Atlas-based Imaging Data Analysis pipeline for functional and structural MRI Data

AIDAmri

v1.3

Code: Leon Scharwächter, Niklas Pallast, Michael Diedenhofen,
Victor Vera Frazão, Markus Aswendt
Manual: Leon Scharwächter, Niklas Pallast, Tim Fleiner,
Victor Vera Frazão, Markus Aswendt

Status: October 2023

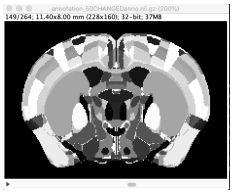
Department of Neurology
University Hospital Cologne

Contents

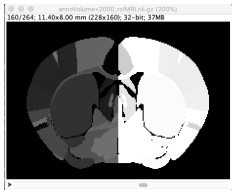
1 Introduction

The Atlas-based Processing Pipeline for functional and structural MRI data (AIDAmri) was developed for automated processing of mouse brain MRI. AIDAmri works with T2-weighted MRI (T2w), diffusion weighted MRI or diffusion tensor imaging (DTI) and resting-state functional MRI (fMRI). The Allen Mouse Brain Reference Atlas (ARA, CCF v3) is registered on each of these MRI data sets and is used to analyse regions of interest. Furthermore, the regions of the ARA are used as seed-points for the connectivity and activity matrices. The lib User-defined ROIs and masks can be generated separately and used for analysis, e.g. stroke lesion masks and peri-infarct regions. AIDAmri comes with different atlas and template versions, which are

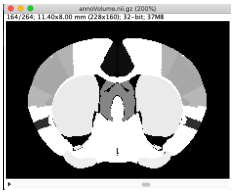
necessary for the registration to work (Figure 1): a) annotation 50 changed anno: original ARA CCF v3 labels (regions with grey values $> 100,000$ changed to new values starting with 2000), b) anno volume 2000 rsfMRI: atlas from a) with reduced number of atlas regions by selective region fusion, 96 regions in total, split between hemispheres (right side +2000), c) same as in b) but no split, d) same as in a) but split, e) original ARA template with 50 um isotropic resolution, f) custom-made MRI template. For the complete list of atlas labels see: `annoVolume+2000_rsfMRI.nii.txt`



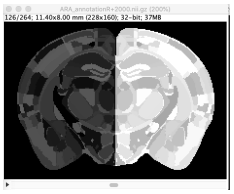
(a) Annotation 50 Changed Anno



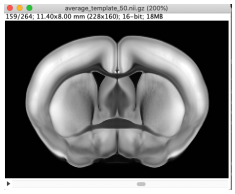
(b) Anno Volume 2000 rsfMRI



(c) Anno Volume



(d) ARA Annotation R2000



(e) Average Template 50

(f) NP Template sc0

Figure 1: Atlases included in the `/lib` *folder.*

2 Installation

AIDAmri is distributed as a Docker image. If you prefer the built-in installation instead, see the legacy installation section. We highly recommend the usage of the Docker image.

2.1 Prerequisites

The following things are required to launch your AIDAmri instance.

- Docker engine (click **here**)
  - **Getting started tutorial for Docker**
- (*Windows only*) Bash subsystem, e.g. **Git BASH**
- at least 10.6 GB free disk memory

We advise you to get comfortable with Shell/command line interface (CLI) usage. The Ubuntu webpage provides tutorials such as this. Download or clone our repository:

```
git clone https://github.com/aswendtlab/AIDAmri.git
```

2.2 Docker usage

This guide introduces the usage of Docker-based containers of the AIDAmri tools for Unix/Linux-based systems (i.e. Linux and MacOS). Hence, the commands shown are written for as such and you may copy the commands into your shell including backslashes as they indicate line breaks. As mentioned in the previous section, Windows user may use a subsystem like Git BASH to use the software. You may also use the Docker Desktop application to get access to our Docker image.

2.3 General overview

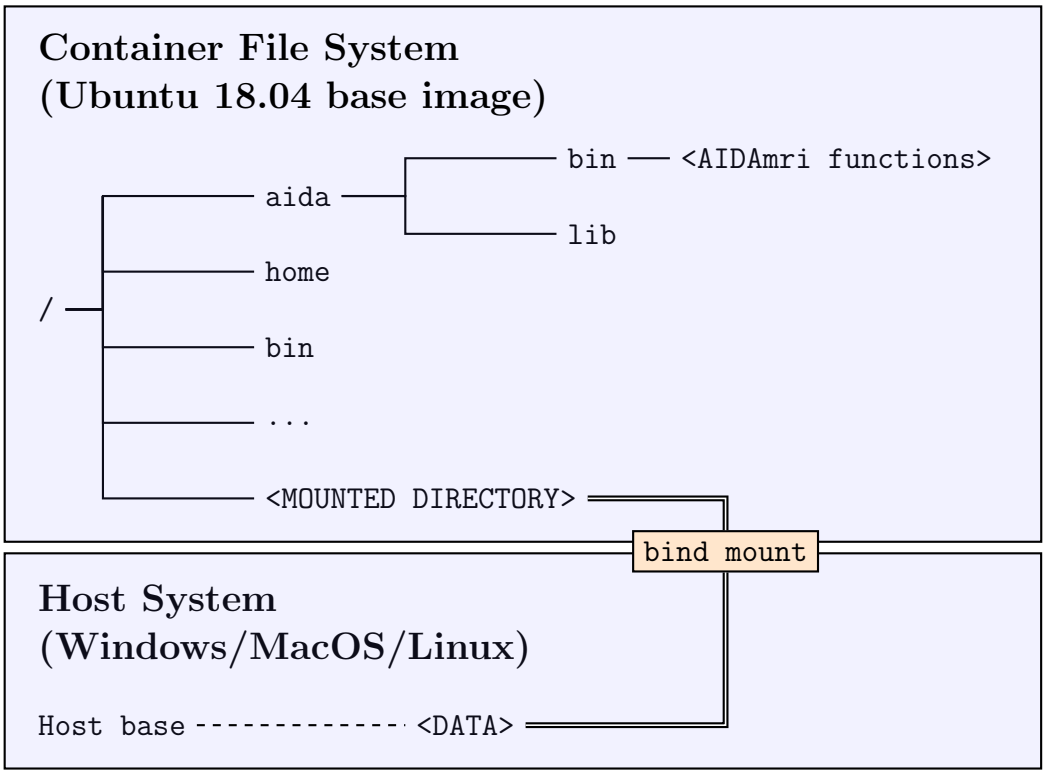


Figure 2: General structure of container file structure. The paths are constructed from left to right (e.g. the absolute path of the bin folder in aida would be "/aida/bin". Keep in mind that "/" is its own directory. The <MOUNTED DIRECTORY> and <DATA> directories are the same but can be named differently, depending on how it was named when mounted.

The AIDAmri pipeline is containerized and structured as depicted in figure 3. The Dockerfile located in our repository provides the installation routine for every required dependency. the `docker build` command constructs the image, i.e. the installed software on your system. The `docker run` command creates an runnable instance of this image, called a container. The container provides an interface (CLI). It can be accessed by directly attaching to an interactive interface that lets you input the AIDAmri commands within the isolated file system of the container or via the `docker exec` command from

your host shell. How to build and use the pipeline is explained in the following sections.

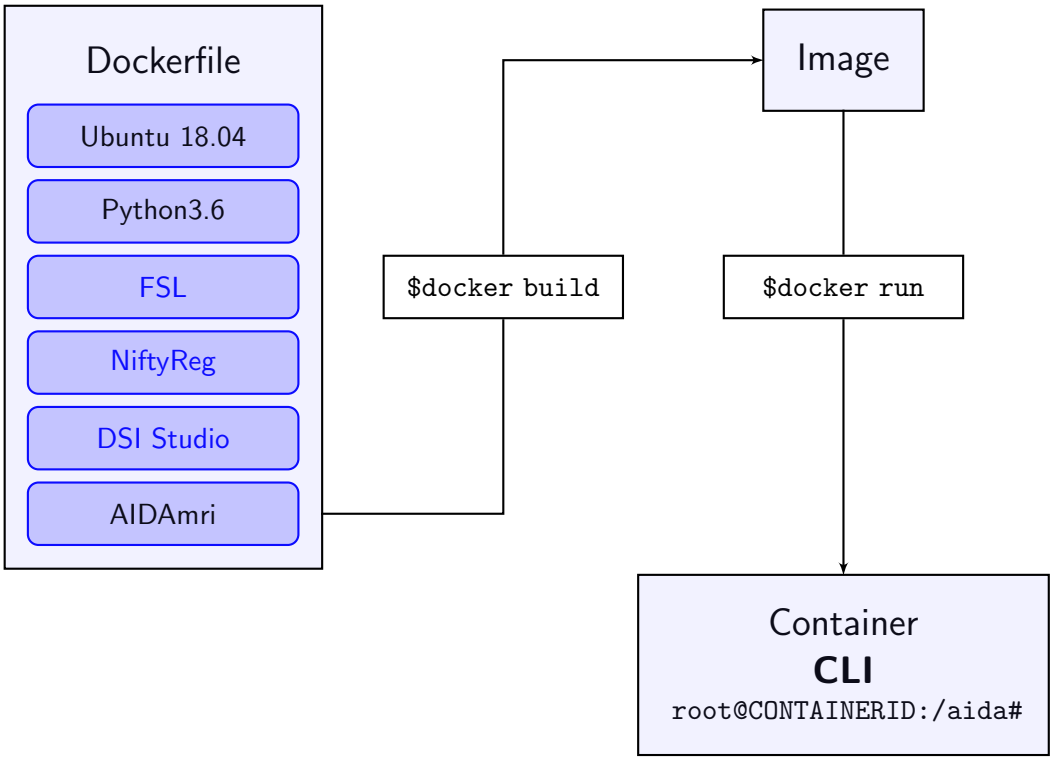


Figure 3: Docker architecture draft. The blue boxes within the Dockerfile box depict the main content layers. The boxes preceded with an $ are command line codes. Click on texts within the boxes for further information.

The container follows a basic Ubuntu 18.04 system (see figure 4), meaning that the root directory is called `"/"`. To share a volume between the host system and the container, we commonly use the bind mount function (see section 2.5 for further information). Please note that the mounted volume is assumed to be located in the root directory. When referring to it while in the container, one needs to refer to it by using the path given at mounting, in this case `/<MOUNTED DIRECTORY>`. This path likely will be different on your host system.

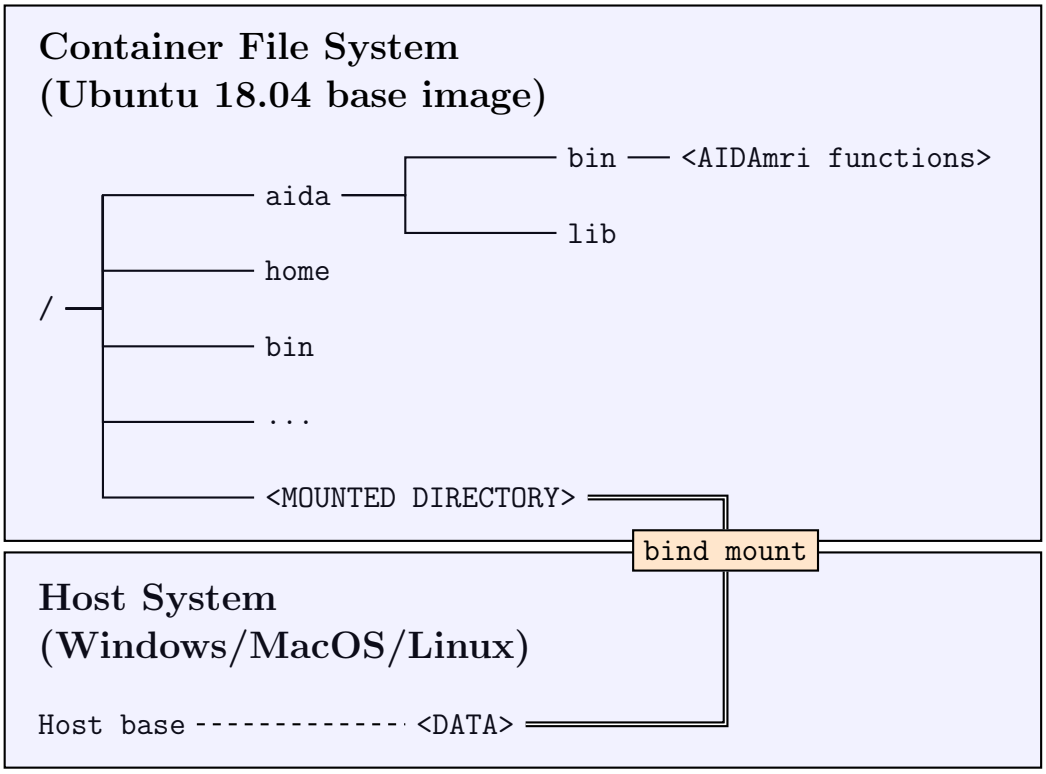


Figure 4: General structure of container file structure. The paths are constructed from left to right (e.g. the absolute path of the bin folder in aida would be "/aida/bin". Keep in mind that "/" is its own directory. The <MOUNTED DIRECTORY> and <DATA> directories are the same but can be named differently, depending on how it was named when mounted.

2.4 Creating image

To initiate the image building process, open your shell to grant yourself access to the command line interface (CLI). Change your directory to the cloned GitHub repository.

```
cd PATH/TO/AIDAmri
```

You can check the folder contents by using the `ls` command. A file named Dockerfile, as well as the `fslinstaller_mod.py`, a `bin/` and a `lib/` folder should be located in this directory. Then, launch the docker daemon to build the image:

```
docker build -t aidamri:latest -f Dockerfile .
```

The created image is a template for running so-called containers (see the below section) and instantiating the pipeline. Be aware that the period at the end is part of the command and refers to the corresponding directory. The `-t` flag calls for the name and the tag that you wish to give your image, in this case it is called 'aidamri' and tagged 'latest'. You may change name and tag to your liking, but keep in mind to change them accordingly in later steps that invoke the image. The `-f` tag calls for the Dockerfile in the current directory. Let the process run. Docker desktop should show you the image within the image tap once it is built.
You only need to run the building process once initially or after updating the github after new changes were made. If an update occurred, the building process will only update those layers that were changed, so the process will not take as long as the initial build.

2.5 Running container and mount data

Before running a container, make sure you know where exactly your data you wish to process is located on your host system. The container will be an instance written from the built image and serves as an environment for you to use the AIDAmri pipeline. To run the container, enter the following command in your command line:

```
docker run -dit \
--name aidamri \
--mount type=bind,source=PATH/TO/DATA,target=/aida/DATA \
aidamri:latest
```

The first flag, `-dit` starts the container as a detached mode, hence the `d`, as well as an interactive mode, that allows you to actively use this environment, hence the `it`. Alternatively, you can directly enter the container environment by only using the `-it` flag. With the `--name` tag you give your container a name, in this case it is called "aidamri". Note, that the image name and the container are named individually. It is recommended to give your container a different name from the image to avoid confusion. Note further that if you wish to use more than one instance of running containers (e.g. to process multiple datasets simultaneously) you need to name each container differently.
Binding mounts create a reference to a given directory, allowing the container

to access and transmute data on the host system. In this case, it should allow the pipeline to process your MRI data without copying or re-allocating your data. The `--mount` flag with the `type=bind` argument allows you to grant your container access to your target directory. Type in the absolute path at the `source` placeholder. Do not use relative paths. Within the container environment, a working directory called `aida/` was created. It is recommended to allocate your reference path within this directory by typing in the name of your data folder according to the target placeholder, i.e. `/aida/DATA`. You can give the target folder another name than the source directory as it still will refer to the source but it is advised to keep the name to avoid confusion. Be aware not to place any spaces between the commas. Pass the name and tag of your image, i.e. here `aidamri:latest` at the end of the command line. After initializing an ID will appear and the Docker Desktop app should show the running container. You can also check whether the container is running by typing `docker container ls`.
To enter the running container, use the following command:

```
docker attach aidamri
```

Windows user You can leave a running container without stopping by pressing `CTRL+P` and `CTRL+Q` consecutively. Alternatively, you can type in `exit`, however, this will stop the container. Use

```
docker start aidamri
```

to re-run the container. Type in "stop" instead of "start" if you like to stop an already running container.
When successfully attached, the shell prompt will look like the following:

```
root@<SOME NUMBERS AND CHARACTERS>:/aida#
```

The number shows the first part of the container ID. From here, you can use the AIDAmri commands as explained in the former usage sections. As a start, change your directory to the binary folder (`cd bin/`) and run `python batchProg.py -h` to see the helping instructions of the batch processing tool and check functionality. Alternatively, you can use the `docker exec` command. Basically, this pipes a command that you wish to run within the container to the running container without directly entering its interactive environment. As an example, to run batchProg help page, type in the following command.

```
docker exec -w /aida/bin aidamri \
python batchProg.py -h
```

The `-w` flag acts as the directory change (`cd`) as you input the directory where batchProg is located (/aida/bin). Afterwards, the command is entered like in the interactive shell.

2.6 Built-in installation

AIDAmri can be installed without the Docker usage at your host machine. Be advised that the installation process is tedious and error prone due to the different dependencies and their installation processes.

1. Download the folders `/bin` and `/lib` by using this link.
   `/bin` and `/lib` should be located in the same directory. Alternatively, clone the repository:

   ```
   git clone https://github.com/Aswendt-lab/AIDAmri
   ```

   The folder `/bin` contains the python scripts necessary for all preprocessing steps. The folder `/lib` contains several information about six atlases. Please do not conduct any changes to these files as it could influence the access from the python scripts.

2. Download & Install the appropriate DSI-Studio and copy the install path into `.../bin/3.2_DTIConnectivity/dsi_studioPath.txt`.

   - On Ubuntu: if an error is raised that states that libQt6Charts.so.6 could not be found, execute the command

     ```
     sudo apt install libqt6charts6-dev
     ```

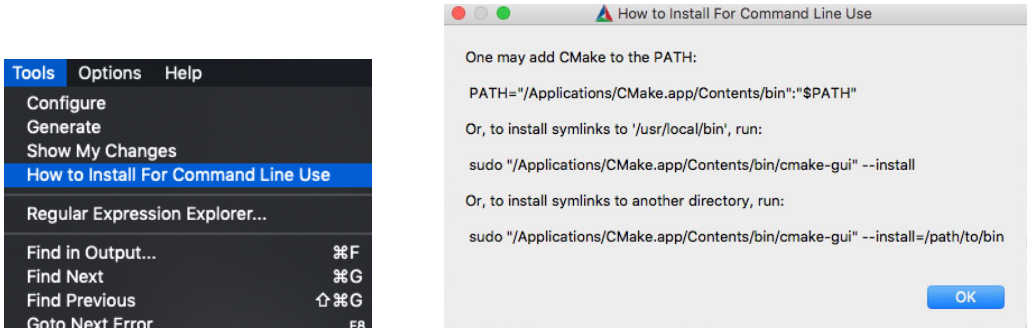
3. Download FSL installer 3.3.0 (for further information on FSL, click here. To install FSL use the following command:

   ```
   python3 fslinstaller.py -V 5.0.11
   ```

4. Download and the latest version of Cmake.

   - Unpack the downloaded archive and switch to the directory using `cd` command.
   - Type in the following command to install Cmake:

     ```
     ./bootstrap && make && sudo make install
     ```



(a) Menu of Cmake software *(b) Install for command line use*

Figure 5: Installation instructions to install Cmake.

5. Download & Install Python 3.6 or higher using Anaconda and enter the following command to install necessary packages

```
pip install nipype==1.1.2 lmfit==0.9.11 progressbar2==3.38.0 \
nibabel shutil
```

Anaconda will tell you if additional packages are necessary. We recommend to install AIDAmri in a separate Anaconda environment (see the related documentation). Alternatively, use the `requirements.txt` from the AIDAmri git repository to install the dependencies. Open the terminal at the folder where the requirement file is located and launch the installation.

```
pip install --upgrade pip && pip install -r requirements.txt
```

6. Install NiftyReg by conducting the following steps:

   a. Generate your source folder `.../NiftyReg/niftyreg_source`.

   b. Download NiftyReg from the git by replacing `<path>` by your personal path and enter the following command:

```
git clone \
git://git.code.sf.net/p/niftyreg/git \
<path>/niftyreg_source
```

   c. Change the folder by typing in the terminal:

```
cd <path>/niftyreg_source
```

d. Type in the following command:

```
git reset --hard 83d8d1182ed4c227ce4764f1fdab3b1797eecd8d
```

e. Follow the steps described here.

3 Functions

List of functions:

- **1_PV2NIfTiConverter**: Bruker to NIfTy converter
- **2.1_T2PreProcessing:** T2w MRI pre-processing including brain extraction, bias field correction and atlas registration
- **2.2_DTIPreprocessing:** DTI pre-processing including brain extraction, bias field correction and atlas registration
- **2.3_fMRIPreProcessing**: DTI pre-processing including brain extraction, bias field correction and atlas registration
- **3.1_T2Processing**: Stroke mask calculations across all subjects per group (Incidence mapping), SNR calculations
- **3.2_DTIConnectivity**: Whole brain fiber tracking using DSI Studio and calculation of diffusion measures (FA, AD, RD, MD) for every brain region
- **3.2.1_DTIdata_extract**: Creates a .txt-file containing DTI values from all brain regions
- **3.3_fMRIActivity**: functional connectivity analysis for all atlas regions
- **4.1_ROI_analysis**: Analysis of T2w, DTI, and rs-fMRI with user-defined atlas regions, e.g.: peri-infarct regions around the stroke lesion
- **5.1 Tools/Create_Groupmapping.py**: Creates a list of subjects within a group. User needs to specify which subject belongs to which group. Needs to be started manually in a terminal.

All program examples are only listed with the mandatory input parameters. For more details/help, call python `.../python <command> -h`. The command line examples are given with the identifier `testData<No.>` `.nii.gz` and can be identically applied to other data. The test data is freely available: https://doi.org/10.12751/g-node.70e11f.

After a successful download, you can choose to either process single files manually or automate the processing for the whole dataset. In both cases, processing includes file conversion from the raw Bruker format into the NIfTI format, several preprocessing steps and registration with the Allen Brain Reference Atlas. The functions in `/bin` are named according to the MRI sequence to be processed (T2, DTI, fMRI) and labeled with 1-4.1 in the order of processing: for example `3.1_T2Processing` requires `1_PV2nIfTiConverter` and `2.1_T2PreProcessing` to be executed sequentially.

4 Batch processing

To process the whole project folder at once, only two scripts are necessary: At first `conv2Nifti_auto.py` creates a new project folder, converts all files to the NIfTI format and stores them in the new project folder, `batchProc.py` then applies pre-processing steps and the registration with the atlas. Remember that our test data set already is converted into the NIfTI format. In this case only the second script needs to be applied. In general, the raw (Bruker format) data needs to be in the following structure for the first script to work: projectfolder/days/subjects/

To convert the whole project folder into the NIfTI format, open the terminal and change the directory to the `/bin` folder of the AIDAmri installation using

```
cd <path to AIDAmri>/bin
```

Enter a similar line as in the example below to start the the first script (Bruker to NIfTI conversion). For the first script to work, the table `groupMapping.csv` within `/bin` needs to be adjusted beforehand: specify the group name of every subject within the project folder (e.g. using Excel). This is necessary for the script to properly generate the new processed project folder structure. Remember that the algorithm does not read the first row of the table containing

the titles (Subject, Group). You can use the *Create_GroupMapping.py* script which is located in the tools folder to do so. It takes a string as input parameter. This string should be contained in every foldername of the datasets you want to process. After the creation of the csv file you need to adjust the group column manually for aidamri to work.

Example:

```
python conv2Nifti_auto.py -f /path/to/raw_dataset \
-d Baseline P1 P7 P14 -g Group1 Group2
```

As you can see, this script computes the conversion either for all data in the raw project folder or for certain days and/or groups specified through the optional arguments -d and -g. During the conversion, a new folder called proc_data is being created in the same directory where the raw data folder is located.

After a successful Bruker to NIfTI conversion, the second script can be applied to the new project folder proc_data.

Example:

```
python batchProc.py -f /path/to/proc_data -g Group1 Group2 \
-d Baseline P1 P7 P14 -t T2w DTI fMRI -stc False
```

This script runs every necessary script for all (pre-)processing and registration steps. The data needs to be ordered like after the Bruker2NIfTI conversion: projectfolder/days/groups/subjects/data/. Again, you can specify which days -d and groups -g to compute and which dataformat -t (T2w, DTI, fMRI) to process. Furthermore you can specify if a slice time correction should be performed on your data. Per default the -stc flag is set to False and it is optional to set this parameter. Please keep in mind that the hereby executed scripts are related to each other and therefore T2w always needs to be specified before DTI.

Dependent on the size of your project, this process may take a while. After finishing, your project folder is ready for network graph analysis, e.g. using AIDAconnect.

5 Processing single files step-by-step

5.1 Convert raw data

Convert Bruker raw data to NIfTI files by specifying the folder containing all raw folders of each scan. A file with exactly the same name is created in the given input folder. It contains all sorted NIfTI files. The raw data should have the same orientation as the example dataset.

```
python pv_conv2Nifti.py -i .../testData
```

Remember to move the new generated file to a new project folder, if you want to separate the raw Bruker files with the processed NIfTI files. We recommend to build a folder structure as follows: projectfolder/days/groups/subjects/data/ especially if you want to use AIDAconnect for graph analysis.

5.2 Processing of T2w & T2mapping data

Apply the reorientation, bias field correction and brain extraction to the T2w data set. The automatically attached endings of the processed filenames indicate which steps have been performed. Brain extraction should be of good quality and must be manually checked or corrected by adapting the default parameter.

```
python preProcessing_T2.py -i .../testData/T2w/testData.5.1.nii.gz
```

The next step includes the registration of the Allen Brain Reference Atlas with the brain extracted T2 dataset. It is necessary to check the registration result, e.g. by superimposing the brain extracted file with the atlas annotations (ends with `..._Anno.nii.gz`). There is the option to segment an additional region of interest, e.g. the stroke lesion. You can segment the region using the brain extracted dataset as reference (ends with `...BET.nii.gz`). We recommend to conduct this step with itk-SNAP. The saved file should end with the extension `...Stroke_mask.nii.gz`.

```
python registration_T2.py -i .../testData/T2w/testDataBiasBet.nii
```

To improve the registration: try to optimize the brain extraction and generated mask (e.g. manually using ImageJ). Then run the registration again.

If you previously defined a region of interest (e.g. the stroke lesion), it is possible to calculate the region size, segmented (parental) atlas regions. Here, the segmented region `.../Stroke_mask.nii.gz` is overplayed with the Allen Brain Reference Atlas and saved in the file `...Anno_mask.nii.gz`. Use the path to the `.../T2w` as input.

```
python getIncidenceSize_par.py -i ... /testData/T2w
python getIncidenceSize.py -i ... /testData/T2w
```

The results, such as affected regions and ROI volume are stored in the folder `.../T2w` in the following files:

```
affectedRegions.txt
affectedRegions.nii.gz
affectedRegions_Parental.txt
affectedRegions_Parental.nii.gz
```

5.3 Processing of T2 data

From the masks drawn on the T2-weighted images, it is possible to determine both the incidence map and the size of affective regions. For example, if a `day1` folder contains multiple `Mouse_1-Mouse_15` folders and the processed T2 data is in those folders, the command would be as follows:

```
python getIncidenceMap.py -i .../day1 -s "Mouse*"
```

It is also possible to determine the Region size as Voxels and Volume(mm^3):

```
python getRegionSize_par.py -i .../T2w
```

5.4 Processing of DTI data

The DTI processing procedure includes a dimension reduction, bias correction, a threshold application, and the subsequent brain extraction. The endings on the filenames indicate which steps have been performed.

```
python preProcessing_DTI.py -i .../DTI/testData.7.1.nii.gz
```

The next step includes the registration of the Allen Brain Reference Atlas with the brain extracted DTI dataset. For processing a reference (stroke) mask, two options are available: a) Registration of a reference mask that is related to another dataset/day, e.g. to always use the same mask, append command `-r <filename of ref>` b) else, the algorithm will automatically use the corresponding reference mask from the respective subject folder. If no mask is defined, the registration will proceed without.

```
python registration_DTI.py -i .../DTI/testDataSmoothMicoBet.nii.gz
```

The connectivity is finally calculated using DSI-Studio. All connectivity matrices are based on the reference atlas.

```
python dsi_main.py -i .../DTI/testData.7.1.nii.gz
```

The connectivity matrices of the parental ARA, the original ARA and the related ROI are stored in the folder `.../DTI/connectivity` as `.txt` and `.mat`. DSI-Studio differentiates between matrices that count how many fibers pass and end in each region. The adjacency matrices can be visualised using the related plot function:

```
python plotDTI_mat.py -i .../testData/DTI/connectivity/testData* \
.connectivity.mat
```

The folder `.../DTI/DSI_studio` also contains the diffusion value maps (e.g. FA map) registered with the atlas. This data can be extracted and saved as `.txt` with the region name and the corresponding FA/RD/MD/AD using the function:

```
python DTIdata_extract.py image_file roi_file
```

in the 3.2.1_DTIdata_extract folder. To iteratively process all subjects use the `iterativeRun.py` function.

5.5 Processing of fMRI data

The fMRI processing is roughly comparable to the preprocessing of the DTI datasets. Brain extraction should be of good quality and must be manually checked or corrected by adapting the given parameters.

```
preProcessing_fMRI.py -i .../fMRI/testData.6.1.nii.gz
```

The step includes the registration of the Allen Brain Reference Atlas with the brain extracted fMRI dataset. The result is a variety of files. An impression of the registration can be obtained by superimposing the brain extracted file with the annotations of the Allen Brain (ends with `..._Anno.nii.gz`)

```
python registration_fMRI.py -i .../testData/fMRI/testSmoothBet.nii
```

If physiological data are not available, the step will be conducted without the included regression. All activity matrices are based on the reference atlas.

```
python process_fMRI -i .../fMRI/testData.6.1.nii.gz
```

The activity matrices of the parental Atlas (original Atlas) are stored in the folder `.../fMRI/regr` as `.txt` and `.mat` with the prefix `MasksTCs.` and `MasksTCsSplit.`. The related adjacency matrices can be visualised using the related plot function:

```
python plotfMRI_mat.py -i .../testData/fMRI/regr/MasksTCsSplit*.mat
```

5.6 Peri-infarct ROI analysis

You can create custom peri-infarct masks to further analyze stroke related regions. Go to the folder `bin/4.1_ROI_analysis` and open the file `proc_tools.py` with an arbitrary editor that can open python files. Adjust all directories, paths and further specifications as described in the script. To decide which regions to include in the peri-infarct region, modify:

`cortex_labels_1.txt` and `cortex_labels_2.txt`

(for the full list of atlas labels, see `../lib/annoVolume+2000_rsfMRI.nii.txt`)

Proceed with the scripts in the order 1 to 4. The first script creates peri-infarct masks for all time points:

```
python 01_dilate_mask_process.py
```

The second script aligns the peri-infarct masks in the rsfMRI and DTI space:

```
python 02_apply_xfm_process.py
```

The result of the third script depends on the imaging type: In case of rsfMRI, a MATLAB file which contains two text files is being created. 1) for each region one column with the averaged rsfMRI time series and 2) the atlas labels names. In case of DTI, a modified atlas labels file which includes individually shaped peri-infarct brain regions is being created. These new generated regions replace the original regions in the file:

```
python 03_create_seed_rois_process.py
```

The fourth script is not mandatory, but a helper tool to compare the number of voxels included in the peri-infarct region for each subject.

```
python 04_examine_rois.py
```

Attention: The scripts for peri-infarct ROI analysis are provided for analysis of time point 7 only (e.g. 7 days post stroke). For other time points, manual modifications are necessary.