MRI Preprocessing (Prisma)

Our current MRI preprocessing pipeline is handled by a combination of FreeSurfer and FSL tools, with Python wrappers.

If a subject has been run for the first time, we need to do anatomical processing as well as functional preprocessing assuming fMRI data were collected. For anatomical processing, see MRI Anatomical. For functional pre-processing, see below.

Software dependencies

- Scientific python environment (e.g. Anaconda)
- Nipype
- FSL (5.0.9 or above) (NOTE: HAVING THIS VERSION IS IMPORTANT, otherwise there will be an error in the header of "distortion_merged_corrected_mean.nii.gz" and misalign this with the EPI)
- OBSOLETE: Freesurfer
- The Python wrapper script from the MRI_tools GitHub repository, in the preprocessing directory
- Neuropythy (required only for to_freesurfer.py, also in the preprocessing directory of MRI_tools)

Note: If you install Nipype from pip (`pip install nipype`) you may get an obscure error similar to this:

```
AttributeError: 'DiGraph' object has no attribute 'in_edges_iter'
```

This can be fixed by uninstalling nipype ('pip uninstall nipype') then re-installing it from the github repository; download the github repository locally: `git clone https://github.com/nipy/nipype` then from inside the nipype directory tell pip to install from local source: `cd nipype && pip install .`

Note that this error was observed on 11-2-2017 and will probably only remain an issue until the nipype developers update their pipi package.

Note that neuropythy can be installed with pip:

```
> pip install -U neuropythy
```

You may have to sudo or pass the --user flag to pip as well. You need neuropythy version 0.4.2 or greater to use to_freesurfer.py.

Preprocessing overview

The preprocessing strategy follows several steps.

1. All volumes from all EPIs are realigned to one of the single-band reference images (typically from the first EPI scan).
2. The single band reference is registered to the distortion scan with the same phase encoding direction.
3. The two distortion scans with opposite phase encoding are used to estimate the susceptibility-induced warp field, which can be used to unwarp the EPIs.
4. The motion realignment, the registration to the distortion scan, and the unwarping are together applied in a single step to each volume of each EPI and saved out as a new set of EPI data sets, typically with the names `timeseries_corrected_runXX.nii`, etc.
5. The mean unwarped distortion scan is aligned to the high resolution whole-brain T1, which is often obtained in a prior session and should already be processed by Freesurfer's recon-all.
6. This registration to the whole brain anatomy is not applied, but saved out in a file called `distort2anat_tkreg.dat`.

Running the preprocessing on BIDS-formatted data

Once data have been reconstructed, moved to the lab server, and converted to BIDS (this can happen via Flywheel, using CBI's BIDS converter, or hacking together something with MRI_tools's prisma_to_BIDS.py, see here for an example use) you can run a pre-processing script from a terminal. Currently, we run a single Python function, prisma_preproc.py, from MRI_tools. Because your data is BIDS-formatted, many of the arguments required when run on prisma-formatted data are not required (and will, in fact, be ignored). We preprocess one session at a time.

See the help string for the script (run python prisma_preproc.py -h) for more details on all the arguments. There are many other optional arguments, which are detailed in the help string, but not below.
Running the preprocessing on prisma-formatted data

Once data have been reconstructed and moved to the lab server, you can run a pre-processing script from a terminal. Currently, we run a single Python function, `prisma_preproc.py`, from `MRI_tools`:

**USAGE:**

```bash
prisma_preproc.py [-h] -subject SUBJECT -datadir DATADIR -outdir OUTDIR -epis EPIS [EPIS...] -sbref SBREF -distortPE DISTORTPE -distortrevPE DISTORTREVPE [-PEdim PEDIM]
```

# Example

```bash
export FSLOUTPUTTYPE=NIFTI_GZ #See notes below
python prisma_preproc.py -subject wl_subj001 -datadir /Volumes/server/Projects/BAIR/MRI/data/wl_subj001_20170823/RAW/ -outdir /Volumes/server/Projects/BAIR/MRI/data/wl_subj001_20170823/preproc -epis 8 10 12 14 16 18 20 22 24 26 28 30 -sbref 7 -distortPE 6 -distortrevPE 5
```

For the example above, the RAW folder looks like this:

```bash
python prisma_preproc.py
-datadir /Volumes/server/Projects/BAIR/Data/BIDS/visual/sub-wlsubj051/ses-nyu3T01
-dir_structure bids
```

Other helpful arguments: `-working_dir` (location of your working directory), `-plugin` and `-plugin_args` (nipype plugin specifying how to run data, see later on this page for details on how to use), `-bids_derivative_name` (name of the directory within your BIDS derivatives folder to contain the output of this, default "preprocessed"), and `-bids_suffix` (filename suffix to specify that this data has been preprocessed, replaces "bold", default "preproc").
Notes:

- **subject** must be the subject ID found in the FreeSurfer subjects directory.
- **datadir** is the folder where the raw data reside.
- **outdir** is where the preprocessed data will go.
- **epis** is a list of integers specifying which folders in RAW have epi data. In the example above, the EPIs are even numbers starting at 8. The odd numbers are the single band reference scans from the EPIs and not the full EPI runs.
- **sbref** is an integer indicating which single band reference scan should be used as the base for motion co-registration of the EPIs. Typically, it will be the sbref for the first EPI acquired, in the example above, 7.
- **distorPE** is an integer indicating which scan was the distortion scan with the same phase encode direction as the EPIs. In the example, the EPI phase-encode direction is PA, so the distortion scan with PA is 6.
- **distortrevPE** is an integer indicating which scan was the distortion scan with the opposite phase encode direction as the EPIs. In the example, this would be scan 5, which was AP.
- **PEdim** is the phase encoding dimension for the EPIs (x, y, or z). This argument is optional and is set to a default of y (anterior-posterior axis) in accordance with our current sequence.
- The line "export FSLOUTPUTTYPE=NIFTI_GZ" is only necessary if you haven't already configured nii.gz to be the default output format for FSL; some versions of FSL default to NIFTI instead of NIFTI_GZ, and this causes problems with parts of the pipeline.
- In general, obscure errors that appear to come from random parts of the pipeline failing are probably due to insufficient FSL configuration: if the FSL initialization doesn't get run, then many errors will likely occur during the pipeline. To fix this, make sure that you have configured FSL correctly: [https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation/ShellSetup](https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation/ShellSetup)

### Running preprocessing on the NYU HPC Cluster

The preprocessing pipeline can be run on the HPC cluster as well. To do so, make sure you copy over your data (both functional data to process as well as the relevant Freesurfer subject directory) from Acadia using `scp`, install nipype into your user directory (something like `pip install --user -e git://github.com/nipy/nipype.git#egg=nipype` should do the trick), and then submit a script like the following to the job scheduler. To understand how to do so, review the NYU HPC wiki, especially the pages on Prince and Slurm. If you copy the following into a script called `preproc_submit.sbatch`, then typing `sbatch preproc_submit.sbatch` should submit it. You can then observe its status by checking `squeue -u username`.

NOTE: I've occasionally had issues with Nipype and the SLURM job scheduler, where it manages to lose track of the job id and so fails (for me, this happens during the `merge_epis` step). In this case, switching the `-plugin` option to `Linear` should fix it. This will then take a very long time (3 hours instead of 30 minutes) because it's not taking advantage of parallel processing at all, but it should run without any problems.
#!/bin/bash
# See NYU HPC documentation for more details on these options
#SBATCH --job-name=preproc
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=10GB
#SBATCH --time=12:00:00

# This makes sure we have anaconda, fsl, and freesurfer available.
# YOU WILL NEED TO INSTALL NIPYPE ON YOUR OWN, it's currently not included
# in any of the HPC modules.
module purge
module load anaconda2/4.3.1
module load fsl/5.0.10
module load freesurfer/6.0.0

# so that freesurfer can find the relevant data.
export SUBJECTS_DIR=~/Freesurfer_subjects

cd ~/MRI_tools/preprocessing

python prisma_preproc.py
-subject wl_subj042
-datadir ~/spatial_frequency_pilot/raw/wl_subj042/20170823_prisma_pilot
-outdir ~/spatial_frequency_pilot/preproc/wl_subj042/pilot
-epis 8 10 12 14 16 18 20 22 24
-sbref 7
-distortPE 6
-distortrevPE 5
-working_dir /scratch/user/preproc
-plugin SLURM

It is also possible to use the MultiProc plugin on the cluster, but you need to specify a couple of plugin arguments to prevent nipype from trying to take over all the memory on the cluster nodes and get killed by the job manager. We do this by using the plugin_args argument for prisma_preproc.py, like the following. The key is that you must let both the job scheduler (through the #SBATCH options cpus-per-task and mem at the top of the file) and the nipype plugin (through the plugin_args argument) know how many cpus and how much memory you want the job to take. See the nipype documentation for more details on the possible plugin arguments and note that they must be formatted a very specific way on the command line: each should be a key:value pair (NO spaces) and each should be separated by a comma (and NO spaces).
#!/bin/bash
# See NYU HPC documentation for more details on these options
#SBATCH --job-name=preproc
#SBATCH --nodes=1
#SBATCH --cpus-per-task=10
#SBATCH --mem=48GB
#SBATCH --time=12:00:00

# This makes sure we have anaconda, fsl, and freesurfer available.
# YOU WILL NEED TO INSTALL NIPYPE ON YOUR OWN, it's currently not included
# in any of the HPC modules.
module purge
module load anaconda2/4.3.1
module load fsl/5.0.10
module load freesurfer/6.0.0

# so that freesurfer can find the relevant data.
export SUBJECTS_DIR=~/Freesurfer_subjects

cd ~/MRI_tools/preprocessing

python prisma_preproc.py
-subject wl_subj042
-datadir ~/spatial_frequency_pilot/raw/wl_subj042/20170823_prisma_pilot
-outdir ~/spatial_frequency_pilot/preproc/wl_subj042/pilot
-epis 8 10 12 14 16 18 20 22 24
-sbref 7
-distortPE 6
-distortrevPE 5
-working_dir /scratch/user/preproc
-plugin MultiProc
-plugin_args n_procs:10,memory_gb:48

Aligning volumes to FreeSurfer anatomy and Surface Interpolation (to_freesurfer.py)

The script to_freesurfer.py can be used to prepare your timeseries data for analysis with FreeSurfer. The script does two things:

1. Overwrites the header data of your timeseries_corrected_runXX.nii.gz files such that their affine transforms align them with the FreeSurfer native volume orientation of your subject.
2. Interpolates the timeseries onto the subject's cortical surface and exports these as a series of lh.timeseries_corrected_runXX.mgz and rh.timeseries_corrected_runXX.mgz files, each of which contains an N x 1 x 1 x T 'volume' where N is the number of vertices in the hemisphere and T is the number of TRs in your volume file.

To perform these operations, run the script from your subject’s Preproc directory (containing the timeseries_corrected_runXX.nii.gz files):

> to_freesurfer.py -v -s ./distort2anat_tkreg.dat ./timeseries_corrected_*

If you wish to overwrite the headers but skip the surface-data export, you may omit the -s flag.

Notes

- This **overwrites** the nifti file headers, based on the current headers. Therefore you should **not** run this twice on the same directory. If you wish to save the original orientations, then you should make backup copies of the nifti files before running the command.
- The surface interpolation performed is linear by default, but may be changed to nearest or heaviest with options (see to_freesurfer.py -h).
- The surface interpolation uses the midgray surface, but this may also be changed via the options.
- MGZ files are exported due to a potential problem storing nifti files with large dimensions (I don’t understand this problem, but python throws a warning when writing out large-dimensions in nifti files). If you want to convert these into nifti files, you can use mri_convert (mri_convert in_file.mgz out_file.nii.gz).