fMRIprep Documentation

Release version

Craig A. Moodie, Krzysztof J. Gorgolewski, Oscar Esteban, Ross I

Mar 24, 2017

Contents

This pipeline is developed by the Poldrack lab at Stanford University for use at the Center for Reproducible Neuroscience (CRN), as well as for open-source software distribution.

CHAPTER 1

About

`fmriprep` is a functional magnetic resonance imaging (fMRI) data preprocessing pipeline that is designed to provide an easily accessible, state-of-the-art interface that is robust to differences in scan acquisition protocols and that requires minimal user input, while providing easily interpretable and comprehensive error and output reporting. It performs basic processing steps (coregistration, normalization, unwarping, noise component extraction, segmentation, skullstripping etc.) providing outputs that make running a variety of group level analyses (task based or resting state fMRI, graph theory measures, surface or volume, etc.) easy.

Note: fmriprep performs minimal preprocessing. Here we define 'minimal preprocessing' as motion correction, field unwarping, normalization, field bias correction, and brain extraction. See the workflows for more details.

The fmriprep pipeline primarily utilizes FSL tools, but also utilizes ANTs tools at several stages such as skull stripping and template registration. This pipeline was designed to provide the best software implementation for each state of preprocessing, and will be updated as newer and better neuroimaging software become available.

This tool allows you to easily do the following:

- Take fMRI data from raw to full preprocessed form.
- Implement tools from different software packages.
- Achieve optimal data processing quality by using the best tools available.
- Generate preprocessing quality reports, with which the user can easily identify outliers.
- Receive verbose output concerning the stage of preprocessing for each subject, including meaningful errors.
- Automate and parallelize processing steps, which provides a significant speed-up from typical linear, manual processing.

More information and documentation can be found here:

https://fmriprep.readthedocs.io/

CHAPTER 2

Principles

`fmriprep` is built around three principles:

1. **Robustness** - the pipeline adapts the preprocessing steps depending on the input dataset and should provide results as good as possible independently of scanner make, scanning parameters or presence of additional correction scans (such as fieldmaps)
2. **Ease of use** - thanks to dependance on the BIDS standard manual parameter input is reduced to a minimum allow the pipeline to run in an automatic fashion.
3. **"Glass box"** philosophy - automation should not mean that one should not visually inspect the results or understand the methods. Thus `fmriprep` provides for each subject visual reports detailing the accuracy of the most importatnt processing steps. This combined with the documentation can help researchers to understand the process and decide which subjects should be kept for the group level analysis.

CHAPTER 3

Acknowledgements

Please acknowledge this work mentioning explicitly the name of this software (fmriprep) and the version, along with the link to the GitHub repository (https://github.com/poldracklab/fmriprep).

CHAPTER 4

License information

We use the 3-clause BSD license; the full license is in the file `LICENSE` in the `fmriprep` distribution.

CHAPTER 5

Authors

This open-source neuroimaging data processing tool is being developed as a part of the MRI image analysis and reproducibility platform offered by the CRN.

The CRN (Center for Reproducible Neuroscience) developers team:

- Chris F. Gorgolewski
- Craig Moodie
- Ross Blair
- Shoshana Berleant
- Oscar Esteban
- Christopher J. Markiewicz
- Russell A. Poldrack

Poldrack Lab, Psychology Department, Stanford University.

CHAPTER 6

Contents

Installation

There are three ways to use fmriprep: in a *Docker Container*, in a *Singularity Container*, or in a *Manually Prepared Environment*. Using a container method is highly recommended. Once you are ready to run fmriprep, see Usage for details.

Docker Container

Make sure command-line Docker is installed.

See *External Dependencies* for more information (e.g., specific versions) on what is included in the fmriprep Docker image.

Now, assuming you have data, you can run fmriprep. You will need an active internet connection the first time.

```
$ docker run -ti --rm \
    -v filepath/to/data/dir:/data:ro \
    -v filepath/to/output/dir:/out \
    poldracklab/fmriprep:latest \
    /data /out/out \
    participant
```

For example:

```
$ docker run -ti --rm \
    -v $HOME/fullds005:/data:ro \
    -v $HOME/dockerout:/out \
    poldracklab/fmriprep:latest \
    /data /out/out \
    participant \
    --ignore fieldmaps
```

Singularity Container

For security reasons, many HPCs (e.g., TACC) do not allow Docker containers, but do allow Singularity containers. In this case, start with a machine (e.g., your personal computer) with Docker installed. Use docker2singularity to create a singularity image. You will need an active internet connection and some time.

```
$ docker run --privileged -t --rm \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v D:\host\path\where\to\output\singularity\image:/output \
    singularityware/docker2singularity \
    poldracklab/fmriprep:latest
```

Transfer the resulting Singularity image to the HPC, for example, using `scp`.

```
$ scp poldracklab_fmriprep_latest-*.img user@hcpserver.edu:/path/to/downloads
```

If the data to be preprocessed is also on the HPC, you are ready to run fmriprep.

```
$ singularity run path/to/singularity/image.img \
    path/to/data/dir path/to/output/dir \
    participant \
    --participant_label label
```

For example:

```
$ singularity run ~/poldracklab_fmriprep_latest-2016-12-04-5b74ad9a4c4d.img \
    /work/04168/asdf/lonestar/ $WORK/lonestar/output \
    participant \
    --participant_label sub-387 --nthreads 16 -w $WORK/lonestar/work \
    --ants-nthreads 16
```

Note: Singularity by default exposes all environment variables from the host inside the container. Because of this your host libraries (such as nipype) could be accidentally used instead of the ones inside the container - if they are included in PYTHONPATH. To avoid such situation we recommend unsetting PYTHONPATH in production use. For example:

```
$ PYTHONPATH="" singularity run ~/poldracklab_fmriprep_latest-2016-12-04-5b74ad9a4c4d.
↪img \
  /work/04168/asdf/lonestar/ $WORK/lonestar/output \
  participant \
  --participant_label sub-387 --nthreads 16 -w $WORK/lonestar/work \
  --ants-nthreads 16
```

Manually Prepared Environment

Note: This method is not recommended! Make sure you would rather do this than use a *Docker Container* or a *Singularity Container*.

Make sure all of fmriprep's *External Dependencies* are installed. These tools must be installed and their binaries available in the system's `$PATH`.

If you have pip installed, install fmriprep

```
$ pip install fmriprep
```

If you have your data on hand, you are ready to run fmriprep:

```
$ fmriprep data/dir output/dir participant --participant_label label
```

External Dependencies

`fmriprep` is implemented using nipype, but it requires some other neuroimaging software tools:

- FSL (version 5.0.9)
- ANTs (version 2.1.0.Debian-Ubuntu_X64)
- AFNI (version Debian-16.2.07)
- C3D (version 1.0.0)

What's new

0.3.1 (24th of March 2017)

- [ENH] Perform bias field correction of EPI images prior to coregistration (#409)
- [FIX] Fix an orientation issue affecting some datasets when bbregister was used (#408)

0.3.0 (20th of March 2017)

- [FIX] Affine and warp MNI transforms are now applied in the correct order
- [ENH] Added preliminary support for reconstruction of cortical surfaces using FreeSurfer
- [ENH] Switched to bbregister for BOLD to T1 coregistration
- [ENH] Switched to sinc interpolation of preprocessed BOLD and T1w outputs
- [ENH] Preprocessed BOLD volumes are now saved in the T1w space instead of mean BOLD
- [FIX] Fixed a bug with MCFLIRT interpolation inducing slow drift
- [ENH] All files are now saved in Float32 instead of Float64 to save space

0.2.0 (13th of January 2017)

- Initial public release

0.1.2 (3rd of October 2016)

- [FIX] Downloads from OSF, remove data downloader (now in niworkflows)
- [FIX] pybids was missing in the install_requires
- [DEP] Deprecated -S/–subject-id tag
- [ENH] Accept subjects with several T1w images (#114)

- [ENH] Documentation updates (#130, #131)
- [TST] Re-enabled CircleCI tests on one subject from ds054 of OpenfMRI
- [ENH] Add C3D to docker image, updated poldracklab hub (#128, #119)
- [ENH] CLI is now BIDS-Apps compliant (#123)

0.1.1 (30th of July 2016)

- [ENH] Grabbit integration (#113)
- [ENH] More outputs in MNI space (#99)
- [ENH] Implementation of phase-difference fieldmap estimation (#91)
- [ENH] Fixed bug using non-RAS EPI
- [ENH] Works on ds005 (datasets without fieldmap nor sbref)
- [ENH] Outputs start to follow BIDS-derivatives (WIP)

0.0.1

- [ENH] Added Docker images
- [DOC] Added base code for automatic publication to RTD.
- Set up CircleCI with a first smoke test on one subject.
- BIDS tree scrubbing and subject-session-run selection.
- Refactored big workflow into consistent pieces.
- Migrated Craig's original code

Usage

Execution and the BIDS format

The `fmriprep` workflow takes as principal input the path of the dataset that is to be processed. The only requirement to the input dataset is that it has a valid BIDS (Brain Imaging Data Structure) format. This can be easily checked online using the BIDS Validator.

The exact command to run `fmriprep` depends on the Installation method. The common parts of the command follow the BIDS-Apps definition. Example:

```
fmriprep data/bids_root/ out/ participant -w work/
```

Command-Line Arguments

```
usage: run_workflow.py [-h]
                       [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
                       [-v] [-s SESSION_ID] [-r RUN_ID] [--task-id TASK_ID]
                       [-d {anat,func}] [--debug] [--nthreads NTHREADS]
                       [--mem_mb MEM_MB] [--write-graph]
```

```
                       [--use-plugin USE_PLUGIN] [-w WORK_DIR]
                       [--ignore {fieldmaps} [{fieldmaps} ...]]
                       [--reports-only] [--skip-native]
                       [--ants-nthreads ANTS_NTHREADS] [--skull-strip-ants]
                       [--no-skull-strip-ants] [--no-freesurfer]
                       bids_dir output_dir {participant}

fMRI Preprocessing workflow

positional arguments:
  bids_dir
  output_dir
  {participant}

optional arguments:
  -h, --help            show this help message and exit
  --participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]
  -v, --version         show program's version number and exit

fMRIprep specific arguments:
  -s SESSION_ID, --session-id SESSION_ID
  -r RUN_ID, --run-id RUN_ID
  --task-id TASK_ID     limit the analysis only ot one task
  -d {anat,func}, --data-type {anat,func}
  --debug               run debug version of workflow
  --nthreads NTHREADS   number of threads
  --mem_mb MEM_MB       try to limit the total amount of requested memory for all␣
↪workflows to this number
  --write-graph         Write workflow graph.
  --use-plugin USE_PLUGIN
                        nipype plugin configuration file
  -w WORK_DIR, --work-dir WORK_DIR
  --ignore {fieldmaps} [{fieldmaps} ...]
                        In case the dataset includes fieldmaps but you chose not to␣
↪take advantage of them.
  --reports-only        only generate reports, don't run workflows. This will only␣
↪rerun report aggregation, not reportlet generation for specific nodes.
  --skip-native         don't output timeseries in native space

specific settings for ANTs registrations:
  --ants-nthreads ANTS_NTHREADS
                        number of threads that will be set in ANTs processes
  --skull-strip-ants    use ANTs-based skull-stripping (default, slow))
  --no-skull-strip-ants
                        don't use ANTs-based skull-stripping (use  AFNI instead, fast)

settings for FreeSurfer preprocessing:
  --no-freesurfer       disable FreeSurfer preprocessing
```

Debugging

Logs and crashfiles are outputted into the `<output dir>/logs` directory. Information on how to customize and understand these files can be found on the nipype debugging page.

Support and communication

The documentation of this project is found here: http://fmriprep.readthedocs.org/en/latest/.

If you have a problem or would like to ask a question about how to use `fmriprep`, please submit a question to NeuroStars.org with an `fmriprep` tag. NeuroStars.org is a platform similar to StackOverflow but dedicated to neuroinformatics.

All previous `fmriprep` questions are available here: http://neurostars.org/tags/fmriprep/

To participate in the `fmriprep` development-related discussions please use the following mailing list: http://mail.python.org/mailman/listinfo/neuroimaging Please add *[fmriprep]* to the subject line when posting on the mailing list.

All bugs, concerns and enhancement requests for this software can be submitted here: https://github.com/poldracklab/fmriprep/issues.

Workflows

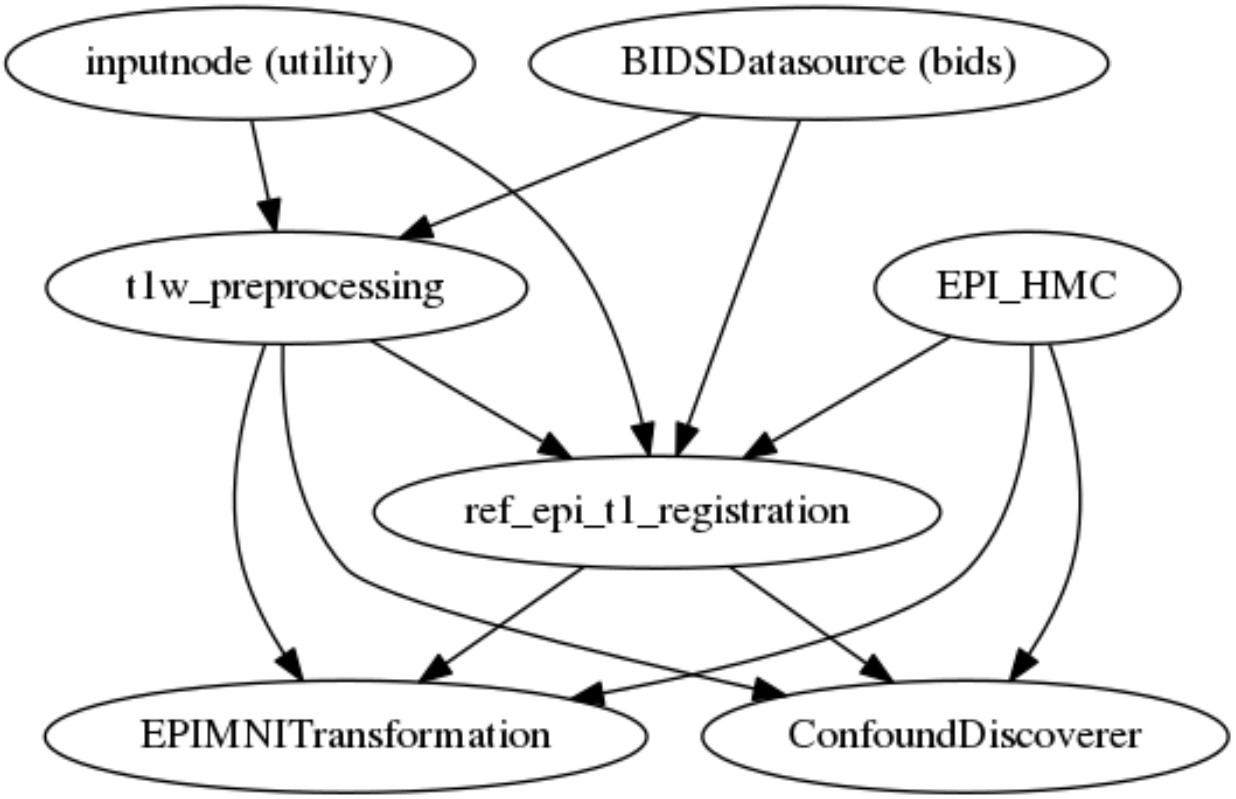
Basic workflow (no fieldmaps)

`fmriprep`'s basic pipeline is used on datasets for which there are only t1ws and at least one functional (EPI) file, but no SBRefs or fieldmaps. To force using this pipeline on datasets that do include fieldmaps and SBRefs use the `--ignore fieldmaps` flag.

Several steps are added or modified if *Surface preprocessing* is enabled.

What It Does

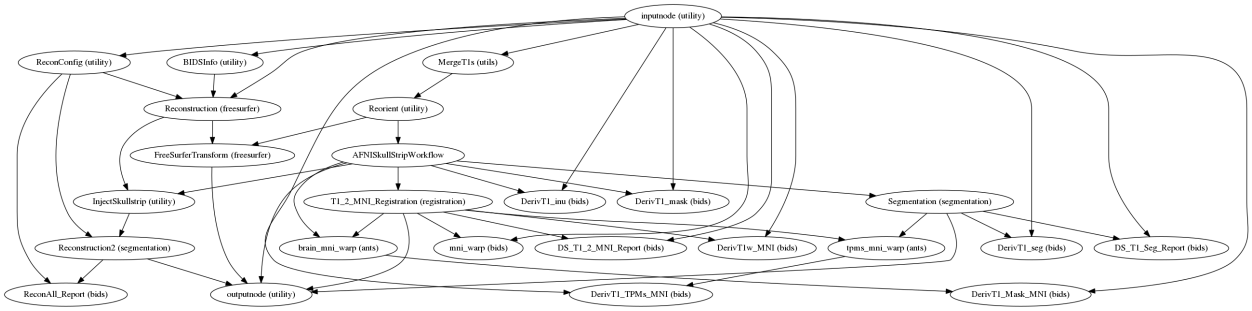
High-level view of the basic pipeline:



BIDSDatasource

This node reads the BIDS-formatted T1 data.

t1w_preprocessing



The `t1w_preprocessing` sub-workflow finds the skull stripping mask and the white matter/gray matter/cerebrospinal fluid segments and finds a non-linear warp to the MNI space.

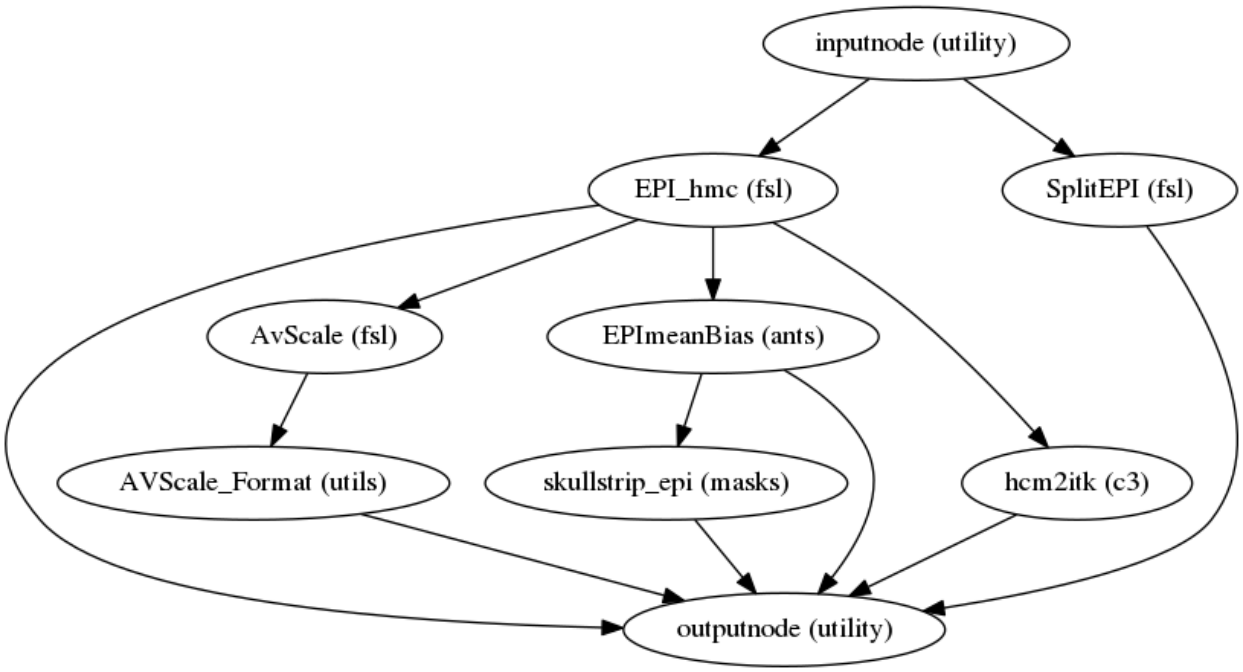
Fig. 6.1: Brain extraction (ANTs).

Fig. 6.2: Segmentation (FAST).

Fig. 6.3: Animation showing T1 to MNI normalization (ANTs)

If enabled, FreeSurfer surfaces are reconstructed from T1-weighted structural image(s), using the ANTs-extracted brain mask. See *Reconstruction* for details.

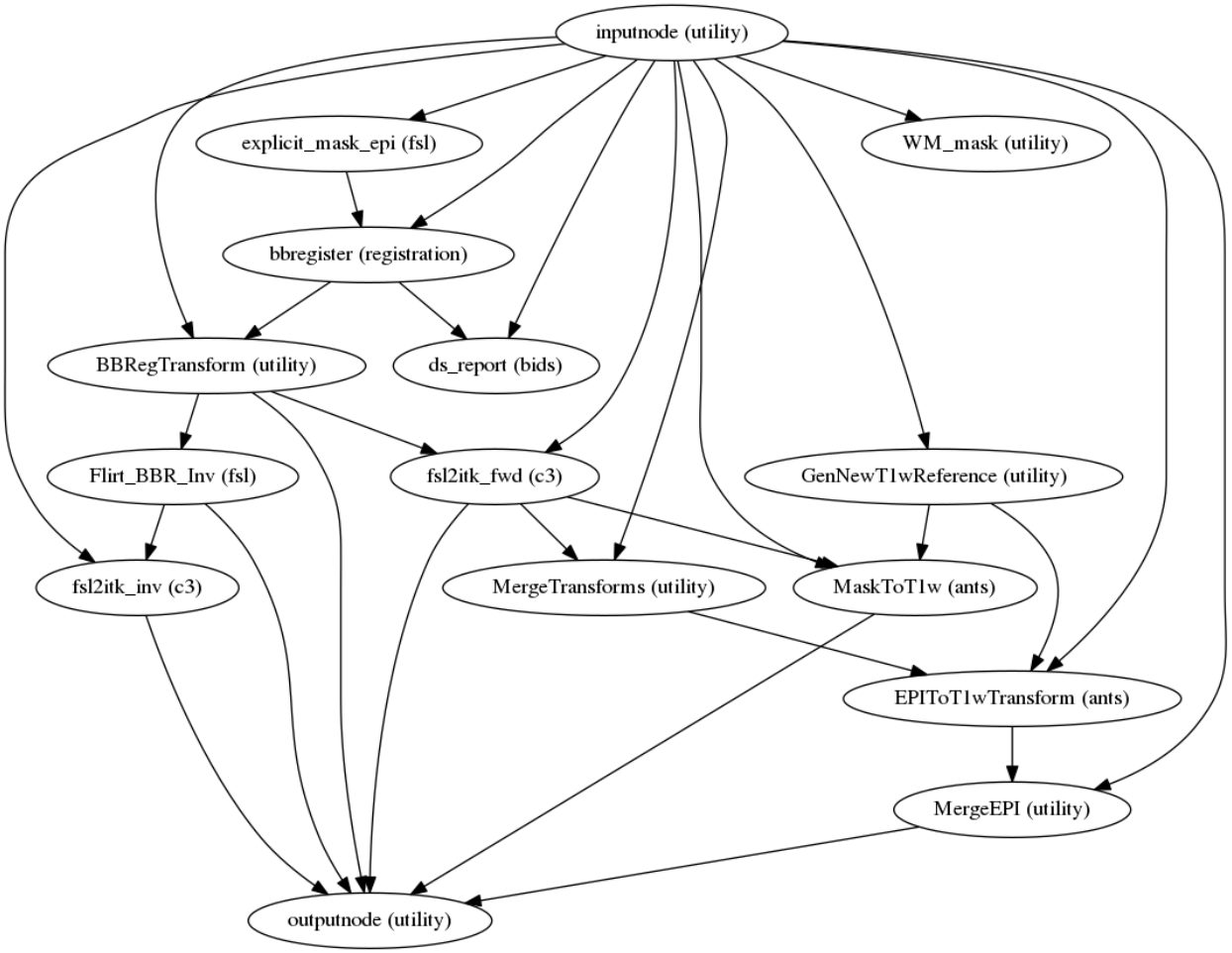
EPI_HMC



The EPI_HMC sub-workflow collects BIDS-formatted EPI files, performs head motion correction, and skullstripping. FSL MCFLIRT is used to estimate motion transformations and ANTs is used to apply them using Lanczos interpolation. Nilearn is used to perform skullstripping of the mean EPI image.

Fig. 6.4: Brain extraction (nilearn).

ref_epi_t1_registration

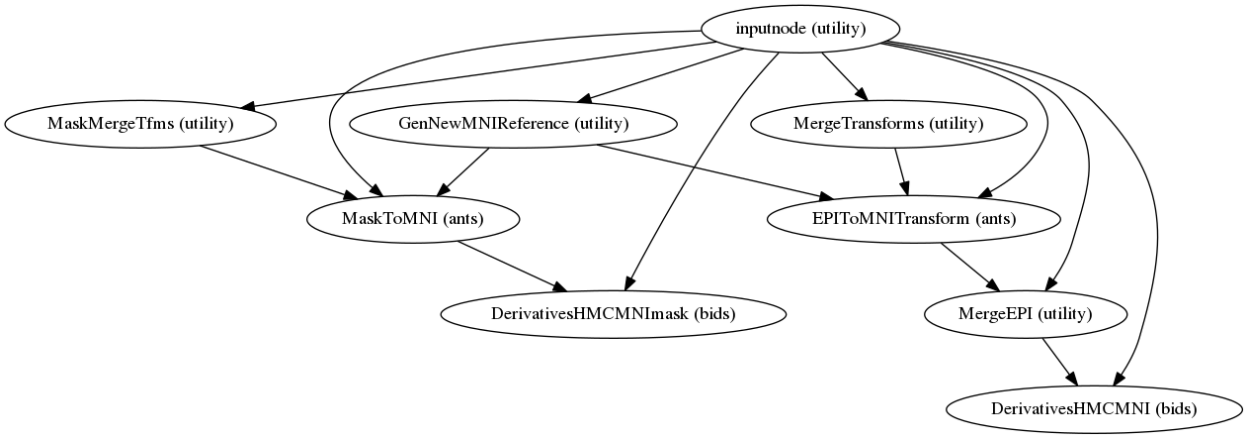


The ref_epi_t1_registration sub-workflow uses FSL FLIRT with the BBR cost function to find the transform that maps the EPI space into the T1-space.

Fig. 6.5: Animation showing EPI to T1 registration (FSL FLIRT with BBR)

If surface processing is enabled, `bbregister` is used instead. See *Boundary-based Registration (BBR)* for details.

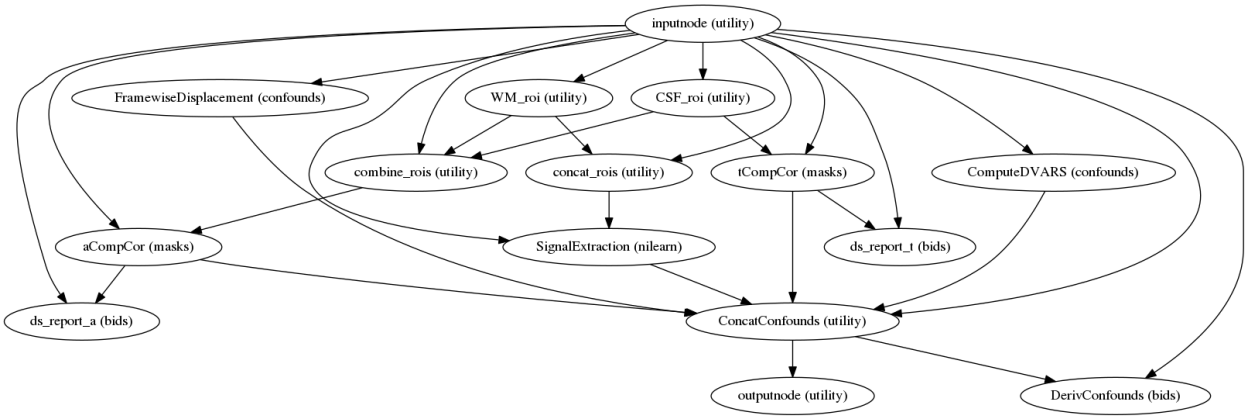
EPIMNITransformation



The EPIMNITransformation sub-workflow uses the transform from *ref_epi_t1_registration* and a T1-to-MNI transform from *t1w_preprocessing* to map the EPI image to standardized MNI space. It also maps the t1w-based mask to MNI space.

Transforms are concatenated and applied all at once, with one interpolation step, so as little information is lost as possible.

ConfoundDiscoverer



Given a motion-corrected fMRI, a brain mask, MCFLIRT movement parameters and a segmentation, the ConfoundDiscoverer sub-workflow calculates potential confounds per volume.

Calculated confounds include the mean global signal, mean tissue class signal, tCompCor, aCompCor, Framewise Displacement, 6 motion parameters and DVARS.

Reports

`fmriprep` outputs summary reports, outputted to `<output dir>/fmriprep/sub-<subject_label>.html`. These reports provide a quick way to make visual inspection of the results easy. Each report is self contained and thus can be easily shared with collaborators (for example via email). View a sample report.

Derivatives

There are additional files, called "Derivatives", outputted to `<output dir>/fmriprep/sub-<subject_label>/`. See the BIDS spec for more information.

Derivatives related to t1w files are in the `anat` subfolder:

- `*T1w_brainmask.nii.gz` Brain mask derived using ANTS or AFNI, depending on the command flag `--skull-strip-ants`
- `*T1w_space-MNI152NLin2009cAsym_brainmask.nii.gz` Same as above, but in MNI space.
- `*T1w_dtissue.nii.gz` Tissue class map derived using FAST.
- `*T1w_preproc.nii.gz` Bias field corrected t1w file, using ANTS' N4BiasFieldCorrection
- `*T1w_space-MNI152NLin2009cAsym_preproc.nii.gz` Same as above, but in MNI space
- `*T1w_space-MNI152NLin2009cAsym_class-CSF_probtissue.nii.gz`
- `*T1w_space-MNI152NLin2009cAsym_class-GM_probtissue.nii.gz`
- `*T1w_space-MNI152NLin2009cAsym_class-WM_probtissue.nii.gz` Probability tissue maps, transformed into MNI space
- `*T1w_target-MNI152NLin2009cAsym_warp.h5` Composite (warp and affine) transform to transform t1w into MNI space

Derivatives related to EPI files are in the `func` subfolder:

- `*bold_space-T1w_brainmask.nii.gz` Brain mask for EPI files, calculated by nilearn on the average EPI volume, post-motion correction, in T1w space
- `*bold_space-MNI152NLin2009cAsym_brainmask.nii.gz` Same as above, but in MNI space
- `*bold_confounds.tsv` A tab-separated value file with one column per calculated confound and one row per timepoint/volume
- `*bold_space-T1w_preproc.nii.gz` Motion-corrected (using MCFLIRT for estimation and ANTs for interpolation) EPI file in T1w space
- `*bold_space-MNI152NLin2009cAsym_preproc.nii.gz` Same as above, but in MNI space

Surface preprocessing

`fmriprep` uses FreeSurfer to reconstruct surfaces from T1/T2-weighted structural images. If enabled, several steps in the `fmriprep` pipeline are added or replaced. All surface preprocessing may be disabled with the `--no-freesurfer` flag.

Reconstruction

If FreeSurfer reconstruction is performed, the reconstructed subject is placed in `<output dir>/freesurfer/sub-<subject_label>/` (see *FreeSurfer Derivatives*).

Surface reconstruction is performed in three phases. The first phase initializes the subject with T1- and T2-weighted (if available) structural images and performs basic reconstruction (`autorecon1`) with the exception of skull-stripping. For example, a subject with only one session with T1 and T2-weighted images would be processed by the following command:

```
$ recon-all -sd <output dir>/freesurfer -subjid sub-<subject_label> \
    -i <bids-root>/sub-<subject_label>/anat/sub-<subject_label>_T1w.nii.gz \
    -T2 <bids-root>/sub-<subject_label>/anat/sub-<subject_label>_T2w.nii.gz \
    -autorecon1 \
    -noskullstrip
```

The second phase imports the brainmask calculated in the *t1w_preprocessing* sub-workflow. The final phase resumes reconstruction, using the T2-weighted image to assist in finding the pial surface, if available:

```
$ recon-all -sd <output dir>/freesurfer -subjid sub-<subject_label> \
    -all -T2pial
```

Reconstructed white and pial surfaces are included in the report.

Fig. 6.6: Surface reconstruction (FreeSurfer)

If T1-weighted voxel sizes are less 1mm in all dimensions (rounding to nearest .1mm), submillimeter reconstruction is used.

In order to bypass reconstruction in `fmriprep`, place existing reconstructed subjects in `<output dir>/freesurfer` prior to the run. `fmriprep` will perform any missing `recon-all` steps, but will not perform any steps whose outputs already exist.

Boundary-based Registration (BBR)

The mean EPI image of each run is aligned to the reconstructed subject using the gray/white matter boundary (FreeSurfer's `?h.white` surfaces).

If FreeSurfer processing is disabled, FLIRT is performed with the BBR cost function, using the FAST segmentation to establish the gray/white matter boundary.

FreeSurfer Derivatives

A FreeSurfer subjects directory is created in `<output dir>/freesurfer`.

```
freesurfer/
    fsaverage/
        mri/
        surf/
        ...
    sub-<subject_label>/
        mri/
        surf/
        ...
    ...
```

A copy of the `fsaverage` subject distributed with the running version of FreeSurfer is copied into this subjects directory.

Contributing to FMRIPREP

This document explains how to prepare a new development environment and update an existing environment, as necessary.

Development in Docker is encouraged, for the sake of consistency and portability. By default, work should be built off of poldracklab/fmriprep:latest (see the installation guide for the basic procedure for running).

It will be assumed the developer has a working repository in `$HOME/projects/fmriprep`, and examples are also given for niworkflows and NIPYPE.

Patching working repositories

In order to test new code without rebuilding the Docker image, it is possible to mount working repositories as source directories within the container. In the docker container, the all Python packages are installed in `/usr/local/miniconda/lib/python3.6/site-packages`.

To patch in working repositories of FMRIPREP or its dependencies, for instance contained in `$HOME/projects/`, add the following arguments to your docker command:

```
-v $HOME/projects/fmriprep/fmriprep:/usr/local/miniconda/lib/python3.6/site-packages/
↪fmriprep:ro
-v $HOME/projects/niworkflows/niworkflows:/usr/local/miniconda/lib/python3.6/site-
↪packages/niworkflows:ro
-v $HOME/projects/nipype/nipype:/usr/local/miniconda/lib/python3.6/site-packages/
↪nipype:ro
```

For example,

```
$ docker run --rm -v $HOME/fullds005:/data:ro -v $HOME/dockerout:/out \
    -v $HOME/projects/fmriprep/fmriprep:/usr/local/miniconda/lib/python3.6/site-
↪packages/fmriprep:ro \
    poldracklab/fmriprep:latest /data /out/out participant \
    -w /out/work/
```

In order to work directly in the container, use `--entrypoint=bash`, and omit the fmriprep arguments:

```
$ docker run --rm -v $HOME/fullds005:/data:ro -v $HOME/dockerout:/out \
    -v $HOME/projects/fmriprep/fmriprep:/usr/local/miniconda/lib/python3.6/site-
↪packages/fmriprep:ro --entrypoint=bash \
    poldracklab/fmriprep:latest
```

Patching containers can be achieved in Singularity by using the PYTHONPATH variable:

```
$ PYTHONPATH="$HOME/projects/fmriprep" singularity run fmriprep.img \
     /scratch/dataset /scratch/out participant -w /out/work/
```

Adding dependencies

New dependencies to be inserted into the Docker image will either be Python or non-Python dependencies. Python dependencies may be added in three places, depending on whether the package is large or non-release versions are required. The image *must be rebuilt* after any dependency changes.

Python dependencies should generally be included in the `REQUIRES` list in fmriprep/info.py. If the latest version in PyPI is sufficient, then no further action is required.

For large Python dependencies where there will be a benefit to pre-compiled binaries, conda packages may also be added to the `conda install` line in the Dockerfile.

Finally, if a specific version of a repository needs to be pinned, edit the `requirements.txt` file. See the current file for examples.

Non-Python dependencies must also be installed in the Dockerfile, via a `RUN` command. For example, installing an `apt` package may be done as follows:

```
RUN apt-get update && \
    apt-get install -y <PACKAGE>
```

Rebuilding Docker image

If it is necessary to rebuild the Docker image, a local image named `fmriprep` may be built from within the working fmriprep repository, located in `~/projects/fmriprep`:

```
~/projects/fmriprep$ docker build -t fmriprep .
```

To work in this image, replace `poldracklab/fmriprep:latest` with `fmriprep` in any of the above commands.