Laminar EEG-fMRI pipeline

Documentation

Tommy Clausner

last updated on July 1, 2018

Contents

List of Figures

List of Tables

1 About this Document

This documententation accompanies the scripts that I wrote for my MSc thesis on feature specific neuronal oscillations on a cortical laminar level under the supervision of:

- Mathilde Bonnefond
- Rene Scheeringa

Hence all the functionality is tied to our specific experiment. However, most steps still can serve as a template for laminar EEG-fMRI analyses in general.
Our experiment consisted of 3 major parts: retinotopic field mapping for visual ROI definition, odd ball task employing gabbor patches (main task) and additional support scans as there are: T1 anatomical and functional scans with inversed flip angle. During the main task EEG was recorded in a 3 s gap between blocks of 4 TR (4 functional volumes). This led to a signal drop off over each of the four fMRI volumes, that was absent during retinotopy scans. We exploited the fact that the difference between the 4th and 1st volume of such a block yields a T1-like contrast, to optimise co-registration performance.
To enable myself to write all the scripts I got help from various people, so further thanks goes to:

- Jose Marques
- Koen Haak
- Matthias Ekman
- Simon Homölle
- Tim van Mourik
- TG guys at the DCCN

The goal was to create a set of scripts such that it would be highly standardized and with as little human interaction as possible. This way it is guaranteed, that once the analysis and respective parameters are set up, the scripts will run similar for all subjects. However it also means that changing single scripts within this "ecosystem" should be done very carefully to not interfere with relative paths and file I/O all functions rely on.

2 Before starting

Most of the functionality is optimized for the IT infrastructure provided by the Donders Center for Cognitive Neuroimaging. Since some parts of the analysis require very high amounts of memory, those parts are absolutely infeasible to compute on a local desktop or laptop machine. Given the infrastructure of the DCCN, all scripts were optimized to a reasonable amount for a reduction in computation time but still keeping high spatial

accuracy. Hence parts might need to be adjusted to either a different cluster computing system or scaled down massively.
All scripts as we used them can be found at `https://github.com/TommyClausner/laminarfMRI` or by cloning the GitHub repository via the command line:

```
# download scripts
git clone https://github.com/TommyClausner/laminarfMRI
```

2.1 Pre- requisites

The present pipeline makes use of many modern state of the art analysis toolboxes for Shell and or MATLAB scripting:

- analysePRF (`http://kendrickkay.net/analyzePRF/`)
- FieldTrip (`https://github.com/fieldtrip/fieldtrip`)
- FreeSurfer (`https://surfer.nmr.mgh.harvard.edu/`)
- FSL (`https://fsl.fmrib.ox.ac.uk/fsl/fslwiki`)
- knkutils (`https://github.com/kendrickkay/knkutils/`)
- MATLAB R2015a or later (`https://nl.mathworks.com/products/matlab.html`)
- MRIcron (`https://www.nitrc.org/projects/mricron`)
- Open fMRI Analysis (`https://github.com/TimVanMourik/OpenFmriAnalysis`)
- Pipeline (`https://github.com/Washington-University/Pipeline`)
- SPM (`http://www.fil.ion.ucl.ac.uk/spm/`)
- tc_functions (`https://github.com/TommyClausner/tc_functions`)
- Tommy's Scriptinator 3000 TM (optional; `https://github.com/TommyClausner/Scriptinator`)
- VistaSoft (`https://github.com/vistalab/vistasoft`)
- Workbench (`https://github.com/Washington-University/Workbench`)

Use e.g. MRIcron's dicom2nii to convert raw MRI data to NIfTI files.

Call the function `getToolboxes.sh` to download (most of) the software automatically into a folder called toolboxes/ located in the folder from where the function was called (see Section 6.1 and Figure 1)

```
# download toolboxes
sh getToolboxes.sh
```

2.2 Folder structure

The general folder structure is set up by a call to setupfolders.sh (see also Figure 1). Raw data should be placed in the corresponding subfolder:

- Functional data must be located in rawData/niftis/**functionals**/
- Anatomical data must be located in in rawData/niftis/**t1**/
- Inverted functional data must be located in in rawData/niftis/**inverted**/
- Proton density data must be located in in rawData/niftis/**pd**/
- Inverted proton density data must be located in in rawData/niftis/**pdinverted**/
- EEG data must be located in rawData/**eegfiles**/
- Eye tracking data must be located in rawData/**eyetrackerfiles**/
- EEG sensor position related files must be in rawData/**electrodes**/
- electromagnetic digitizer data must be in rawData/electrodes/**polhemus**/
- photogrammetry data must be in rawData/electrodes/**photogrammetry**/ (note also the respective sub-structure, set up by setupfolders.sh)

2.3 How things work

2.3.1 Bash scripts

Parts of the analysis run in Bash scripts. Those should be called from their root directory using:

```
# run scripts
sh scriptname.sh
```

Each step within the analysis has a different subfolder containing the respective results of this step. Leading numbers indicate in which logical order the corresponding shell scripts should be executed. Note that all files created by each respective step are stored in the corresponding folder. However files that need to be preset (e.g. config files) must be located in A_helperfiles. Note Further, that the entire analysis can be broken down to a few sub-pipelines.

2.3.2 pipelines

Since most operations can be grouped into bigger chunks or "pipelines", all subsections below are wrapped into pipeline scripts, calling them in the respective order. Addtionally a file having the extension .pipe stores the pipeline information in a format readable with Tommy's Scriptinator 3000 TM. This is a java based tool, that can be used to represent and store scripts and their dependencies including a graphical representation. See Section 9.2 of this document of `https://github.com/TommyClausner/Scriptinator` for more information.

2.3.3 MATLAB wrapper

Note, that most EEG related scripts rely on MATLAB and FieldTrip. Those scripts can be run stand-alone inside MATLAB or using the bash-script wrapper that has the exact same name only with the file extension `.sh` instead of `.m` Furthermore some MATLAB scipts require significant computation time, for which reason they are set up to be run as parts of a whole. In this case the words *split* is added to the file name. Those scripts call the corresponding shell scripts that call the MATLAB scripts in turn, in a organized manner to be run on the cluster.

When using such a MATLAB wrapper some MATLAB language specific details and variables are added to the base-file (`targetScript.m`). In every case the respective absolute folder path is provided. The shell script creates a temporary file and run it. It will be constructed by adding variables to an empty file and afterwards adding the actual base-file like so:

```
# guts of MATLAB wrapper
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
cd $DIR
nameadd=$(date +"%m%d%Y%H%M%S")
echo "mainpath=" "'$DIR';">$DIR/tmp_$nameadd.m
cat $DIR/targetScript.m>>$DIR/tmp_$nameadd.m
echo 'matlab2017b -nosplash -r "run('"'"$DIR/
   tmp_$nameadd.m"'"');"'  | qsub -q $jobtype -l
   walltime=$walltime,mem=$memory
```

2.3.4 a note on EEG

Files processing EEG data will have the prefix `do_EEG` and have a in itself closed workflow. This is mostly due to the fact, that scripts merely run in MATLAB at some point. To keep the data somewhat structured, a new datafile is created after each major processing step, adding a new prefix to the file name. E.g. after the time-frequency analysis `TF_` will be added to the current file name making `my_data.mat` becoming `TF_my_data.mat`

Within each function it can be defined, which is the prefix of the data to load and which is the prefix that will be added after results were computed. The default will be stated in the respective detailed explaination in Section 6. However, after running `do_EEGfreqChanSelect.m` all in that file defined prefixes that are found in the EEG folder will be deleted, keeping only essential files like headmodel, sourcemodel, sensors and the results file.

3 Helper files

There are several helper files that are needed in order to perform the analysis:

3.1 acquisition_parameters.txt

indicating the respective acquisition parameters per volume needed in order to perform the distortion correction. One line indicates the respective parameter setting for the respective volume (1st line corresponds to 1st volume, etc.)

e.g.:
0 1 0 0.042
0 -1 0 0.042

The first 3 columns indicate the respective phase coding direction the last column some weird value, that is only important if it changes. Otherwise it's fine to use any value as long as it is the same.

3.2 b02b0.cnf

Settings for distortion correction.

Needed in order to set the parameters for the distortion correction (example provided by FSL). For more information see `https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/topup/TopupUsersGuide/`.

3.3 params.mat

Parameters used for retinotopy scans as obtained from VistaDisp. For more information see `https://web.stanford.edu/group/vista/cgi-bin/wiki/index.php/Stimulus`.

3.4 images.mat

Stimuli used for retinotopy scans. Stimuli file must be such that each frame is represented as a binary image. The matrix must be of shape $Y \times X \times T$ where X and Y are the image dimensions in pixel and T is the respective number of frames.

4 qsub

Scripts that are flagged with the (Q) can be submitted as a qsub job.
When running the computation on the cluster use the wrapper function runonqsub.sh like so

```
# run scripts on cluster using wrapper function
sh runonqsub.sh 32gb targetScript.sh
```

Note that some scripts send a MATLAB script to the qsub cluster. Thus they cannot be run using `runonqsub.sh`, but instead create their own job. However those scripts can also run stand-alone within a interactive MATLAB session.

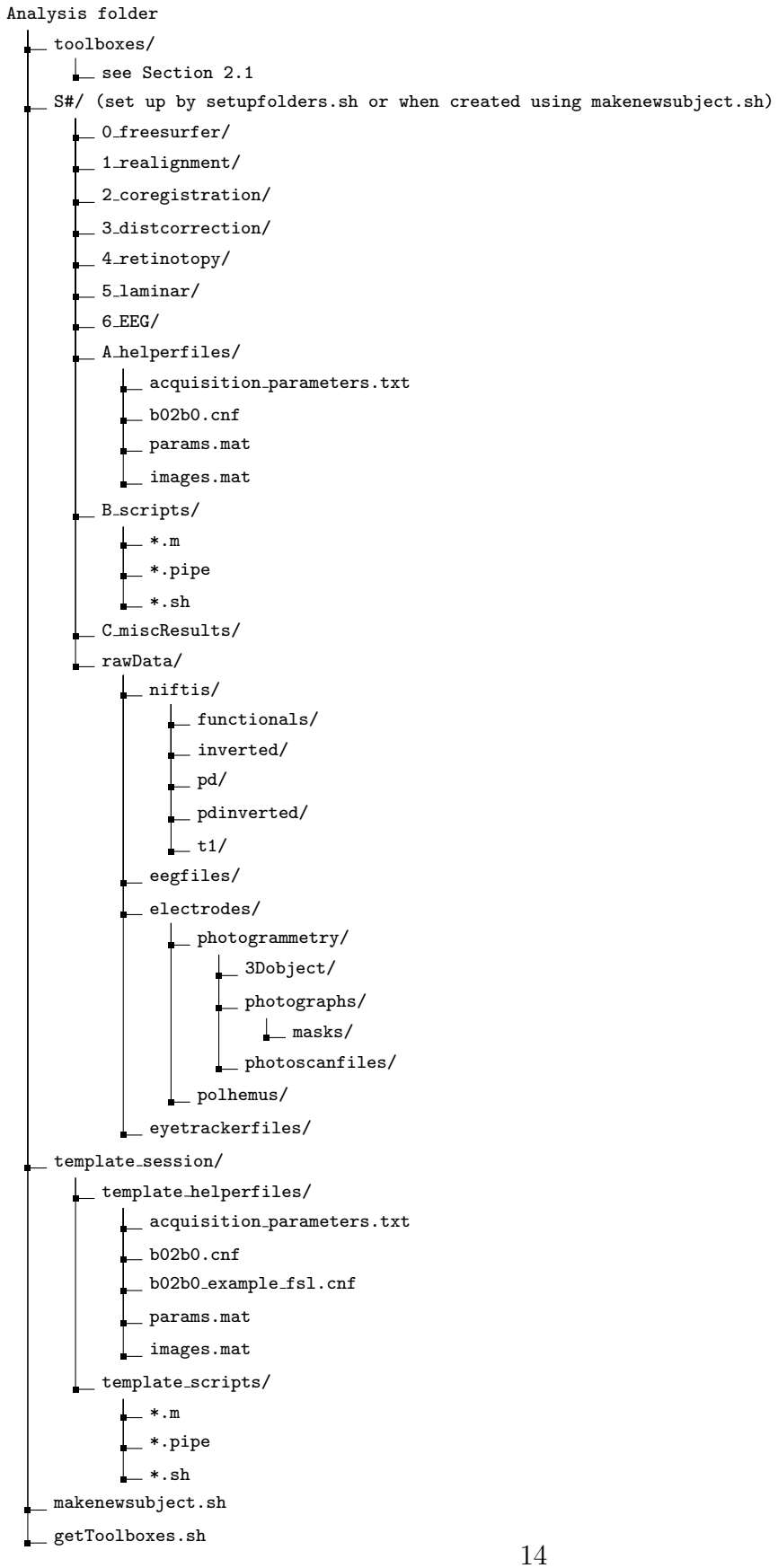
Some scripts are present as `scriptnameQsub.sh` Those scripts contain an internal Qsub-wrapper and do the same as sh `runonqsub.sh XXgb scriptname.sh` Those scripts are created when using Pipelines (see Section 9).

Table 1 gives an overview about requirements on memory and computation time can be expected:

script	memory required	time required
do_preparefunctionals.sh	32GB	$\approx 10min$
do_realignment.sh	32GB	$\approx 30min$
do_distcorr.sh	32GB	$\approx 15min$
do_applysimpledistcorr.sh	32GB	$\approx 30min$
do_preparecoregistration.sh	16GB	$\approx 4.5h$
do_correctavgdiff.sh	4GB	$\approx 3sec$
do_fsrecon.sh	16GB	$\approx 6h$
do_coregistration.sh	16GB	$\approx 30min$
do_makemasksandlabels.sh	16GB	$\approx 1min$
do_tseriesinterpolation.sh	64GB	$\approx 20min$
do_split_analyzePRF.sh	480GB	$\approx 7h$
do_analyzePRF.sh	6GB	$\approx 7h$
combine_split_PRF_results.sh	64GB	$\approx 10min$
make_PRF_overlays.sh	16GB	$\approx 10min$
makeOverlays.sh	16GB	$\approx 5min$
GUI2ROIs.sh	8GB	user dependent
labels2masks.sh	8GB	$\approx 10s$
expandROIs.sh	16GB	$\approx 3min$
do_getLayers.sh	16GB	$\approx 20min$
do_getLayerWeights.sh	16GB	$\approx 5min$
do_EEGpreprocessing.sh	16GB	$\approx 30min$
do_EEGprepareFS.sh	8GB	$\approx XXXmin$
do_EEGprepareHeadmodel.sh	32GB	$\approx XXXmin$
do_EEGelectrodeRegistration.sh	16GB	user dependent
do_EEGprepareSourcemodel.sh	32GB	$\approx XXXmin$
do_EEGtimelock.sh	16GB	$\approx 5min$
do_EEGsplitBeamformer.sh	480GB	$\approx 5h$
do_EEGbeamformer.sh	60GB	$\approx 5h$
do_EEGsplitVirtualChannel.sh	240GB	$\approx 5h$
do_EEGvirtualChannel.sh	30GB	$\approx 5h$
do_EEGsplitFreqOnVirtChan.sh	960GB	$\approx 5h$
do_EEGfreqOnVirtChan.sh	60GB	$\approx 5h$
do_EEGfreqChanSelect.sh	256GB	$\approx 30min$

Table 1: Approximated time and memory requirements for the respective script to run. Note that everything that is more than 4GB should be run on the cluster. Use runonqsub.sh or the respective MATLAB wrapper for that purpose.

Figure 1: Initial folder structure as required for the analysis



5 Quick Start

This document builds upon the analysis rational used during our own analyses. It is build upon the technically facilities of the DCCN (Radboud University Nijmegen, 2018). Hence a copy-paste workflow will only work on a very similar infrastructure.

5.1 first steps

maximum memory required	approx time required
32GB	$\approx 10min + user$

Since you are reading this document, it is unlikely that you haven't done already, but the first step would be to download the required scripts by cloning or downloading the GitHub repository:

```
# download scripts and toolboxes
cd myPath
git clone https://github.com/TommyClausner/laminarfMRI
cd laminarfMRI/quickStart
sh getToolboxes.sh
```

Figure 1 depicts an overview of how the analysis folder should look like. Individual subject folders are created and have the correct sub-structure by calling:

```
# create new subject folder
sh makenewsubject.sh
```

This functions searches for folders starting with **S** (for subject) within it's root directory and will create a new subject $S0$ if none was found and otherwise increment the respective maximum value by 1. Hence if 21 subjects are already present, a call to `makenewsubject.sh` will create subject $S22$.
Once this is done the folder structure should already be in place and the recoreded data can be placed in the subject's rawData folder.
From this point on it's best to navigate to the B_scripts folder of the subject and run all scripts from there. The first to run would be

```
# prepare fMRI raw data
cd S<number>/B_scripts
sh runonqsub.sh 32gb do_preparefunctionals.sh
```

This functions cuts off the first 4 volumes of the recorded data if it has 4 volumes more than expected. If further volumes were recorded (more than pre-defined) they will also be cut at the end.

5.2 fMRI pre-processing

Data will be corrected for motion and field distortion. Motion and distortion correction will be done using the average task block volumes (ignoring retinotopy volumes) as reference. Field distortion will be estimated compared to the inverse volumes that were collected. Both, inverse volumes and as many "normal" volumes will be averaged. See Figure 2 bottom for a graphical depiction.

maximum memory required	approx time required
64GB	$\approx 6h$

Motion correction (realignment) will be performed using FSL's `mcflirt function` (`https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/MCFLIRT`). Before the data will be split into two parts: *retino* and *sparse*, based on their raw basename string. As a reference volume the average of all *sparse* volumes is used. Furthermore the inverted scans will be corrected using the same volume as well.

Results can be found in 1_realignment/

```
# realign fMRI raw data
sh runonqsub.sh 32gb do_realignment.sh
```

Field distortion is estimated based on the average inverse volume and an anverage volume over the same number of volumes from the not inverted data. This average is constructed by taking the mean over every 4th volume of as many blocks of 4, countaing backwards from the number of main task volumes.

$$b0 = \frac{\sum_{i=0}^{N_{inv}-1} V_{(N-4i)}}{N_{inv}} \tag{1}$$

, where V is the set of all volumes within the last task block, N_{inv} is the number of volumes recorded with reversed phase-encoding blips and N is the number of volumes in V. $b0_{inv}$ is the respective time series average of all N_{inv} inverted volumes. A graphical depiction can be found in Figure 2 bottom. Once those two averages are created, the actual field distortion estimation is performed using FSL's `topup` (`https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/topup`) and applies it to the motion corrected data.

Results can be found in 3_distcorrection/

```
# distortion correction for fMRI raw data
sh runonqsub.sh 32gb do_distcorr.sh
sh runonqsub.sh 64gb do_applysimpledistcorr.sh
```

After corrections for motion and field distortion, functional space was registered to anatomical space. In this case the signal drop off can be exploited to provide a T1-like contrast. To achieve this, the set of task volumes across all blocks is split into chunks of four volume. Within each of those chunks, the first volume is substracted from the fourth. The respective difference volumes are then averged and provide a single partial volume V_{ref} exposing a T1-like contrast.

$$V_{ref} = 4\frac{\sum_{m=1}^{\frac{N}{4}}(V_{4m} - V_{4m-3})}{N} \tag{2}$$

, where V is the set of all task volumes across all four blocks and N is the size of V. Figure 2 top illustrates the procedures of volume selection and averaging.

Results can be found in 2_coregistration/

```
# prepare co-registration for functional and anatomical
   MRI raw data
sh runonqsub.sh 16gb do_preparecoregistration.sh
sh do_correctavgdiff.sh
```

The full preprocessing can also be performed in a single step, by calling the pipeline script:

```
# using pipeline
sh preprocessing.sh
```

5.3 fMRI - anatomical co-registration and masks

fMRI data will be co-registered to the anatomical T1 scan based on the gray and white matter boundaries obtained from FreeSurfer. Furthermore partial and full brain masks will be created, including gray and white matter and brain. Co-registration exploits the fact, that due regular interuptions of the sequence in order to obtain a clean EEG signal, the amplitude drops off. In fact every 4 volumes the scanner stoped in our experiment and EEG data was collected. For the field to reach a steady state, the signal will always drop off similarly.
Computing the difference between every 4th and 1st volume and averaging, leads to a T1-like contrast, that was used to otimize functional to anatomical registration. A graphical depiction can be found in Figure 2 top.

Results can be found in 2_coregistration/

maximum memory required	approx time required
16GB	$\approx 6.5h$

The anatomical T1 weighted image will be transformed into various representations (volumetric, surfaces, etc) and partly segmented (gray / white matter, CSF, etc) using FreeSurfer (`https://surfer.nmr.mgh.harvard.edu/`)

Results can be found in 0_freesurfer/

```
# do anatomical segmentation and volume reconstruction
sh runonqsub.sh 16gb do_fsrecon.sh
```

Gray matter boundaries from the FreeSurfer segmentation will be used to register the partial fMRI volumes to the anatomical space. This is done using FreeSurfer's `bbregister` function. The toolbox OpenFmriAnalysis (`https://github.com/TimVanMourik/OpenFmriAnalysis`) is used to perform this step and output a registration movie to C_miscResults. Furthermore gray and white matter masks are transformed into the functional space.

Results can be found in 2_coregistration/

```
# coregistration and mask creation
sh do_coregistration.sh
sh runonqsub.sh 16gb do_makemasksandlabels.sh
```

The full coregistration and creation of functional masks can also be performed in a single step, by calling the pipeline script:

```
# using pipeline
sh CoregistrationAndAnatMasks.sh
```

5.4 pRF mapping

Population receptive field mapping. Estimates retinotopic mapping from screen to cortical locations. This is to be done in order to obtain boundaries for visual areas such as V1, V2 and V3. First the fMRI time series will be interpolated (cubic) to match the numeber of frames presented during the stimulation. Afterwards pRF mapping will be analyzed and paramters will be written to volume files (NIfTI) and surface files for use with FreeSurfer.

maximum memory required	approx time required
64GB (480GB)	$\approx 8h$

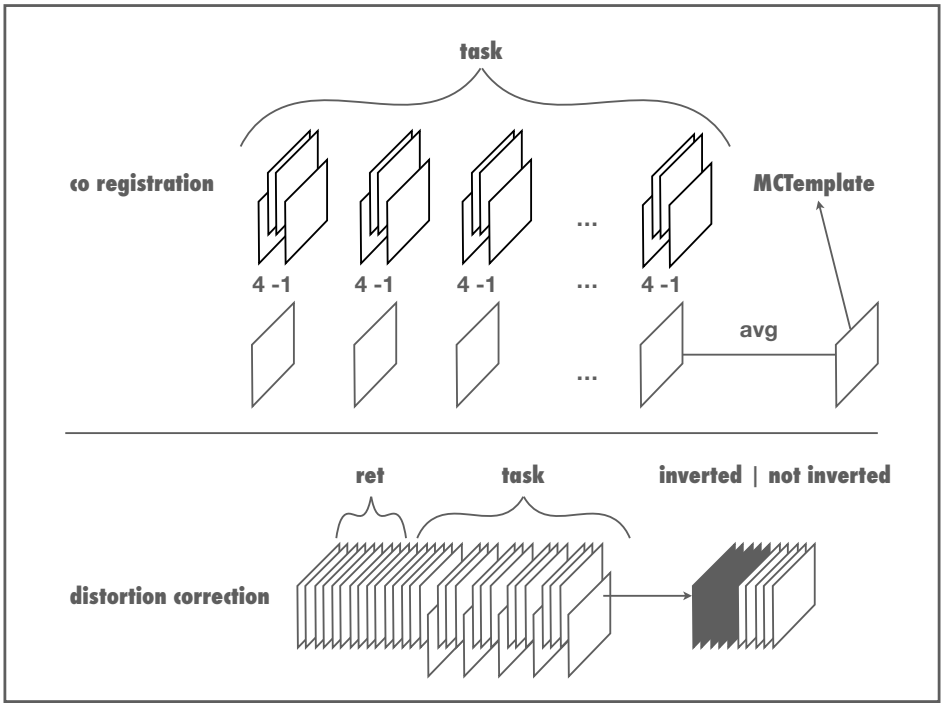


Figure 2: Coregistration and distortion correction visual depiction

Interpolate time series of volumes to match the number of presented frames during stimulation, by resampling the TR to match the "stimulation-frame-rate". The function below calls submits the corresponding MATLAB script as a qsub job.
The data will furthermore be transformed into percent change relative to the mean.

Results can be found in 4_retinotopy/

```
# interpolate volume time series
sh do_tseriesinterpolation.sh
```

Once the data is in the right space to estimate voxel activity, based on active screen pixels, population receptive field mapping can be performed. Since using high resolution fMRI leads inevitably to "big data" the analysis has to be split up into multiple parts. In the present case an assumed computation time of $\approx 8h$ is reached by splitting up the analysis into 80 parts. Hence there is a linear relationship between processing time and parts the data can be split up to. In the present case all together 480 GB of memory are used to perform the analysis in parallel on 80 different cluster nodes with only 6 GB each. Afterwards results are combined into a single results file and temporary results are deleted.

Results can be found in 4_retinotopy/

```
# perform pRF mapping by splitting up the search space
   of voxels and combining the results.
sh do_split_analyzePRF.sh
sh combine_split_PRF_results.sh
```

After for each voxel certain parameters like circular coordinates mapping to screen locations, those have to be transformed into volumes, that can be overlayed with anatomical data in order to define masks for regions of interest based on the pRF mapping.

Results can be found in 4_retinotopy/

```
# transform pRF mapping results into NIfTI volume
   overlays
sh make_PRF_overlays.sh
sh runonqsub.sh 16gb makeOverlays.sh
```

The retinotopy can also be performed in a single step, by calling the pipeline script:

```
# using pipeline
sh retinotopy.sh
```

5.5 fMRI layer segmentation

Gray matter boundaries will be used to compute cortical layer separation. First tksurfer is used to "draw" occipital ROIs based on the retinotopic mapping. Those regions will be transformed into gray matter masks such that an individual masks for V1, V2, V3 for each hemisphere exists. Cortical layer separation yields a probability map for each voxel to lay within a specifcially spaced area between gray matter boundaries. Thus a layer specific ROI specific map is created for each ROI by multiplying the ROI map with each of the layer probability maps. The results are 4D mask volumes separating the layer probabilies in the 4th dimension.

maximum memory required	approx time required
16GB	$\approx 0.5h + user$

FreeSurfer's tksurfer (`https://surfer.nmr.mgh.harvard.edu/fswiki/TkSurfer`) is used to obtained visual ROIs from retinotopic mapping. When running `GUI2ROIs.sh`, an instance of tksurfer is spawned with the correct overlay already set up. In some cases the required 0_freesurfer/mri/orig.mgz is not present, but instead it's called orig_nu.mgz. In that case please make sure that 0_freesurfer/mri/orig.mgz exists by copy-pasting orig_nu.mgz to orig.mgz This can also be achieved by calling:

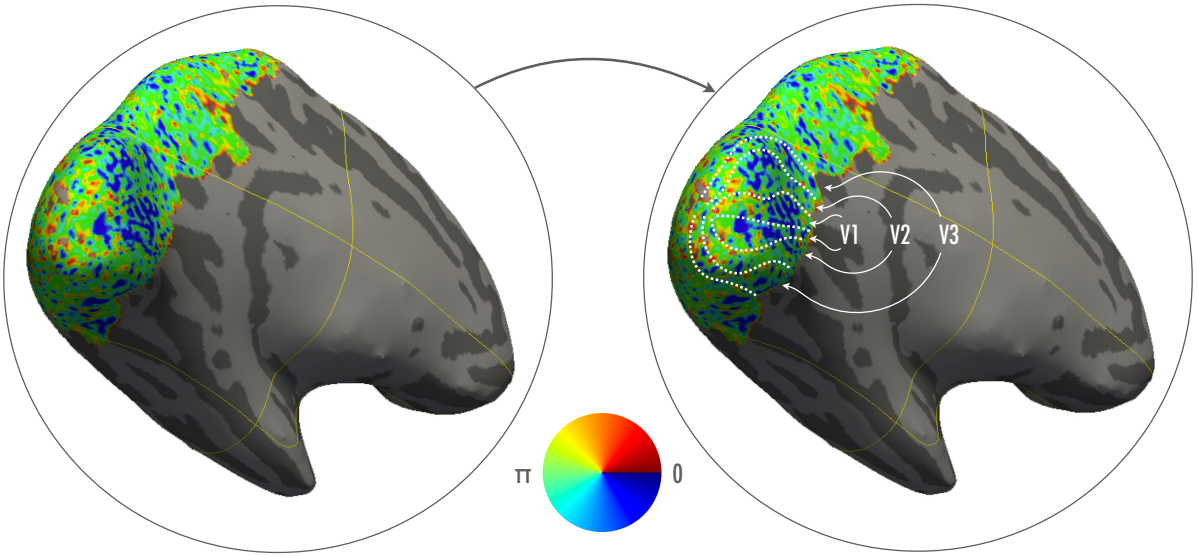


Figure 3: Definition of visual ROIs. Note that along the borders of visual areas a significant shift in color occurs.

```
# cd to /B_scripts folder
cp ../0_freesurfer/mri/orig_nu.mgz  ../0_freesurfer/mri/
   orig.mgz
```

Afterwards run the commands below again. Once the overlay is displayed correctly, proceed as described in Section 7. Afterwards labels are transformed into masks. Since the selection was done on a surface, the corresponding mask volume is only one voxel thick. Hence voxel have to be expanded to fill the gray matter area.

Results can be found in 5_laminar/

```
# making masks for V1,2,3 based on retinotopy
sh GUI2ROIs.sh
sh labels2masks.sh
sh expandROIs.sh
```

Note that using FreeSurfer's freeview yields better plotting functionality, however `tksurfer` provides an easier way to select vertices.
Once this is done gray matter volumes are split into corical layers. This is implemented in OpenFmriAnalysis (`https://github.com/TimVanMourik/OpenFmriAnalysis`). Based on gray and white matter boundaries and the respective curvature, gray matter is segmented into 3 layers. The result is a probability map for each voxel to belong to a certain layer.

Results can be found in 5_laminar/

```
# performing laminar separation
sh do_getLayers.sh
sh do_getLayerWeights.sh
```

The layer segmentation process can also be performed in a single step, by calling the pipeline script:

```
# using pipeline
sh laminar.sh
```

5.6 EEG pre-processing

The raw EEG signal will be transformed into virtual channel data by applying a LCMV beamformer. Frequencies of interest (α, β, γ) are defined that $8\ Hz \leq \alpha \leq 12\ Hz$; $18\ Hz \leq \beta \leq 28\ Hz$; $60\ Hz \leq \gamma \leq 80\ Hz$. Since the covariance matrices of the noise will differ significantly for lower and higher frequencies, the data will be band pass filtered for low and high frequencies separately. This will be achieved by using pass bands of $2 - 32\ Hz$ and $30 - 100\ Hz$ respectively.
A FEM forward model will be used as well as a anatomical constraint source model (see Figure 4 for a graphical depiction).

First data is loaded an segmented into trials. Note that the default trial selection

maximum memory required	approx time required
16 GB	$\approx 2h$

string might vary from experiment to experiment. Target trigger values can be set in `do_EEGpreprocessing.sh`
Additionally the data will be band pass filtered at $2 - 32\ Hz$ for α and β bands and at $30 - 100\ Hz$ for γ. Respective changes in the filterband can be made in `do_EEGpreprocessing.sh`
If desired, trials should be excluded at this stange as well. To achieve this either use FieldTrip's built in functionality (`do_EEGpreprocessing.sh` bottom) or provide a logical array of whether to include a trial to `ft_redefinetrial(cfg, data)`. See `http://www.fieldtriptoolbox.org/reference/ft_redefinetrial` for more information.

Results can be found in 6_EEG/

```
# EEG preprocessing: trial selection, BP filter, (trial
   selection)
sh do_EEGpreprocessing.sh
```

Once the general preprocessing is done, the function `ft_timelockanalysis` (`http://www.fieldtriptoolbox.org/reference/ft_timelockanalysis`) is used to estimate

the noise covariance matrix based on baseline data for high and low frequencies separately. This is necessary since the noise structure for those different bands varies significantly. The default noise estimation window is -0.5 to -0.1 s realtive to stimulus onset.

Results can be found in 6_EEG/

```
# EEG preprocessing: noise estimation
sh do_EEGtimelock.sh
```

As last step the result from `fsrecon` (`do_fsrecon.sh`) must be prepared to be used as source model. For this purpose Workbench (`https://github.com/Washington-University/workbench`) is used, wrapped into `do_EEGprepareFS.sh`

Results can be found in 6_EEG/

```
# EEG preprocessing: source model preparation
sh do_EEGprepareFS.sh
```

5.7 EEG prepare head and source models

Head and Sourcemodel are prepared using FieldTrip. Per default the pipeline is set up to use finite element models (FEM), but can be changed to boundary element model (BEM) as well. As source model, the result from Workbench is used. Loading

maximum memory required	approx time required
32 GB	$\approx 6h + user$

and segmenting anatomical MRI data is done using ft_volumesegment (`http://www.fieldtriptoolbox.org/reference/ft_volumesegment`) and `ft_prepare_mesh` (`http://www.fieldtriptoolbox.org/reference/ft_prepare_mesh`). For more details about the settings of this function see Section 6.40.

Results can be found in 6_EEG/

```
# prepare head model
sh do_EEGprepareHeadmodel.sh
```

Once the headmodel is prepare, sensor locations need to be coregistered. This is achieved by starting an interactive MATLAB session and opening the script `do_EEGelectrodeRegistration.m`. The FieldTrip function `ft_electroderealign` is used to interactively coregister sensor locations to anatomical MR data. See `http://www.fieldtriptoolbox.org/reference/ft_electroderealign` for more information.

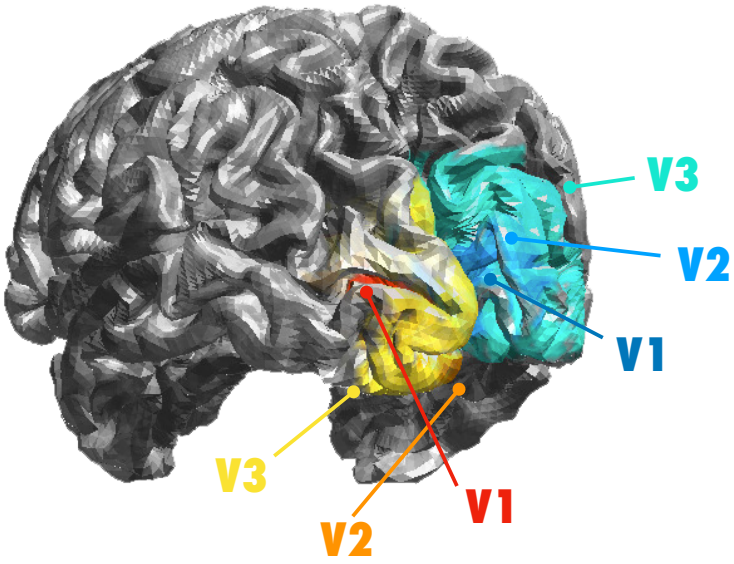


Figure 4: Example source model including visual ROIs. Note that for demonstrative purposes all possible ROIs are show, however per default only V1 for both hemispheres will be used. This plot was created using `EEGvisualizeVirtualChannel.m`

```
# coregister electrode locations to anatomical MR using
   MATLAB
matlab
```

The previously computed data is now prepared to be finally used and the Workbench sourcemodel is loaded.

Results can be found in 6_EEG/

```
# prepare data and load source model
sh do_EEGprepareSourcemodel.sh
```

5.8 EEG compute virtual channels

maximum memory required	approx time required
960 GB	$\approx 12h$

The first step is to estimate time courses for dipoles at each voxel lcoation. To achieve this LCMV beamformer weights are computed separately for high and low frequencies. To reduce search space and hence computation time, previously computed ROI maps are used. By default left and right V1 are used as two separate ROIs. Volumetric ROIs are translated into cartesian coordinates. Beamformer weights are computed using $\lambda = 10\%$ for noise regularization employing `ft_sourceanalysis` (`http://www.fieldtriptoolbox.org/reference/ft_sourceanalysis`). Trial segmented data is then multiplied with beamformer weights, in order to obtain individual time series for all dipole locations.
Note that 480 GB of memory are required to estimate the beamformer weights in a parallelized fashion as below.

Results can be found in 6_EEG/

```
# prepare data and load source model
sh do_EEGsplitBeamformer.sh
sh do_EEGsplitVirtualChannel.sh
```

Afterwards a time frequency analysis (`http://www.fieldtriptoolbox.org/reference/ft_freqanalysis`) is run for all virtual channels. Since this operation is performed on a single trial basis for different filter bands and left and right hemisphere separately, 960 GB of memory are required to compute this step as implemented.

Results can be found in 6_EEG/

```
# prepare data and load source model
sh do_EEGsplitFreqOnVirtChan.sh
```

To relate the data to layer specific fMRI results, a channel selection has to take place, that is optimized for frequency ROIs (α, β, γ). For each of the 3 bands all virtual channels will be searched and the best 100 are selected. Those are defined to have the lowest (α, β) or highest (γ) relative power compared to the baseline period (-0.3 to -0.2 s relative to simulus onset).

Frequency ROIs are defined as:

α	β	γ
$8 - 12$ Hz	$22 - 28$ Hz	$50 - 70$ Hz

Furthermore, the function produces figures saved in C_miscResults similar to what is shown in Figure 5.

Results can be found in 6_EEG/

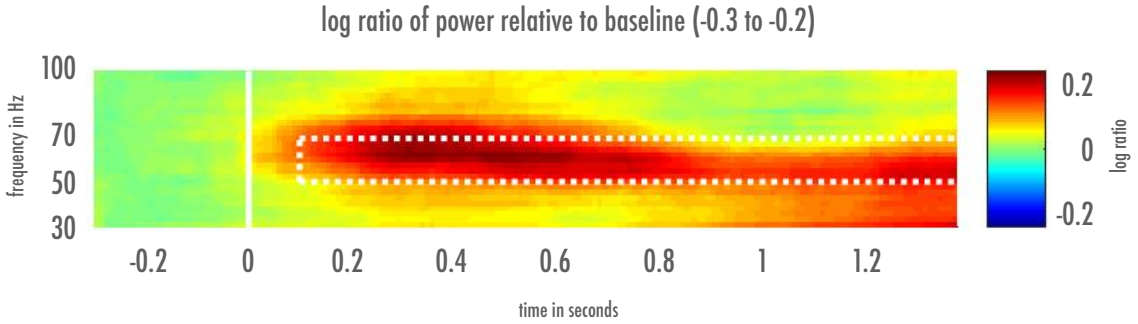


Figure 5: Example for γ response in virtual channel

```
# prepare data and load source model
sh do_EEGfreqChanSelect.sh
```

6 Scripts explained (support for runonqsub.sh: Q)

6.1 getToolboxes.sh

Downloads all necessary toolboxes into a folder called toolboxes/ or displays a URL where the user has to download the software manually (see Section 2.1). That is FSL and SPM have to be downloaded manually due to required user registration. Place them in the same directory as the other toolboxes or install as required.

6.2 runonqsub.sh

Initiates a qsub session and sends it to the cluster. This is necessary, since scripts running on the cluster are copied to a different directory, but within scripts all directories are relative. runonqsub.sh must hence be run in a "normal" session not using a non-interactive qsub, giving the respective script that was intended to run as an argument. runonqsub will then set everything up correctly. Further you have to specify the amount of memory needed.

```
# example
sh runonqsub.sh 32gb myscript.sh
```

Additional arguments can be passed after the script.

```
# example with argument
sh runonqsub.sh 32gb myscript.sh myargument
```

Under the hood:

```
qsub -l walltime=24:00:00,mem=$1 -F "$DIR $\{@:3\}" $DIR
   /$2
```

- $DIR is the current absolute path to B_scripts
- $1 is the amount of memory used (e.g. `32gb`)
- $2 is the respective script to run (e.g. `myscript.sh`)
- ${@:3} is all following arguments that are parsed to the script (e.g. `myargument`)
- every time `runonqsub.sh` is called the absolute path is forwarded to the script (`-F $DIR`) in order to keep the relative dependencies clear

6.3 cleanscriptsfolder.sh

Since qsub puts a output + error log into /B_ scripts `cleanscriptsfolder.sh` can be used to move all qsub outputs to /B_scripts/qsuboutput

6.4 liveupdateqsub.sh

Submits a qstat -u user request every second.

```
# example
sh liveupdateqsub.sh tomcla
```

exit command by hitting ctrl+c

6.5 waitForJobs.sh

Checks all currently running jobs (`qstat -u user`) and waits until the last script up until this point has finished.

6.6 makenewsubject.sh

executable script (double click to execute)

Creates a new subject (S#) folder structure in order to get all folder dependencies right. It automatically detects folders called "S<*number*>" and creates a new folder incrementing <*number*> by one.

6.7 setupfolders.sh (Q)

Sets up the necessary folder structure for the analysis (within a subject folder). This function is automatically called when using `makenewsubject.sh`

6.8 do_preparefunctionals.sh (Q)

Prepares functional data, i.e. removes the first 4 and the last x volumes of every set of functionals, where x is the number of volumes acquired after the stimulation ended. If files were modified (i.e. if volumes were deleted), the original file will remain in /niftis/-functionals/old

6.9 do_realignment.sh (Q)

Merges and motion corrects all files in /niftis/functionals using the average volume of all task volumes.

numberofvolumes.txt is written to /A_helperfiles giving information about which sets had how many volumes. The first column is coresponds to dim4 from `fslinfo`

Under the hood:

- `fslmerge` is called to create a combined .nii for all files in niftis/functionals/*sparse* - files are merged allong the 4th dimension
- a .txt file is created saving which files were merged and how many volumes were in there
- `mcflirt` is used on the combined data doing the motion correction based on the average volume across all task blocks
- all results are stored in /1_realignment
- results have the extension "_mcf"

6.10 do_distcorr.sh (Q)

Uses the averge volume of the inverted set of images in /niftis/inverted as reference for *inverted*

Uses as many volumes n as there were in /niftis/inverted. Respective volumes are selected starting from the last going in $4 \; x \; n$ steps backward. Hence the resulting average of this set was computed over the 4th volume of each of n last sets in the normal images.

Simplified (all 1 are selected):

for n=5

0,0,0,0,...,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1

Both (normalavg and invertedavg) will be used to esimate the distortion correction.

Under the hood:

- reference volumes (normal + inverted) are selected in order to do the field distortion estimate (selection: see above).
- selected reference volumes are merged into a single file (/3_distcorrection/all_b0.nii.gz)
- topup is called using the merged (normal + inverted) .nii and performs field mapping accoriding to the specifications in /A_helperfiles/b02b0.cnf using the acquisition parameters specified in acquisition_parameters.txt
- all results are stored in /3_distcorrection

6.11 do_applysimpledistcorr.sh (Q)

Applies distortion correction to all functional data.

Under the hood:

- distortion correction is applied to the full set of functionals using the `OUTPUT` of topup (`topup [...] --out=OUTPUT`) as well as acquisition_parameters.txt (Using `--method=jac`)
- all results are stored in /3_distcorrection
- all results have the prefix "corrected_"

6.12 do_preparecoregistration.sh (Q)

Computes a reference volume (MCTemplate) used as the basis to co-register with the anatomical image. This image is computed by splitting the entire set of task volumes into chunks of 4. In each chunk the 1st volume is substracted from the 4th. This procedure yields a pseude-T1 contrast. All chunk differences are then averaged to form "MCTemplate"

6.13 do_correctavgdiff.sh (Q)

Due to the subtraction of the 1st from the 4th volume the area outside the brain has the highest value. In order to correct this the lowest value will be shifted to zero the a certain part of the higher values are cut (e.g. 0.975).

The result should be checked using
`fsleyes ../2_coregistration/Inplane/MCTemplateThr*.nii.gz` to ensure that the area outside the brain is nulled. Note that it can happen that some areas within the brain are nulled as well. If they are not too wide spread they do not affect later co-registration.

6.14 do_fsrecon.sh (Q)

Performs segmentation using Freesurfer.

If running on OSX the script assumes:

```
# FreeSurfer Home Mac
FREESURFER_HOME=/Applications/freesurfer
```

If running on linux the script assumes:

```
# FreeSurfer Home Linux
FREESURFER_HOME=/opt/freesurfer/version
```

The target image that is used for the segmentation must be located in /niftis/t1

Under the hood:

```
recon-all -i rawData/niftis/t1/* -subjid  0_freesurfer -
   all OutputVolume.nii.gz
```

, all results are stored in /0_freesurfer

6.15 do_coregistration.sh

Will submit a qsub session using MATLAB, that in turn will be using FreeSurfer (bbregister) and OpenFmriAnalysis to perform a linear and non-linear boundary registration. Movie files of the respective co-registration are stored in /C_miscResults
The file used to execute MATLAB commands is /B_scripts/do_coregistration.m Note that it will be modified using the correct folder settings, which yields tmp.m that will be called and removed after MATLAB was closed.

6.16 do_coregistration.m

The script doing the actual co-registration. Uses wrapper functions from OpenFmriAnalysis to evoke FreeSurfers bbregister and computes a recursive boundary based

registration.

The results are several files storing the boundaries as a point cloud, and the respective transformation in diferent formats.
Under the hood:

- calls bbregister
- results are stored in /2_coregistration
- yields boundaries.mat, matrix.mat, bbregister.dat, transmat.txt, transmatinv.txt

6.17 do_makemasksandlabels.sh (Q)

Uses FreeSurfer reconstruction to create masks for CSF, gray matter and white matter and creates labeled volume for use within retinotopy containing left / right + gray / white matter. All masks will be created as full-brain and partial-brain masks.

If the orientation needs to be changed, parse the respective FreeSurfer compatible orientation parameter (`https://surfer.nmr.mgh.harvard.edu/pub/docs/html/mri_convert.help.xml.html`)

Under the hood:

```
sh do_movesinglevolume.sh InputVolume.nii.gz transmat.
   txt OutputVolume.nii.gz
```

- results are stored in /2_coregistration
- coregistered functional masks have the prefix "fct" and the suffix "coreg"

6.18 do_movesinglevolume.sh

Simple wrapper function to apply a transformation from one volumetric space to another, given a transformation matrix using FSL's `flirt` (`https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT`).

Under the hood:

```
flirt -applyxfm -in $1 -ref $1 -init $2 -out $3
```

- $1: volume to move
- $2: transformation matrix in form of a .txt file containing a 4×4 transformation matrix
- $3: volume to write

6.19 do_tseriesinterpolation.sh

Wrapper function for running `do_tseriesinterpolation.m` via qsub. Variables used by do_tseriesinterpolation.m can be defined in the header section of the script.
Under the hood:

- submits a MATLAB job using qsub calling `do_tseriesinterpolation.m` using the below settings
- default number of blocks: 3
- default TR: 2.7s
- default Mask Threshold: 0.01
- default rescaled stimulus size: 100px

6.20 do_tseriesinterpolation.m

Interpolates time series for retinotopy scans to match the number of frames during the stimulation (cubic interpolation). The new TR will be $TR_{new} = \frac{TR_{old}N_{volumes}}{N_{imageframes}}$. The function implements "tseriesinterp" from the analyzePRF toolboxe. The data will be normalized to percent signal change.

Furthermore all images will be downsampled to the predefined size (e.g. 100px)

Under the hood:

- interpolates time series of functional data to match the numer of frames during the stimulation
- results: tIntData.mat (interpolated time series)
- results: mask.mat (volume mask)
- results: voxelindices.mat (indices of mask voxels, ratio between $N_{imageframes}$ and $N_{volumes}$, mask threshold)
- results: downsampled images

6.21 do_split_analyzePRF.sh

Wrapper function for running `do_analyzePRF.sh` in muslitple instances. The function calls a predefined number of instances of `do_analyzePRF.sh` to run on the cluster, submitted using qsub. Per default 80 jobs requesting 6 GB of memory each (see Section 6.22), will be spawned. This procedure speeds up the population receptive field estimation significantly and would be otherwise impractical on high resolution functional data.

6.22 do_analyzePRF.sh

Wrapper function for running `do_analyzePRF.m` via qsub. Spawns a qsub job using 6GB of memory (default) and runs a specific part of a set of parts. The first input argument is the part that is to be computed and the second input argument is the sum of all parts.

```
# example split search space into 5 parts an run the
   first
sh do_analyzePRF.sh 1 5

# example not splitting up search space
sh do_analyzePRF.sh 1 1
```

6.23 do_analyzePRF.m

Computes pRF mapping implemeting "analyzePRF" from the analyzePRF toolbox. It can be decided whether to average across blocks or if the estimates shall be done for each block separately and averaging the results. Note, that running the analysis on separate blocks slows down computation time, but to my experience yields slightly better results.

Under the hood:

- estimates pRFs
- computes pRF mapping for part x of X by splitting the list of indices and selecting the corresponding set
- set avgdata=1 to average data before the analysis (otherwise it will be averaged afterwards)
- result: structure containing pRF parameters of part x of X

6.24 combine_split_PRF_results.sh

Wrapper function for running `combine_split_PRF_results.m` via qsub. Combines split results from `do_split_analyzePRF.sh` (see Section 6.21). Note that the exact amount of parts that the data was split into must be defined here (default: 80).

6.25 combine_split_PRF_results.m

Combines split results from do_split_analyzePRF.sh (see Section 6.21) into a single MATLAB file. Note that given the respective numerical naming of all parts (x_of_X) parts will be concatenated along values of x. After this operation split parts will be deleted.

6.26 make_PRF_overlays.sh

Wrapper function running `make_PRF_overlays.m` via qsub

6.27 make_PRF_overlays.m

Takes result of pRF mapping and creates .nii files for separate parameters (ang, ecc, expt, r2, rfsize, xpos, ypos). The data will be saved in ../4_retinotopy as parameterName_map.nii

Under the hood:

- angle (θ) will be devided by 180 and multiplied by π
- eccentricity (r) will be devided by the image size (in px) and multiplied by the original value range (° visual angle; default: 7). Only ecc values that have a positive R^2 and are smaller than the defined field of view, will be written to file. All other values will be set to NaN
- X position (xpos) $\cos(\theta) \cdot r$
- Y position (ypos) $\sin(\theta) \cdot r$

6.28 makeOverlays.sh (Q)

Makes FreeSurfer compatible overlays from NIfTI files. Calls FreeSurfer's `mri_vol2surf` in order to transform a volumetric .nii file into a surface file. This will be done for both hemispheres separately. Parameters transformed to surface files are: ang, ecc, xpos, ypos, r2

Under the hood:

```
mri_vol2surf --src $DIR/4_retinotopy/ang_map.nii  --
   srcreg $DIR/2_coregistration/bbregister.dat --hemi ?h
    --surf pial --out $DIR/4_retinotopy/?h.ang.mgh --
   out_type paint
```

- –src: NIfTI to be transformed
- –srcreg: result from `bbregister` of `do_coregistration.sh` (default: bbregister.dat)
- –hemi: respective hemisphere (lh for left hemisphere or rh for right hemisphere)
- –surf: surface used as reference (obtained from FreeSurfer in ../0_freesurfer/surf/)
- –out: output file name
- –out_type: influences how the file is outputted (see `mri_vol2surf --help` for more information)

6.29 GUI2ROIs.sh

This script spawns an instance for `tksurfer` (FreeSurfer)

6.30 labels2masks.sh (Q)

Converts surface labels defined for ?h, V1,2,3 into a volume. For this purpose FreeSurfer's `mri_label2vol` will be called using the following configuration:

- –label /0_freesurfer/label/?h.V?.ret.label (the label to be converted)
- –temp /2_coregistration/refVol (shape template volume)
- –reg /2_coregistration/bbregister.dat (co-registration file)
- -o /4_retinotopy/?hV?maks.nii.gz (output volume's file name)

To indicate which hemisphere or visual ROI, the questionmarks must be replace with the respective information (lh, rh; V1, V2, V3).

The full command looks internally like this:

```
mri_label2vol --label $DIR/0_freesurfer/label/lh.V1.ret.
   label --temp $DIR/2_coregistration/$refVol --reg $DIR
   /2_coregistration/bbregister.dat --o $DIR/4
   _retinotopy/lhV1mask.nii.gz
```

, where `DIR` is the subject's absolute path.

6.31 expandROIs.sh

Wrapper function for running `expandROIs.m` via qsub. In order to perform the correct voxel expansion, the expansion factor (`ExpandBy`) and mask threshold (`mskThr`) can be specified. Those settings will be forwarded to `expandROIs.m`.

6.32 expandROIs.m

Thresholds a given volume utilizing the predefined mask threshold (`maskthreshold`). Afterwards a cubic box around each voxel will be created, that goes N voxels in each of the 6 directions. Hence a volume containing only a single ROI of size $1 \times 1 \times 1$ will be expanded to $9 \times 9 \times 9$, if the given expansion factor `ExpansionFactor = 4`. The actual voxel expansion algorithm was implemented in `tc_expandVoxelSelection.m` provided by the tc_functions script collection (`https://github.com/TommyClausner/tc_functions`).

6.33 do_getLayers.sh

Wrapper function for running `do_getLayers.m` via qsub. Using the variable `registerTo` the volume to perform the layer segmentation on is defined and by setting `Nlayers`, the number of layers in which to split the data.

6.34 do_getLayers.m

Utilizes the toolbox OpenFmriAnalysis (`https://github.com/TimVanMourik/OpenFmriAnalysis`) to segment `numLayers` anatomical layers between the gray and white matter boundaries of the volume `regvol`.

Under the hood:

```
% boundaries used for co-registration
cfg.i_Boundaries       = '2_coregistration/boundaries.mat';

% gray and white matter files obtained from FreeSurfer
cfg.o_ObjWhite         = '5_laminar/?h.white.reg.obj';
cfg.o_ObjPial          = '5_laminar/?h.pial.reg.obj';

% volume to use for segmentation
cfg.i_ReferenceVolume = ['2_coregistration/' regvol '.nii'];

% Output volumes for gray and white matter
cfg.o_SdfWhite         = '5_laminar/?h.white.sdf.nii';
cfg.o_SdfPial          = '5_laminar/?h.pial.sdf.nii';
cfg.o_White            = '5_laminar/brain.white.sdf.nii';
cfg.o_Pial             = '5_laminar/brain.pial.sdf.nii';

% output gradient and curvature
cfg.o_Gradient         = '5_laminar/brain.gradient.nii';
cfg.o_Curvature        = '5_laminar/brain.curvature.nii';

% define layer space (4 boundaries for 3 Layers)
cfg.i_Levels           = linspace(0,1,numberOfLayers+1);

% output for layer data
cfg.o_LaplacePotential= '5_laminar/LaplacePotential.nii';
cfg.o_LevelSet         = '5_laminar/brain.levels.nii';
cfg.o_Layering         = '5_laminar/brain.layers.nii';
```

6.35 do_getLayerWeights.sh

Wrapper function for running `do_getLayerWeights.m` via qsub

6.36 do_getLayerWeights.m

Combines masks obtained from retinotopy with layer masks obtained from laminar segmentation. The respective expanded volumes of the pRF mapping are multiplied with the layer mask and yield a layer specific ROI selection masks. As a result a NIfTI file is create having the same shape, space and orientation as the functional data. Thereby the 4th dimension of the volume represents the respective number of layers.

6.37 do_EEGpreprocessing.sh

Wrapper function for running `do_EEGpreprocessing.m` via qsub

6.38 do_EEGpreprocessing.m

```
% for trial and channel selection
% Selprefix = ''; does not exist
Addprefix = 'TCsel_';

% band pass filtering
Selprefix = 'TCsel';
Addprefix=['BP' num2str(filterPassBand(1)) '_' num2str(
    filterPassBand(2)) '_']; % result is e.g. BP2_32_
```

Does a basic preprocessing to the data. First trigger values will be extracted and trials are defined according to the specifications in the script. Further only EEG channels are selected. The Data is downsampled to 1024 Hz. This in-between step is stored. Afterwards the data is further split into different sets of band pass filtered data. Note, that the noise structure will be different for high and low frequencies and thus this step is highly recommendable. Per default two band pass filters are chosen ($2 - 32$ Hz and $30 - 100$ Hz). The baseline period will be from -0.5 s until -0.1 s relative to the stimulus onset. For the high frequency case a dft filter to filter line noise is applied. Using the default version of this function will store the computed data using the respective filter settings as a prefix. However if desired trials can / should be excluded in this step as well.

Under the hood (uses the following fieldtrip functions):

- ft_definetrial (sets up trial definition according to predefined trigger values in the data)
- ft_preprocessing (selects EEG channel)
- ft_redefinetrial (defines channel and creates trials)
- ft_resampledata (downsampling)
- ft_preprocessing (frequency filters)
- (ft_rejectvisual()) (trial rejection)

6.39 do_EEGprepareFS.sh

This script was provided by Simon Homölle (DCCN, 2018). Uses Workbench (`https://github.com/Washington-University/workbench`) to prepare FreeSurfer results for use with the EEG pipeline.

6.40 do_EEGprepareHeadmodel.sh

Wrapper function for running `do_EEGprepareHeadmodel.m` via qsub. Changing the variable `Model` will cause the function to produce a different forward model. Possible options are `Model=FEM` and `Model=BEM`.

6.41 do_EEGprepareHeadmodel.m

Creates volume conduction model either as finite-element model (FEM) or boundary element model (BEM). Per default all scripts are set up to use a FEM model. The respective MRI file to use will be 0_freesurfer/mri/orig_nu.mgz. If desired the variable `convertCoordSys` can be set to `convertCoordSys = 1;` to convert the coordinate system to ctf (otherwise will be ras).

Models will have the following properties: Underlying FieldTrip functions are:

Model	Propertiy	Value
FEM mesh	cfg.output	{'gray','white','csf','skull','scalp'}
	cfg.scalpsmooth	10
	cfg.method	'hexahedral'
	cfg.shift	0.3
FEM	cfg.method	'simbio'
	cfg.conductivity	[0.33 0.14 1.79 0.01 0.43]
BEM mesh	cfg.output	{'brain','skull','scalp'}
	cfg.method	'projectmesh'
	cfg.numvertices	[1000 2000 3000]
BEM	cfg.method	'openmeeg'
	cfg.conductivity	[0.33 0.01 0.43]

Table 2: Specifications for creating headmodel.

- ft_volumesegment (segments anatomical data into distinct structures (e.g. skull))
- ft_prepare_mesh (sets up 3D mesh for head model preparation)
- ft_prepare_headmodel (creates the conductivity model)

6.42 do_EEGelectrodeRegistration.m

This function cannot be run using Qsub or a headless MATLAB instance, since it requires user interaction via a graphical user interface. The script will load the respective data, save and exit (after the user is done). Since this script is just a wrapper function for `ft_electroderealign` I will refer to the corresponding FieldTrip documentation: `http://www.fieldtriptoolbox.org/reference/ft_electroderealign`

6.43 do_EEGprepareSourcemodel.sh

Wrapper function for running `do_EEGprepareSourcemodel.m` via qsub. Changing the variable `Model` will cause the function to use a different forward model. Possible options are `Model=FEM` and `Model=BEM` (must be in accordance with `do_EEGprepareHeadmodel.sh`).

6.44 do_EEGprepareSourcemodel.m

Sets up sourcemodel (dipole model) for source analysis using the result of `do_EEGprepareFS.sh` It utilizes FieldTrip's `ft_prepare_vol_sens` to ensure that headmodel and sensors are set up correctly and prepares the source model by reading the Workbench result as a headshape (`ft_read_headshape`).

6.45 do_EEGtimelock.sh

Wrapper function for running `do_EEGtimelock.m` via qsub.

6.46 do_EEGtimelock.m

```
sel_ = {'BP30_100','BP2_32'}; % used in a for loop
Selprefix = sel_{n}; % inside for loop
Addprefix = 'TL_';
```

Average (re-) referencing, timelock analysis and noise covariance estimation for the data. Estimates noise between $-0.5\ s$ until $-0.1\ s$ relative to the stimulus onset. Basically a wrapper function to call `ft_timelockanalysis` using `cfg.keeptrials = 'yes'`. See `http://www.fieldtriptoolbox.org/reference/ft_timelockanalysis` for more information.

Subsequent processing steps include average referencing and noise covariance estimation based on the variable noiseEstWin=[-0.5 -0.1];

6.47 do_EEGsplitBeamformer.sh

Wrapper function for running `do_EEGbeamformer.sh` in multiple instances. The function calls a predefined number of instances of `do_EEGbeamformer.sh` to run on the cluster, submitted using qsub. Per default 8 jobs requesting 62 GB of memory each, will be spawned. This procedure speeds up the processing significantly and would be otherwise

impractical.

6.48 do_EEGbeamformer.sh

Wrapper function for running `do_EEGbeamformer.m` via qsub

6.49 do_EEGbeamformer.m

```
Selprefix = 'TL';
Addprefix='Beamf_centV1_';
```

Performing LCMV beamforming on the data. Since the grid size will be scaled to match the functional (high-res) data, scanning the full space would require too much computation time. Hence a ROI mask is used, that is in the present case the white matter voxels in V1, as obtained from retinotopy scans. Noise regularization was set to $\lambda = 10\%$

6.50 do_EEGsplitVirtualChannel.sh

Wrapper function for running `do_EEGvirtualChannel.sh` in multiple instances. The function calls a predefined number of instances of `do_EEGvirtualChannel.sh` to run on the cluster, submitted using qsub. Per default 8 jobs requesting 30 GB of memory each, will be spawned. This procedure speeds up the processing significantly and would be otherwise impractical.

6.51 do_EEGvirtualChannel.sh

Wrapper function for running `do_EEGvirtualChannel.m` via qsub.

6.52 do_EEGvirtualChannel.m

```
Selprefix='Beamf_centV1';
Addprefix='VirtCh_';
```

Computes virtual channel time series, by mutliplying the data with the previously computed spatial filters.

6.53 do_EEGsplitFreqOnVirtChan.sh

Wrapper function to call `do_EEGfreqOnVirtChan.sh` The function calls a predefined number of instances of `do_EEGfreqOnVirtChan.sh` to run on the cluster, submitted using qsub. Per default 16 jobs requesting 60 GB of memory each, will be spawned. This procedure speeds up the processing significantly and would be otherwise impractical.

6.54 do_EEGfreqOnVirtChan.sh

Wrapper function for running `do_EEGfreqOnVirtChan.m` via qsub.

6.55 do_EEGfreqOnVirtChan.m

```
Selprefix='VirtCh';
Addprefix='TF_';
```

Computes time-frequency analysis for virtual channel data using `ft_freqanalysis`. For more information see `http://www.fieldtriptoolbox.org/reference/ft_freqanalysis`.

6.56 do_EEGfreqChanSelect.sh

Wrapper function for running `do_EEGfreqChanSelect.m` via qsub.

6.57 do_EEGfreqChanSelect.m

```
Selprefix='TF';
% Addprefix=''; does not exist
```

6.58 EEGvisualizeVirtualChannel.m

7 How to: Selecting visual areas based on pRF mapping

8 How to: EEG electrodes from photogrammetry

Please find the respective information within the janus3D documentation (`https://github.com/janus3D/janus3D_toolbox/blob/master/janus3D_users_manual.pdf`)

9 How to: Pipelines

9.1 Scripts

9.2 Tommy's Scriptinator 3000 TM

Tommy's Scriptinator 3000 TM is a graphical user interface that allows drag and drop based arrangements of scripts on a canvas. The code of the script as well as defined variables can directly be viewed by double clicking one of the icons representing a script (examples below). This way analysis pipelines can easily be comprehended and shared. Tommy's Scriptinator 3000 TM can be downloaded from:

`https://github.com/TommyClausner/Scriptinator/`

It requires at least JDK9 or newer to be installed:

`http://www.oracle.com/technetwork/java/javase/downloads/`

After downloading the repository from GitHub, one can start the program by double clicking Scriptinator.jar. All further help will be provided from there.

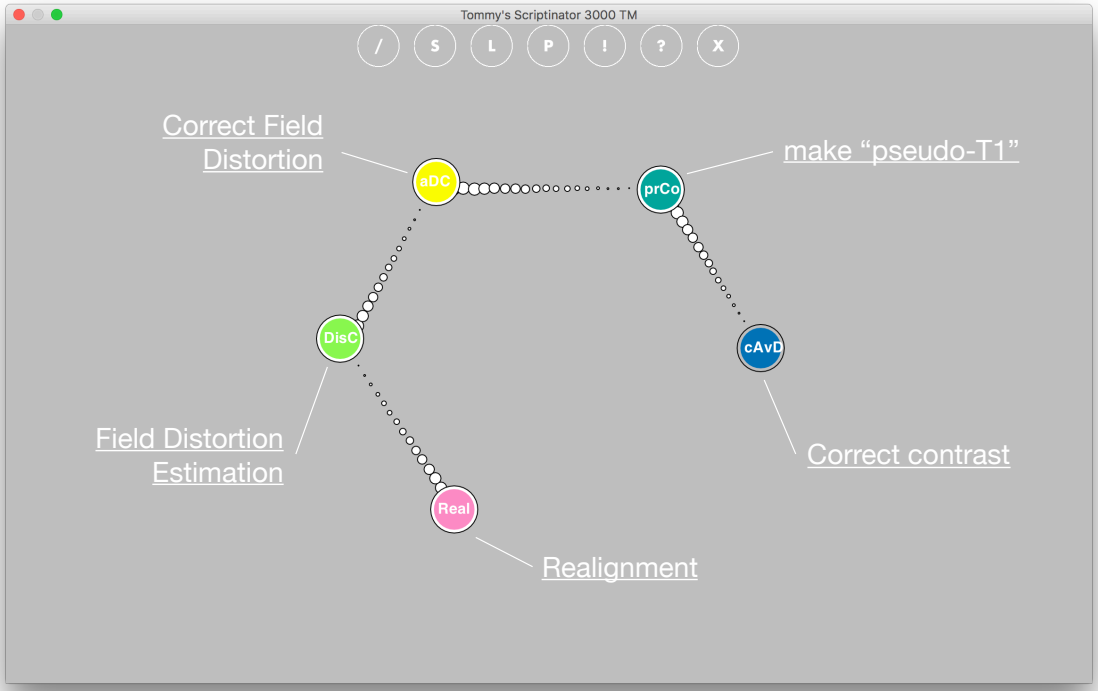


Figure 6: Example pipeline for the pre-processing, using Tommy's Scriptinator 3000 TM: preprocessing.pipe

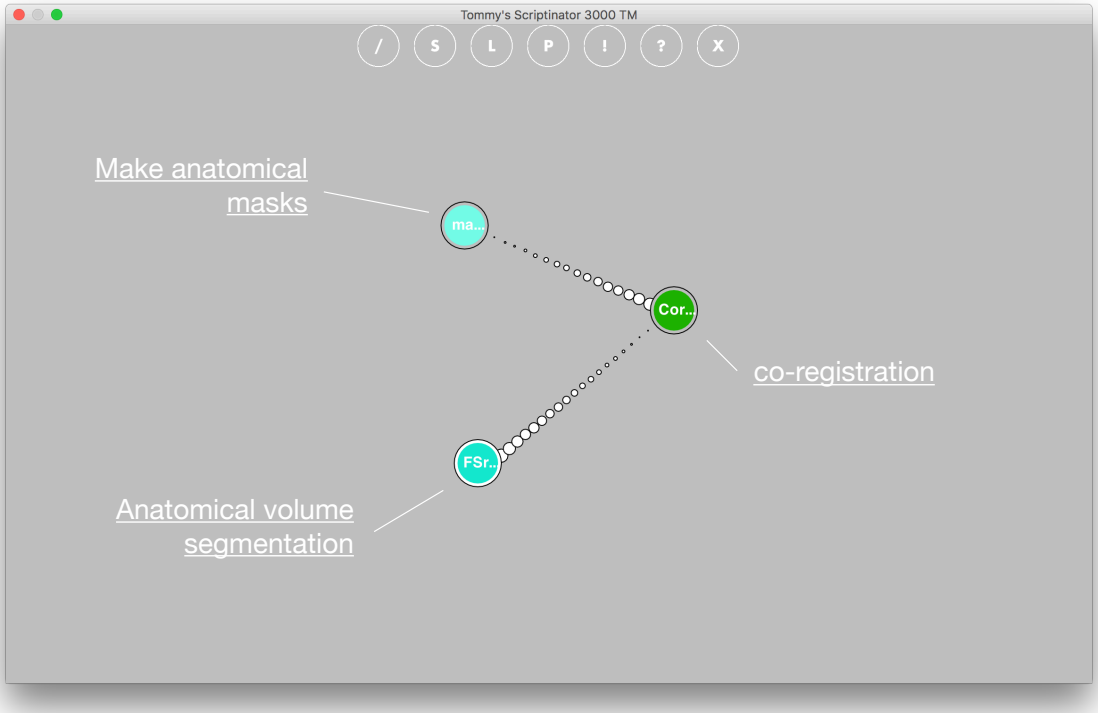


Figure 7: Example pipeline for the anatomical segmentation, using Tommy's Scriptinator 3000 TM: CoregistrationAndAnatMasks.pipe

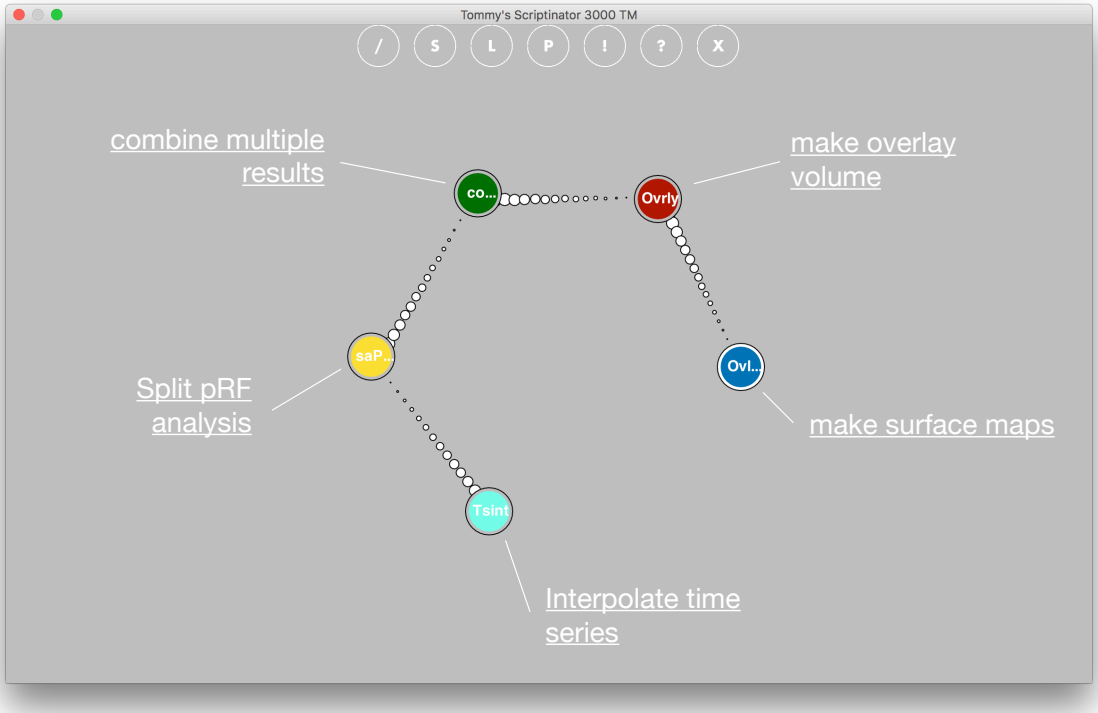


Figure 8: Example pipeline for the retinotopy, using Tommy's Scriptinator 3000 TM: retinotopy.pipe

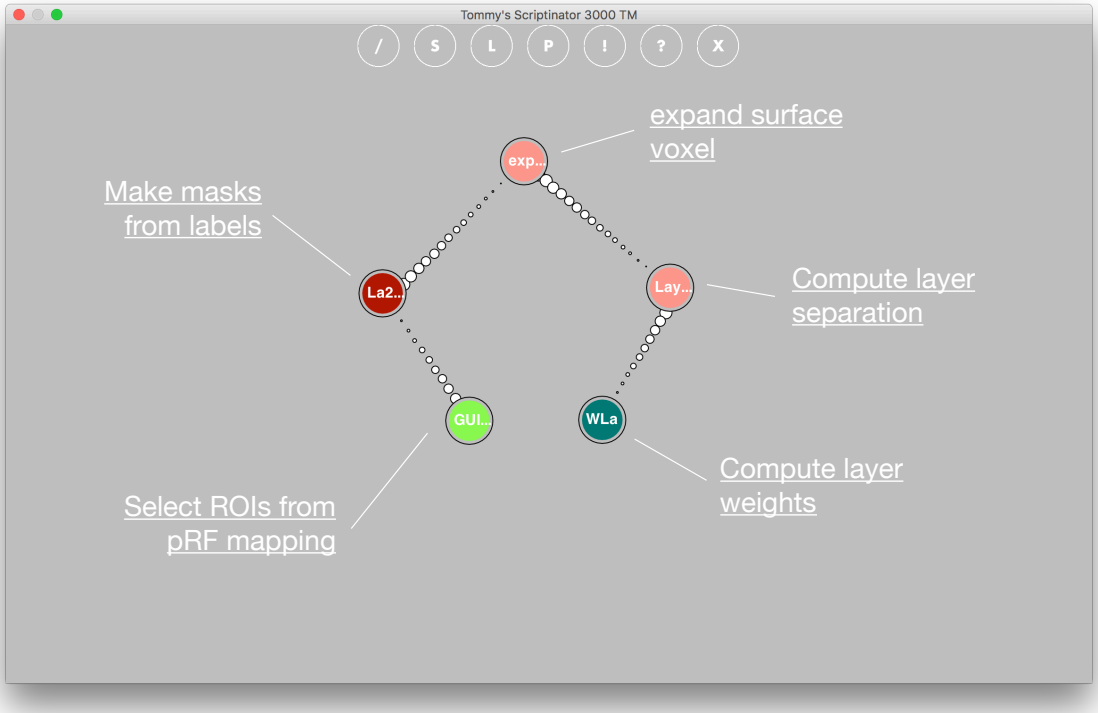


Figure 9: Example pipeline for the cortical layer definition, using Tommy's Scriptinator 3000 TM: laminar.pipe