FreeSurfer Multiprocessing Pipeline

Contents

Module `freesurfer_wrapper`

freesurfer_wrapper

freesurfer_wrapper aims to facilitate the creation of a multiprocessing pipeline using FreeSurfer. It is a Python wrapper to execute parallel runs of cross, base and long recon-all. Some pial edits algorithms are also available.

Requirements

- Docker[1]
- FreeSurfer license key[2]
- Ubuntu OS[3]: instructions are given, and were tested, considering an Ubuntu OS. It is possible to run using other OS, like Windows, since the wrapper uses Docker. However, keep in mind that adaptations may be necessary.

Files and folders overview

```
ADNI  # ADNI test data
    ...
docs # documentation files
    ...
FS_OUTPUTS # output folder for processing (freesurfer SUBJECTS_DIR).

QC # output folder for quality control analysis

scripts # additional scripts used by the wrapper
    ...
run.py # wrapper main script
```

Preparation

1) Place the license in a license.txt file in the same folder as the Dockerfile.

2) Place your dataset folder in the same folder as the Dockerfile.

3) Build the docker image:

[1] https://www.docker.com/
[2] https://surfer.nmr.mgh.harvard.edu/registration.html
[3] https://ubuntu.com/desktop

```
sudo docker build -t fs_wrapper .
```

Workflow overview

In this section, a usage example is shown using data from the Alzheimer's Disease Neuroimaging Initiative[4] dataset. The ADNI folder contains MP-RAGE data from 3 visits of subject 137_S_1414.

recon-all [CROSS] processing

Cross-sectionally process all time points with the default workflow.

Input file

CROSS processing requires a tab separated file with named columns:

- Mandatory columns:
  - id: unique id.
  - dcm_path: path to one dcm/nii file.
- Additional columns (**required only in case of BASE and LONG processing**):
  - subject: subject base ID
  - session: session ID
  - date: folder named with scan date
  - visit: time point relative to the ones contained in the subject folder.

For the ADNI dataset example, you can create this file using `create_recon_input.py all -i <PATH_TO_SAMPLES_FOLDER>`. This will create a recon_all_input.txt file. The script will combine the subject ID and the session ID to create the unique ID.

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 scripts/create_reco
```

recon_all_input.txt example:

```
id  subject session date    visit   dcm_path
137_S_1414_I64472   137_S_1414  I64472  2007-08-01_10_14_02.0   1   ADNI/137_S_1414/MP-RAGE/2007-08-
137_S_1414_I153787  137_S_1414  I153787 2009-08-26_11_06_33.0   2   ADNI/137_S_1414/MP-RAGE/2009-08-
137_S_1414_I190917  137_S_1414  I190917 2010-08-18_14_20_16.0   3   ADNI/137_S_1414/MP-RAGE/2010-08-
```

Run recon-all [CROSS]

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 run.py recon_all -i
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

recon-all [BASE] processing

Create an unbiased template from all time points for each subject.

Input file

BASE processing requires a single column file were each line must be a command string with the following template:

```
recon-all -base <subject> -tp <unique_id> -tp <unique_id> ... -all
```

For the ADNI dataset example, you can create this file using `create_recon_input.py base -i recon_all_input.txt`. This will use the recon_all_input.txt file created previously for the CROSS processing. If necessary, edit this recon_all_input.txt to contain **only the samples that have been successfully processed**.

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 scripts/create_reco
```

recon_base_input.txt example:

```
recon-all -base 137_S_1414 -tp 137_S_1414_I64472 -tp 137_S_1414_I153787 -tp 137_S_1414_I190917 -all
```

[4] https://adni.loni.usc.edu/

Run recon-all [BASE]

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 run.py recon_base -
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

recon-all [LONG] processing

Longitudinally process all timepoints.

Input file

LONG processing requires a single column file were each line must be a command string with the following template:

```
recon-all -long <unique_id> <subject> -all
```

For the ADNI dataset example, you can create this file using `create_recon_input.py long -i recon_all_input.txt`. This will use the recon_all_input.txt file created previously for the CROSS processing. If necessary, edit this recon_all_input.txt to contain **only the samples that have been successfully processed**.

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 scripts/create_reco
```

recon_long_input.txt example:

```
recon-all -long 137_S_1414_I64472 137_S_1414 -all
recon-all -long 137_S_1414_I153787 137_S_1414 -all
recon-all -long 137_S_1414_I190917 137_S_1414 -all
```

Run recon-all [LONG]

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 run.py recon_long -
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

Segmentation of hippocampal subfields and nuclei of the amygdala [CROSS] processing

Original script by Juan Eugenio Iglesias. For more information and citation requirements, please consult FS official documentation[5].

Input file

CROSS processing requires a tab separated file with named columns:

- Mandatory columns:
  - id: unique id.

For the ADNI dataset example, you can use the recon_all_input.txt file created previously for the recon-all CROSS processing. If necessary, edit this recon_all_input.txt to contain **only the samples that have been successfully processed**.

Run segment_HA [CROSS]

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 run.py segment_HA -
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

Segmentation of hippocampal subfields and nuclei of the amygdala [LONG] processing

Original script by Juan Eugenio Iglesias. For more information and citation requirements, please consult FS official documentation[6].

[5] https://surfer.nmr.mgh.harvard.edu/fswiki/HippocampalSubfieldsAndNucleiOfAmygdala
[6] https://surfer.nmr.mgh.harvard.edu/fswiki/HippocampalSubfieldsAndNucleiOfAmygdala

Input file

LONG processing requires a tab separated file with named columns:

- Mandatory columns:
    - subject: base ID from subject processed with recon-all [BASE]

For the ADNI dataset example, you can use the recon_all_input.txt file created previously for the recon-all CROSS processing. If necessary, edit this recon_all_input.txt to contain **only the samples that have been successfully processed**.

Run segment_HA [LONG]

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 run.py segment_HA_l
```

This command will use the maximum number of CPUs. You can append the -p <INT> flag where <INT> is the number of parallel runs you want.

Tissue ratio correction

After running recon_all you can check your results using freeview. Please refer to Manual quality analysis section to use a custom script.

If any skull edits are necessary, you need to create an input table to run edit with the following characteristics:

- Mandatory columns:
    - id: unique id.
    - ratio: the threshold to value (%) of WM intensity. The value should be >0 and <1; larger values would correspond to cleaner skull-strip but higher chance of brain erosion.

You can copy the recon_all_input.txt content, keep only the lines for the scans that need edits, and add the column with the tissue ratio values.

As an example, an edit_input.txt file for some ADNI records would look like this:

```
id  subject session date    visit   dcm_path    ratio
137_S_1414_I64472   137_S_1414  I64472  2007-08-01_10_14_02.0   1   ADNI/137_S_1414/MP-RAGE/2007-08-
137_S_1414_I153787  137_S_1414  I153787 2009-08-26_11_06_33.0   2   ADNI/137_S_1414/MP-RAGE/2009-08-
```

Run edit

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper \
python3 run.py edit -i edit_input.txt
```

This command will use the maximum number of CPUs. You can append the -p <INT> flag where <INT> is the number of parallel runs you want.

You can check the resulting edited mask using freeview:

```
SUBJECTS_DIR=$(pwd)/FS_OUTPUTS
freeview -recon <UNIQUE_ID> -v brainmask.gcutsT$<TISSUE_RATIO>.mgz:colormap=heat:opacity=0.5
```

If still not good, change the tissue ratio value in the input file and run edit again. When all masks are OK, proceed to recon_edit command.

recon_edit

recon_edit will re-run parts of FS recon-all using the edited masks. The input file is the table used for edit with the final values for tissue ratio.

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 run.py recon_edit -
```

This command will use the maximum number of CPUs. You can append the -p <INT> flag where <INT> is the number of parallel runs you want.

How to check for completed runs and hard recon-all errors

FreeSurfer's recon-all command creates different logs while running. The `recon-all.done` log is created only for completed runs. The `recon-all.error` is created for hard failures.

You can check these logs using a custom script. The script was written to work on ADNI folder structure. For other datasets you can try to edit the PATH_PATTERN variable in scripts/check_logs.py.

Done

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper \
python3 scripts/check_logs.py done
```

You can also pipe the output to bash word count command to get a quick count:

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper \
python3 scripts/check_logs.py done | wc -l
```

Error

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper \
python3 scripts/check_logs.py error
```

How to restart after a computer failure

If the execution of the pipeline is halted by a computer failure or system restart then you have to update the input file of the recon commands.

To update recon_all_input.txt:

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 scripts/update_reco
```

To update recon_base_input.txt:

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 scripts/update_reco
```

To update recon_long_input.txt:

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper python3 scripts/update_reco
```

This will remove all "done" samples from the original input and create a new input file (`<YYYY-MM-DD>_recon_<all|base|long>_in`

To delete the folders from the samples that were running when the failure happened:

```
# first check the list
IsRunning=(ls FS_OUTPUTS/*/scripts/*IsRunning* | cut -f 1,2 -d /)
echo $IsRunning

# if it is ok, delete
sudo rm -R $IsRunning
```

Quality control

Automated quality analysis

The tool is packaged with **qatools-python**[7] version 1.2 for quality control measurements. This script was developed by Reuter DeepMI Lab[8] as a revision, extension, and translation to the Python language of the Freesurfer QA Tools.

```
sudo docker run --rm -it -v "$(pwd):/root/freesurfer_wrapper" fs_wrapper \
python3 scripts/qatools-python/qatools.py --subjects_dir FS_OUTPUTS --output_dir QC \
--screenshots --outlier --fornix
```

This will create `qatools-results.csv` file; screenshots, outliers and fornix folders inside the QC folder. Please consult qatools-python docs[9] for a full explanation of each QC measurement.

[7] https://github.com/Deep-MI/qatools-python
[8] https://deep-mi.org/
[9] scripts/qatools-python/README.md#description

Manual quality analysis

You can visually inspect each result using freeview. We provide a script to speed up the opening process of each scan. The script also prompts the user about the result of the QC after each window of freeview is closed. The result is saved to manual_QC.txt

It is not possible to run freeview using Docker, the graphical user interface cannot be displayed. Therefore, you need to have FreeSurfer/freeview installed in your host machine.

Step 1

Set SUBJECTS_DIR environment variable. Here, the results are stored inside the FS_OUTPUTS directory.

```
export SUBJECTS_DIR=$(pwd)/FS_OUTPUTS
```

Step 2

Run the view.py script.

```
python3 scripts/view.py
```

Sub-modules

- freesurfer_wrapper.run
- freesurfer_wrapper.scripts

Module `freesurfer_wrapper.run`

Command-line wrapper tool to execute parallel runs of FreeSurfer recon-all and some pial edits algorithms.

This file can also be imported as a module and contains the following functions:

```
* argument_parser -  parser for command-line options, arguments and sub-commands.
* run_command - select and run the commands
* worker - invokes a subprocess running the command.
```

Functions

Function `argument_parser`

```
def argument_parser(
    args: list
) -> ArgumentParser.parse_args
```

Parser for command-line options, arguments and sub-commands.

Parameters

`args` : list Command-line arguments list

Returns

Parser

Function `run_command`

```
def run_command(
    args: list
)
```

Pass the appropriate command function to the worker.

Parameters

`args` : list Command-line arguments list

Returns

None

Function `worker`

```
def worker(
    cmd: str
) -> <function run at 0x7ff2132785e0>
```

Invokes a subprocess running the command.

Parameters

`cmd` **: str** Command-line string

Returns

subprocess.run()

Namespace `freesurfer_wrapper.scripts`

Sub-modules

- freesurfer_wrapper.scripts.check_logs
- freesurfer_wrapper.scripts.create_recon_input
- freesurfer_wrapper.scripts.update_recon_input
- freesurfer_wrapper.scripts.view

Module `freesurfer_wrapper.scripts.check_logs`

Script to check recon-all logs for each run

usage: python check_logs.py <done|error>

Please edit the **PATH_PATTERN** variable with the appropriate pathname pattern to find each file.

This file can also be imported as a module and contains the following functions:

```
* get_logs - get the path for each log based on pathname pattern.
* print_id_from_logs - prints the IDs from a list of logs.
```

Functions

Function `get_logs`

```
def get_logs(
    path_pattern: str
) -> list
```

Get the path for each log based on pathname pattern.

Parameters

`path_pattern` **: str** Glob pathname pattern to find each log.

Returns

List of log paths

Function `print_id_from_logs`

```
def print_id_from_logs(
    logs: list
)
```

Prints the IDs from a list of logs.

Parameters

`logs` **: list** List of log paths

Returns

None

Module `freesurfer_wrapper.scripts.create_recon_input`

Script to create recon input tables

This file can also be imported as a module and contains the following functions:

```
* create_recon_all_input - creates the recon-all input table.
* create_recon_base_input - creates the recon-all base input table.
* create_recon_long_input - creates the recon-all base input table.
```

Functions

Function `argument_parser`

```
def argument_parser(
    args: list
) -> ArgumentParser.parse_args
```

Parser for command-line options, arguments and sub-commands.

Parameters

`args` **: list** Command-line arguments list

Returns

Parser

Function `create_recon_all_input`

```
def create_recon_all_input(
    base_dir: str
)
```

Creates a 6 column text file to be used as input for the main script recon-all command. First column: unique ID (combines SUBJECT ID and SESSION ID). Second column: SUBJECT ID. Third column: SESSION ID. Fourth column: Scan date. Fifth column: Time point relative to the ones contained in the subject folder. Sixth column: path to DICOM file.

Parameters

`base_dir` **: str** Path to directory containing the samples.

Returns

None

Function `create_recon_base_input`

```
def create_recon_base_input(
    recon_input: str
)
```

Creates a single column text file to be used as input for the main script recon-all base command. Each line is a complete command to execute.

Parameters

`recon_input` **: str** Path to recon_all_input.txt file used for cross processing.

Returns

None

Function `create_recon_long_input`

```
def create_recon_long_input(
    recon_input: str
)
```

Creates a single column text file to be used as input for the main script recon-all long command. Each line is a complete command to execute.

Parameters

`recon_input` : str Path to recon_all_input.txt file used for cross processing.

Returns

None

Module `freesurfer_wrapper.scripts.update_recon_input`

Script to create a new input file after a PC failure (power, restart, ...). This will remove all "done" samples from the original input. It will also delete the folders from the samples that were running when the failure happened. These samples will run again from scratch.

This file can also be imported as a module and contains the following functions:

```
* update_recon_all_input - updates the recon-all input table.
* update_recon_base_long_input - updates the recon-all base or long input table.
```

Functions

Function `argument_parser`

```
def argument_parser(
    args: list
) -> ArgumentParser.parse_args
```

Parser for command-line options, arguments and sub-commands.

Parameters

`args` : list Command-line arguments list

Returns

Parser

Function `update_recon_all_input`

```
def update_recon_all_input(
    recon_input: str
)
```

Updates the recon_all_input.txt file to remove successfully processed data.

Parameters

`recon_input` : str Path to recon_all_input.txt file used for cross processing.

Returns

None

Function `update_recon_base_long_input`

```
def update_recon_base_long_input(
    recon_input: str,
    long=False
)
```

Updates the recon_base_input.txt or recon_long_input.txt file to remove successfully processed data.

Parameters

`recon_input` : str Path to recon_base_input.txt or recon_long_input.txt file used for base or long processing.

Returns

None

Module `freesurfer_wrapper.scripts.view`

Script to open the scans in sequence and register if passed or not in QC analysis

This script identifies the subjects present in the folder set in SUBJECTS_DIR environment variable. Using this list, it automatically opens freeview and asks the user input for the result of QC analysis. Results are saved in manual_QC.txt file.

This file can also be imported as a module and contains the following functions:

```
* get_subjects - returns a list of subjects inside the folder.
* freeview - returns a freeview command formated as string
```

Functions

Function `freeview`

```
def freeview(
    subject_id: str
)
```

Returns a freeview command formated as string.

Parameters

`subject_id` : str FreeSurfer subject_id.

Returns

str

Function `get_subjects`

```
def get_subjects(
    subjects_dir='/subjects'
)
```

Returns a list of subjects inside the folder.

Parameters

`subjects_dir` : str, default=`os.environ['SUBJECTS_DIR']` FreeSurfer SUBJECTS_DIR.

Returns

list

Generated by *pdoc* 0.10.0 (https://pdoc3.github.io).