255] prior to running the code with this option. The projection step of the TV algorithm will otherwise clip values to 255.

  - `"do_nlm_denoising"` (optional) indicates whether the NLM denoising preprocessing should be performed prior to motion estimation. (default is False)
  - `"do_reconstruct_labels"` (optional) indicates whether the reconstruction of LR label maps should be performed together with T2w images. (default is False)
  - `"skip_stacks_ordering"` (optional) indicates whether the order of stacks specified in `"stacks"` should be kept or re-computed. (default is False)
  - `"do_anat_orientation"` (optional) indicates whether the alignement into anatomical planes should be performed. If True, path to a directory containing STA atlas (Gholipour et al., 2017[1],[2]) must be mounted to */sta*. (default is False)
  - `"preproc_do_registration"` (optional) indicates whether the Slice-to-Volume Registration should be computed in the `"preprocessing"` run (default is False).
  - `"do_multi_parameters"` (optional) enables running the super-resolution reconstruction with lists of parameters. The algorithm will

  then run a grid search over all combinations of parameters. (default is False)

  - `"do_srr_assessment"` (optional) enables comparing the quality of the super-resolution reconstruction with a reference image. (default is False)

  If True, it will require a reference isotropic T2w image, mask and labels located in the data folder.

[1] Gholipour et al.; A normative spatiotemporal MRI atlas of the fetal brain for automatic segmentation and analysis of early brain growth, Scientific Reports 7, Article number: 476 (2017). **(link to article)<http://www.nature.com/articles/s41598-017-00525-w>_** .

[2] (link to download)[23]

[23] http://crl.med.harvard.edu/research/fetal_brain_atlas/

References

Important: Before using any BIDS App, we highly recommend you to validate your BIDS structured dataset with the free, online BIDS Validator[24].

2.3.2 Running *MIALSRTK*

You can run the *MIALSRTK* using the lightweight Docker or Singularity wrappers we created for convenience or you can interact directly with the Docker / Singularity Engine via the docker or singularity run command. (See *Installation Instructions for Users* (page 5))

New

You can now be aware about the adverse impact of your processing on the environment !

With the new *–track_carbon_footprint* option of the *mialsuperresolutiontoolkit_docker* and *mialsuperresolutiontoolkit_singularity* BIDS App python wrappers, you can use codecarbon[25] to estimate the amount of carbon dioxide (CO2) produced to execute the code by the computing resources and save the results in `<bids_dir>/code/emissions.csv`.

Then, to visualize, interpret and track the evolution of the CO2 emissions incurred, you can use the visualization tool of *codecarbon* aka *carbonboard* that takes as input the *.csv* created:

```
carbonboard --filepath="<bids_dir>/code/emissions.csv" --port=xxxx
```

With the wrappers

When you run `mialsuperresolutiontoolkit_docker`, it will generate a Docker command line for you, print it out for reporting purposes, and then execute it without further action needed, e.g.:

```
$ mialsuperresolutiontoolkit_docker \
    /home/localadmin/data/ds001 /media/localadmin/data/ds001/
↪derivatives \
    participant --participant_label 01 \
    --param_file /home/localadmin/data/ds001/code/participants_
↪params.json \
    --track_carbon_footprint \
    (--openmp_nb_of_cores 4) \
    (--nipype_nb_of_cores 4)
```

When you run `mialsuperresolutiontoolkit_singularity`, it will generate a Singularity command line for you, print it out for reporting purposes, and then execute it without further action needed, e.g.:

```
$ mialsuperresolutiontoolkit_singularity \
    /home/localadmin/data/ds001 /media/localadmin/data/ds001/
↪derivatives \
    participant --participant_label 01 \
    --param_file /home/localadmin/data/ds001/code/participants_
↪params.json \
```

(continues on next page)

[24] http://bids-standard.github.io/bids-validator/
[25] https://codecarbon.io/

(continued from previous page)

```
    --track_carbon_footprint \
    (--openmp_nb_of_cores 4) \
    (--nipype_nb_of_cores 4)
```

With the Docker / Singularity Engine

If you need a finer control over the container execution, or you feel comfortable with the Docker or Singularity Engine, avoiding the extra software layer of the wrapper might be a good decision.

For instance, the previous call to the `mialsuperresolutiontoolkit_docker` wrapper corresponds to:

```
$ docker run -t --rm -u $(id -u):$(id -g) \
        -v /home/localadmin/data/ds001:/bids_dir \
        -v /media/localadmin/data/ds001/derivatives:/output_dir \
        (-v /path/to/CRL_Fetal_Brain_Atlas:/sta \)
        sebastientourbier/mialsuperresolutiontoolkit:v2.1.0 \
        /bids_dir /output_dir participant --participant_label 01 \
        --param_file /bids_dir/code/participants_params.json \
        (--openmp_nb_of_cores 4) \
        (--nipype_nb_of_cores 4)
```

Note: We use the *-v /path/to/local/folder:/path/inside/container* docker run option to access local files and folders inside the container such that the local directory of the input BIDS dataset (here: `/home/localadmin/data/ds001`) and the output directory (here: `/media/localadmin/data/ds001/derivatives`) used to process are mapped to the folders `/bids_dir` and `/output_dir` in the container respectively.

The previous call to the `mialsuperresolutiontoolkit_singularity` wrapper corresponds to:

```
$ singularity run --containall \
        --bind /home/localadmin/data/ds001:/bids_dir \
        --bind /media/localadmin/data/ds001/derivatives:/output_dir \
        library://tourbier/default/mialsuperresolutiontoolkit:v2.1.0 \
        /bids_dir /output_dir participant --participant_label 01 \
        --param_file /bids_dir/code/participants_params.json \
        (--openmp_nb_of_cores 4) \
        (--nipype_nb_of_cores 4)
```

Note: Similarly as with Docker, we use the *–bind /path/to/local/folder:/path/inside/container* singularity run option to access local files and folders inside the container such that the local directory of the input BIDS dataset (here: `/home/localadmin/data/ds001`) and the output directory (here: `/media/localadmin/data/ds001/derivatives`) used to process are mapped to the folders `/bids_dir` and `/output_dir` in the container respectively.

2.3.3 Debugging

Logs are outputted into `<output dir>/nipype/sub-<participant_label>/anatomical_pipeline/rec<srId>/pypeline.log`.

2.3.4 Support, bugs and new feature requests

All bugs, concerns and enhancement requests for this software are managed on GitHub and can be submitted at https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/issues.

2.3.5 Not running on a local machine? - Data transfer

If you intend to run *MIALSRTK* on a remote system, you will need to make your data available within that system first. Comprehensive solutions such as Datalad[26] will handle data transfers with the appropriate settings and commands. Datalad also performs version control over your data.

2.4 Outputs of MIALSRTK BIDS App

Processed, or derivative, data are outputed to `<bids_dataset/derivatives>/` and follow the BIDS v1.4.1 standard (see BIDS Derivatives[27]) whenever possible.

2.4.1 BIDS derivatives entities

Entity	Description
`sub-<label>`	Distinguish different subjects
`ses-<label>`	Distinguish different T2w scan acquisition sessions
`run-<label>`	Distinguish different T2w scans
`rec-<label>`	Distinguish images reconstructed using scattered data interpolation (SDI) or super-resolution (SR) methods
`id-<label>`	Distinguish outputs of reconstructions run multiple times with different configuration

See Original BIDS Entities Appendix[28] for more description.

Note: A new entity `id-` has been introduced to distinguish between outputs when the pipeline is run with multiple configurations (such a new order of scans) on the same subject.

[26] http://www.datalad.org/
[27] https://bids-specification.readthedocs.io/en/v1.4.1/05-derivatives/01-introduction.html
[28] https://bids-specification.readthedocs.io/en/v1.4.1/99-appendices/09-entities.html

Main MIALSRTK BIDS App Derivatives

Main outputs produced by MIALSRTK BIDS App are written to `<bids_dataset/derivatives>/pymialsrtk-<variant>/sub-<label>/(_ses-<label>/)`. The execution log of the full workflow is saved as `sub-<label>(_ses-<label>)_id-<label>_log.txt`.

2.4.2 Anatomical derivatives

- Anatomical derivatives are placed in each subject's `anat/` subfolder, including:
  - The brain masks of the T2w scans:
    * `anat/sub-<label>(_ses-<label>)_run-01_id-<label>_desc-brain_mask.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_run-02_id-<label>_desc-brain_mask.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_run-03_id-<label>_desc-brain_mask.nii.gz`
    * ...
  - The preprocessed T2w scans used for slice motion estimation and scattered data interpolation (SDI) reconstruction:
    * `anat/sub-<label>(_ses-<label>)_run-01_id-<label>_desc-preprocSDI_T2w.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_run-02_id-<label>_desc-preprocSDI_T2w.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_run-03_id-<label>_desc-preprocSDI_T2w.nii.gz`
    * ...
  - The preprocessed T2w scans used for super-resolution reconstruction:
    * `anat/sub-<label>(_ses-<label>)_run-01_id-<label>_desc-preprocSR_T2w.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_run-02_id-<label>_desc-preprocSR_T2w.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_run-03_id-<label>_desc-preprocSR_T2w.nii.gz`
    * ...
  - The high-resolution image reconstructed by SDI:
    * `anat/sub-<label>(_ses-<label>)_rec-SDI_id-<label>_T2w.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_rec-SDI_id-<label>_T2w.json`
  - The high-resolution image reconstructed by SDI:
    * `anat/sub-<label>(_ses-<label>)_rec-SR_id-<label>_T2w.nii.gz`
    * `anat/sub-<label>(_ses-<label>)_rec-SR_id-<label>_T2w.json`
- The slice-by-slice transforms of all T2W scans estimated during slice motion estimation and SDI reconstruction and used in the super-resolution forward model are placed in each subject's `xfm/` subfolder:
  - `xfm/sub-<label>(_ses-<label>)_run-1_id-<label>_T2w_from-origin_to-SDI_mode-image_xfm.txt`

  - `xfm/sub-<label>(_ses-<label>)_run-2_id-<label>_T2w_from-origin_to-SDI_mode-image_xfm.txt`
  - `xfm/sub-<label>(_ses-<label>)_run-3_id-<label>_T2w_from-origin_to-SDI_mode-image_xfm.txt`
  - `...`
- The pipeline execution log file can be found in each subject's `logs/` subfolder as:
  - `logs/sub-01_rec-SR_id-1_log.txt`
- The HTML processing report which provides in one place: pipeline/workflow configuration summary, Nipype workflow execution graph, links to the log and the profiling output report, plots for the quality check of the automatic reordering step based on the motion index, three orthogonal cuts of the reconstructed image, and computing environment summary. It is placed in each subject's `report/` subfolder:
    - `report/sub-<label>.html`

Super-Resolution Reconstruction Report

- Subject: 01
- Reconstruction ID: 1

Configuration

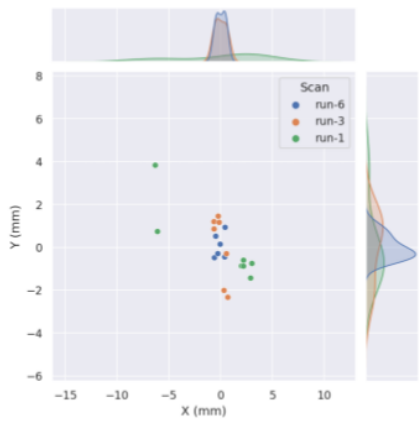
- Number of scans: 3
- Original scan run index order: [3, 1, 6]
- Motion-based scan auto-ordering: ✓
- Automatic brain extraction: ✓
- NLM denoising: ✓
- Slice-to-volume registration: ✓
- SR brain mask refinement: ✗
- Computational resources profiling: ✓

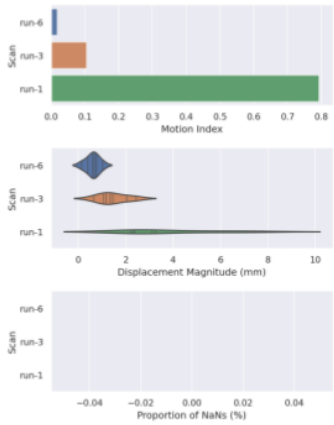
Workflow

Execution

- Starting processing date / time: July 26, 2021 / 07:58:54
- Elapsed time: 17 minutes and 41 seconds
- Log file: ../logs/sub-01_rec-SR_id-1_log.txt
- Computational resources report: ../logs/sub-01_rec-SR_id-1_desc-profiling_log.html

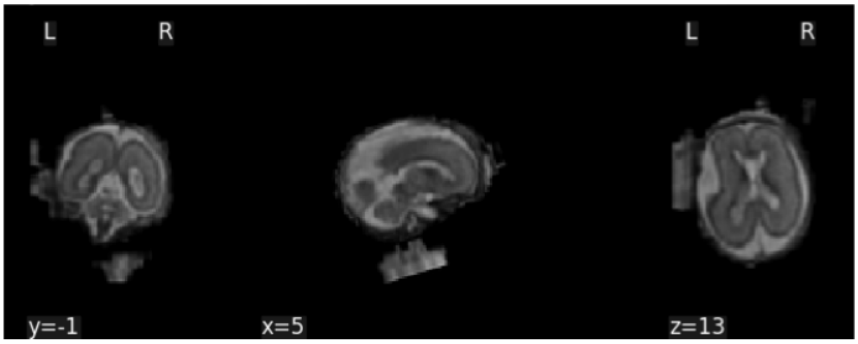
Motion and stack order





Super-Resolution

- Input scan run index order: [6, 3, 1]
- Output resolution: 1.125 x 1.125 x 1.125 mm^3
- Algorithm: Total-Variation
- Regularization weight (Lambda): 0.75
- Optimization time step: 0.01
- Number of primal/dual iterations: 10



Computing Environment

- Python: 3.7.10 | packaged by conda-forge | (default, Feb 19 2021, 16:07:37) [GCC 9.3.0]
- OS: Linux 4.15.0-1027-gcp

Multithreading

Number of threads for:

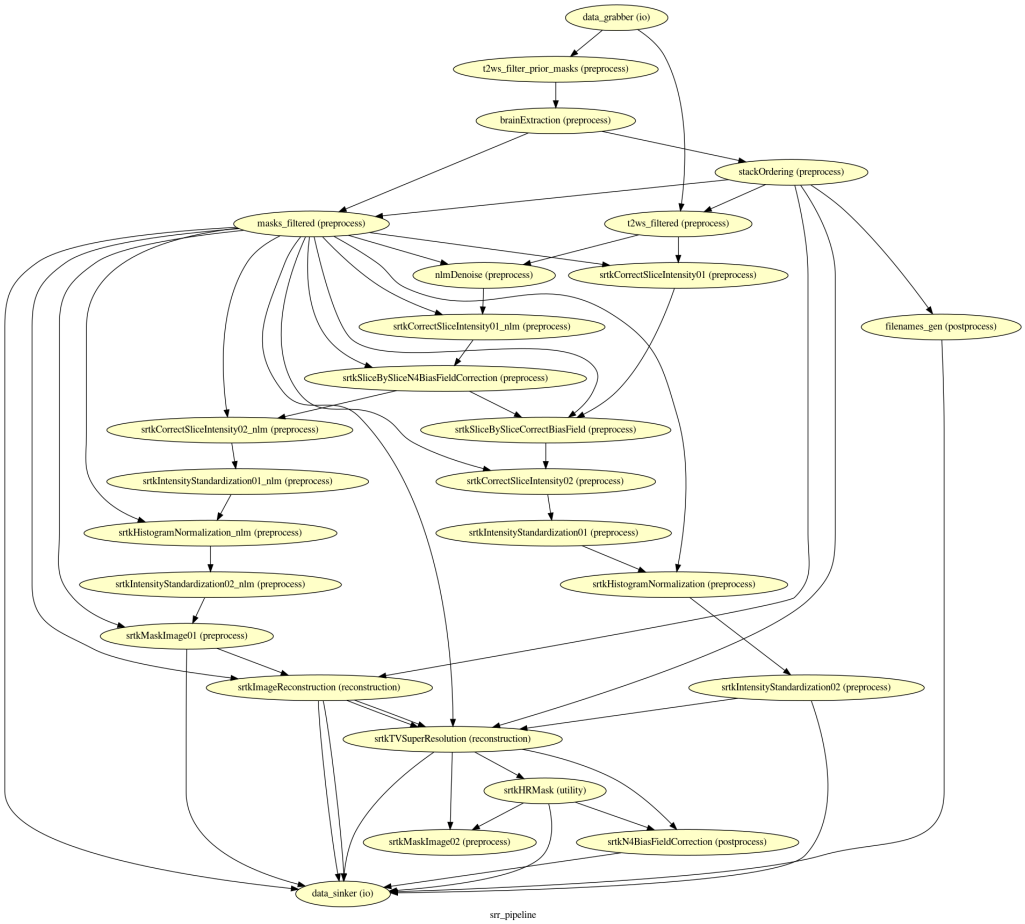
- Nipype: 1
- OpenMP: 1

Auto-generated by MIALSRTK version 2.0.2-dev using Jinja 3.0.1

Copyright © 2016-2021 Medical Image Analysis Laboratory, University Hospital Center and University of Lausanne (UNIL-CHUV), Switzerland, and contributors

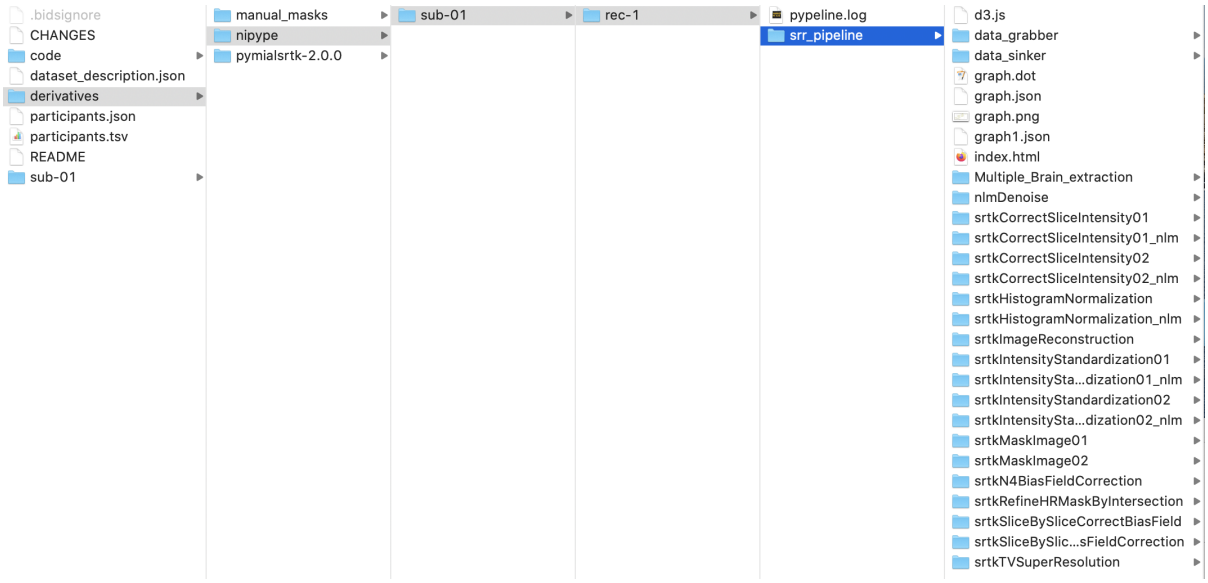
Powered by w3.css

- The PNG image generated by the stack auto-reordering node and used in the HTML processing report can be found in each subject's `figures/` subfolder as:
  - `figures/sub-01_rec-SR_id-1_desc-motion_stats.png`
- The Nipype workflow execution graph used in the HTML processing report, which summarizes all processing nodes involves in the given processing pipeline, can be found in each subject's `figures/` folder as `sub-01_rec-SR_id-1_desc-processing_graph.png`:



Nipype Workflow Derivatives

The execution of the Nipype workflow (pipeline) involves the creation of a number of intermediate outputs for each subject `sub-` and each run `rec-` which are written to `<bids_dataset/derivatives>/nipype/sub-<label>/rec-<id_label>/srr_pipeline` where `<id_label>` corresponds to the label used previously for the entity `id-`:



Execution details (data provenance) of each interface (node) of a given pipeline are reported in `srr_pipeline/<interface_name>/_report/report.rst`

```
report.rst
Node: nlmDenoise (preprocess)
=============================


 Hierarchy : srr_pipeline.nlmDenoise
 Exec ID : nlmDenoise


Original Inputs
---------------


* bids_dir : /bids_dir
* input_images : ['/bids_dir/sub-01/anat/sub-01_run-1_T2w.nii.gz', '/bids_dir/sub-01/anat/sub-01_run-2_T2w.nii.gz', '/bids_dir/sub-01/anat
/sub-01_run-3_T2w.nii.gz', '/bids_dir/sub-01/anat/sub-01_run-4_T2w.nii.gz', '/bids_dir/sub-01/anat/sub-01_run-5_T2w.nii.gz', '/bids_dir/
sub-01/anat/sub-01_run-6_T2w.nii.gz']
* input_masks : <undefined>
* out_postfix : _nlm
* stacks_order : [1, 3, 5]
* weight : 0.1


Execution Inputs
----------------


* bids_dir : /bids_dir
* input_images : ['/bids_dir/sub-01/anat/sub-01_run-1_T2w.nii.gz', '/bids_dir/sub-01/anat/sub-01_run-2_T2w.nii.gz', '/bids_dir/sub-01/anat
/sub-01_run-3_T2w.nii.gz', '/bids_dir/sub-01/anat/sub-01_run-4_T2w.nii.gz', '/bids_dir/sub-01/anat/sub-01_run-5_T2w.nii.gz', '/bids_dir/
sub-01/anat/sub-01_run-6_T2w.nii.gz']
* input_masks : <undefined>
* out_postfix : _nlm
* stacks_order : [1, 3, 5]
* weight : 0.1


Execution Outputs
-----------------


* output_images : ['/output_dir/nipype/sub-01/rec-1/srr_pipeline/nlmDenoise/sub-01_run-1_T2w_nlm.nii.gz', '/output_dir/nipype/
sub-01/rec-1/srr_pipeline/nlmDenoise/sub-01_run-3_T2w_nlm.nii.gz', '/output_dir/nipype/sub-01/rec-1/srr_pipeline/nlmDenoise/
sub-01_run-5_T2w_nlm.nii.gz']


Runtime info
------------


* duration : 95.262152
* hostname : 70e74524c9c5
* prev_wd : /app
* working_dir : /output_dir/nipype/sub-01/rec-1/srr_pipeline/nlmDenoise
```

2.5 Command-line interface (CLI) module

This module defines the `mialsuperresolutiontoolkit_bidsapp_docker` script that wraps calls to the Docker BIDS APP image.

`pymialsrtk.cli.mialsuperresolutiontoolkit_docker.`**`create_docker_cmd`**(*args*)

Function that creates and returns the BIDS App docker run command.

Parameters **args** (*dict*) –

Dictionary of parsed input argument in the form:

```
{
    'bids_dir': "/path/to/bids/dataset/directory",
    'output_dir': "/path/to/output/directory",
    'param_file': "/path/to/configuration/parameter/file",
    'analysis_level': "participant",
    'participant_label': ['01', '02', '03'],
    'openmp_nb_of_cores': 1,
    'nipype_nb_of_cores': 1,
    'masks_derivatives_dir': 'manual_masks'
}
```

Returns **cmd** – String containing the command to be run via `subprocess.run()`

Return type string

`pymialsrtk.cli.mialsuperresolutiontoolkit_docker.main()`

Main function that creates and executes the BIDS App docker command.

Returns

exit_code –

An exit code given to `sys.exit()` that can be:

- '0' in case of successful completion
- '1' in case of an error

Return type {0, 1}

This module defines the `mialsuperresolutiontoolkit_bidsapp_singularity` script that wraps calls to the Singularity BIDS APP image.

`pymialsrtk.cli.mialsuperresolutiontoolkit_singularity.create_singularity_cmd(args)`

Function that creates and returns the BIDS App singularity run command.

Parameters **args** (*dict*) –

Dictionary of parsed input argument in the form:

```
{
    'bids_dir': "/path/to/bids/dataset/directory",
    'output_dir': "/path/to/output/directory",
    'param_file': "/path/to/configuration/parameter/file",
    'analysis_level': "participant",
    'participant_label': ['01', '02', '03'],
    'openmp_nb_of_cores': 1,
    'nipype_nb_of_cores': 1,
    'masks_derivatives_dir': 'manual_masks'
}
```

Returns **cmd** – String containing the command to be run via `subprocess.run()`

Return type string

`pymialsrtk.cli.mialsuperresolutiontoolkit_singularity.main()`

Main function that creates and executes the BIDS App singularity command.

Returns

exit_code –

An exit code given to `sys.exit()` that can be:

- '0' in case of successful completion
- '1' in case of an error

Return type {0, 1}

2.6 Preprocess module

PyMIALSRTK preprocessing functions.

It includes BTK Non-local-mean denoising, slice intensity correction slice N4 bias field correction, slice-by-slice correct bias field, intensity standardization, histogram normalization and both manual or deep learning based automatic brain extraction.

2.6.1 ApplyAlignmentTransform

Link to code[29]

Bases: nipype.interfaces.base.core.BaseInterface

Apply a rigid 3D transform.

Examples

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪ApplyAlignmentTransform
>>> align_img = ApplyAlignmentTransform()
>>> align_img.inputs.input_image = "sub-01_acq-haste_run-1_T2w.
↪nii.gz"
>>> align_img.inputs.input_template = "STA30.nii.gz"
>>> align_img.inputs.input_mask = "sub-01_acq-haste_run-1_T2w_
↪mask.nii.gz"
>>> align_img.inputs.input_transform = "sub-01_acq-haste_run-1_
↪rigid.tfm"
>>> align_img.run()
```

Mandatory Inputs

- **input_image** (*a pathlike object or string representing a file*) – Input image to realign.
- **input_template** (*a pathlike object or string representing a file*) – Input reference image.
- **input_transform** (*a pathlike object or string representing a file*) – Input alignment transform to apply.

Optional Inputs **input_mask** (*a pathlike object or string representing a file*) – Input mask to realign.

Outputs

- **output_image** (*a pathlike object or string representing a file*) – Output reoriented image.
- **output_mask** (*a pathlike object or string representing a file*) – Output reoriented mask.

[29] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L2631-L2713

2.6.2 BrainExtraction

Link to code[30]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the automatic brain extraction module.

This module is based on a 2D U-Net (Ronneberger et al. [1]_) using the pre-trained weights from Salehi et al. [2]_.

References

Examples

```
>>> from pymialsrtk.interfaces.preprocess import BrainExtraction
>>> brainMask = BrainExtraction()
>>> brainmask.inputs.base_dir = '/my_directory'
>>> brainmask.inputs.in_file = 'sub-01_acq-haste_run-1_2w.nii.gz'
>>> brainmask.inputs.in_ckpt_loc = 'my_loc_checkpoint'
>>> brainmask.inputs.threshold_loc = 0.49
>>> brainmask.inputs.in_ckpt_seg = 'my_seg_checkpoint'
>>> brainmask.inputs.threshold_seg = 0.5
>>> brainmask.inputs.out_postfix = '_brainMask.nii.gz'
>>> brainmask.run()
```

Mandatory Inputs

- **in_ckpt_loc** (*a pathlike object or string representing a file*) – Network_checkpoint for localization.
- **in_ckpt_seg** (*a pathlike object or string representing a file*) – Network_checkpoint for segmentation.
- **in_file** (*a pathlike object or string representing a file*) – Input image.

Optional Inputs

- **out_postfix** (*a string*) – Suffix of the automatically generated mask. (Nipype **default** value: `_brainMask`)
- **threshold_loc** (*a float*) – Threshold determining cutoff probability (0.49 by default).
- **threshold_seg** (*a float*) – Threshold for cutoff probability (0.5 by default).

Outputs **out_file** (*a pathlike object or string representing a file*) – Output brain mask image.

[30] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L1178-L2027

2.6.3 BtkNLMDenoising

Link to code[33]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the non-local mean denoising module.

It calls the Baby toolkit implementation by Rousseau et al. [1]_ of the method proposed by Coupé et al. [2]_.

References

Example

```
>>> from pymialsrtk.interfaces.preprocess import BtkNLMDenoising
>>> nlmDenoise = BtkNLMDenoising()
>>> nlmDenoise.inputs.in_file = 'sub-01_acq-haste_run-1_T2w.nii.gz
↪'
>>> nlmDenoise.inputs.in_mask = 'sub-01_acq-haste_run-1_mask.nii.
↪gz'
>>> nlmDenoise.inputs.weight = 0.2
>>> nlmDenoise.run()
```

Mandatory Inputs **in_file** (*a pathlike object or string representing a file*) – Input image filename.

Optional Inputs

- **in_mask** (*a pathlike object or string representing a file*) – Input mask filename.
- **out_postfix** (*a string*) – Suffix to be added to input image filename to construst denoised output filename. (Nipype **default** value: `_nlm`)
- **verbose** (*a boolean*) – Enable verbosity.
- **weight** (*a float*) – NLM smoothing parameter (high beta produces smoother result). (Nipype **default** value: `0.1`)

Outputs **out_file** (*a pathlike object or string representing a file*) – Output denoised image file.

2.6.4 CheckAndFilterInputStacks

Link to code[36]

Bases: nipype.interfaces.base.core.BaseInterface

Runs a filtering and a check on the input files.

This module filters the input files matching the specified run-ids. Other files are discarded.

[33] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L82-L137

[36] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L637-L728

Examples

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪CheckAndFilterInputStacks
>>> stacksFiltering = CheckAndFilterInputStacks()
>>> stacksFiltering.inputs.input_masks = ['sub-01_run-1_mask.nii.
↪gz', 'sub-01_run-4_mask.nii.gz', 'sub-01_run-2_mask.nii.gz']
>>> stacksFiltering.inputs.stacks_id = [1,2]
>>> stacksFiltering.run()
```

Optional Inputs

- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Input images.
- **input_labels** (*a list of items which are a pathlike object or string representing a file*) – Input label maps.
- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Input masks.
- **stacks_id** (*a list of items which are any value*) – List of stacks id to be kept.

Outputs

- **output_images** (*a list of items which are a string*) – Filtered list of image files.
- **output_labels** (*a list of items which are any value*) – Filtered list of label files.
- **output_masks** (*a list of items which are any value*) – Filtered list of mask files.
- **output_stacks** (*a list of items which are any value*) – Filtered list of stack files.

CheckAndFilterInputStacks.**m_output_images = []**

CheckAndFilterInputStacks.**m_output_labels = []**

CheckAndFilterInputStacks.**m_output_masks = []**

CheckAndFilterInputStacks.**m_output_stacks = []**

2.6.5 ComputeAlignmentToReference

Link to code[37]

Bases: nipype.interfaces.base.core.BaseInterface

Reorient image along reference, based on principal brain axis.

This module relies on the implementation **[1]_** from EbnerWang2020 **[2]_**.

[37] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L2436-L2609

References

segmentation and super-resolution reconstruction of fetal brain MRI. NeuroImage, 206, 116324. (link to paper)[39]

Examples

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪ComputeAlignmentToReference
>>> align_to_ref = ComputeAlignmentToReference()
>>> align_to_ref.inputs.input_image = "sub-01_acq-haste_run-1_T2w.
↪nii.gz"
>>> align_to_ref.inputs.input_template = "STA30.nii.gz"
>>> align_to_ref.run()
```

Mandatory Inputs

- **input_image** (*a pathlike object or string representing a file*) – Input image to realign.
- **input_template** (*a pathlike object or string representing a file*) – Input reference image.

Outputs **output_transform** (*a pathlike object or string representing a file*) – Output 3D rigid tranformation file.

`ComputeAlignmentToReference.`**`m_best_transform = None`**

2.6.6 ListsMerger

Link to code[40]

Bases: nipype.interfaces.base.core.BaseInterface

Interface to merge list of paths or list of list of paths.

Examples

```
>>> from pymialsrtk.interfaces.preprocess import ListsMerger
>>> merge_lists = ListsMerger()
>>> merge_lists.inputs.inputs = ["sub-01_acq-haste_run-1_labels_1.
↪nii.gz", "sub-01_acq-haste_run-1_labels_2.nii.gz"]
>>> merge_lists.run()
```

Optional Inputs **inputs** (*a list of items which are any value*)

Outputs **outputs** (*a list of items which are any value*)

`ListsMerger.`**`m_list_of_files = None`**

[39] https://www.sciencedirect.com/science/article/pii/S1053811919309152
[40] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L2286-L2327

2.6.7 MialsrtkCorrectSliceIntensity

Link to code[41]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK mean slice intensity correction module.

Example

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪MialsrtkCorrectSliceIntensity
>>> sliceIntensityCorr = MialsrtkCorrectSliceIntensity()
>>> sliceIntensityCorr.inputs.in_file = 'sub-01_acq-haste_run-1_
↪T2w.nii.gz'
>>> sliceIntensityCorr.inputs.in_mask = 'sub-01_acq-haste_run-1_
↪mask.nii.gz'
>>> sliceIntensityCorr.run()
```

Mandatory Inputs **in_file** (*a pathlike object or string representing a file*) – Input image filename.

Optional Inputs

- **in_mask** (*a pathlike object or string representing a file*) – Input mask filename.
- **out_postfix** (*a string*) – Suffix to be added to input image file to construct corrected output filename. (Nipype **default** value: "")
- **verbose** (*a boolean*) – Enable verbosity.

Outputs **out_file** (*a pathlike object or string representing a file*) – Output image with corrected slice intensities.

2.6.8 MialsrtkHistogramNormalization

Link to code[42]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK histogram normalizaton module.

This module implements the method proposed by Nyul et al. **[1]**_.

References

Example

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪MialsrtkHistogramNormalization
>>> histNorm = MialsrtkHistogramNormalization()
>>> histNorm.inputs.input_images = ['sub-01_acq-haste_run-1_T2w.
↪nii.gz','sub-01_acq-haste_run-2_T2w.nii.gz']
```

(continues on next page)

[41] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L163-L207

[42] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L485-L544

(continued from previous page)

```
>>> histNorm.inputs.input_masks = ['sub-01_acq-haste_run-1_mask.
↪nii.gz','sub-01_acq-haste_run-2_mask.nii.gz']
>>> histNorm.run()
```

Optional Inputs

- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Input image filenames to be normalized.
- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Input mask filenames.
- **out_postfix** (*a string*) – Suffix to be added to normalized input image filenames to construct ouptut normalized image filenames. (Nipype **default** value: `_histnorm`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs **output_images** (*a list of items which are a pathlike object or string representing a file*) – Histogram-normalized images.

2.6.9 MialsrtkIntensityStandardization

Link to code[44]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK intensity standardization module.

This module rescales image intensity by linear transformation

Example

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪MialsrtkIntensityStandardization
>>> intensityStandardization= MialsrtkIntensityStandardization()
>>> intensityStandardization.inputs.input_images = ['sub-01_acq-
↪haste_run-1_T2w.nii.gz','sub-01_acq-haste_run-2_T2w.nii.gz']
>>> intensityStandardization.run()
```

Optional Inputs

- **in_max** (*a float*) – Maximal intensity.
- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Files to be corrected for intensity.
- **out_postfix** (*a string*) – Suffix to be added to intensity corrected input_images. (Nipype **default** value: `""`)
- **stacks_order** (*a list of items which are any value*) – Order of images index. To ensure images are processed with their correct corresponding mask.
- **verbose** (*a boolean*) – Enable verbosity.

Outputs **output_images** (*a list of items which are a pathlike object or string representing a file*) – Intensity-standardized images.

[44] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L406-L455

2.6.10 MialsrtkMaskImage

Link to code[45]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK mask image module.

Example

```
>>> from pymialsrtk.interfaces.preprocess import MialsrtkMaskImage
>>> maskImg = MialsrtkMaskImage()
>>> maskImg.inputs.in_file = 'sub-01_acq-haste_run-1_T2w.nii.gz'
>>> maskImg.inputs.in_mask = 'sub-01_acq-haste_run-1_mask.nii.gz'
>>> maskImg.inputs.out_im_postfix = '_masked'
>>> maskImg.run()
```

Mandatory Inputs

- **in_file** (*a pathlike object or string representing a file*) – Input image filename to be masked.
- **in_mask** (*a pathlike object or string representing a file*) – Input mask filename.

Optional Inputs

- **out_im_postfix** (*a string*) – Suffix to be added to masked in_file. (Nipype **default** value: "")
- **verbose** (*a boolean*) – Enable verbosity.

Outputs **out_im_file** (*a pathlike object or string representing a file*) – Masked image.

2.6.11 MialsrtkSliceBySliceCorrectBiasField

Link to code[46]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK independant slice by slice bias field correction module.

Example

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪MialsrtkSliceBySliceCorrectBiasField
>>> biasFieldCorr = MialsrtkSliceBySliceCorrectBiasField()
>>> biasFieldCorr.inputs.in_file = 'sub-01_acq-haste_run-1_T2w.
↪nii.gz'
>>> biasFieldCorr.inputs.in_mask = 'sub-01_acq-haste_run-1_mask.
↪nii.gz'
>>> biasFieldCorr.inputs.in_field = 'sub-01_acq-haste_run-1_field.
↪nii.gz'
>>> biasFieldCorr.run()
```

[45] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L568-L610

[46] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L329-L372

Mandatory Inputs

- **in_field** (*a pathlike object or string representing a file*) – Input bias field file.
- **in_file** (*a pathlike object or string representing a file*) – Input image file.
- **in_mask** (*a pathlike object or string representing a file*) – Input mask file.

Optional Inputs

- **out_im_postfix** (*a string*) – Suffix to be added to bias field corrected `in_file`. (Nipype **default** value: `_bcorr`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs **out_im_file** (*a pathlike object or string representing a file*) – Bias field corrected image.

2.6.12 MialsrtkSliceBySliceN4BiasFieldCorrection

Link to code[47]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK slice by slice N4 bias field correction module.

This module implements the method proposed by Tustison et al. **[1]**_.

References

Example

```
>>> from pymialsrtk.interfaces.preprocess import␣
↪MialsrtkSliceBySliceN4BiasFieldCorrection
>>> N4biasFieldCorr = MialsrtkSliceBySliceN4BiasFieldCorrection()
>>> N4biasFieldCorr.inputs.in_file = 'sub-01_acq-haste_run-1_T2w.
↪nii.gz'
>>> N4biasFieldCorr.inputs.in_mask = 'sub-01_acq-haste_run-1_mask.
↪nii.gz'
>>> N4biasFieldCorr.run()
```

Mandatory Inputs

- **in_file** (*a pathlike object or string representing a file*) – Input image.
- **in_mask** (*a pathlike object or string representing a file*) – Input mask.

Optional Inputs

- **out_fld_postfix** (*a string*) – Suffix to be added to input image filename to construct output bias field filename. (Nipype **default** value: `_n4bias`)
- **out_im_postfix** (*a string*) – Suffix to be added to input image filename to construct corrected output filename. (Nipype **default** value: `_bcorr`)

[47] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L245-L302

- **verbose** (*a boolean*) – Enable verbosity.

Outputs

- **out_fld_file** (*a pathlike object or string representing a file*) – Filename bias field extracted slice by slice from input image.
- **out_im_file** (*a pathlike object or string representing a file*) – Filename of corrected output image from N4 bias field (slice by slice).

2.6.13 ReduceFieldOfView

Link to code[49]

Bases: nipype.interfaces.base.core.BaseInterface

Interface to reduce the Field-of-View.

Examples

```
>>> from pymialsrtk.interfaces.preprocess import ReduceFieldOfView
>>> reduce_fov = ReduceFieldOfView()
>>> reduce_fov.inputs.input_image = 'sub-01_acq-haste_run-1_T2w.
↪nii.gz'
>>> reduce_fov.inputs.input_mask = 'sub-01_acq-haste_run-1_T2w_
↪mask.nii.gz'
>>> reduce_fov.run()
```

Mandatory Inputs

- **input_image** (*a pathlike object or string representing a file*) – Input image filename.
- **input_mask** (*a pathlike object or string representing a file*) – Input mask filename.

Optional Inputs **input_label** (*a pathlike object or string representing a file*) – Input label filename.

Outputs

- **output_image** (*a pathlike object or string representing a file*) – Cropped image.
- **output_label** (*a pathlike object or string representing a file*) – Cropped labels.
- **output_mask** (*a pathlike object or string representing a file*) – Cropped mask.

[49] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L2045-L2192

2.6.14 ResampleImage

Link to code[50]

Bases: nipype.interfaces.base.core.BaseInterface

Retrieve atlas of the same age and resample it to subject's in-plane resolution.

Examples

```
>>> from pymialsrtk.interfaces.preprocess import ResampleImage
>>> resample_image = ResampleImage()
>>> resample_image.inputs.input_image = "sub-01_acq-haste_run-1_
↪T2w.nii.gz"
>>> resample_image.inputs.input_reference = "STA30.nii.gz"
>>> resample_image.run()
```

Mandatory Inputs

- **input_image** (*a pathlike object or string representing a file*) – Input image to resample.
- **input_reference** (*a pathlike object or string representing a file*) – Input image with reference resolution.

Optional Inputs **verbose** (*a boolean*) – Enable verbosity.

Outputs **output_image** (*a pathlike object or string representing a file*) – Masked image.

2.6.15 SplitLabelMaps

Link to code[51]

Bases: nipype.interfaces.base.core.BaseInterface

Split a multi-label labelmap into one label map per label.

Examples

```
>>> from pymialsrtk.interfaces.preprocess import SplitLabelMaps
>>> split_labels = SplitLabelMaps()
>>> split_labels.inputs.in_labelmap = 'sub-01_acq-haste_run-1_
↪labels.nii.gz'
>>> split_labels.run()
```

Mandatory Inputs **in_labelmap** (*a pathlike object or string representing a file*) – Input label map.

Optional Inputs **all_labels** (*a list of items which are any value*)

Outputs

- **out_labelmaps** (*a list of items which are a pathlike object or string representing a file*) – Output masks.

[50] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L2345-L2419

[51] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L2208-L2272

- **out_labels** (*a list of items which are any value*) – List of labels ids that were extracted.

2.6.16 StacksOrdering

Link to code[52]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the automatic ordering of stacks.

This module is based on the tracking of the brain mask centroid slice by slice.

Examples

```
>>> from pymialsrtk.interfaces.preprocess import StacksOrdering
>>> stacksOrdering = StacksOrdering()
>>> stacksOrdering.inputs.input_masks = ['sub-01_run-1_mask.nii.gz
↪',
>>>                                      'sub-01_run-4_mask.nii.gz
↪',
>>>                                      'sub-01_run-2_mask.nii.gz
↪']
>>> stacksOrdering.run()
```

Note: In the case of discontinuous brain masks, the centroid coordinates of the slices excluded from the mask are set to `numpy.nan` and are not anymore considered in the motion index computation since `v2.0.2` release. Prior to this release, the centroids of these slices were set to zero that has shown to drastically increase the motion index with respect to the real motion during acquisition. However the motion in the remaining slices that were actually used for SR reconstruction might not correspond to the high value of this index.

Mandatory Inputs sub_ses (*a string*) – Subject and session BIDS identifier.

Optional Inputs

- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Input brain masks on which motion is computed.
- **verbose** (*a boolean*) – Enable verbosity.

Outputs

- **motion_tsv** (*a pathlike object or string representing a file*) – Output TSV file with results used to create `report_image`.
- **report_image** (*a pathlike object or string representing a file*) – Output PNG image for report.
- **stacks_order** (*a list of items which are any value*) – Order of image `run-id` to be used for reconstruction.

StacksOrdering.**m_stack_order = []**

[52] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/preprocess.py#L755-L1142

2.7 Postprocess module

PyMIALSRTK postprocessing functions.

It encompasses a High Resolution mask refinement and an N4 global bias field correction.

2.7.1 BinarizeImage

Link to code[53]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK mask image module.

Example

```
>>> from pymialsrtk.interfaces.postprocess import BinarizeImage
>>> maskImg = MialsrtkMaskImage()
>>> maskImg.inputs.input_image = 'input_image.nii.gz'
```

Mandatory Inputs input_image (*a pathlike object or string representing a file*) – Input image filename to be binarized.

Outputs output_srmask (*a pathlike object or string representing a file*) – Image mask (binarized input).

2.7.2 ConcatenateImageMetrics

Link to code[54]

Bases: nipype.interfaces.base.core.BaseInterface

Concatenate metrics CSV files with the metrics computed on each labelmap.

Example

```
>>> from pymialsrtk.interfaces.postprocess import␣
↪ConcatenateImageMetrics
>>> concat_metrics = ConcatenateImageMetrics()
>>> concat_metrics.inputs.input_metrics = ['sub-01_acq-haste_run-
↪1_paramset_1_metrics.csv', 'sub-01_acq-haste_run-1_paramset2_
↪metrics.csv']
concat_metrics.inputs.input_metrics_labels = ['sub-01_acq-haste_
↪run-1_paramset_1_metrics_labels.csv', 'sub-01_acq-haste_run-1_
↪paramset2_metrics_labels.csv']
>>> concat_metrics.run()
```

Optional Inputs

- **input_metrics** (*a list of items which are a pathlike object or string representing a file*)

[53] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L689-L729

[54] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L971-L1027

- **input_metrics_labels** (*a list of items which are a pathlike object or string representing a file*)

Outputs

- **output_csv** (*a pathlike object or string representing a file*)
- **output_csv_labels** (*a pathlike object or string representing a file*)

2.7.3 FilenamesGeneration

Link to code[55]

Bases: nipype.interfaces.base.core.BaseInterface

Generates final filenames from outputs of super-resolution reconstruction.

Example

```
>>> from pymialsrtk.interfaces.postprocess import␣
↪FilenamesGeneration
>>> filenamesGen = FilenamesGeneration()
>>> filenamesGen.inputs.sub_ses = 'sub-01'
>>> filenamesGen.inputs.stacks_order = [3,1,4]
>>> filenamesGen.inputs.sr_id = 3
>>> filenamesGen.inputs.use_manual_masks = False
>>> filenamesGen.run()
```

Mandatory Inputs

- **multi_parameters** (*a boolean*) – Whether multiple SR were reconstructed.
- **run_type** (*a string*) – Type of run (preproc or sr).
- **sr_id** (*an integer*) – Super-Resolution id.
- **stacks_order** (*a list of items which are any value*) – List of stack run-id that specify the order of the stacks.
- **sub_ses** (*a string*) – Subject and session BIDS identifier to construct output filename.
- **use_manual_masks** (*a boolean*) – Whether masks were computed or manually performed.

Optional Inputs **TV_params** (*a list of items which are any value*) – List iterables TV parameters processed.

Outputs **substitutions** (*a list of items which are any value*) – Output correspondance between old and new filenames.

`FilenamesGeneration.m_substitutions = []`

[55] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L306-L673

2.7.4 ImageMetrics

Link to code[56]

Bases: nipype.interfaces.base.core.BaseInterface

Compute various image metrics on a SR reconstructed image compared to a ground truth.

Example

```
>>> from pymialsrtk.interfaces.postprocess import ImageMetrics
>>> compute_metrics = ImageMetrics()
>>> compute_metrics.inputs.input_image = 'sub-01_acq-haste_rec-SR_
↪id-1_T2w.nii.gz'
>>> compute_metrics.inputs.input_ref_image = 'sub-01_acq-haste_
↪desc-GT_T2w.nii.gz'
>>> compute_metrics.inputs.input_ref_mask = 'sub-01_acq-haste_
↪desc-GT_T2w_mask.nii.gz'
>>> compute_metrics.inputs.input_TV_parameters = {'in_loop': '10',
↪ 'in_deltat': '0.01', 'in_lambda': '2.5', 'in_bregman_loop': '3
↪', 'in_iter': '50', 'in_step_scale': '1', 'in_gamma': '1', 'in_
↪inner_thresh': '1e-05', 'in_outer_thresh': '1e-06'}
>>> concat_metrics.run()
```

Mandatory Inputs

- **input_TV_parameters** (*a dictionary with keys which are any value and with values which are any value*)
- **input_image** (*a pathlike object or string representing a file*) – Input image filename.
- **input_ref_image** (*a pathlike object or string representing a file*) – Input reference image filename.
- **input_ref_mask** (*a pathlike object or string representing a file*) – Input reference mask filename.

Optional Inputs

- **input_ref_labelmap** (*a pathlike object or string representing a file*) – Input reference labelmap filename.
- **mask_input** (*a boolean*) – Whether the image and reference should be masked when computing the metrics. (Nipype **default** value: `True`)
- **normalize_input** (*a boolean*) – Whether the image and reference should be individually normalized before passing them into the metrics. (Nipype **default** value: `True`)

Outputs

- **output_metrics** (*a pathlike object or string representing a file*) – Output CSV.
- **output_metrics_labels** (*a pathlike object or string representing a file*) – Output per-label CSV.

`ImageMetrics.`**`norm_data`**`(`*`data`*`)`

Normalize data into 0-1 range

[56] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L766-L955

2.7.5 MergeMajorityVote

Link to code[57]

Bases: nipype.interfaces.base.core.BaseInterface

Perform majority voting to merge a list of label-wise labelmaps.

Example

```
>>> from pymialsrtk.interfaces.postprocess import␣
↪MergeMajorityVote
>>> merge_labels = MergeMajorityVote()
>>> merge_labels.inputs.input_images = ['sub-01_acq-haste_run-1_
↪labels_1.nii.gz', 'sub-01_acq-haste_run-1_labels_2.nii.gz',
↪'sub-01_acq-haste_run-1_labels_3.nii.gz']
>>> merge_labels.run()
```

Mandatory Inputs **input_images** (*a list of items which are a pathlike object or string representing a file*) – Inputs label-wise labelmaps to be merged.

Outputs **output_image** (*a pathlike object or string representing a file*) – Output label map.

2.7.6 MialsrtkN4BiasFieldCorrection

Link to code[58]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIAL SRTK slice by slice N4 bias field correction module.

This tools implements the method proposed by Tustison et al. **[1]_** slice by slice.

References

Example

```
>>> from pymialsrtk.interfaces.postprocess import␣
↪MialsrtkSliceBySliceN4BiasFieldCorrection
>>> N4biasFieldCorr = MialsrtkSliceBySliceN4BiasFieldCorrection()
>>> N4biasFieldCorr.inputs.input_image = 'sub-01_acq-haste_run-1_
↪SR.nii.gz'
>>> N4biasFieldCorr.inputs.input_mask = 'sub-01_acq-haste_run-1_
↪mask.nii.gz'
>>> N4biasFieldCorr.run()
```

Mandatory Inputs **input_image** (*a pathlike object or string representing a file*) – Input image filename to be normalized.

Optional Inputs

- **input_mask** (*a pathlike object or string representing a file*) – Input mask filename.

[57] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L1043-L1095

[58] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L205-L266

- **out_fld_postfix** (*a string*) – (Nipype **default** value: `_gbcorrfield`)
- **out_im_postfix** (*a string*) – (Nipype **default** value: `_gbcorr`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs

- **output_field** (*a pathlike object or string representing a file*) – Output bias field extracted from input image.
- **output_image** (*a pathlike object or string representing a file*) – Output corrected image.

2.7.7 MialsrtkRefineHRMaskByIntersection

Link to code[60]

Bases: nipype.interfaces.base.core.BaseInterface

Runs the MIALSRTK mask refinement module.

It uses the Simultaneous Truth And Performance Level Estimate (STAPLE) by Warfield et al. **[1]**_.

References

Example

```
>>> from pymialsrtk.interfaces.postprocess import␣
↪MialsrtkRefineHRMaskByIntersection
>>> refMask = MialsrtkRefineHRMaskByIntersection()
>>> refMask.inputs.input_images = ['sub-01_acq-haste_run-1_T2w.
↪nii.gz','sub-01_acq-haste_run-2_T2w.nii.gz']
>>> refMask.inputs.input_masks = ['sub-01_acq-haste_run-1_mask.
↪nii.gz','sub-01_acq-haste_run-2_mask.nii.gz']
>>> refMask.inputs.input_transforms = ['sub-01_acq-haste_run-1_
↪transform.txt','sub-01_acq-haste_run-2_transform.nii.gz']
>>> refMask.inputs.input_sr = 'sr_image.nii.gz'
>>> refMask.run()
```

Mandatory Inputs **input_sr** (*a pathlike object or string representing a file*) – SR image filename.

Optional Inputs

- **in_use_staple** (*a boolean*) – Use STAPLE for voting (default is True). If False, Majority voting is used instead. (Nipype **default** value: `True`)
- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Image filenames used in SR reconstruction.
- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Mask filenames.
- **input_rad_dilatation** (*an integer*) – Radius of the structuring element (ball). (Nipype **default** value: 1)

[60] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L94-L178

- **input_transforms** (*a list of items which are a pathlike object or string representing a file*) – Transformation filenames.
- **out_lrmask_postfix** (*a string*) – Suffix to be added to the Low resolution input_masks. (Nipype **default** value: `_LRmask`)
- **out_srmask_postfix** (*a string*) – Suffix to be added to the SR reconstruction filename to construct output SR mask filename. (Nipype **default** value: `_srMask`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs

- **output_lrmasks** (*a list of items which are a pathlike object or string representing a file*) – Output low-resolution reconstruction refined masks.
- **output_srmask** (*a pathlike object or string representing a file*) – Output super-resolution reconstruction refined mask.

2.7.8 ReportGeneration

Link to code[62]

Bases: nipype.interfaces.base.core.BaseInterface

Generate a report summarizing the outputs of mialsrtk.

This comprises:

- Details of which parameters were run.
- A visualization of the SR-reconstructed image
- A visualization of the computational graph.

Example

```
>>> from pymialsrtk.interfaces.postprocess import ReportGeneration
>>> report_gen = ReportGeneration()
>>> report_gen.inputs.subject = "sub-01"
>>> report_gen.inputs.session = "ses-01"
>>> report_gen.inputs.stacks = [2,3,5,1]
>>> report_gen.inputs.stacks_order = [5,2,3,1]
>>> report_gen.inputs.sr_id = "1"
>>> report_gen.inputs.run_type = "sr"
>>> report_gen.inputs.output_dir = "report_dir/
>>> report_gen.inputs.run_start_time = ""
>>> report_gen.inputs.run_elapsed_time = ""
>>> report_gen.inputs.do_refine_hr_mask = True
>>> report_gen.inputs.do_nlm_denoising = True
>>> report_gen.inputs.skip_stacks_ordering = False
>>> report_gen.inputs.skip_svr = False
>>> report_gen.inputs.masks_derivatives_dir = "derivatives/masks"
>>> report_gen.inputs.do_reconstruct_labels = False
>>> report_gen.inputs.do_anat_orientation = True
>>> report_gen.inputs.do_srr_assessment = False
>>> report_gen.inputs.openmp_number_of_cores = 3
```

(continues on next page)

[62] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/postprocess.py#L1178-L1349

(continued from previous page)

```
>>> report_gen.inputs.nipype_number_of_cores = 1
>>> report_gen.inputs.input_sr = "outputs/sub-01_ses-01_res-SR_
↪T2w.nii.gz"
>>> report_gen.inputs.input_json_path = "outputs/sub-01_ses-01_
↪res-SR_T2w.json"
>>> report_gen.run()
```

Mandatory Inputs

- **do_anat_orientation** (*a boolean*)
- **do_multi_parameters** (*a boolean*)
- **do_nlm_denoising** (*a boolean*)
- **do_reconstruct_labels** (*a boolean*)
- **do_refine_hr_mask** (*a boolean*)
- **do_srr_assessment** (*a boolean*)
- **input_json_path** (*a pathlike object or string representing a file*)
- **input_sr** (*a pathlike object or string representing a file*)
- **masks_derivatives_dir** (*a string*)
- **nipype_number_of_cores** (*an integer*)
- **openmp_number_of_cores** (*an integer*)
- **output_dir** (*a string*)
- **run_elapsed_time** (*a float*)
- **run_start_time** (*a float*)
- **run_type** (*a string*) – Type of run (preproc or sr).
- **session** (*a string*) – Session BIDS identifier to construct output filename.
- **skip_stacks_ordering** (*a boolean*)
- **skip_svr** (*a boolean*)
- **sr_id** (*an integer*) – Super-Resolution id.
- **stacks** (*a list of items which are any value*) – List of stack run-id that specify the order of the stacks.
- **stacks_order** (*a list of items which are any value*) – List of stack run-id that specify the order of the stacks.
- **subject** (*a string*) – Subject BIDS identifier to construct output filename.

Optional Inputs **substitutions** (*a list of items which are any value*) – Output correspondance between old and new filenames.

Outputs **report_html** (*a pathlike object or string representing a file*)

2.8 Reconstruction module

PyMIALSRTK reconstruction functions.

2.8.1 MialsrtkImageReconstruction

Link to code[63]

Bases: nipype.interfaces.base.core.BaseInterface

Creates a high-resolution image from a set of low resolution images.

It is built on the BTK implementation and implements the method, presented in [Rousseau2006], that iterates between slice-to-volume registration and reconstruction of a high-resolution image using scattered data interpolation. The method converges when the mean square error between the high-resolution images reconstructed in the last two iterations is lower than `1e-6`.

It is used to estimate slice motion prior to `MialsrtkTVSuperResolution`.

References

Example

```
>>> from pymialsrtk.interfaces.reconstruction import␣
↪MialsrtkImageReconstruction
>>> srtkImageReconstruction = MialsrtkImageReconstruction()
>>> srtkImageReconstruction.input_images = ['sub-01_ses-01_run-1_
↪T2w.nii.gz',
>>>                                          'sub-01_ses-01_run-2_
↪T2w.nii.gz',
>>>                                          'sub-01_ses-01_run-3_
↪T2w.nii.gz',
>>>                                          'sub-01_ses-01_run-4_
↪T2w.nii.gz']
>>> srtkImageReconstruction.input_masks = ['sub-01_ses-01_run-1_
↪mask.nii.gz',
>>>                                         'sub-01_ses-01_run-2_
↪mask.nii.gz',
>>>                                         'sub-01_ses-01_run-3_
↪mask.nii.gz',
>>>                                         'sub-01_ses-01_run-4_
↪mask.nii.gz']
>>> srtkImageReconstruction.inputs.stacks_order = [3,1,2,4]
>>> srtkImageReconstruction.inputs.sub_ses = 'sub-01_ses-01'
>>> srtkImageReconstruction.inputs.in_roi = 'mask'
>>> srtkImageReconstruction.run()
```

Mandatory Inputs

- **in_roi** (*'mask' or 'all' or 'box' or 'mask'*) –

  Define region of interest (required):

  - `box`: Use intersections for roi calculation
  - `mask`: Use masks for roi calculation

[63] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/reconstruction.py#L107-L230

- `all`: Use the whole image FOV.

(Nipype **default** value: `mask`)

- **stacks_order** (*a list of items which are any value*) – List of stack run-id that specify the order of the stacks.

Optional Inputs

- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Input images.
- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Masks of the input images.
- **input_rad_dilatation** (*a float*) – Radius dilatation used in prior step to construct output filename. (Nipype **default** value: `1.0`)
- **out_prefix** (*a string*) – Prefix added to construct output super-resolution filename. (Nipype **default** value: `SRTV_`)
- **out_sdi_prefix** (*a string*) – Suffix added to construct outputscattered data interpolation filename. (Nipype **default** value: `SDI_`)
- **out_transf_postfix** (*a string*) – Suffix added to construct output transformation filenames. (Nipype **default** value: `_transform`)
- **skip_svr** (*a boolean*) – Skip slice-to-volume registration.
- **sub_ses** (*a string*) – Subject and session BIDS identifier to construct output filename. (Nipype **default** value: `x`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs

- **output_sdi** (*a pathlike object or string representing a file*) – Output scattered data interpolation image file.
- **output_transforms** (*a list of items which are a pathlike object or string representing a file*) – Output transformation files.

2.8.2 MialsrtkSDIComputation

Link to code[65]

Bases: nipype.interfaces.base.core.BaseInterface

Creates a high-resolution image from a set of low resolution images and their slice-by-slicemotion parameters.

Mandatory Inputs

- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Input images.
- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Masks of the input images.
- **input_reference** (*a pathlike object or string representing a file*) – Input reference image.
- **input_transforms** (*a list of items which are a pathlike object or string representing a file*) – Input transformation files.

[65] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/reconstruction.py#L651-L755

- **stacks_order** (*a list of items which are any value*) – List of stack run-id that specify the order of the stacks.

Optional Inputs

- **label_id** (*an integer*) – (Nipype **default** value: `-1`)
- **sub_ses** (*a string*) – Subject and session BIDS identifier to construct output filename. (Nipype **default** value: `x`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs **output_sdi** (*a pathlike object or string representing a file*) – Output scattered data interpolation image file.

`MialsrtkSDIComputation.`**`get_empty_ref_image`**(*input_ref*)
Generate an empty reference image for `_run_interface`.

2.8.3 MialsrtkTVSuperResolution

Link to code[66]

Bases: nipype.interfaces.base.core.BaseInterface

Apply super-resolution algorithm using one or multiple input images.

It implements the super-resolution algorithm with Total-Variation regularization presented in [Tourbier2015].

Taking as input the list of input low resolution images and the list of slice-by-slice transforms estimated with `MialsrtkImageReconstruction`, it builds the forward model `H` that relates the low resolution images `y` to the high resolution `x` to be estimated by `y = Hx`, and it solves optimally using convex optimization the inverse (i.e. super-resolution) problem (finding `x`) with exact Total-Variation regularization.

The weight of TV regularization is controlled by `in_lambda`. The lower it is, the lower will be the weight of the data fidelity and so the higher will the regularization. The optimization time step is controlled by `in_deltat` that defines the time step of the optimizer.

References

Example

```
>>> from pymialsrtk.interfaces.reconstruction import␣
↪MialsrtkTVSuperResolution
>>> srtkTVSuperResolution = MialsrtkTVSuperResolution()
>>> srtkTVSuperResolution.input_images = ['sub-01_ses-01_run-1_
↪T2w.nii.gz', 'sub-01_ses-01_run-2_T2w.nii.gz',     'sub-01_ses-
↪01_run-3_T2w.nii.gz', 'sub-01_ses-01_run-4_T2w.nii.gz']
>>> srtkTVSuperResolution.input_masks = ['sub-01_ses-01_run-1_
↪mask.nii.gz', 'sub-01_ses-01_run-2_mask.nii.gz',     'sub-01_
↪ses-01_run-3_mask.nii.gz', 'sub-01_ses-01_run-4_mask.nii.gz']
>>> srtkTVSuperResolution.input_transforms = ['sub-01_ses-01_run-
↪1_transform.txt', 'sub-01_ses-01_run-2_transform.txt',     'sub-
↪01_ses-01_run-3_transform.txt', 'sub-01_ses-01_run-4_transform.
↪txt']
>>> srtkTVSuperResolution.input_sdi = 'sdi.nii.gz'
```

(continues on next page)

[66] http://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/blob/cce7cfc/pymialsrtk/interfaces/reconstruction.py#L325-L611

(continued from previous page)

```
>>> srtkTVSuperResolution.inputs.stacks_order = [3,1,2,4]
>>> srtkTVSuperResolution.inputs.sub_ses = 'sub-01_ses-01'
>>> srtkTVSuperResolution.inputs.in_loop = 10
>>> srtkTVSuperResolution.inputs.in_deltat = 0.01
>>> srtkTVSuperResolution.inputs.in_lambda = 0.75
>>> srtkTVSuperResolution.run()
```

Mandatory Inputs

- **in_deltat** (*a float*) – Parameter deltat of TV optimizer.
- **in_lambda** (*a float*) – TV regularization factor which weights the data fidelity term in TV optimizer.
- **in_loop** (*an integer*) – Number of loops (SR/denoising).
- **input_sdi** (*a pathlike object or string representing a file*) – Reconstructed image for initialization. Typically the output of MialsrtkImageReconstruction is used.

Optional Inputs

- **deblurring** (*a boolean*) – Flag to set deblurring PSF during SR (double the neighborhood). (Nipype **default** value: `False`)
- **in_bregman_loop** (*an integer*) – Number of Bregman loops. (Nipype **default** value: `3`)
- **in_gamma** (*an integer*) – Parameter gamma. (Nipype **default** value: `10`)
- **in_inner_thresh** (*a float*) – Inner loop convergence threshold. (Nipype **default** value: `1e-05`)
- **in_iter** (*an integer*) – Number of inner iterations. (Nipype **default** value: `50`)
- **in_outer_thresh** (*a float*) – Outer loop convergence threshold. (Nipype **default** value: `1e-06`)
- **in_step_scale** (*an integer*) – Parameter step scale. (Nipype **default** value: `10`)
- **input_images** (*a list of items which are a pathlike object or string representing a file*) – Input image filenames for super-resolution.
- **input_masks** (*a list of items which are a pathlike object or string representing a file*) – Masks of input images for super-resolution.
- **input_rad_dilatation** (*a float*) – Radius dilatation used in prior step to construct output filename. (Nipype **default** value: `1.0`)
- **input_transforms** (*a list of items which are a pathlike object or string representing a file*) – Estimated slice-by-slice ITK transforms of input images.
- **out_prefix** (*a string*) – Prefix added to construct output super-resolution filename. (Nipype **default** value: `SRTV_`)
- **stacks_order** (*a list of items which are any value*) – List of stack run-id that specify the order of the stacks.
- **sub_ses** (*a string*) – Subject and session BIDS identifier to construct output filename. (Nipype **default** value: `x`)

- **use_manual_masks** (*a boolean*) – Use masks of input files. (Nipype **default** value: `False`)
- **verbose** (*a boolean*) – Enable verbosity.

Outputs

- **output_TV_parameters** (*a dictionary with keys which are any value and with values which are any value*) – Dictionary of all TV parameters used.
- **output_json_path** (*a pathlike object or string representing a file*) – Output json file where super-resolution reconstruction parameters are summarized.
- **output_sr** (*a pathlike object or string representing a file*) – Output super-resolution image file.
- **output_sr_png** (*a pathlike object or string representing a file*) – Output super-resolution PNG image file for quality assessment.

`MialsrtkTVSuperResolution.`**`m_TV_parameters = {}`**

`MialsrtkTVSuperResolution.`**`m_out_files = ''`**

`MialsrtkTVSuperResolution.`**`m_output_dict = {}`**

2.9 Utility (utils) module

PyMIALSRTK utils functions.

`pymialsrtk.interfaces.utils.`**`get_emission_car_miles_equivalent`**(*emissions*)

Return the equivalent of CO2 emissions [Kg] in terms of kms traveled by an average car.

References

https://github.com/mlco2/codecarbon/blob/c6aebb9681186a71573748e381b6a3c9731de2d3/codecarbon/viz/data.py#L53

`pymialsrtk.interfaces.utils.`**`get_emission_tv_time_equivalent`**(*emissions*)

Return the equivalent of CO2 emissions [Kg] in terms of kms traveled by an average car.

References

https://github.com/mlco2/codecarbon/blob/c6aebb9681186a71573748e381b6a3c9731de2d3/codecarbon/viz/data.py#L66

`pymialsrtk.interfaces.utils.`**`reorder_by_run_ids`**(*p_files*, *p_order*)

Function used to reorder images by their run-id IDS tag.

The images are sorted according to the input parameters. If more images are available, they remaining are sorted in scending order.

Parameters

- **p_files** (*list of string*) – List of image paths - containing a 'run-' id tag, to be reordered
- **p_order** (*list of int*) – List of expecting run id order.

Examples

```
>>> in_files = ['sub-01_run-2_T2w.nii.gz', 'sub-01_run-5_T2w.nii.gz',
↪'sub-01_run-3_T2w.nii.gz', 'sub-01_run-1_T2w.nii.gz']
>>> my_order = [1,5,3]
>>> reorder_by_run_ids(in_files, my_order)
```

`pymialsrtk.interfaces.utils.run(command, env=None, cwd=None)`

Function calls by each MIALSRTK interface.

It runs the command specified as input via `subprocess.run()`.

Parameters

- **command** (*string*) – String containing the command to be executed (required)
- **env** (*os.environ*) – Specify a custom os.environ
- **cwd** (*Directory*) – Specify a custom current working directory

Examples

```
>>> cmd = 'btkNLMDenoising -i "/path/to/in_file" -o "/path/to/out_file
↪" -b 0.1'
>>> run(cmd)
```

`pymialsrtk.interfaces.utils.sort_ascending(p_files)`

Function used to sort images at the input of a nipype node.

Parameters **p_files** (*list*) – List of image paths to be sorted in ascending order

Examples

```
>>> in_files = ['sub-01_run-2_T2w.nii.gz', 'sub-01_run-5_T2w.nii.gz',
↪'sub-01_run-3_T2w.nii.gz', 'sub-01_run-1_T2w.nii.gz']
>>> sort_ascending(in_files)
```

2.10 Pipelines module

Abstract base class for the anatomical pipeline.

class pymialsrtk.pipelines.anatomical.abstract.**AbstractAnatomicalPipeline**(*p_bids_dir, p_output_dir, p_subject, p_ga=None, p_stacks=None, p_sr_id=1, p_session=None, p_masks_derivatives, p_masks_desc=, p_dict_custom_i, p_verbose=None, p_openmp_num, p_nipype_numb, p_run_type=No*)

Bases: `object`

Class used to represent the workflow of the anatomical pipeline.

Attributes

- **m_bids_dir** (*string*) – BIDS root directory (required)
- **m_output_dir** (*string*) – Output derivatives directory (required)
- **m_subject** (*string*) – Subject ID (in the form `sub-XX`)
- **m_wf** (*nipype.pipeline.Workflow*) – Nipype workflow of the reconstruction pipeline
- **m_sr_id** (*string*) – ID of the reconstruction useful to distinguish when multiple reconstructions with different order of stacks are run on the same subject
- **m_session** (*string*) – Session ID if applicable (in the form `ses-YY`)
- **m_stacks** (*list(int)*) – List of stack to be used in the reconstruction. The specified order is kept if `skip_stacks_ordering` is True.
- **m_masks_derivatives_dir** (*string*) – directory basename in BIDS directory derivatives where to search for masks (optional)
- **m_do_nlm_denoising** (*bool*) – Whether the NLM denoising preprocessing should be performed prior to motion estimation. (default is False)
- **m_skip_stacks_ordering** (*bool (optional)*) – Whether the automatic stacks ordering should be skipped. (default is False)

Examples

```
>>> from pymialsrtk.pipelines.anatomical.srr import AnatomicalPipeline
>>> # Create a new instance
>>> pipeline = AnatomicalPipeline(bids_dir='/path/to/bids_dir',
                                  output_dir='/path/to/output_dir',
                                  subject='sub-01',
                                  p_stacks=[1,3,2,0],
                                  sr_id=1,
                                  session=None,
                                  paramTV={deltat_TV = "0.001",
                                           lambda_TV = "0.75",
                                           num_primal_dual_loops = "20
↪"},
```

(continues on next page)

(continued from previous page)

```
                                    masks_derivatives_dir="/custom/mask_
↪dir",
                                    masks_desc=None,
                                    p_dict_custom_interfaces=None)
>>> # Create the super resolution Nipype workflow
>>> pipeline.create_workflow()
>>> # Execute the workflow
>>> res = pipeline.run(number_of_cores=1)
```

abstract create_workflow()

Create the Niype workflow of the super-resolution pipeline.

It is composed of a succession of Nodes and their corresponding parameters, where the output of node i goes to the input of node i+1.

The more specific definition given in each node implementing the method.

run(*memory=None, logger=None*)

Execute the workflow of the super-resolution reconstruction pipeline.

Nipype execution engine will take care of the management and execution of all processing steps involved in the super-resolution reconstruction pipeline. Note that the complete execution graph is saved as a PNG image to support transparency on the whole processing.

Parameters **memory** (*int*) – Maximal memory used by the workflow

Module for the preprocessing pipeline.

class pymialsrtk.pipelines.anatomical.preprocessing.**PreprocessingPipeline**(*p_bids_dir, p_output_dir, p_subject, p_ga=None, p_stacks=None, p_sr_id=1, p_session=None, p_masks_deriva, p_masks_desc=, p_dict_custom_i, p_verbose=Non, p_openmp_num, p_nipype_numb*

Bases: *pymialsrtk.pipelines.anatomical.abstract.AbstractAnatomicalPipeline* (page 48)

Class used to represent the workflow of the Preprocessing pipeline.

Attributes

- **m_bids_dir** (*string*) – BIDS root directory (required)
- **m_output_dir** (*string*) – Output derivatives directory (required)
- **m_subject** (*string*) – Subject ID (in the form `sub-XX`)
- **m_wf** (*nipype.pipeline.Workflow*) – Nipype workflow of the preprocessing pipeline
- **m_sr_id** (*string*) – ID of the preprocessing useful to distinguish when multiple preprocessing with different order of stacks are run on the same subject
- **m_session** (*string*) – Session ID if applicable (in the form `ses-YY`)

- **m_stacks** (*list(int)*) – List of stack to be used in the preprocessing. The specified order is kept if `skip_stacks_ordering` is True.
- **m_masks_derivatives_dir** (*string*) – directory basename in BIDS directory derivatives where to search for masks (optional)
- **m_do_nlm_denoising** (*bool*) – Whether the NLM denoising preprocessing should be performed prior to motion estimation. (default is False)
- **m_skip_stacks_ordering** (*bool (optional)*) – Whether the automatic stacks ordering should be skipped. (default is False)

Examples

```
>>> from pymialsrtk.pipelines.anatomical.srr import␣
↪PreprocessingPipeline
>>> # Create a new instance
>>> pipeline = PreprocessingPipeline(bids_dir='/path/to/bids_dir',
                                     output_dir='/path/to/output_dir',
                                     subject='sub-01',
                                     p_stacks=[1,3,2,0],
                                     sr_id=1,
                                     session=None,
                                     paramTV={deltatTV = "0.001",
                                              lambdaTV = "0.75",
                                              num_primal_dual_loops = "20
↪"},
                                     masks_derivatives_dir="/custom/mask_
↪dir",
                                     masks_desc=None,
                                     p_dict_custom_interfaces=None)
>>> # Create the super resolution Nipype workflow
>>> pipeline.create_workflow()
>>> # Execute the workflow
>>> res = pipeline.run(number_of_cores=1)
```

check_parameters_integrity(*p_dict_custom_interfaces*)

Check parameters integrity.

This checks whether the custom interfaces dictionary contains only keys that are used in preprocessing, and raises an exception if it doesn't.

Parameters **p_dict_custom_interfaces** (*dict*) – dictionary of custom inferfaces for a given subject that is to be processed.

create_workflow()

Create the Niype workflow of the super-resolution pipeline.

It is composed of a succession of Nodes and their corresponding parameters, where the output of node i goes to the input of node i+1.

Module for the super-resolution reconstruction pipeline.

class pymialsrtk.pipelines.anatomical.srr.**SRReconPipeline**(*p_bids_dir, p_output_dir, p_subject, p_ga=None, p_stacks=None, p_sr_id=1, p_session=None, p_paramTV=None, p_masks_derivatives_dir=None, p_labels_derivatives_dir=None, p_masks_desc=None, p_dict_custom_interfaces=None, p_verbose=None, p_openmp_number_of_cores=None, p_nipype_number_of_cores=None, p_all_outputs=None*)

Bases: *pymialsrtk.pipelines.anatomical.abstract.AbstractAnatomicalPipeline* (page 48)

Class used to represent the workflow of the Super-Resolution reconstruction pipeline.

Attributes

- **m_bids_dir** (*string*) – BIDS root directory (required)
- **m_output_dir** (*string*) – Output derivatives directory (required)
- **m_subject** (*string*) – Subject ID (in the form `sub-XX`)
- **m_wf** (*nipype.pipeline.Workflow*) – Nipype workflow of the reconstruction pipeline
- **m_paramTV** (`dict`) – Dictionary of parameters for the super-resolution reconstruction. Contains: - deltatTV : string

  Super-resolution optimization time-step

  - **lambdaTV** [float] Regularization weight (default is 0.75)
  - **num_iterations** [string] Number of iterations in the primal/dual loops used in the optimization of the total-variation super-resolution algorithm.
  - **num_primal_dual_loops** [string] Number of primal/dual (inner) loops used in the optimization of the total-variation super-resolution algorithm.
  - **num_bregman_loops** [string] Number of Bregman (outer) loops used in the optimization of the total-variation super-resolution algorithm.
  - **step_scale** [string] Step scale parameter used in the optimization of the total- variation super-resolution algorithm.
  - **gamma** [string] Gamma parameter used in the optimization of the total-variation super-resolution algorithm.
- **m_sr_id** (*string*) – ID of the reconstruction useful to distinguish when multiple reconstructions with different order of stacks are run on the same subject
- **m_keep_all_outputs** (*bool*) – Whether intermediate outputs must be issued. (default: False)
- **m_session** (*string*) – Session ID if applicable (in the form `ses-YY`)

- **m_stacks** (*list(int)*) – List of stack to be used in the reconstruction. The specified order is kept if `skip_stacks_ordering` is True.
- **m_masks_derivatives_dir** (*string*) – directory basename in BIDS directory derivatives where to search for masks (optional)
- **m_skip_svr** (*bool*) – Whether the Slice-to-Volume Registration should be skipped in the image reconstruction. (default is False)
- **m_do_refine_hr_mask** (*bool*) – Whether a refinement of the HR mask should be performed. (default is False)
- **m_do_nlm_denoising** (*bool*) – Whether the NLM denoising preprocessing should be performed prior to motion estimation. (default is False)
- **m_skip_stacks_ordering** (*bool (optional)*) – Whether the automatic stacks ordering should be skipped. (default is False)

Examples

```
>>> from pymialsrtk.pipelines.anatomical.srr import SRReconPipeline
>>> # Create a new instance
>>> pipeline = SRReconPipeline(bids_dir='/path/to/bids_dir',
                                 output_dir='/path/to/output_dir',
                                 subject='sub-01',
                                 p_stacks=[1,3,2,0],
                                 sr_id=1,
                                 session=None,
                                 paramTV={deltatTV = "0.001",
                                          lambdaTV = "0.75",
                                          num_primal_dual_loops = "20
↪"},
                                 masks_derivatives_dir="/custom/mask_
↪dir",
                                 masks_desc=None,
                                 p_dict_custom_interfaces=None)
>>> # Create the super resolution Nipype workflow
>>> pipeline.create_workflow()
>>> # Execute the workflow
>>> res = pipeline.run(number_of_cores=1)
```

create_workflow()

Create the Niype workflow of the super-resolution pipeline.

It is composed of a succession of Nodes and their corresponding parameters, where the output of node i goes to the input of node i+1.

run(*memory=None, logger=None*)

Execute the workflow of the super-resolution reconstruction pipeline.

Nipype execution engine will take care of the management and execution of all processing steps involved in the super-resolution reconstruction pipeline. Note that the complete execution graph is saved as a PNG image to support transparency on the whole processing.

Parameters **memory** (*int*) – Maximal memory used by the workflow

2.11 Workflows module

Workflow for the management of super-resolution reconstruction pipeline inputs.

`pymialsrtk.workflows.input_stage.`**`create_input_stage`**(*p_bids_dir, p_sub_ses, p_sub_path, p_use_manual_masks, p_masks_desc, p_masks_derivatives_dir, p_labels_derivatives_dir, p_skip_stacks_ordering, p_do_reconstruct_labels, p_stacks, p_do_srr_assessment, p_verbose, name='input_stage'*)

Create a input management workflow for srr pipeline.

Parameters

- **name** (*string*) – name of workflow (default: input_stage)
- **p_bids_dir** (*string*) – Path to the bids directory
- **p_sub_ses** (*string*) – String containing subject-session information.
- **p_use_manual_masks** (*boolean*) – Whether manual masks are used
- **p_masks_desc** (*string*) – BIDS description tag of masks to use (optional)
- **p_masks_derivatives_dir** (*string*) – Path to the directory of the manual masks.
- **p_skip_stacks_ordering** (*boolean*) – Whether stacks ordering should be skipped. If true, uses the order provided in `p_stacks`.
- **p_stacks** (*list of integer*) – List of stack to be used in the reconstruction. The specified order is kept if `skip_stacks_ordering` is True.
- **p_do_srr_assessment** (`bool`) – If super-resolution assessment should be done.

Outputs

- **outputnode.t2ws_filtered** (*list of filenames*) – Low-resolution T2w images
- **outputnode.masks_filtered** (*list of filenames*) – Low-resolution T2w masks
- **outputnode.stacks_order** (*list of ids*) – Order in which the stacks should be processed
- **outputnode.report_image** (*filename*) – Output PNG image for report
- **outputnode.motion_tsv** (*filename*) – Output TSV file with results used to create `report_image`
- **outputnode.ground_truth** (*filename*) – Ground truth image used for `srr_assessment` (optional, if p_do_srr_assessment=True)

Example

```
>>> from pymialsrtk.pipelines.workflows import input_stage
>>> input_mgmt_stage = input_stage.create_input_stage(
        p_bids_dir="bids_data",
        p_sub_ses="sub-01_ses-1",
        p_sub_path="sub-01/ses-1/anat",
        p_use_manual_masks=False,
        p_skip_stacks_ordering=False,
        p_do_srr_assessment=False,
        name="input_mgmt_stage",
    )
>>> input_mgmt_stage.run()
```

Module for the preprocessing stage of the super-resolution reconstruction pipeline.

`pymialsrtk.workflows.preproc_stage.`**`create_preproc_stage`**(*p_skip_preprocessing=False, p_do_nlm_denoising=False, p_do_reconstruct_labels=False, p_verbose=False, name='preproc_stage'*)

Create a SR preprocessing workflow.

Parameters

- **p_do_nlm_denoising** (*boolean*) – Whether to proceed to non-local mean denoising (default: `False`)
- **p_do_reconstruct_labels** (*boolean*) – Whether we are also reconstruction label maps. (default: `False`)
- **p_verbose** (*boolean*) – Whether verbosity should be enabled (default False)
- **name** (*string*) – name of workflow (default: "preproc_stage")

Inputs

- **input_images** (*list of items which are a pathlike object or string representing a file*) – Input T2w images
- **input_masks** (*list of items which are a pathlike object or string representing a file*) – Input mask images

Outputs

- **output_images** (*list of items which are a pathlike object or string representing a file*) – Processed images
- **output_masks** (*list of items which are a pathlike object or string representing a file*) – Processed images
- **output_images_nlm** (*list of items which are a pathlike object or string representing a file*) – Processed images with NLM denoising, required if `p_do_nlm_denoising = True`

Example

```
>>> from pymialsrtk.pipelines.workflows import preproc_stage as preproc
>>> preproc_stage = preproc.create_preproc_stage(p_do_nlm_
↪denoising=False)
>>> preproc_stage.inputs.inputnode.input_images =
        ['sub-01_run-1_T2w.nii.gz',
         'sub-01_run-2_T2w.nii.gz']
>>> preproc_stage.inputs.inputnode.input_masks =
        ['sub-01_run-1_T2w_mask.nii.gz',
         'sub-01_run-2_T2w_mask.nii.gz']
>>> preproc_stage.inputs.inputnode.p_do_nlm_denoising = 'mask.nii'
>>> preproc_stage.run()
```

Module for the registration stage of the super-resolution reconstruction pipeline.

`pymialsrtk.workflows.registration_stage.`**`create_registration_stage`**(*p_do_nlm_denoising=Fals*
p_skip_svr=False,
p_sub_ses='',
p_verbose=False,
name='registration_stage

Create a a registration workflow, used as an optional stage in the preprocessing only pipeline.

Parameters

- **p_do_nlm_denoising** (*boolean*) – Enable non-local means denoising (default: `False`)
- **p_skip_svr** (*boolean*) – Skip slice-to-volume registration (default: `False`)
- **p_sub_ses** (*string*) – String containing subject-session information.
- **name** (*string*) – name of workflow (default: "registration_stage")

Inputs

- **input_images** (*list of items which are a pathlike object or string representing a file*) – Input low-resolution T2w images
- **input_images_nlm** (*list of items which are a pathlike object or string representing a file*) – Input low-resolution denoised T2w images, Optional - only if `p_do_nlm_denoising = True`
- **input_masks** (*list of items which are a pathlike object or string representing a file*) – Input mask images from the low-resolution T2w images
- **stacks_order** (*list of integer*) – Order of stacks in the registration

Outputs

- **output_sdi** (*pathlike object or string representing a file*) – SDI image
- **output_tranforms** (*list of items which are a pathlike object or string representing a file*) – Estimated transformation parameters

Example

```
>>> from pymialsrtk.pipelines.workflows import registration_stage as␣
↪reg
>>> registration_stage = reg.create_registration_stage(
        p_sub_ses=p_sub_ses,
    )
>>> registration_stage.inputs.input_images = [
        'sub-01_run-1_T2w.nii.gz',
        'sub-01_run-2_T2w.nii.gz'
    ]
>>> registration_stage.inputs.input_masks = [
        'sub-01_run-1_T2w.nii_mask.gz',
        'sub-01_run-2_T2w.nii_mask.gz'
    ]
>>> registration_stage.inputs.stacks_order = [2,1]
>>> registration_stage.run()
```

Module for the reconstruction stage of the super-resolution reconstruction pipeline.

`pymialsrtk.workflows.recon_stage.`**`create_recon_stage`**(*p_paramTV, p_use_manual_masks, p_do_multi_parameters=False, p_do_nlm_denoising=False, p_do_reconstruct_labels=False, p_do_refine_hr_mask=False, p_skip_svr=False, p_sub_ses='', p_verbose=False, name='recon_stage'*)

Create a super-resolution reconstruction workflow.

Parameters

- **p_paramTV** (*dictionary*) – Dictionary of TV parameters
- **p_use_manual_masks** (*boolean*) – Whether masks were done manually.
- **p_do_nlm_denoising** (*boolean*) – Whether to proceed to non-local mean denoising. (default: `False`)
- **p_do_multi_parameters** (*boolean*) – Perform super-resolution reconstruction with a set of multiple parameters. (default: `False`)
- **p_do_reconstruct_labels** (*boolean*) – Whether we are also reconstruction label maps. (default: `False`)
- **p_do_refine_hr_mask** (*boolean*) – Whether to do high-resolution mask refinement. (default: `False`)
- **p_skip_svr** (*boolean*) – Whether slice-to-volume registration (SVR) should be skipped. (default: `False`)
- **p_sub_ses** (*string*) – String describing subject-session information (default: ‘’)
- **p_verbose** (*boolean*) – Whether verbosity should be enabled (default: `False`)
- **name** (*string*) – Name of workflow (default: “recon_stage”)

Inputs

- **input_images** (*list of items which are a pathlike object or string representing a file*) – Input T2w images
- **input_images_nlm** (*list of items which are a pathlike object or string representing a file*) – Input T2w images, required if `p_do_nlm_denoising=True`
- **input_masks** (*list of items which are a pathlike object or string representing a file*) – Input mask images
- **stacks_order** (*list of integer*) – Order of stacks in the reconstruction

Outputs

- **output_sr** (*pathlike object or string representing a file*) – SR reconstructed image
- **output_sdi** (*pathlike object or string representing a file*) – SDI image
- **output_hr_mask** (*pathlike object or string representing a file*) – SRR mask
- **output_tranforms** (*list of items which are a pathlike object or string representing a file*) – Estimated transformation parameters
- **outputnode.output_json_path** (*pathlike object or string representing a file*) – Path to the json sidecar of the SR reconstruction
- **outputnode.output_sr_png** (*pathlike object or string representing a file*) – Path to the PNG of the SR reconstruction
- **outputnode.output_TV_parameters** (*dictionary*) – Parameters used for TV reconstruction

Example

```
>>> from pymialsrtk.pipelines.workflows import recon_stage as rec
>>> recon_stage = rec.create_preproc_stage(
        p_paramTV,
        p_use_manual_masks,
        p_do_nlm_denoising=False)
>>> recon_stage.inputs.inputnode.input_images =
        ['sub-01_run-1_T2w.nii.gz', 'sub-01_run-2_T2w.nii.gz']
>>> recon_stage.inputs.inputnode.input_masks =
        ['sub-01_run-1_T2w_mask.nii.gz', 'sub-01_run-2_T2w_mask.nii.gz
↪']
>>> recon_stage.inputs.stacks_order = [2,1]
>>> recon_stage.run()
```

Module for the high-resolution reconstruction stage of low-resolution labelmaps in the super-resolution reconstruction pipeline.

`pymialsrtk.workflows.recon_labelmap_stage.`**`create_recon_labelmap_stage`**(*p_sub_ses, p_verbose=False, name='recon_labels*

Create a SR reconstruction workflow for tissue label maps.

Parameters

- **p_sub_ses** (*string*) – String containing subject-session information for output formatting
- **p_verbose** (*boolean*) – Whether verbosity should be enabled (default: `False`)

- **name** (*string*) – Name of workflow (default: "recon_labels_stage")

Inputs

- **input_labels** (*list of items which are a pathlike object or string representing a file*) – Input LR label maps
- **input_masks** (*list of items which are a pathlike object or string representing a file*) – Input mask images
- **input_transforms** (*list of items which are a pathlike object or string representing a file*) – Input tranforms
- **input_reference** (*pathlike object or string representing a file*) – Input HR reference image
- **label_ids** (*list of integer*) – Label IDs to reconstruct
- **stacks_order** (*list of integer*) – Order of stacks in the reconstruction

Outputs **output_labelmap** (*pathlike object or string representing a file*) – HR labelmap

Example

```
>>> from pymialsrtk.pipelines.workflows import recon_labelmap_stage as␣
↪rec_label
>>> recon_labels_stage = rec_label.create_recon_labelmap_stage(
        p_sub_ses=p_sub_ses,
        p_verbose=p_verbose
    )
>>> recon_labels_stage.inputs.input_labels = [
        'sub-01_run-1_labels.nii.gz',
        'sub-01_run-2_labels.nii.gz'
    ]
>>> recon_labels_stage.inputs.input_masks = [
        'sub-01_run-1_T2w.nii_mask.gz',
        'sub-01_run-2_T2w.nii_mask.gz'
    ]
>>> recon_labels_stage.inputs.input_transforms = [
        'sub-01_run-1_transforms.txt',
        'sub-01_run-2_transforms.txt'
    ]
>>> recon_labels_stage.inputs.input_reference = 'sub-01_desc-GT_T2w.
↪nii.gz'
>>> recon_labels_stage.inputs.label_ids = 'sub-01_desc-GT_T2w.nii.gz'
>>> recon_labels_stage.inputs.stacks_order = [2,1]
>>> recon_labels_stage.run()
```

Module for the assessment of the super-resolution reconstruction quality with a reference.

`pymialsrtk.workflows.srr_assessment_stage.`**`create_srr_assessment_stage`**(*p_do_multi_paramet*
p_do_reconstruct_lal
p_input_srtv_node=N
p_verbose=False,
p_openmp_number_c
name='srr_assessme

Create an assessment workflow to compare a SR-reconstructed image and a reference target.

Parameters

- **name** (*string*) – Name of workflow (default: "sr_assessment_stage")
- **p_do_multi_parameters** (*boolean*) – whether multiple SR are to be assessed with different TV parameters (default: `False`)
- **p_input_srtv_node** (*string*) – when p_do_multi_parameters is set, name of the sourcenode from which metrics must be merged
- **p_openmp_number_of_cores** (*integer*) – number of threads possible for ants registration (default : 1)

Inputs

- **input_reference_image** (*pathlike object or string representing a file*) – Path to the ground truth image against which the SR will be evaluated.
- **input_reference_mask** (*pathlike object or string representing a file*) – Path to the mask of the ground truth image.
- **input_reference_labelmap** (*pathlike object or string representing a file*) – Path to the labelmap (tissue segmentation) of the ground truth image.
- **input_sr_image** (*pathlike object or string representing a file*) – Path to the SR reconstructed image.
- **input_sdi_image** (*pathlike object or string representing a file*) – Path to the SDI (interpolated image) used as input to the SR.
- **input_TV_parameters** (*dictionary*) – Dictionary of parameters that were used for the TV reconstruction.

Outputs **outputnode.output_metrics** (*list of float*) – List of output metrics

Example

```
>>> from pymialsrtk.pipelines.workflows import srr_assessment_stage as␣
↪srr_assessment
>>> srr_eval = srr_assessment.create_srr_assessment_stage()
>>> srr_eval.inputs.input_reference_image = 'sub-01_desc-GT_T2w.nii.gz'
>>> srr_eval.inputs.input_reference_mask = 'sub-01_desc-GT_mask.nii.gz'
>>> srr_eval.inputs.input_reference_mask = 'sub-01_desc-GT_labels.nii.
↪gz'
>>> srr_eval.inputs.input_sr_image = 'sub-01_id-1_rec-SR_T2w.nii.gz'
>>> srr_eval.inputs.input_sdi_image = 'sub-01_id-1_desc-SDI_T2w.nii.gz'
>>> srr_eval.inputs.input_TV_parameters = {
        'in_loop': '10',
        'in_deltat': '0.01',
        'in_lambda': '0.75',
        'in_bregman_loop': '3',
        'in_iter': '50',
        'in_step_scale': '1',
        'in_gamma': '1',
        'in_inner_thresh':
        '1e-05',
        'in_outer_thresh': '1e-06'
    }
>>> srr_eval.run()
```

Workflow for the management of super-resolution reconstruction pipeline outputs.

pymialsrtk.workflows.output_stage.**create_preproc_output_stage**(*p_sub_ses, p_sr_id, p_run_type, p_use_manual_masks, p_do_nlm_denoising=False, p_skip_stacks_ordering=False, p_do_registration=False, name='preproc_output_stage'*)

Create an output management workflow for the preprocessing only pipeline.

Parameters

- **p_sub_ses** (*string*) – String containing subject-session information for output formatting
- **p_sr_id** (*integer*) – ID of the current run
- **p_run_type** (*"preprocessing" or "super resolution"*) – Type of run
- **p_use_manual_masks** (*boolean*) – Whether manual masks were used in the pipeline
- **p_do_nlm_denoising** (*boolean*) – Enable non-local means denoising (default: False)
- **p_skip_stacks_ordering** (*boolean*) – Skip stacks ordering (default: False) If disabled, `report_image` and `motion_tsv` are not generated
- **p_do_registration** (*boolean*) – Whether registration is performed in the preprocessing pipeline
- **name** (*string*) – name of workflow (default: "preproc_output_stage")

Inputs

- **sub_ses** (*string*) – String containing subject-session information for output formatting
- **sr_id** (*integer*) – ID of the current run
- **stacks_order** (*list of integer*) – Order of stacks in the registration (list of integer)
- **final_res_dir** (*pathlike object or string representing a file*) – Output directory
- **run_type** (*"preprocessing" or "super resolution"*) – Type of run
- **input_masks** (*list of items which are a pathlike object or string representing a file*) – Input mask images from the low-resolution T2w images
- **input_images** (*list of items which are a pathlike object or string representing a file*) – Input low-resolution T2w images
- **input_sdi** (*pathlike object or string representing a file*) – Interpolated high resolution volume, obtained after slice-to-volume registration (SVR) Optional - only if `p_do_registration = True`
- **input_transforms** (*list of items which are a pathlike object or string representing a file*) – Transforms obtained after SVR Optional - only if `p_do_registration = True`
- **report_image** (*pathlike object or string representing a file*) – Report image obtained from the StacksOrdering module Optional - only if `p_skip_stacks_ordering = False`

- **motion_tsv** (*pathlike object or string representing a file*) – Motion index obtained from the StacksOrdering module Optional - only if `p_skip_stacks_ordering = False`
- **input_images_nlm** (*list of items which are a pathlike object or string representing a file*) – Input low-resolution denoised T2w images, Optional - only if `p_do_nlm_denoising = True`

`pymialsrtk.workflows.output_stage.`**`create_srr_output_stage`**(*p_sub_ses, p_sr_id, p_run_type, p_keep_all_outputs=False, p_use_manual_masks=False, p_do_nlm_denoising=False, p_do_reconstruct_labels=False, p_do_srr_assessment=False, p_skip_stacks_ordering=False, p_do_multi_parameters=False, p_subject=None, p_session=None, p_stacks=None, p_output_dir=None, p_run_start_time=None, p_run_elapsed_time=None, p_skip_svr=None, p_do_anat_orientation=None, p_do_refine_hr_mask=None, p_masks_derivatives_dir=None, p_openmp_number_of_cores=None, p_nipype_number_of_cores=None, name='srr_output_stage'*)

Create a output management workflow for the super-resolution reconstruction pipeline.

Parameters

- **p_sub_ses** – String containing subject-session information for output formatting
- **p_sr_id** – ID of the current run
- **p_run_type** – Type of run (preprocessing/super resolution/ . . .)
- **p_keep_all_outputs** – Whether intermediate outputs must be issues
- **p_use_manual_masks** – Whether manual masks were used in the pipeline
- **p_do_nlm_denoising** (`bool`) – Enable non-local means denoising (default: False)
- **p_do_reconstruct_labels** (`bool`) – Enable the reconstruction of labelmaps
- **p_do_srr_assessment** (`bool`) – Enables output of srr assessment stage
- **p_skip_stacks_ordering** (`bool`) – Skip stacks ordering (default: False) If disabled, `report_image` and `motion_tsv` are not generated
- **p_do_multi_parameters** (`bool`) – Whether recon_stage was performed in a multi-TV mode
- **name** (`str`) – name of workflow (default: “srr_output_stage”)

Inputs

- **stacks_order** (*list of integer*) – Order of stacks in the registration (list of integer)
- **use_manual_masks** (`bool`) – Whether manual masks were used in the pipeline
- **final_res_dir** (*pathlike object or string representing a file*) – Output directory
- **input_masks** (*list of items which are a pathlike object or string representing a file*) – Input mask images from the low-resolution T2w images
- **input_images** (*list of items which are a pathlike object or string representing a file*) – Input low-resolution T2w images
- **input_transforms** (*list of items which are a pathlike object or string representing a file*) – Transforms obtained after SVR
- **input_sdi** (*pathlike object or string representing a file*) – Interpolated high resolution volume, obtained after slice-to-volume registration (SVR)
- **input_sr** (*pathlike object or string representing a file*) – High resolution volume, obtained after the super- resolution (SR) reconstruction from the SDI volume.
- **input_hr_mask** (*pathlike object or string representing a file*) – Brain mask from the high-resolution reconstructed volume.
- **input_json_path** (*pathlike object or string representing a file*) – Path to the JSON file describing the parameters used in the SR reconstruction.
- **input_sr_png** (*pathlike object or string representing a file*) – PNG image summarizing the SR reconstruction.
- **report_image** (*pathlike object or string representing a file*) – Report image obtained from the StacksOrdering module Optional - only if `p_skip_stacks_ordering = False`
- **motion_tsv** (*pathlike object or string representing a file*) – Motion index obtained from the StacksOrdering module Optional - only if `p_skip_stacks_ordering = False`
- **input_images_nlm** (*pathlike object or string representing a file*) – Input low-resolution denoised T2w images Optional - only if `p_do_nlm_denoising = True`

2.12 BSD 3-Clause License

2.13 Citing

Important:

- If your are using the MIALSRTK BIDS App, a manuscript is in preparation, but for now, please acknowledge this software with the following three entries:

  1. Tourbier S, De Dumast P., Kebiri H., Sanchez T., Lajous H., Hagmann P., Bach Cuadra M. (2023, January 31). Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit: MIAL Super-Resolution Toolkit (Version v2.1.0). Zenodo. http://doi.org/10.5281/zenodo.4290209
  2. Tourbier S, Bresson X, Hagmann P, Meuli R, Bach Cuadra M, *An efficient total variation algorithm for super-resolution in fetal brain MRI with adaptive regularization*, Neuroimage 118 (2015) 584-597. doi:10.1016/j.neuroimage.2015.06.018
  3. Tourbier S, Velasco-Annis C, Taimouri V, Hagmann P, Meuli R, Warfield SK, Bach Cuadra M, A. Gholipour, *Automated template-based brain localization and extraction for fetal brain MRI reconstruction*, Neuroimage (2017) In Press. doi:10.1016/j.neuroimage.2017.04.004

2.14 Changes

2.14.1 Version 2.0.3

Date: December 24, 2021

This corresponds to the release of MIAL Super-Resolution Toolkit 2.0.3, that includes in particular the following changes.

New feature

- You can now be aware about the adverse impact of your processing on the environment **:raw-html:🌍:raw-html:✨:raw-html:🎄**! A new `--track_carbon_footprint` option has been added to the `mialsuperresolutiontoolkit_docker` and `mialsuperresolutiontoolkit_singularity` python wrappers of the BIDS App, which will use codecarbon[68] to estimate the amount of carbon dioxide (CO2) produced to execute the BIDS App. Results are saved in `<bids_dir>/code/emissions.csv`.
- Functions `get_emission_car_miles_equivalent` and `get_emission_tv_time_equivalent` that convert the CO2 emission in terms of (i) kms traveled by an average car and (ii) time of watching a 32-inch screen have been added to `pymialsrtk.interfaces.util` module.

More...

Please check pull request 113[69] for more change details and development discussions.

2.14.2 Version 2.0.2

Date: November 22, 2021

This corresponds to the release of MIAL Super-Resolution Toolkit 2.0.2, that includes in particular the following changes.

New feature

- `pymialsrtk` enables to fix the maximal amount of memory (in Gb) that could be used by the pipelines at execution with the `--memory MEMORY_Gb` option flag. (See pull request 92[70]).
- `pymialsrtk` generates a HTML processing report for each subject in `sub-<label>/report/sub-<label>.html`. It includes the following:
  - Pipeline/workflow configuration summary
  - Nipype workflow execution graph
  - Link to the processing log
  - Plots for the quality check of the automatic reordering step based on the motion index.
  - Three orthogonal cuts of the reconstructed image
  - Computing environment summary

  (See pull requests 97[71], 102[72], and 103[73]).

[68] https://codecarbon.io/
[69] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/113
[70] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/92
[71] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/97
[72] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/102
[73] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/103

Major change

- The method `pymialsrtk.postprocess.binarize_image()` has been modified and encapsulated in a new interface called `pymialsrtk.postprocess.BinarizeImage`.

Python update

- From `3.6.8` to `3.7.10`

New package

- pandas `1.1.5`
- sphinxcontrib-apidoc `0.3.0` (required to build documentation)
- sphinxcontrib-napoleon `0.7` (required to build documentation)

Package update

- traits from `5.1.2` to `6.3.0`
- nipype from `1.6.0` to `1.7.0`
- nilearn from `0.7.1` to `0.8.1`
- numpy from `1.16.6` to `1.21.3`
- scikit-learn from `0.20` to `1.0.1`
- scikit-image from `0.14` to `0.16.2`

Bug fix

- Correct the filename of the high-resolution brain mask generated by the data sinker in `mialsrtk-<variant>/sub-<label>/anat` (See pull request 92[74]).
- `mialsrtkImageReconstruction` updates the reference image used for slice-to-volume registration using the high-resolution image reconstructed by SDI at the previous iteration.
- The following Sphinx extension packages were added to the conda environment, that were required if one wish to build the documentation locally:
  - sphinxcontrib-apidoc `0.3.0`
  - sphinxcontrib-napoleon `0.7`

Note

It was not possible to update the version of tensorflow for the moment. All versions of tensorflow greater than 1.14 are in fact compiled with a version of GCC much more recent than the one available in Ubuntu 14.04. This seems to cause unresponsiveness of the `preprocess.BrainExtraction` interface node which can get stuck while getting access to the CPU device.

[74] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/92

Software development life cycle

- Use PEP 8 Speaks[75], a GitHub app to automatically review Python code style over Pull Requests. (Configuration described by `pep8speaks.yml`)

More...

Please check pull request 70[76] and pull request 110[77] for more change details and development discussions.

2.14.3 Version 2.0.1

Date: December 24, 2020

This corresponds to the release of MIAL Super-Resolution Toolkit 2.0.1, that includes in particular the following changes.

Major change

- Review `setup.py` for publication of future release of `pymialsrtk` to PyPI (See pull request 59[78]).
- Review creation of entrypoint scripts of the container for compatibility with Singularity (See pull request 60[79]).
- Use `MapNode` for all interfaces that apply a processing independently to a list of images (See pull request 68[80]).
- Use the nipype sphinx extension to generate API documentation (See pull request 65[81]).
- Review the `--manual` option flag which takes as input a directory with brain masks (See pull request 51[82]).

New feature

- `pymialsrtk` enables to skip different steps in the super-resolution pipeline (See pull request 63[83]).
- Support of Singularity to execute MIALSTK on high-performance computing cluster (See pull request 60[84]).
- `pymialsrtk` implements for convenience a Python wrapper that generates the Singularity command line of the BIDS App for you, prints it out for reporting purposes, and then executes it without further action needed (See pull request 61[85]).

[75] https://pep8speaks.com/
[76] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/70
[77] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/110
[78] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/59
[79] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/60
[80] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/68
[81] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/65
[82] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/51
[83] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/63
[84] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/60
[85] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/61

Software development life cycle

- Add `test-python-install` job to CircleCI to test the creation of the distribution wheel to PyPI and test its installation via `pip` (See pull request 34[86]).
- Add `deploy-pypi-release` job to CircleCI to publish the package of a new release to PyPI (See pull request 59[87]).
- Add `build-singularity`, `test-singularity`, `deploy-singularity-latest`, and `deploy-singularity-release` jobs in CircleCI to build, test and deploy a Singularity image of `MIALSRTK` to Sylabs.io[88] (See pull request 34[89]). The tests includes:
  - Test 03: Run BIDS App on the sample `data/` BIDS dataset with the `--manual_masks` option without code coverage.
  - Test 04: Run BIDS App on the sample `data/` BIDS dataset with automated brain extraction (masking) without code coverage.

More...

Please check pull request 53[90] for more change details and development discussions.

2.14.4 Version 2.0.0

Date: November 25, 2020

This corresponds to the first release of the second version of the MIAL Super-Resolution Toolkit, which has evolved massively over the last years in terms of the underlying codebase and the scope of the functionality provided, following recent advances in standardization of neuroimaging data organization and processing workflows.

Major changes

- Adoption of the Brain Imaging Data Structure standard[91] for data organization and the sample dataset available in data/ has been modified accordingly. (See *BIDS and BIDS App standards* (page 7) for more details)
- MIALSRTK is going to Python with the creation of the `pymialsrtk` workflow library which extends the Nipype dataflow library[92] with the implementation of interfaces to all C++ MIALSRTK tools connected in a common workflow to perform super-resolution reconstruction of fetal brain MRI with data provenance and execution detail recordings. (See *API Documentation* (page 22))
- Docker image encapsulating MIALSRTK is distributed as a BIDS App, a standard for containerized workflow that handles BIDS datasets with a set of predefined commandline input argument. (See *BIDS App Commadline Usage* (page 9) for more details)
- Main documentation of MIALSRTK is rendered using readthedocs at https://mialsrtk.readthedocs.io/.

[86] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/34
[87] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/59
[88] https://sylabs.io
[89] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/34
[90] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/53
[91] https://bids.neuroimaging.io/
[92] https://nipype.readthedocs.io/en/latest/

New feature

- `pymialsrtk` implements an automatic brain extraction (masking) module based on a 2D U-Net (Ronneberger et al. [Ref1]) using the pre-trained weights from Salehi et al. [Ref2] (See pull request 4[93]). It is integrated in the BIDS App workflow by default.
- `pymialsrtk` implements a module for automatic stack reference selection and ordering (masking) based on the tracking of the brain mask centroid slice by slice (See pull request 34[96])
- `pymialsrtk` implements for convenience a Python wrapper that generates the Docker command line of the BIDS App for you,

prints it out for reporting purposes, and then executes it without further action needed (See pull request 47[97])

Software development life cycle

- Adopt CircleCI for continuous integration testing and run the following regression tests:
  - Test 01: Run BIDS App on the sample `data/` BIDS dataset with the `--manual_masks` option.
  - Test 02: Run BIDS App on the sample `data/` BIDS dataset with automated brain extraction (masking).

  See CircleCI project page[98].
- Use Codacy[99] to support code reviews and monitor code quality over time.
- Use coveragepy[100] in CircleCI during regression tests of the BIDS app and create code coverage reports published on our Codacy project page[101].

More...

Please check pull request 2[102] and pull request 4[103] for more change details and development discussions.

2.15 Contributing

This project follows the all contributors specification[104]. Contributions in many different ways are welcome!

[93] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/4
[96] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/34
[97] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/47
[98] https://app.circleci.com/pipelines/github/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit
[99] https://www.codacy.com/
[100] https://coverage.readthedocs.io/en/coverage-5.2/
[101] https://app.codacy.com/gh/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/dashboard
[102] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/2
[103] https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/pull/4
[104] https://allcontributors.org/

2.15.1 Contribution Types

Report Bugs

Report bugs at https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/issues.

If you are reporting a bug, please include:

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

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

The best way to send feedback is to create an issue at https://github.com/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.15.2 Get Started!

Ready to contribute? Here's how to set up `MIALSRTK` for local development.

1. Fork the `mialsuperresolutiontoolkit` repo on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/mialsuperresolutiontoolkit.git
cd mialsuperresolutiontoolkit
```

3. Create a branch for local development:

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

4. Now you can make your changes locally. If you add a new node in a pipeline or a completely new pipeline, we encourage you to rebuild the BIDS App Docker image (See *BIDS App build instructions* (page 71)) and test it on the sample dataset (`mialsuperresolutiontoolkit/data/`).

Note: Please keep your commit the most specific to a change it describes. It is highly advice to track unstaged files with `git status`, add a file involved in the change to the stage one by one with `git add <file>`. The use of `git add .` is highly disencouraged. When all the files for a given change are staged, commit the files with a brieg message using `git commit -m "[COMMIT_TYPE]: Your detailed description of the change. "` that describes your change and where `[COMMIT_TYPE]` can be `[FIX]` for a bug fix, `[ENH]` for a new feature, `[MAINT]` for code maintenance and typo fix, `[DOC]` for documentation, `[CI]` for continuous integration testing.

---

5. When you're done making changes, push your branch to GitHub:

```
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, the docs should be updated (See *documentation build instructions* (page 72)).
2. The pull request should work for Python 3.6. Check https://app.circleci.com/pipelines/github/Medical-Image-Analysis-Laboratory/mialsuperresolutiontoolkit and make sure that the tests pass.

How to build the BIDS App locally

1. Go to the clone directory of your fork and run the script `build_bidsapp.sh`

```
cd mialsuperresolutiontoolkit
sh build_bidsapp.sh
```

Note that the tag of the version of the image will be extracted from `pymialsrtk/info.py` where you might need to change the version to not overwrite an other existing image with the same version.

How to install `pyMIALSTK` locally

1. Install the `MIALSRTK` conda environment `pymialsrtk-env` that provides a Python 3.6 environment:

```
cd mialsuperresolutiontoolkit
conda env create -f docker/bidsapp/environment.yml
```

2. Activate the `pymialsrtk-env` conda environment and install `pymialsrtk`

```
conda activate pymialsrtk-env
pip install .
```

How to build the documentation locally

1. Install the MIALSRTK conda environment pymialsrtk-env with sphinx and all extensions to generate the documentation:

```
cd mialsuperresolutiontoolkit
conda env create -f docker/bidsapp/environment.yml
```

2. Activate the MIALSRTK conda environment pymialsrtk-env and install pymialsrtk

```
conda activate pymialsrtk-env
pip install .
```

3. Run the script build_sphinx_docs.sh to generate the HTML documentation in documentation/_build/html:

```
bash build_sphinx_docs.sh
```

Note: Make sure to have activated the conda environment pymialsrtk-env before running the script build_sphinx_docs.sh.

2.15.3 Not listed as a contributor?

This is easy, MIALSRTK has the all contributors bot[105] installed.

Just comment on Issue or Pull Request (PR), asking @all-contributors to add you as contributor:

```
@all-contributors please add <github_username> for <contributions>
```

<contribution>: See the Emoji Key Contribution Types Reference[106] for a list of valid contribution types.

The all-contributors bot will create a PR to add you in the README and reply with the pull request details.

When the PR is merged you will have to make an extra Pull Request where you have to:

1. add your entry in the zenodo.json (for that you will need an ORCID ID - https://orcid.org/). Doing so, you will appear as a contributor on Zenodo in the future version releases of MIALSRTK. Zenodo is used by MIALSRTK to publish and archive each of the version release with a unique Digital Object Identifier (DOI), which can then be used for citation.
2. update the content of the table in documentation/contributors.rst with the new content generated by the bot in the README. Doing so, you will appear in the *Contributing Page* (page 69).

This document has been inspired and adapted from these great contributing guidelines[107].

[105] https://allcontributors.org/docs/en/bot/usage
[106] https://github.com/all-contributors/all-contributors/blob/master/docs/emoji-key.md
[107] https://github.com/dPys/MIALSRTK/edit/master/docs/contributing.rst

2.16 Contributors

Bibliography

[Rousseau2006] Rousseau et al.; Acad Radiol., 2006. (link to paper)[64]

[Tourbier2015] Tourbier et al.; NeuroImage, 2015. (link to paper)[67]

[Ref1] Ronneberger et al.; Medical Image Computing and Computer Assisted Interventions, 2015. (link to paper)[94]

[Ref2] Salehi et al.; arXiv, 2017. (link to paper)[95]

[64] https://doi.org/10.1016/j.acra.2006.05.003
[67] https://doi.org/10.1016/j.neuroimage.2015.06.018
[94] https://arxiv.org/abs/1505.04597
[95] https://arxiv.org/abs/1710.09338

Python Module Index

p

Index

R

S