Pipeline - visual guide

These slides serve as a visual reference for the steps outlined in the README for infant_neuropipe

Many of the details here will be the same, but with accompanying visualizations to aid in your understanding of the pipeline. There will also be explanations for the choices that were made and potential consequences of them.

It is recommended that you read the README first and then read this text

Expected background

It would be wonderful if there was sufficient infrastructure and support here to guide someone working with fMRI data for the first time.

Realistically, there will be many points of assumed knowledge.

A reasonable, minimal background would be that the user is comfortable with FSL, the FSL terminology and FSL commands, and can read but not necessarily write in MATLAB and Bash. Python experience would be a plus.

Legend

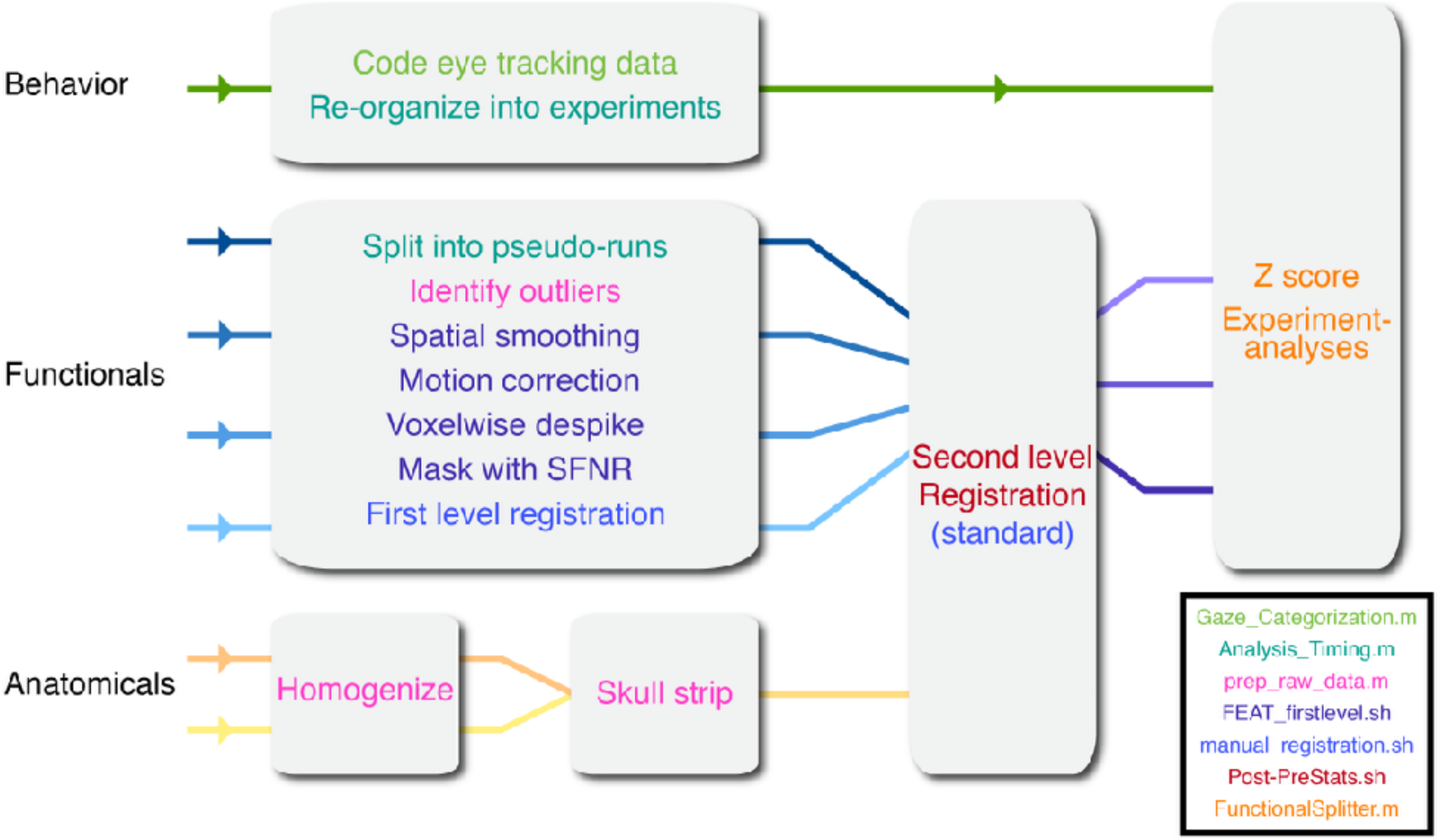
Number in the titles of each slide indicate the step (according to the README) this slide belongs too

‘$’ indicate variables that need to be changed, such as ‘$ppt’ meaning you should change this to be the participant name. $PROJ_DIR means the path to the project directory, $SUBJ_DIR means the path within that to the subject directory you are analyzing.

Grey boxes in the top right indicate the function that is being used in a given step

*** means that this is a free parameter that could be changed. For instance, you could use a different parameter or function. A gray box near those stars is an explanation of what other options are possible

Preprocessing pipeline



1. Set up Neuropipe

When adding a new participant, the first step is to make their directory by scaffolding, as below:

```
./scaffold $ppt;
```

Pull data from scanner and run initial QA (if using XNAT). The next slide explains this in more detail

```
cd subjects/$ppt; ./prep_xnat.sh
```

1. Set up Neuropipe

Using the '*$SUBJ_DIR/prep_xnat.sh*' script, it is necessary to set up a run file ('*$SUBJ_DIR/run-order.txt*') where each line names a scan (as defined in a dicom file), collected in order.

EPI=functionalXX
PETRA=petraXX
MPRAGE=mprageXX
SPACE=spaceXX
Scout=scoutXXX

Note, petra refers to the T1 anatomical sequence we use and scout refers to our localizers. For young infants we use space since it is a T2 scan sensitive to grey/white matter contrast in infants

What counts as a scan is specific to each sequence and scanner. For instance, scouts often come in as 11 separate scans.

If you want to ignore a scan, such as a calibration you ran that is irrelevant, still include the line but call it 'ERROR_RUN'

1. Set up Neuropipe

The prep_xnat.sh script has several initial steps for setting up the data that assume you are using XNAT for loading in the data

If you don't use XNAT you need to heavily edit this script to suit your data retrieval. Critically, if you do not use this procedure, you need to create nifti files with the appropriate naming conventions:

EPI=${SUBJ_NAME}_functionalXX.nii.gz
PETRA=${SUBJ_NAME}_petraXX.nii.gz
MPRAGE=${SUBJ_NAME}_mprageXX.nii.gz

Note, if you can't use `prep_xnat` you don't have to make the QA work (although it is extremely insightful). This will not interfere with any functionality, except with the '*$PROJ_DIR/scripts/Participant_Index.m*' function: you won't be able to use SNR/SFNR as a screening tool for data selection.

2. Add participant info

Add participant information to the file that keeps track of the mapping of fMRI names and matlab names (AKA the name used to save the data in the experiment menu), as well as age.

Edit $PROJ_DIR*/scripts/Participant_Data.txt* by appending a row of fMRI name, matlab name and age (in months)

3. Mask anatomicals

PETRA has a huge FOV for an infant. To help with skull stripping (and hence registration) it helps to remove non-head from the field of view of the anatomicals

If you don't do this then you won't created skullstripped versions of the data by the subsequent steps. If you don't want to mask but want this functionality then just copy your anatomical with '_masked' as a suffix.

Identify the boundaries of the volume (X, Y, Z)

fslmaths $Anatomical_unmasked -mul 0 -add 1 -roi $Xmin $Xlength $Ymin $Ylength $Zmin $Zlength 0 1 mask.nii.gz

fslmaths $Anatomical_unmasked -mas mask.nii.gz ${Anatomical}_masked.nii.gz

4. Prepare for merging

Register anatomicals that are to be merged into the same space, if you want to.

You should use the masked versions to best help the alignment and subsequent skullstripping

Rename the reference anatomical to also have ‘_registered’ as a suffix

flirt -in $Low_quality_Scan -ref $High_quality_Scan -out ${Low_quality_Scan}_registered.nii.gz -dof 6

Copy high quality scan so you know that should be merged too

cp $High_quality_Scan ${High_quality_Scan}_registered.nii.gz

5. Preprocess anatomicals

prep_raw_data.m

As part of *prep_raw_data.m* all of the anatomicals to be registered are merged.

All anatomicals are homogenized (helps if bottom head coil only)

Skull stripping is then done ***

Freesurfer is then run on all anatomicals***

5. Preprocess anatomicals

prep_raw_data.m

As part of *prep_raw_data.m* all of the anatomicals to be registered ar

Skull stripping doesn't work well but good enough for most steps. We tried BET, manual stripping, 3dSkullStrip, and FreeSurfer. Perfect skullstripping is essential for nonlinear alignment

All anatomicals head coil only)

Skull stripping is then done ***

Freesurfer is then run on all anatomicals***

Need to improve freesurfer for youngest children (under 1 year)

5. Extract centroid TR

prep_raw_data.m

Run mcFLIRT as per usual

Extract the centroid TR

Finds the euclidean distance of XYZ values of all TRs against all TRs. Finds the minimum mean*** distance

5. Extract centroid TR

prep_raw_data.m

Run mcFLIRT as per usual

Extract the centroid TR

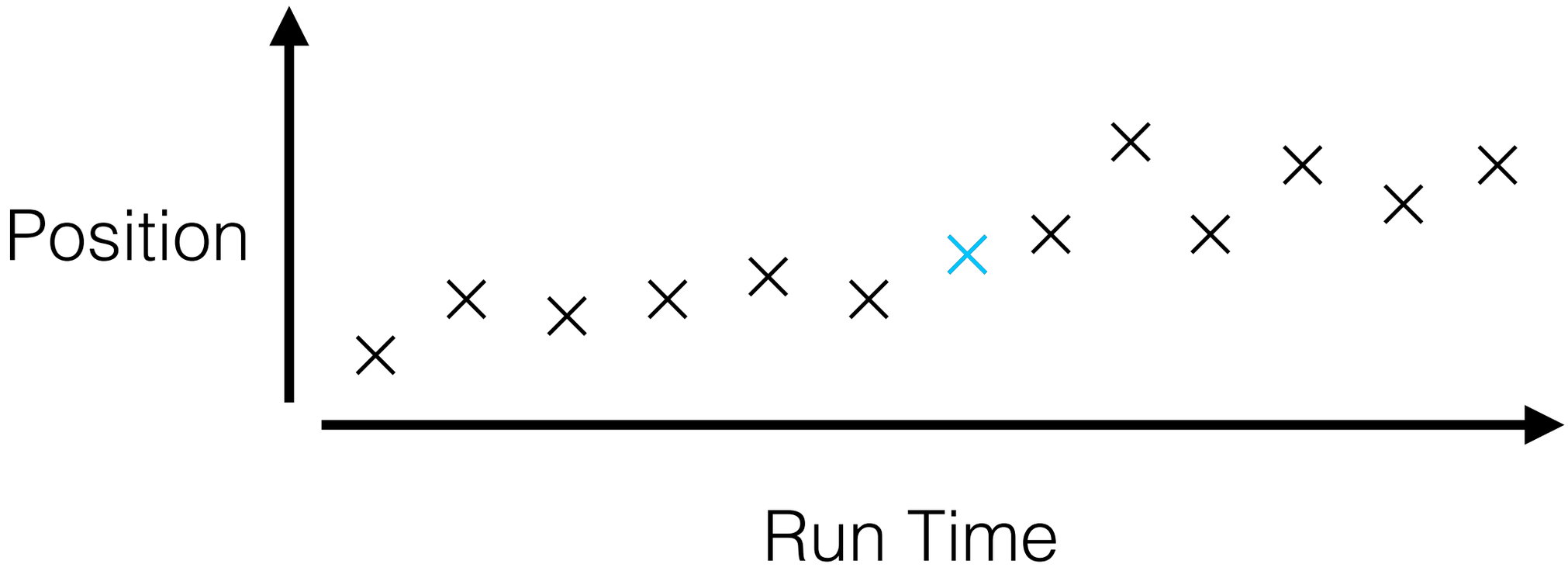
Finds the euclidean distance of XYZ values of all TRs against all TRs. Finds the minimum mean*** distance

Could use median instead. No clear reason why one vs the other

5. Extract centroid TR

prep_raw_data.m

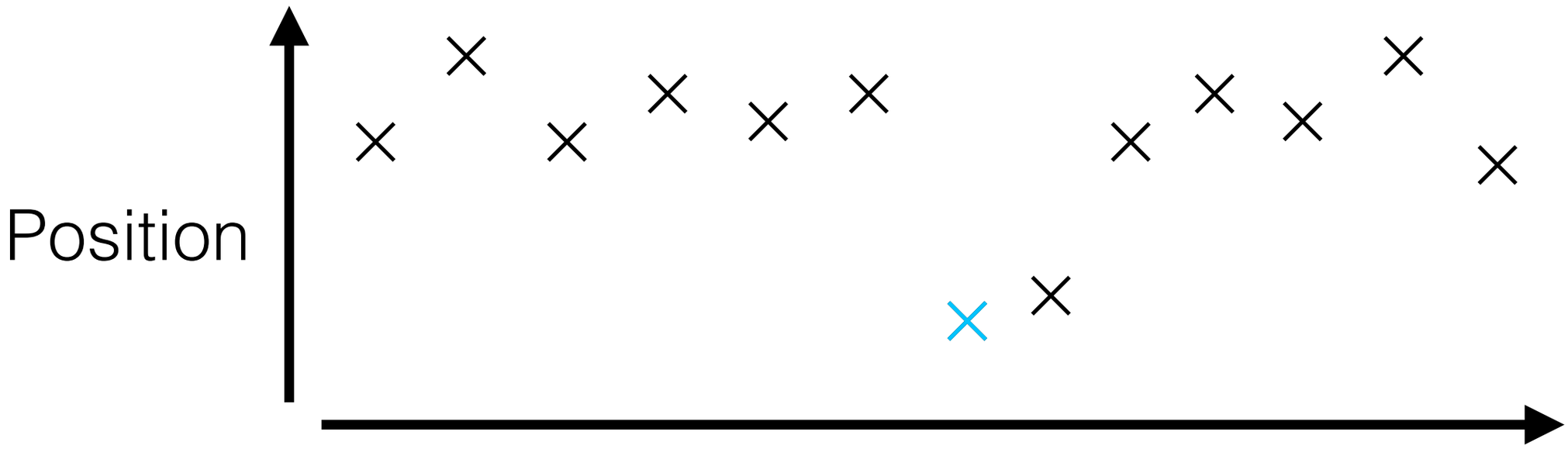
Example data where the middle TR makes sense



5. Extract centroid TR

prep_raw_data.m

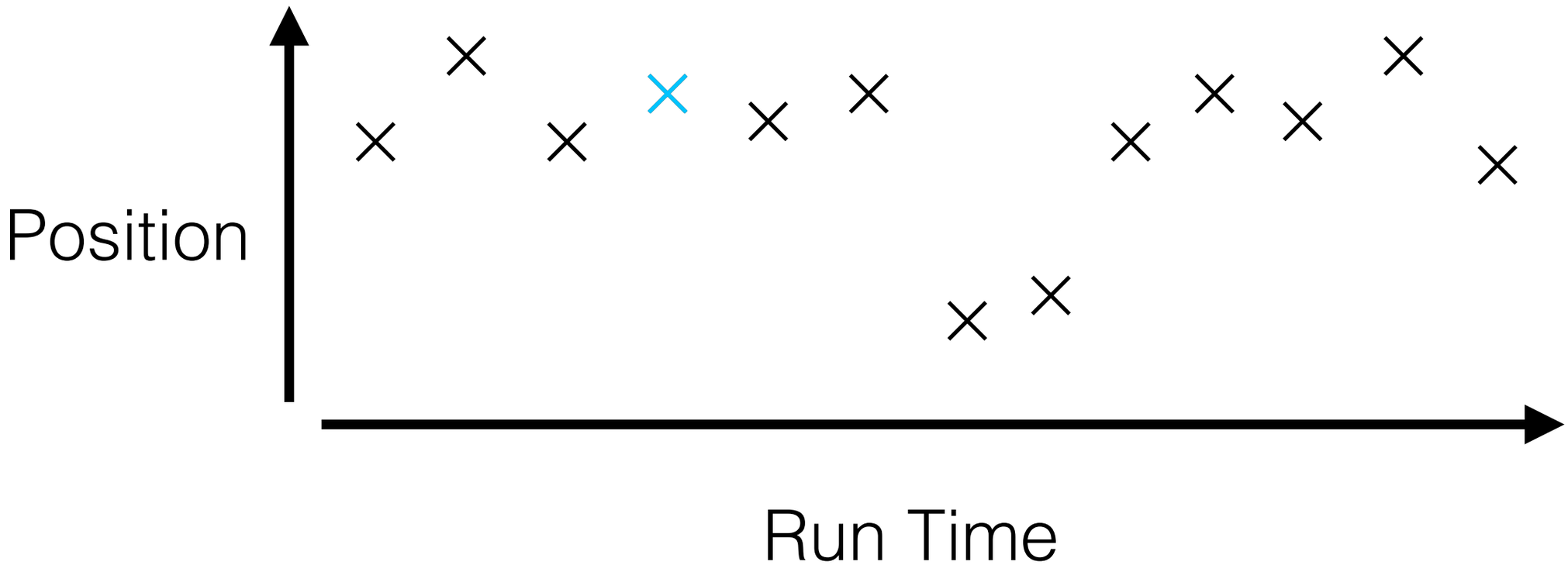
Example data where the middle TR doesn't make sense



5. Extract centroid TR

prep_raw_data.m

Example data where the centroid TR does make sense

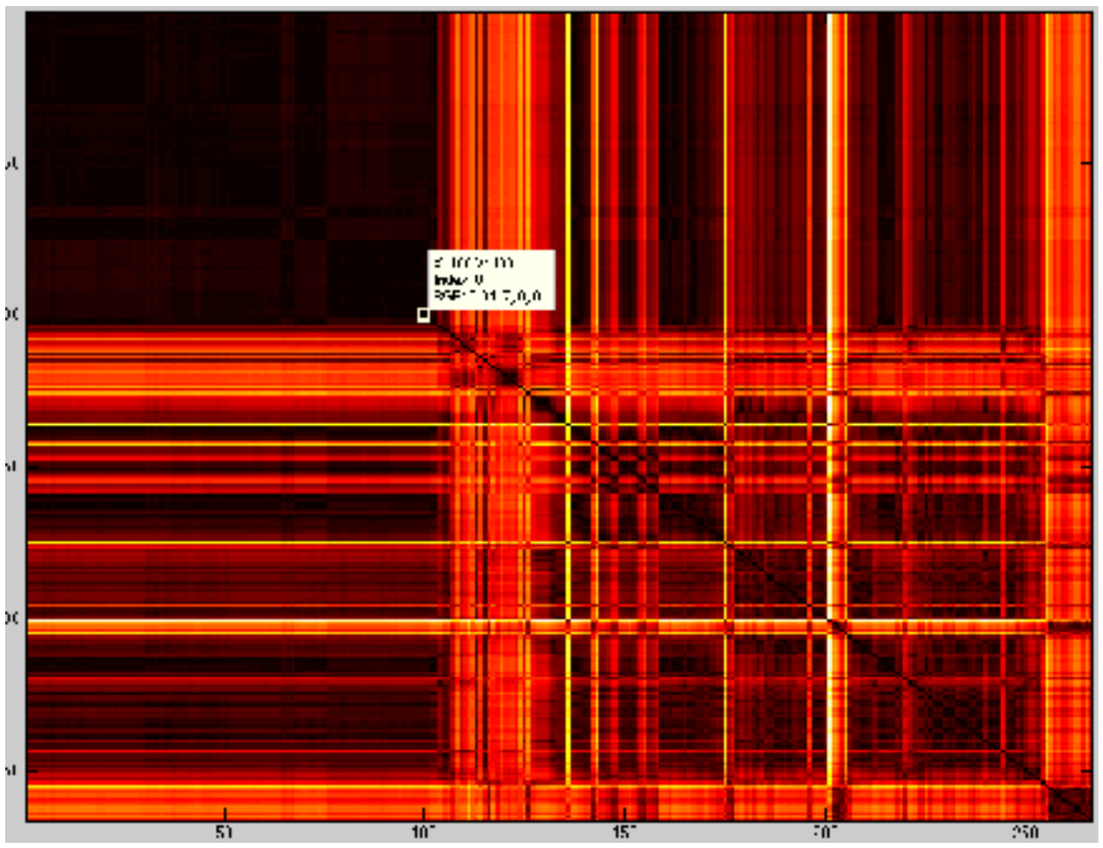


5. Extract centroid TR

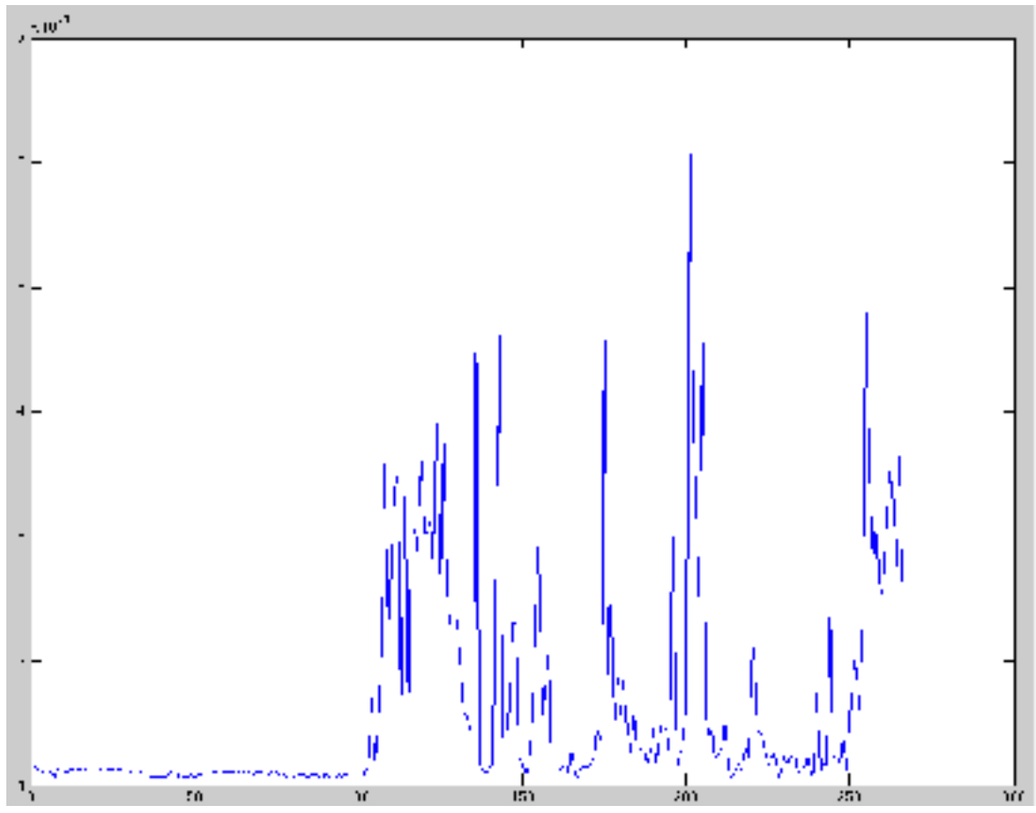
prep_raw_data.m

Default MCFLIRT, using middle TR as reference

Time point by time point distance matrix



Time point displacement

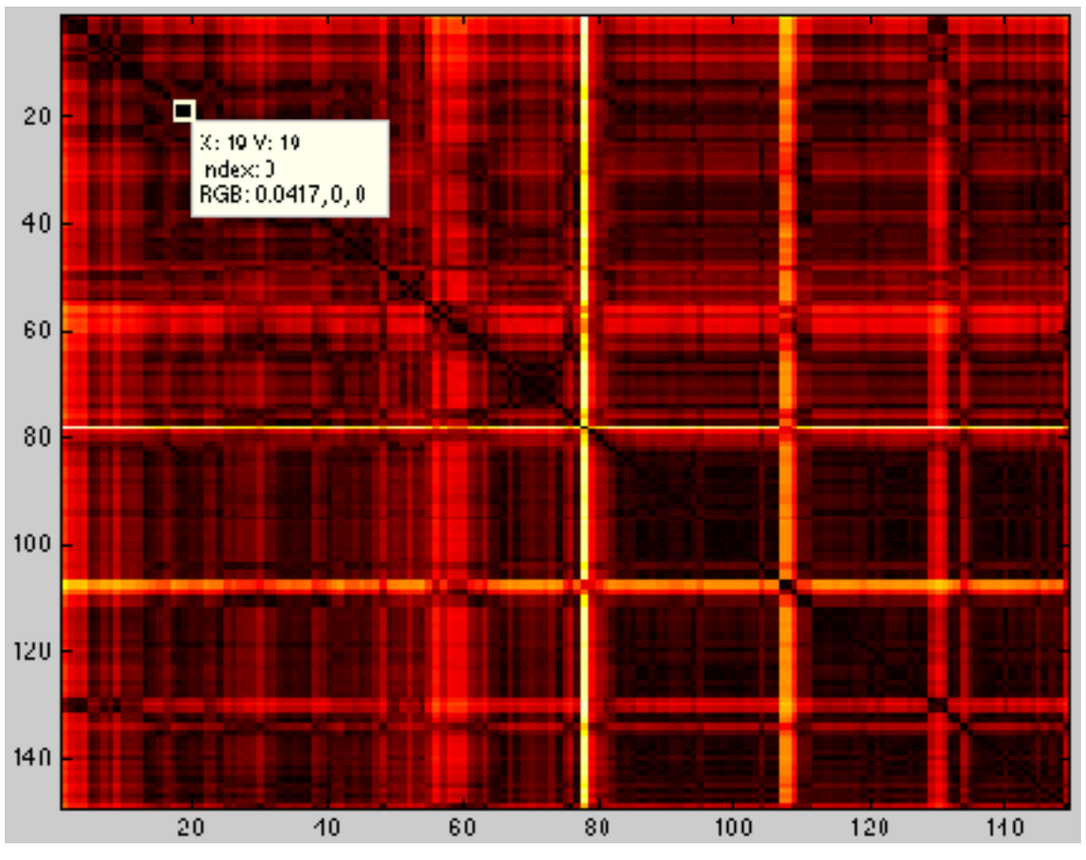


5. Extract centroid TR

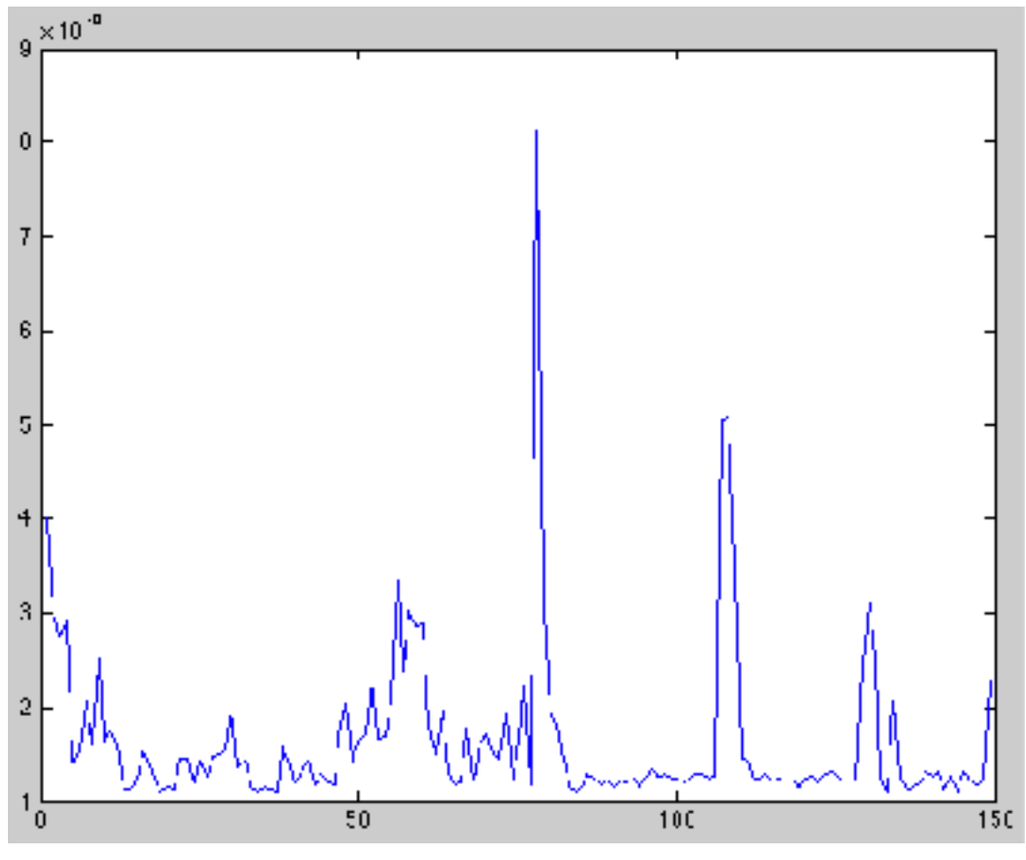
prep_raw_data.m

Using centroid TR as reference

Time point by time point distance matrix



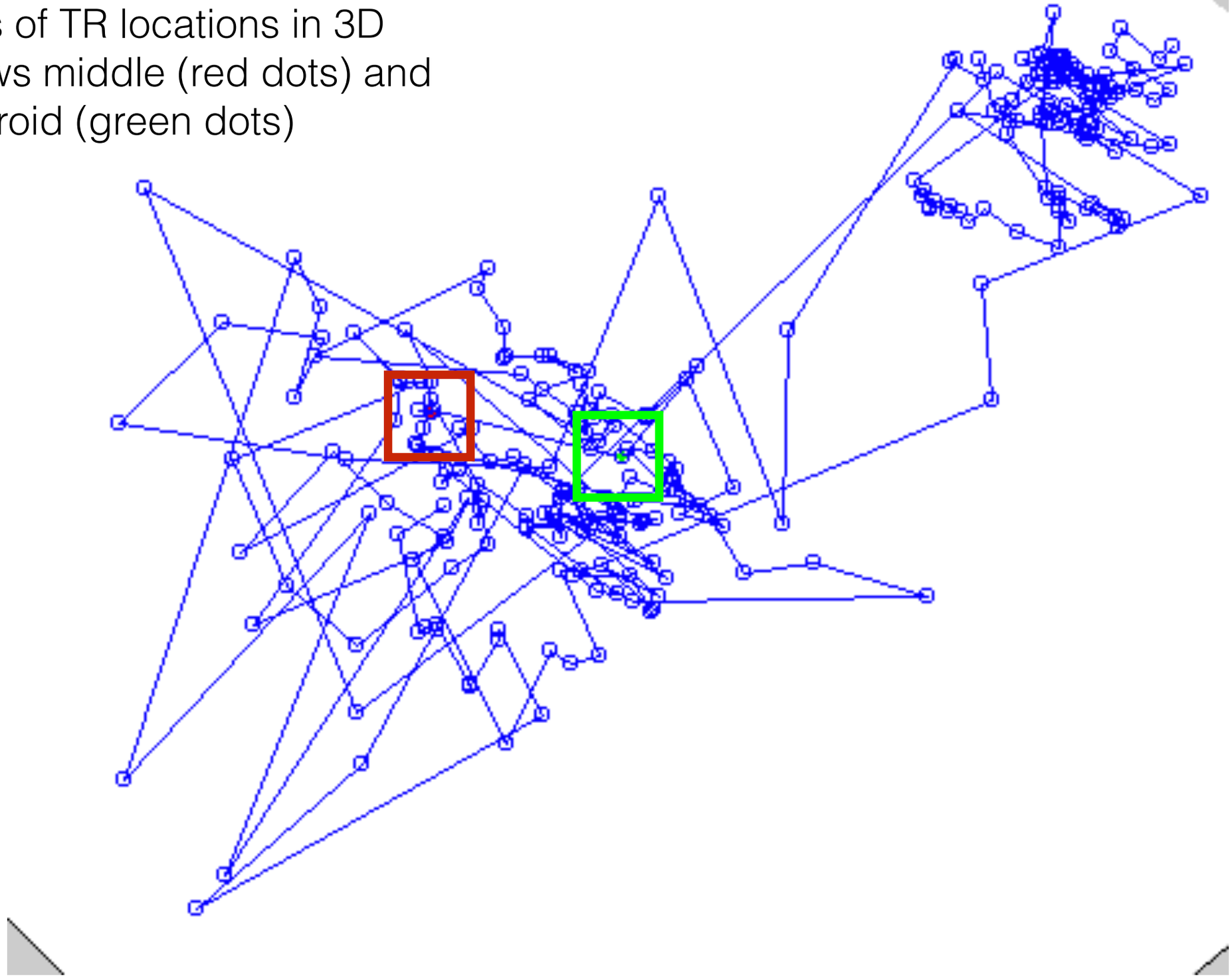
Time point displacement



5. Extract centroid TR

prep_raw_data.m

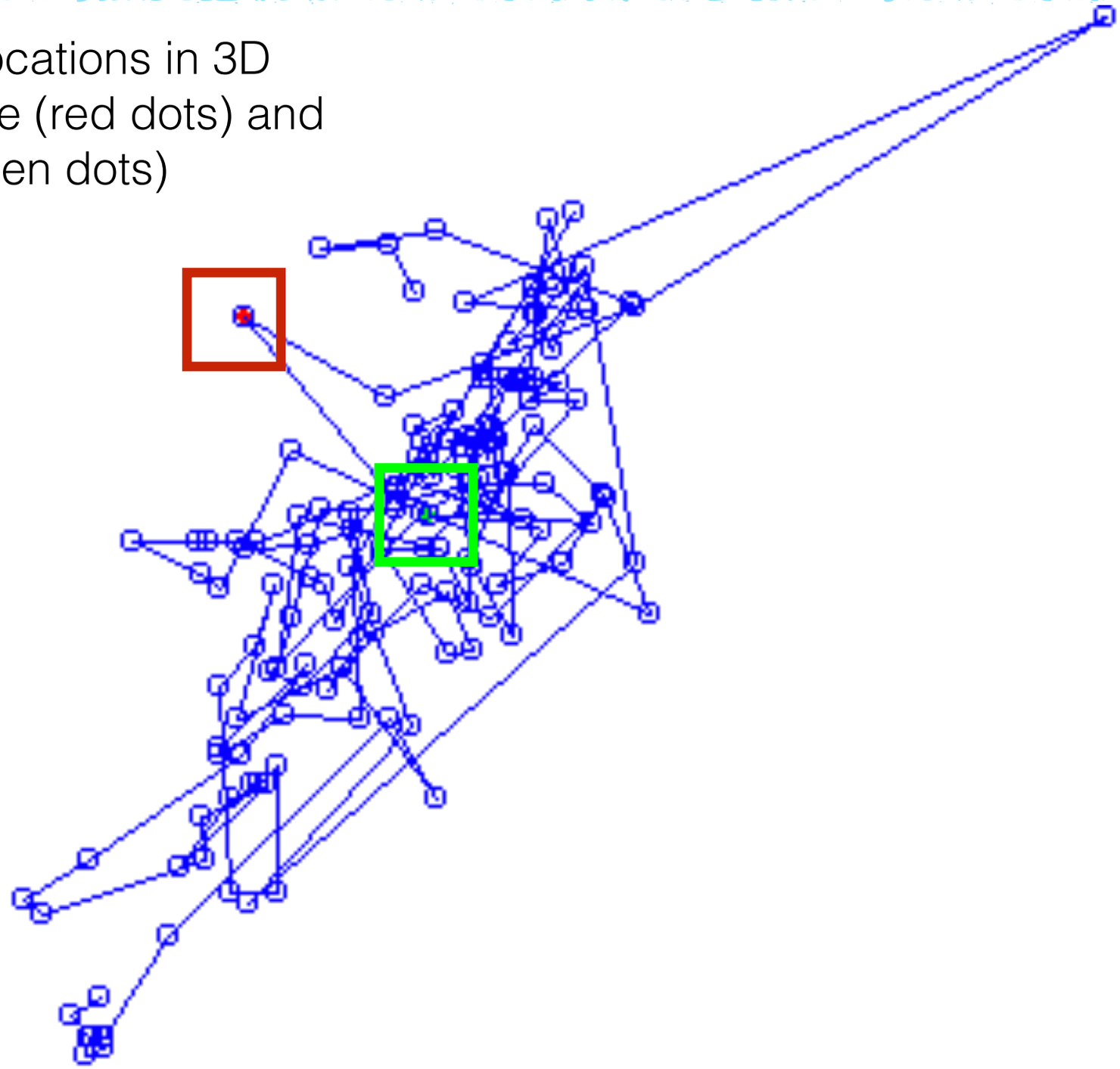
Examples of TR locations in 3D space. Shows middle (red dots) and centroid (green dots)



5. Extract centroid TR

prep_raw_data.m

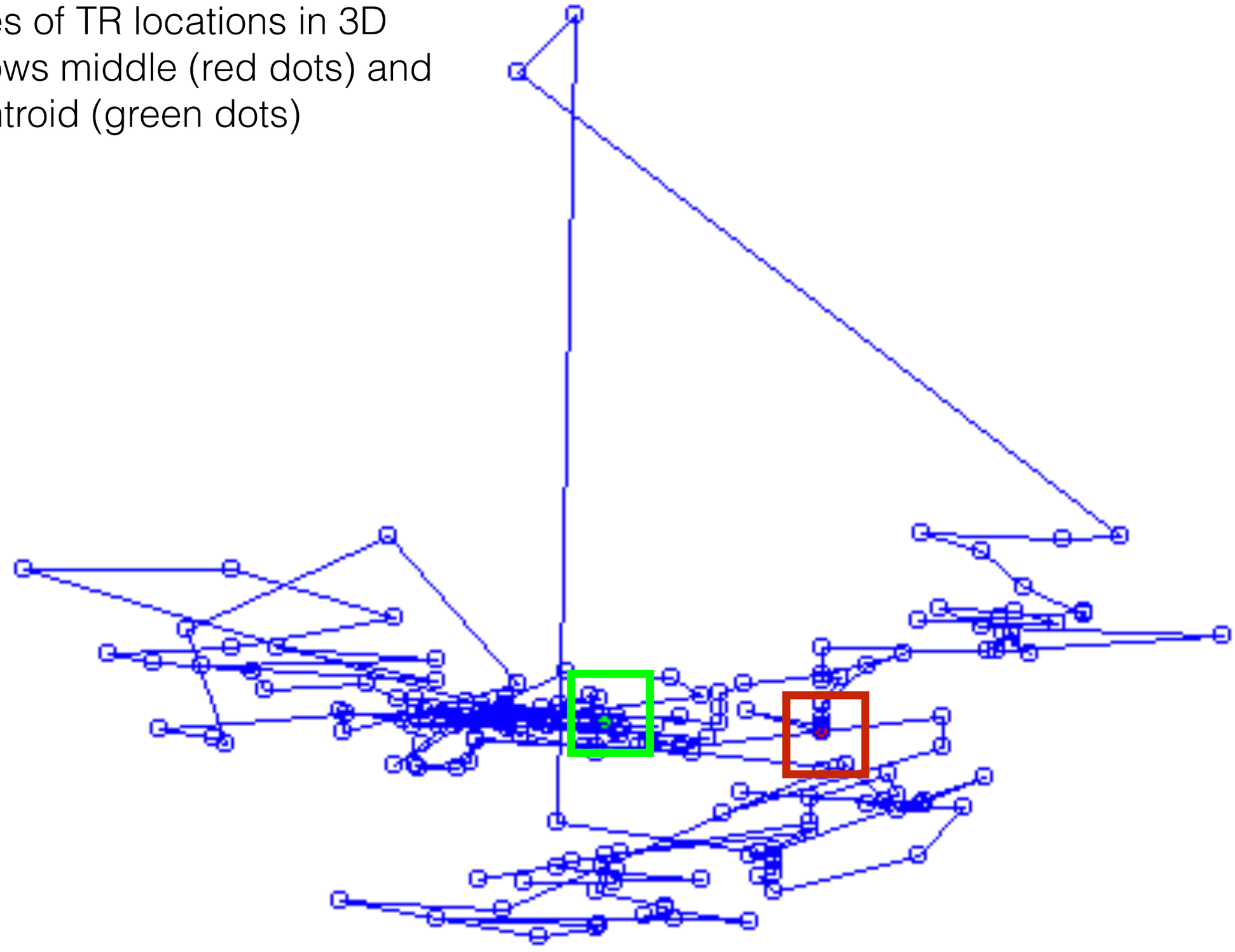
Examples of TR locations in 3D space. Shows middle (red dots) and centroid (green dots)



5. Extract centroid TR

prep_raw_data.m

Examples of TR locations in 3D space. Shows middle (red dots) and centroid (green dots)



5. Extract centroid TR

prep_raw_data.m

If the TR chosen as the centroid is to be excluded (based on recomputed motion parameters) then the algorithm will go to the TR that is next centre most

5. Identify outliers

prep_raw_data.m

The centroid TR is used as a reference to create a vector describing TRs with at least 3mm of motion***, relative to the last TR. Uses a custom version of *fsl_motion_outliers* found in $SUBJ_DIR/*scripts/fsl_motion_outliers.sh* which accepts a reference TR

5. Identify outliers

prep_raw_data.m

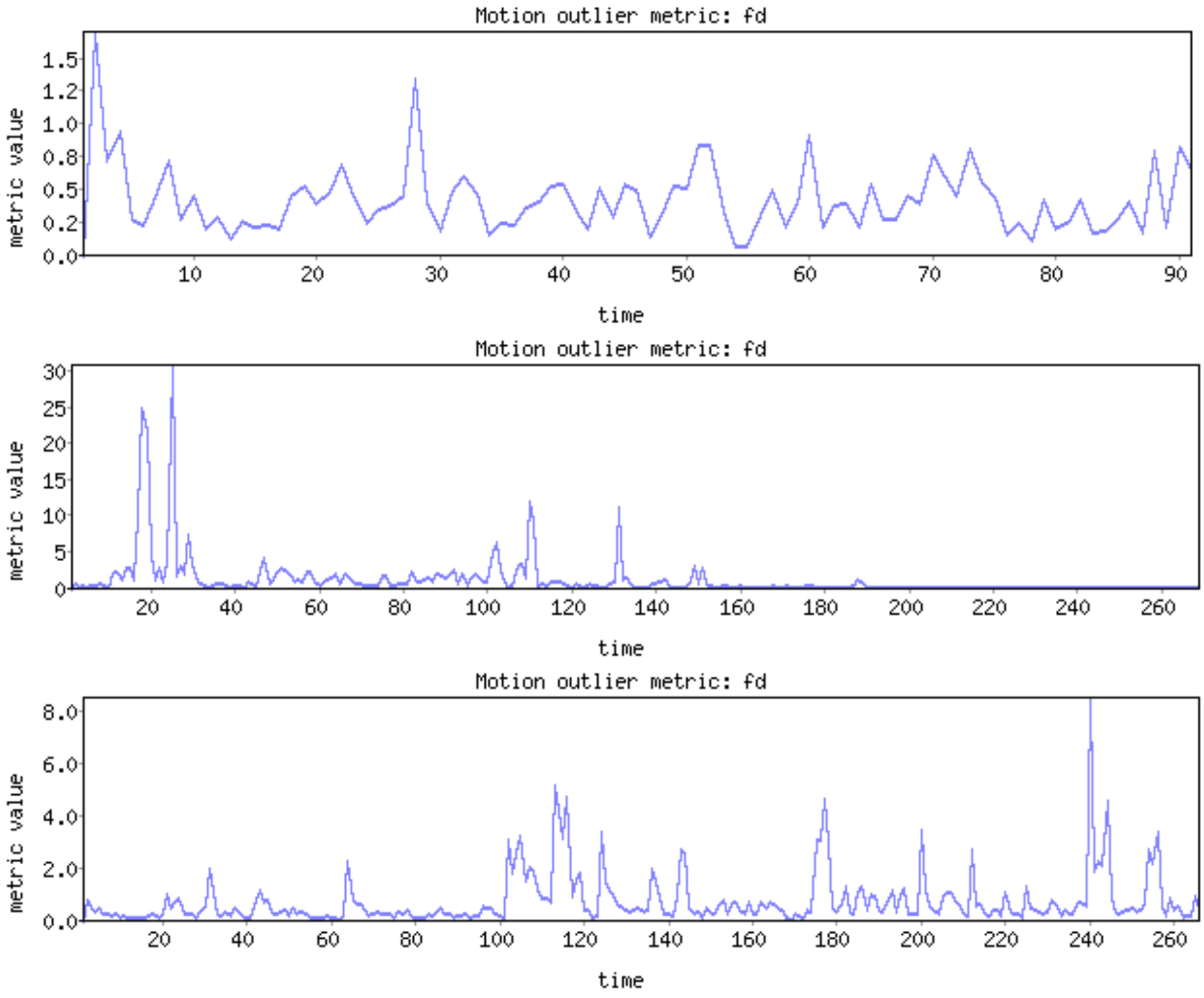
The centroid TR is used as a refer
vector describing TRs with at leas
relative to the last TR. Uses a cust
fsl_motion_outliers found in $SUB
fsl_motion_outliers.sh which accepts a reference TR

Could use a different threshold. Stricter thresholds might increase data quality but decrease data yield

5. Identify outliers

prep_raw_data.m

Example frame wise displacements for infant data, measured in mm



5. Identify outliers

prep_raw_data.m

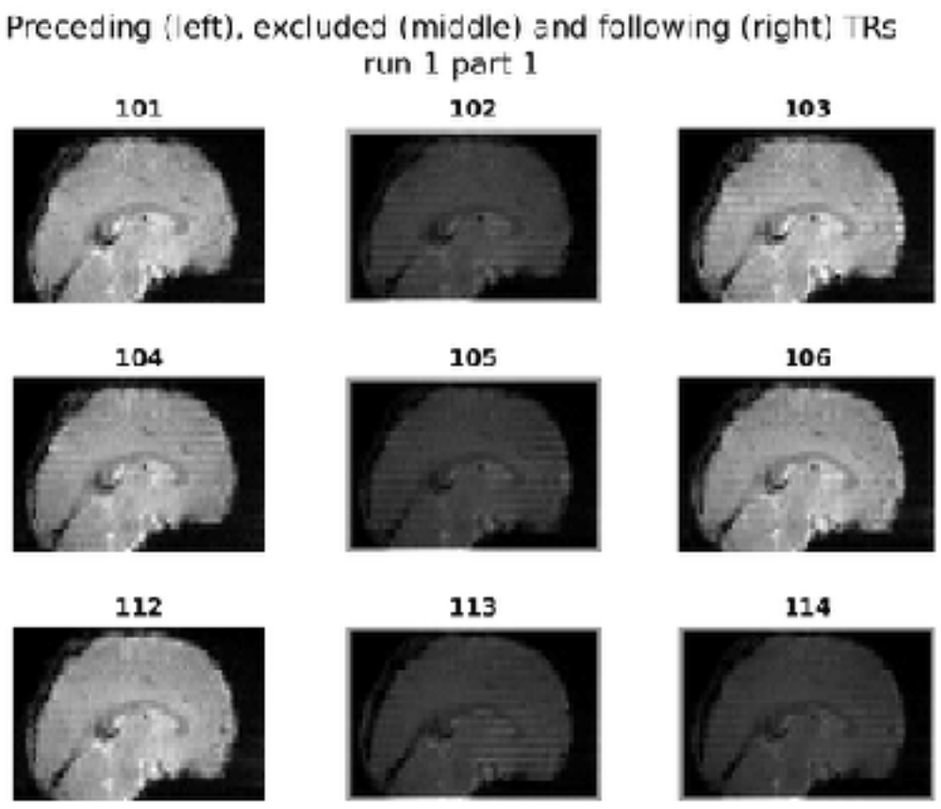
The reason why you make the motion parameters here, rather than using the defaults in FEAT, is so that you can decorrelate the motion parameters and append the confound regressors.

The decorrelation is so that the regressors are not nearly identical. This is because correlated regressors do not pass the SVD check when creating the GLM.

5. Report outliers

prep_raw_data.m

Along with motion parameters and other relevant information, the '$SUBJ_DIR/analysis/firstlevel/Confounds' folder will contain images showing the TRs that are reported as outliers, as well as the TR immediately before and after it



5. Non-standard parameters

prep_raw_data.m

It is possible to use the prep_raw_data.m script to create motion preprocessed data with specific parameters.

For instance you might want to run it with 1mm motion threshold. You could do this by running the command: ‘sbatch $SUBJ_DIR/scripts/run_prep_raw_data.sh [7,8,9] 3 2 fslmotion_threshold 1’

This is also useful if you have a non-standard burn in for a run or a non-standard TR.

6. Set up Gaze coding

Gaze_Categorization.m

If you collected eye data using the assumed data collection procedure (*Utils_EyeTracker_FrameGrabber_Record.py* from the Experiment Menu*)* then you should have two types of outputs: a text file with time stamps and a video (or frames if you used a legacy version of that code), as well as the mat file directly from Experiment Menu.

Text file with timestamps (in eye tracker nanoseconds) and messages sent from the Menu matlab code. These messages have timestamps that allow handshakes to occur
A video, storing each frame separately

We need this data to be organized according to the experiments and blocks it was collected for. The code *$PROJ_DIR/scripts/Gaze_Categorization/generate_timing_file.m* takes the video, splits it into frames and gives each frame the name of the timestamps in the text file (there *must* be the same number of frames as timestamps)

Another file is also created that assigns each frame to an experiment block and event.

The order of events are then shuffled for coding to reduce the ability for coders to guess the condition labels

6. Code Eye Data

Gaze_Categorization.m

Take each epoch of an experiment (might be a 0.5s trial, might be a whole movie) and have coders evaluate it using the criteria specified.

The script *$PROJ_DIR/scripts/Gaze_Categorization_Responses.m* specifies the response options that can be given for a specific experiment. Make sure that you only add to this list, never subtract!

As mentioned in the README it is probably best not to run this script from the cluster due to needing psychtoolbox, instead you should have a dedicated eye coding computer

7. Review

At this stage you want to take stock and review the progress so far.

Review the QA using a browser like firefox pointed to *$SUBJ_DIR/data/qa/index.html*

Review the motion parameters and TRs labelled for exclusion in *$SUBJ_DIR/analysis/firstlevel/Confounds/*

Review the eye tracking coding using *$PROJ_DIR/scripts/Gaze_Categorization/Gaze_Categorization_Replay.m*

8. Exclude runs if necessary

Sometimes you may need to exclude runs manually (rather than wait for the code exclude it), for instance the child moved drastically out of the field of view.

You can do this by writing to *$SUBJ_DIR/analysis/firstlevel/Confounds/Excluded_Runs* the list of the run names

9. Edit matlab data

Although the pipeline deals with ~95% of the problems and nuances that come from having drastic variability across sessions, it cannot do it all

Hence it is sometimes necessary to make edits to the data on an ad hoc basis that will likely be completely unique to this participant.

To do this in a robust way, make a copy of the matlab behavioral file and then make a matlab script that performs all of the edits you desire, returning a file with the original matlab name

The next slide has an example of this edit script

9. Edit matlab data

```
% Edit data for $PPT

clear all

load data/Behavioral/PPT_Original.mat

% In a run where we switched between StatLearning and Posner, we
% accidentally switched to PlayVideo in between. This only lasted for a few
% TRs and no movie was shown (hence it counted as 'rest' after
% StatLearning) but it would interfere with Analysis_Timing.
% Analysis_Timing assumes that PlayVideo Block_1 will only be played when
% the run has ended, hence it will assume that the Posner blocks to come
% after StatLearning are part of a separate run. This assumption is built
% in because it is true in almost all cases, but this one. Hence, you can
% append the PlayVideo TRs onto the ebd of StatLearning and act as if that
% accidental PlayVideo block never existed.
concat_TRs = [Data.Experiment_StatLearning.Block_12_1.Timing.TR, ...
    Data.Experiment_PlayVideo.Block_1_6.Timing.TR];
Data.Experiment_StatLearning.Block_12_1.Timing.TR = concat_TRs;

% Delete the block of PlayVideo
Data.Experiment_PlayVideo = rmfield(Data.Experiment_PlayVideo, 'Block_1_6');

% Remove the run from the run order
Data.Global.RunOrder = [Data.Global.RunOrder(1:18, :); Data.Global.RunOrder(20:end, :)];

% Save data
save('data/Behavioral/PPT.mat');
```

10. Create the fsf scripts

render-fsf-templates.sh

This creates fsf files used for first level analyses. This script makes assumptions about what the anatomical and infant standard should be.

This needs to be run after *prep_raw_data* (step 5) since you should probably use the skullstripped anatomical

This must be run before *Analysis_Timing* (step 11) because that script will edit these files based on whether pseudo runs or exclusions are necessary

10. Create the fsf scripts

render-fsf-templates.sh

Like other scripts, this can be hijacked with additional inputs in order to get non-default behavior.

For instance, the third input is the non-default name you want to save the fsf file with.

11. Organize behavior and timing data

Analysis_Timing.m

$SUBJ_DIR/scripts/Analysis_Timing.m is very complicated script that has the hard task of taking each unique participant session and reorganizing it in such a way that it is possible to have consistency across participants.

In service of this, it has three main functions:

1. Get all the eye tracking data and motion exclusions to decide what blocks to exclude. Functionality is included for analyzing eye tracking data from EyeLink as well as the FrameGrabber solution that we use in our work.

2. Get all the timing information and make timing files

3. Divide the data into pseudo runs if necessary

There are a lot of text outputs in this function that should help to explain what happened in each block and then summarize within a run. When there are pseudo runs there is a lot of output as the data is carved up based on the experiments that were done in an original functional run.

11. Aggregate eye tracking data across coders

Analysis_Timing.m

Script: *$SUBJ_DIR/scripts/Analysis_Timing_functions/EyeTracking_Aggregate.m*

Aggregate all of the eye tracking data across coders

Record the proportion of frames collected

Determines the average timecourse of responses by finding the modal response across coders for a 5 frame window.

Blue box shows how the first frame is coded by taking the available responses within the window (cannot take 2 before the first frame).

Ties are resolved by using previous response as in the green ‘2’

Coder A	1	1	1	1	1	2	2	2	0	0	0	0
Coder B	1	1	1	1	1	1	2	2	2	0	0	0
Coder C	2	2	2	2	2	2	2	2	2	2	2	1
Aggregate	1	1	1	1	2	2	2	2	2	2	0	0

11. Coder reliability

Analysis_Timing.m

Script: *$SUBJ_DIR/scripts/Analysis_Timing_functions/EyeTracking_Reliability.m*

Extract the reliability and validity of the eye tracking coding

Two metrics of reliability (across all response types):

1. Intraframe: proportion of frames coded the same between coders who saw the same frame.
2. Interframe: proportion of frames coded the same between adjacent frames (including within the same coder if the coders coded every frame)

Two possible metrics of validity:

1. Calibration: Accuracy of second half modal response for a given stimulus position
2. Posner: Accuracy of left/right post-target modal resp

11. Do experiment-specific analyses

Analysis_Timing.m

Script: *$SUBJ_DIR/scripts/Analysis_Timing_functions/EyeTracking_Experiments.m*

Accepts functions to perform experiment specific analyses/plots e.g. Posner, StatLearning of eye tracking.

Can also be used to prepare labels for events if behavior is used to organize the timing files

11. Exclude epochs

Analysis_Timing.m

Script: *$SUBJ_DIR/scripts/Analysis_Timing_functions/EyeTracking_Exclude.m*

Decide which events to exclude (>50% off-screen***)

The default value is >50% frames coded as off screen but this can be set to different thresholds for different experiments.

For some experiments, other responses will warrant exclusion, such as looking on the wrong side for a stimulus

It is possible to set a critical window. For instance, in PosnerCuing, the period in which the cue appears is a critical window, if they don't look during that period the trial is dropped.

You can also set the criterion for exclusion to be different for different epochs in the experiment.

11. Exclude epochs

Analysis_Timing.m

Script: *$SUBJ_DIR/scripts/Analysis_Timing_functions/EyeTracking_Exclude.m*

Decide which events to exclude (>50% off-screen***)

The default value is >50% frames coded as off s
different thresholds for different experiments.

For some experiments, other responses will warra
looking on the wrong side for a stimulus

It is possible to set a critical window. For instance
in which the cue appears is a critical window, if they don't look during that
period the trial is dropped.

You can also set the criterion for exclusion to be different for different epochs
in the experiment.

Could use a different threshold. This really depends on which experiments you are doing and whether you are doing GLM or other types of analyses

11. Counting the number of Hrs

Analysis_Timing.m

Sometimes TR triggers are missed during data collection. This is fixed automatically by assuming that the scanner does not start and stop within a block, hence any missing TRs within a run can be interpolated.

For some experiments, e.g. StatLearning, the code was set up in a less standardized way and so the timestamps were not labelled using the appropriate format. These timestamp names are corrected here.

The code determines number of burn in TRs (based on TRs received before experiment starts). Throws warning if not 3 but this default could be changed.

Code makes a guess when the scanner stops to estimate the number of TRs in a run which is then compared to the actual number of TRs.

At the end of each run there is an output in Analysis_Timing that tells the number of estimated TRs and the actual number. Errors here are always important, however these can happen when you have pseudo runs and these will not always be problematic. Nonetheless, always investigate them.

11. Exclude blocks

Analysis_Timing.m

Exclude events or blocks with excessive numbers of motion TRs (>50%***) or because they were quit halfway through or because the eye tracking analysis determined that they should be excluded.

If all blocks from a run are excluded then the run will be too.

11. Exclude blocks

Analysis_Timing.m

Exclude events or blocks with excessive numbers of motion TRs (>50%***) or because they were quit halfway through c ing analysis determined that t

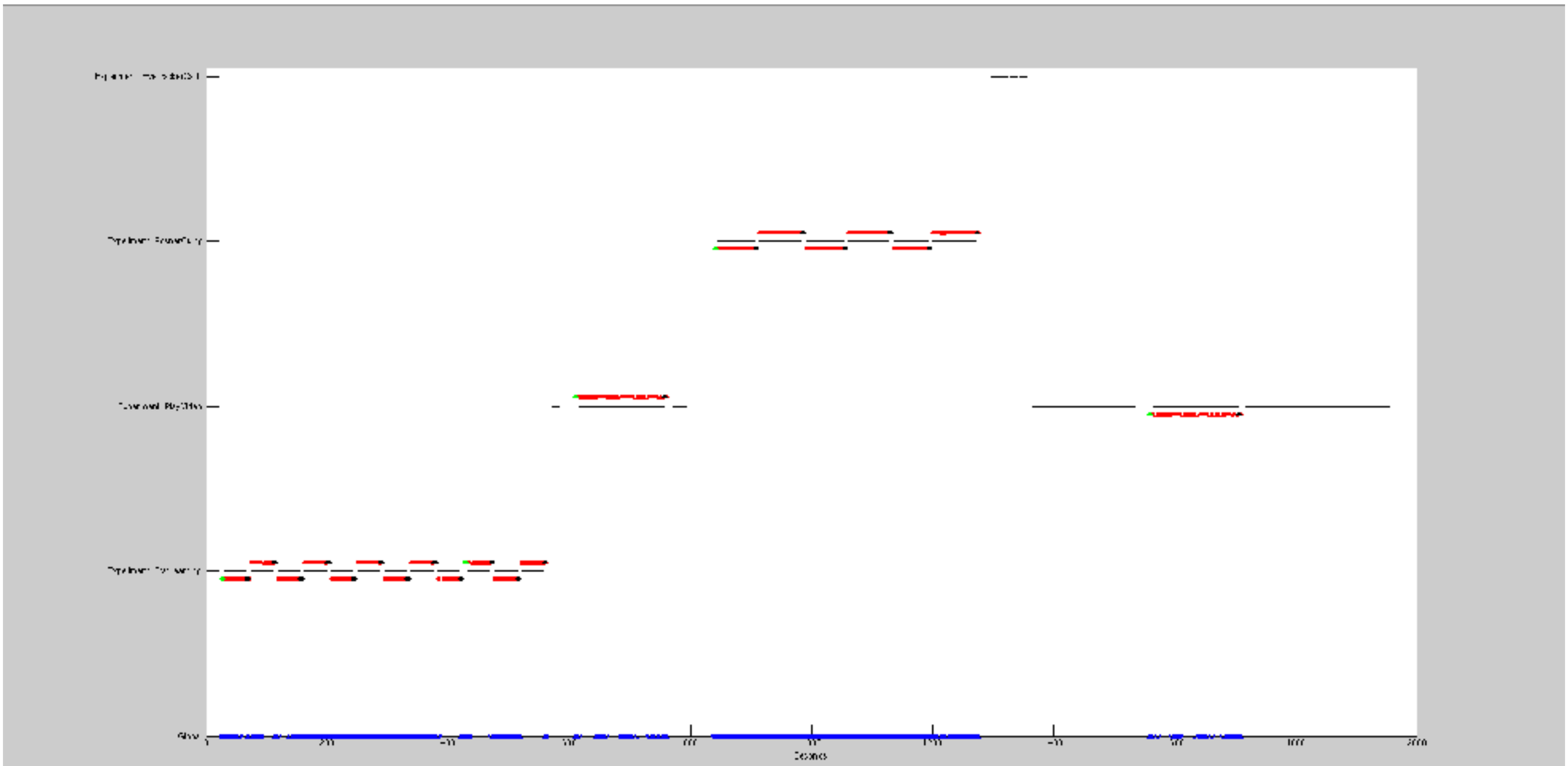
Threshold can be changed and will depend on the type of analyses to determine what is appropriate

If all blocks from a run are excluded then the run will be too.

11. Report experiment sequence

Analysis_Timing.m

Sequence of blocks. Red filled circles are caught TRs, Red unfilled circles are interpolated TRs (because the trigger was missed). Green circles are burn in, Black circles are burn out TRs. Black lines are the block duration time. Blue circles are all TRs, unsorted by experiment



11. Prepare Timing data

Analysis_Timing.m

Collect experiment onset time and duration

All scanning experiments (e.g. not gaze calibration) have timing files created that specify the block start and duration.

If you create new experiments that you want to have timing files then make a file '*$PROJ_DIR/prototype/link/scripts/Analysis_Timing_functions/Timing_$EXPERIMENT.m*'. Critically, these files must determine when each block onsets and its duration.

Some experiments (e.g. PosnerCuing) have event timing specified for each block. These events can then be assigned weights

Some experiments (e.g. PosnerCuing) have condition labels specified. Each event is assigned a condition and conditions are aggregated across blocks

11. Create Timing Files

Analysis_Timing.m

$PROJ_DIR/prototype/link/scripts/Analysis_Timing_functions/Timing_MakeTimingFile.m organizes the timing information for experiments and creates timing files at both the first level (in run time) and at the second level (in experiment time). The former are used by other scripts (e.g. *FunctionalSplitter*) to ensure reliability

Block and event timing files are made ‘automatically’. To make condition files, you must create a function which gives each event a condition label

This function *cannot* handle events with overlapping time courses. In other words, if event A starts at 0s and ends at 10s, you cannot make an event B that starts at 0s and ends at 8s. To get this functionality you would have to make an event A’ that goes from 8s to 10s and add that to B in order to get A.

11. Pseudoruns

Analysis_Timing.m

$PROJ_DIR/prototype/link/scripts/Analysis_Timing_functions/pseudorun_divide.m takes the information from a run and determines whether more than one experiment has usable data in a run. If so then that run is split.

Runs are given a letter appended to their run number and treated in subsequent analyses as if they were separate runs

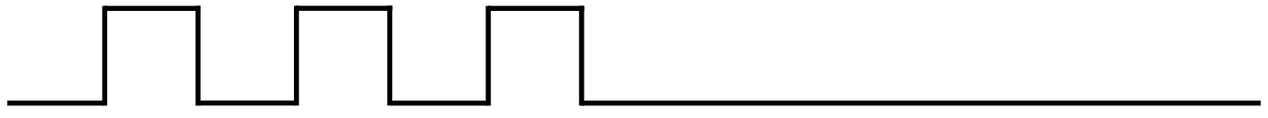
This is so that any preprocessing and exclusions are as isolated as possible. This is particularly important for Z scoring within runs (which FunctionalSplitter would fix anyway). Temporal filtering is also affected. Next slide gives an example why.

11. Reason for pseudoruns

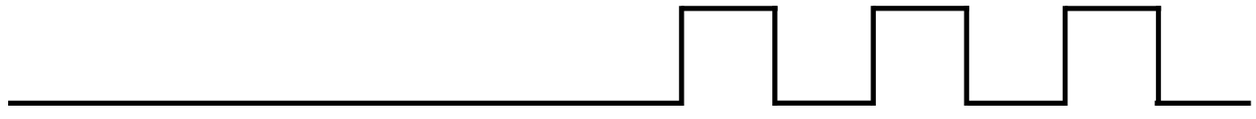
Analysis_Timing.m

Design matrix

Visual expt



Audio expt



Hypothesized voxel responses

Visual

Audio

If you z scored the whole run then voxels that are visually responsive will have a strong negative response (red) during auditory stimulation and vice versa, purely due to z scoring

11. Excluded runs

Analysis_Timing.m

If no blocks from a run (or pseudorun) are usable, the run will not be preprocessed or used any further. To enact this, the fsf files are renamed with the suffix '_excluded_run.fsf' to indicate a feat should not be run.

If a run was previously excluded but now no longer should be (e.g. you added more gaze coding that changed what is included) then rerunning *Analysis_Timing* will revert the fsf name back

12. Preprocess functional

FEAT_firstlevel.sh

FEAT makes some assumptions about the data which we violate in this pipeline; hence we use a slight variant of this script:
FEAT_firstlevel.sh $fsf_file

If you want to run different versions of the same FEAT then add a suffix to the name (e.g. functionalXX_new.feat). Then in subsequent steps you can analyze these runs separately. *However*, the timing files are assumed to be the same across analyses

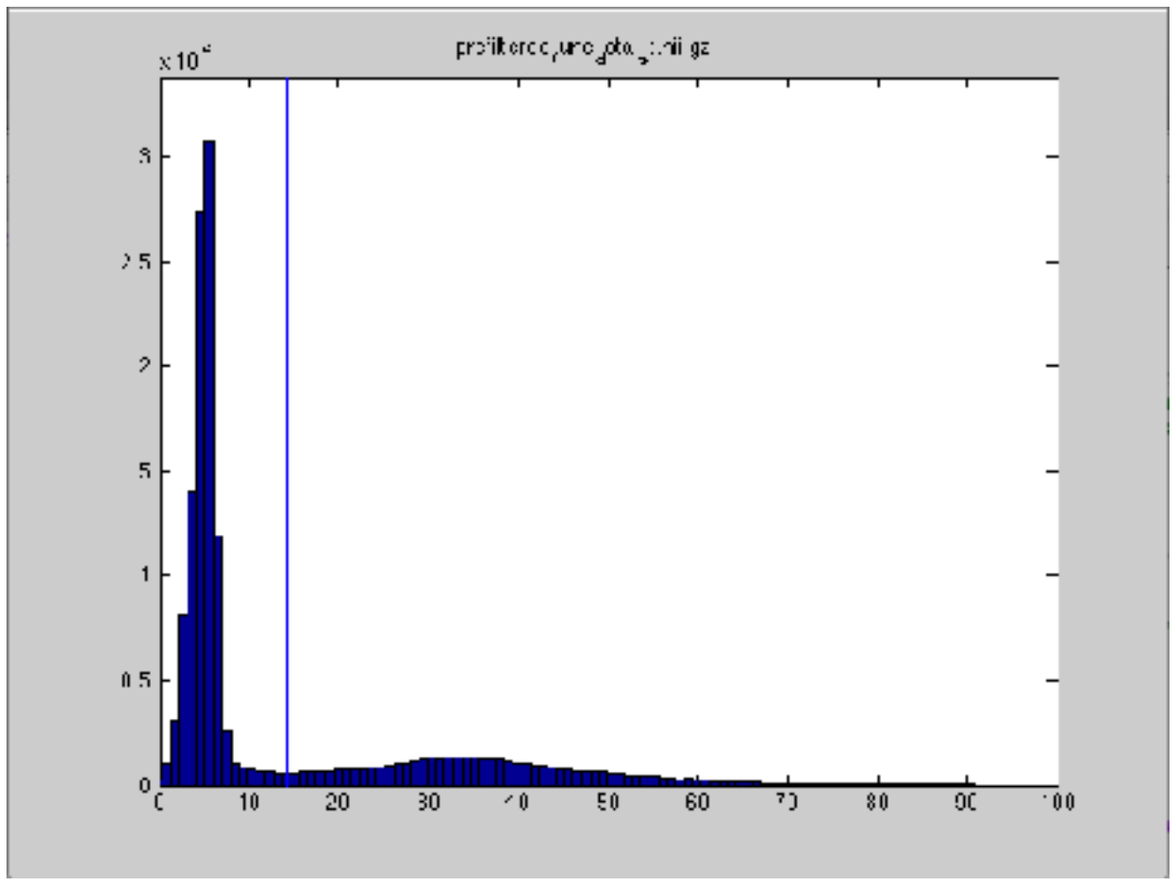
FEAT_prestats.sh performs registration and motion correction with the centroid TR

whole_brain_sfnr.m calculates SFNR of every voxel in a functional. This is used as the mask. Sometimes, if the brain volume is very noisy, this will fail in which case BET is used.

12. SFNR masking

FEAT_firstlevel.sh

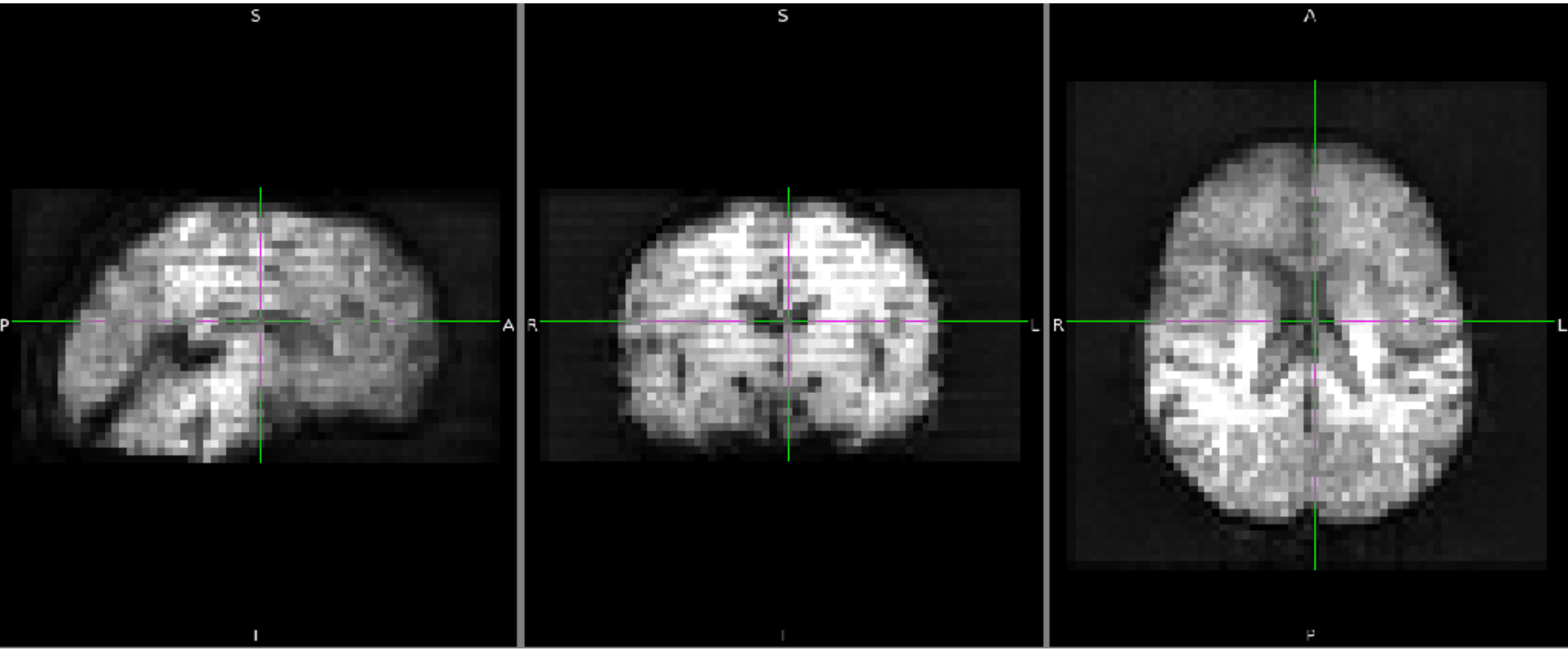
Plot a histogram of SFNR values for all voxels in the brain



12. SFNR masking

FEAT_firstlevel.sh

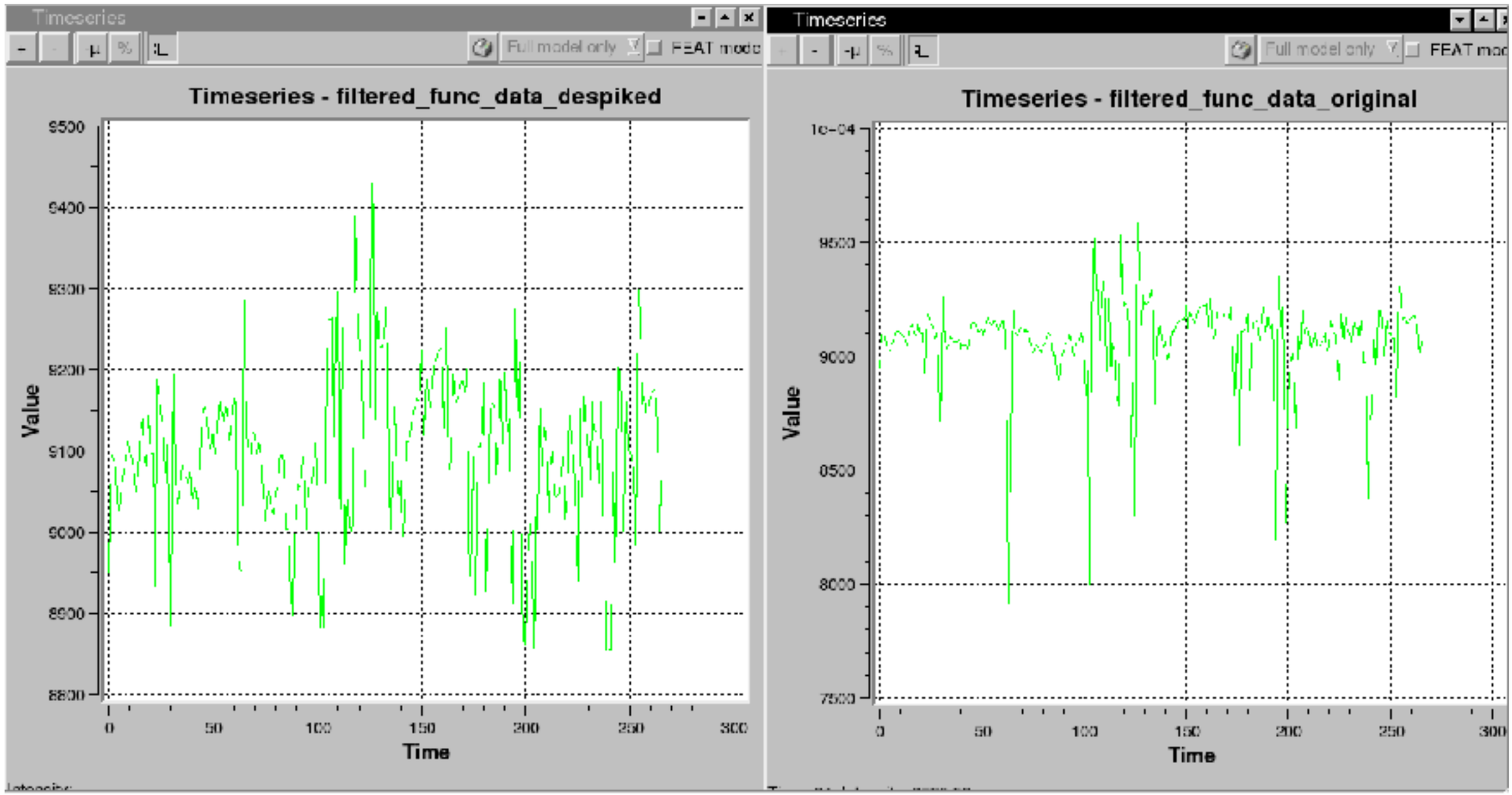
Intensity is the SFNR of each voxel



12. Preprocess functional

FEAT_firstlevel.sh

Voxelwise despiking is performed with *3dDespike*



13. Manual registration (first level)

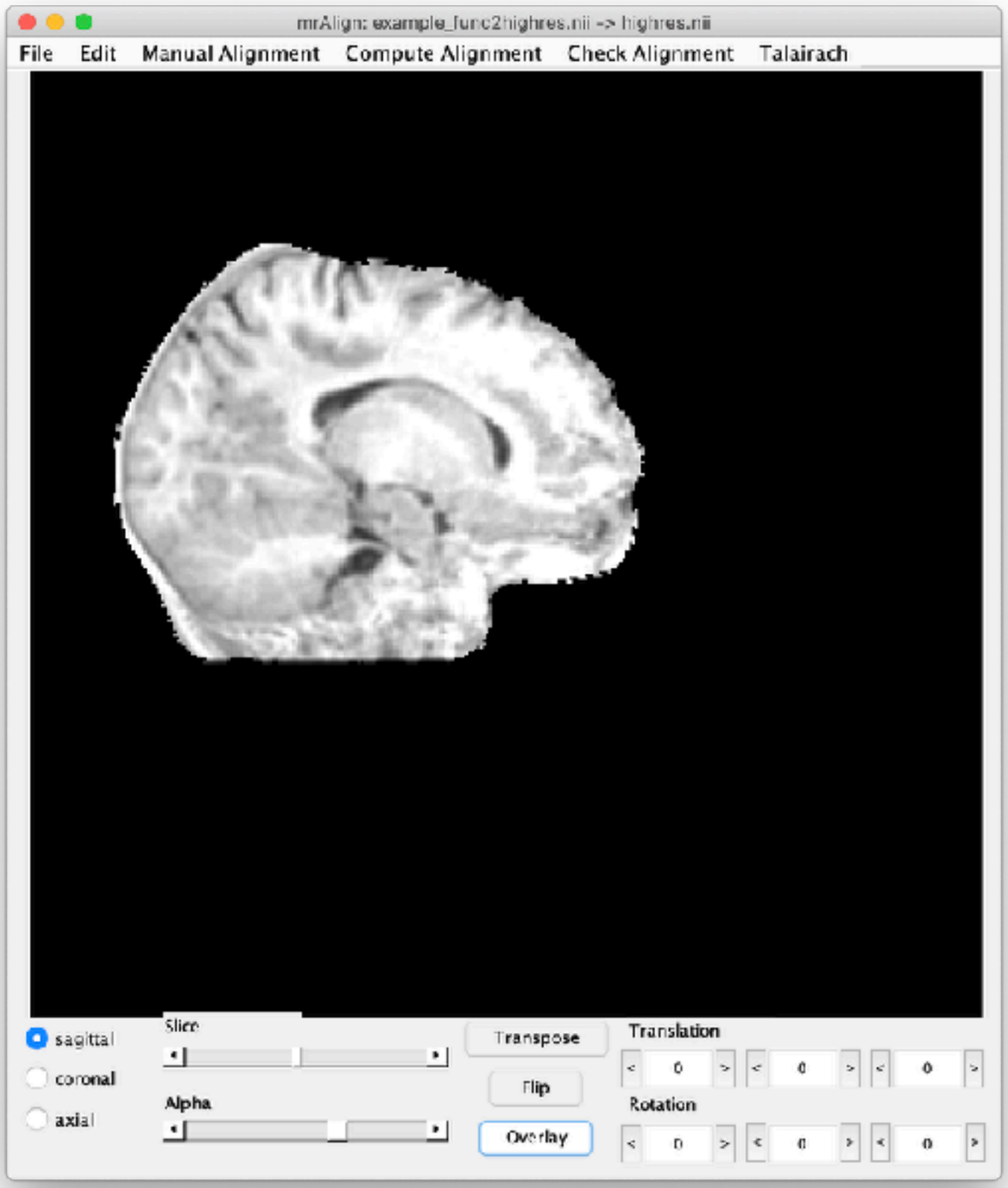
manual_registration.sh

The registration to highres at the first level is almost always poor and so manual registration is performed using mrAlign from the mrTools toolbox in the Gardner lab.

To help with manual registration, *$SUBJ_DIR/scripts/manual_registration.sh* has the step by step of doing this

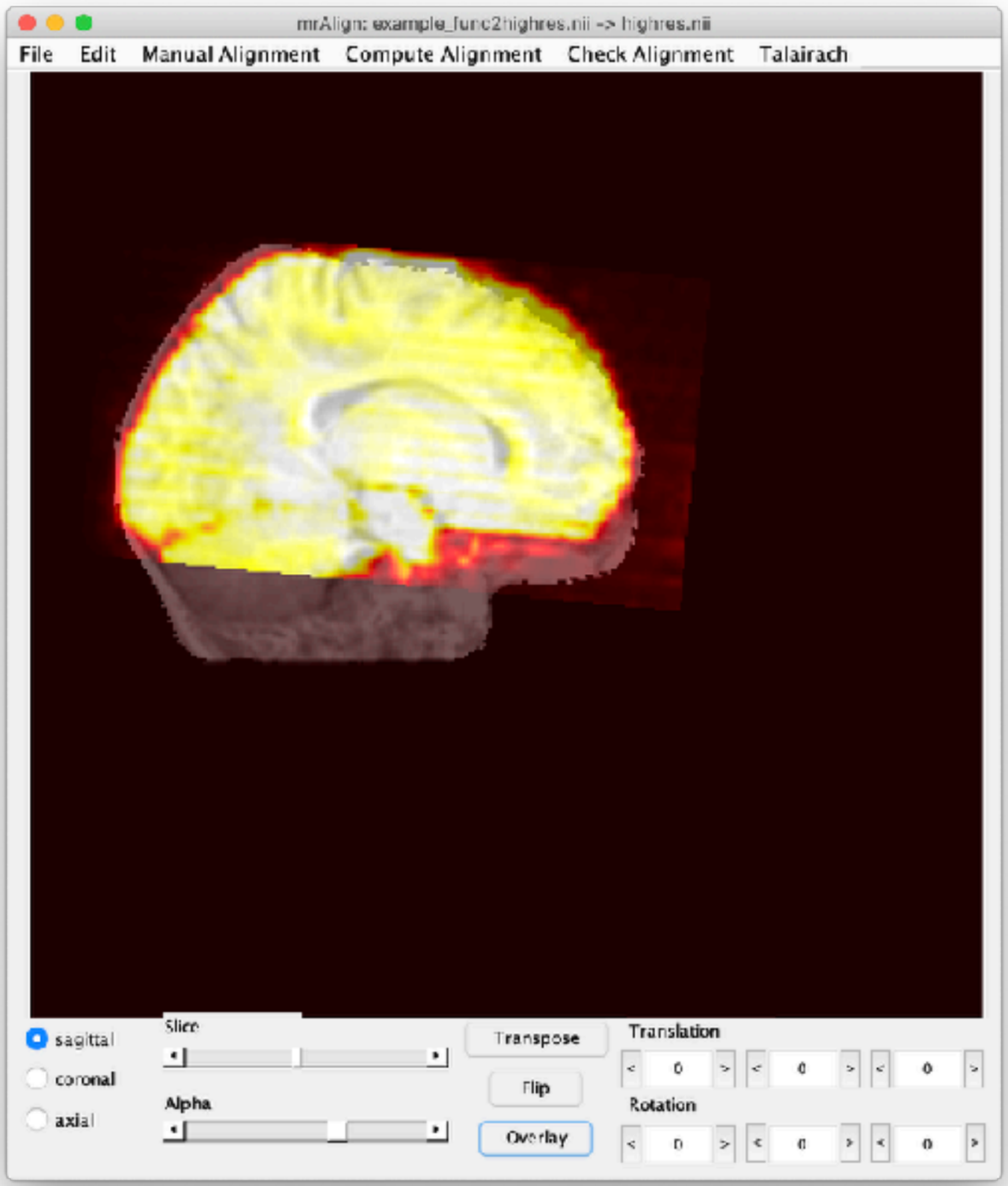
13. Manual registration (first level)

manual_registration.sh



13. Manual registration (first level)

manual_registration.sh



14. Combine runs

Post_PreStats.sh

Once the functionals are all registered to anatomy and preprocessed at first level it is necessary to concatenate across runs

We align the data to anatomical space but in functional voxel resolution

This creates a run mask so that you can decide how to mask the volume at secondlevel, but presumably you want the intersect of all masks for given experiment

Filtered functionals are all concatenated into a single file as are the motion parameters

15. Combine experiments

FunctionalSplitter.m

Extract information about when each experimental block starts and stops for each participant

This uses information from first level timing, rather than second level, to minimize risk of errors. e.g. if one run has extra burnout that is undetected by *Analysis_Timing.m*, this has carry on effects to second level timing for all subsequent runs, but this isn't true for first level timing.

Include a 3 TR rest after each block and ignore any extra TRs that may have been created

This script also enables experiment-specific block balancing. For instance StatLearning has different strategies of balancing the number of blocks within runs

(Only) Z score across blocks within a run that will be used

Experiment specific masking is performed

Motion parameters (decorrelated) and timing files are remade for each experiment according to what blocks are used

15. Combine experiments

FunctionalSplitter.m

15. Combine experiments

FunctionalSplitter.m

Legend:

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

S

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

S 1

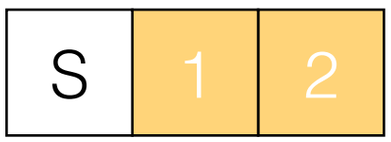
15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X



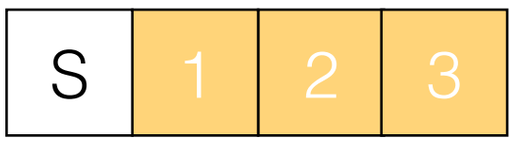
15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X



15. Combine experiments

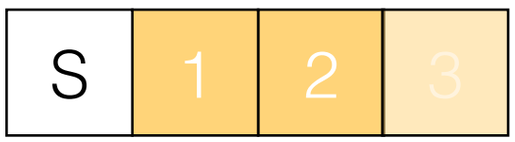
FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt X block X, excluded



15. Combine experiments

FunctionalSplitter.m

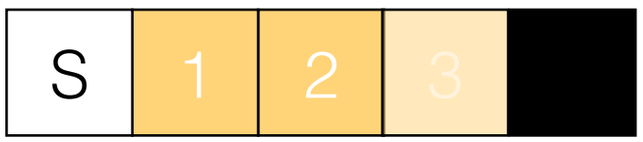
Legend:

S =Scout

X =Expt 1 block X

=Rest

X =Expt X block X, excluded



15. Combine experiments

FunctionalSplitter.m

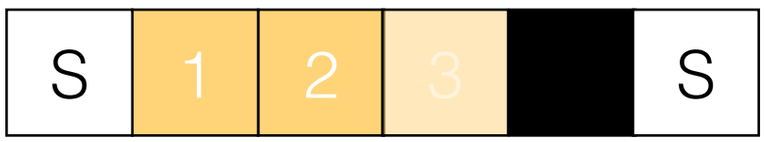
Legend:

S =Scout

X =Expt 1 block X

=Rest

X =Expt X block X, excluded



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

=Rest

X =Expt X block X, excluded

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

=Rest

X =Expt X block X, excluded

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

=Rest

X =Expt X block X, excluded

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

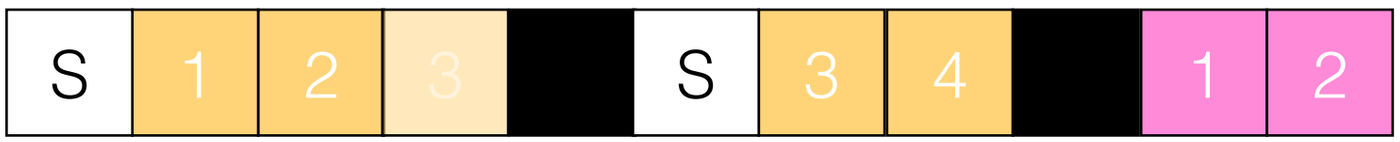
S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt X block X, excluded

=Rest



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

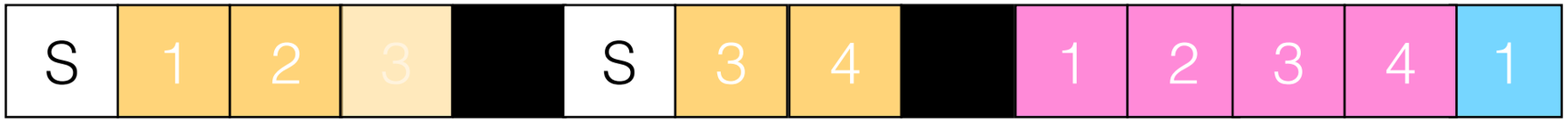
X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest



Pseudorun break

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

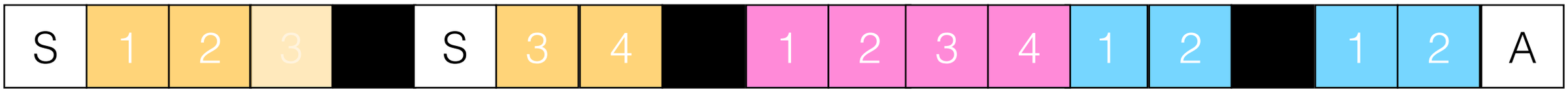
X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

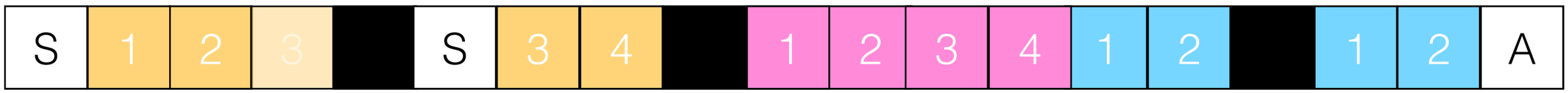
X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

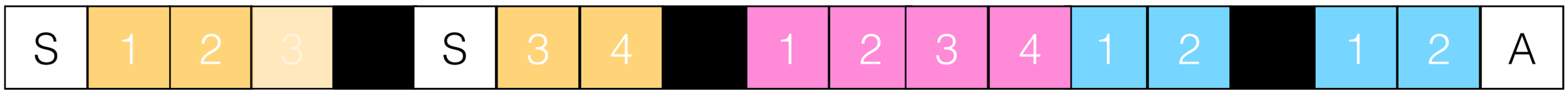
X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

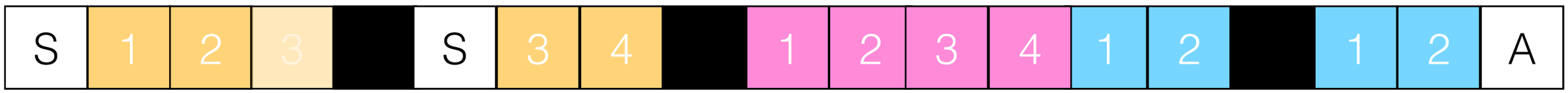
X =Expt 1 block X

X =Expt 2 block X

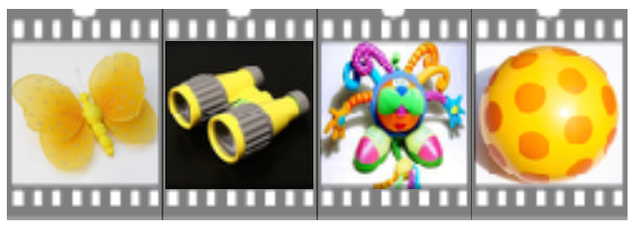
X =Expt 3 block X

X =Expt X block X, excluded

=Rest



Event 1 2 3 4



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

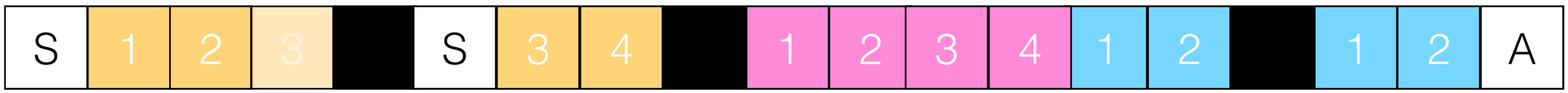
X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

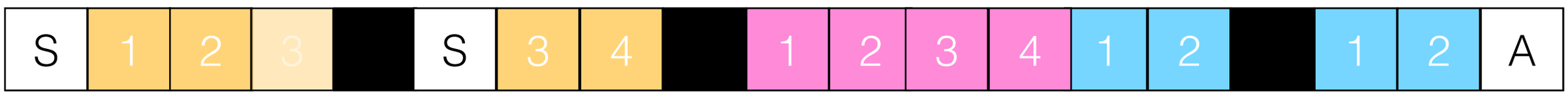
X =Expt 1 block X

X =Expt 2 block X

X =Expt 3 block X

X =Expt X block X, excluded

=Rest



Align, Merge, Split

15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

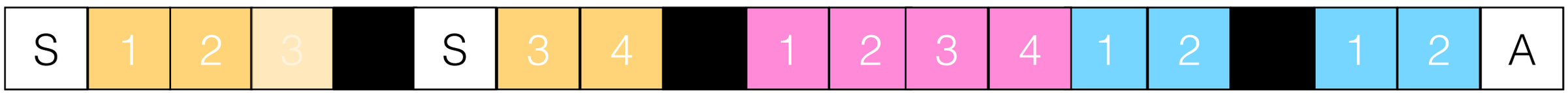
X =Expt 1 block X

X =Expt 2 block X

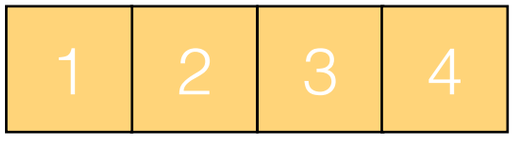
X =Expt 3 block X

X =Expt X block X, excluded

=Rest



Align, Merge, Split



15. Combine experiments

FunctionalSplitter.m

Legend:

=Scout

X =Expt 1 block X

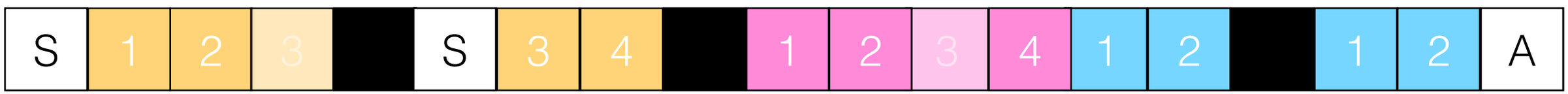
X =Expt 3 block X

=Rest

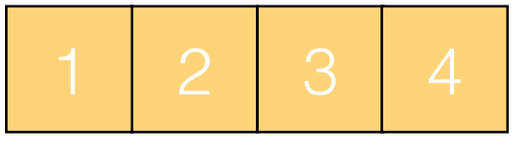
=Anatomical

X =Expt 2 block X

X =Expt X block X, excluded



Align, Merge, Split



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

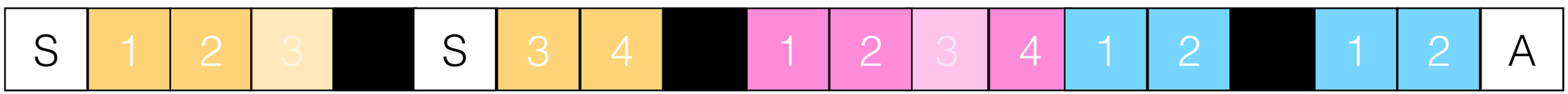
X =Expt 1 block X

X =Expt 2 block X

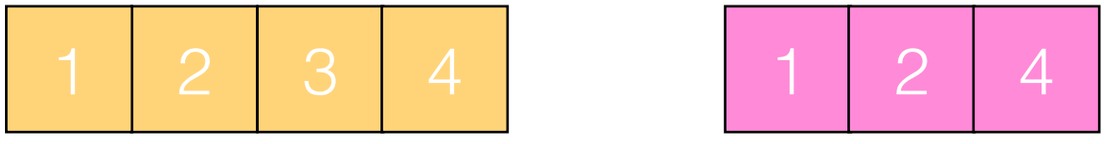
X =Expt 3 block X

X =Expt X block X, excluded

=Rest



Align, Merge, Split



15. Combine experiments

FunctionalSplitter.m

Legend:

S =Scout

A =Anatomical

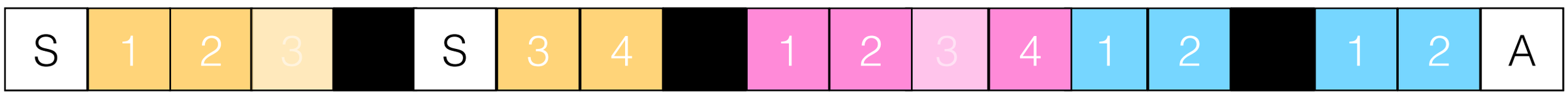
X =Expt 1 block X

X =Expt 2 block X

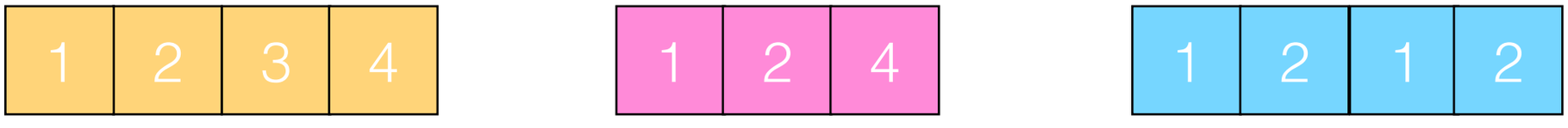
X =Expt 3 block X

X =Expt X block X, excluded

=Rest



Align, Merge, Split



16. Manual registration (second level)

manual_registration.sh

The registration of anatomy to standard space is typically bad.

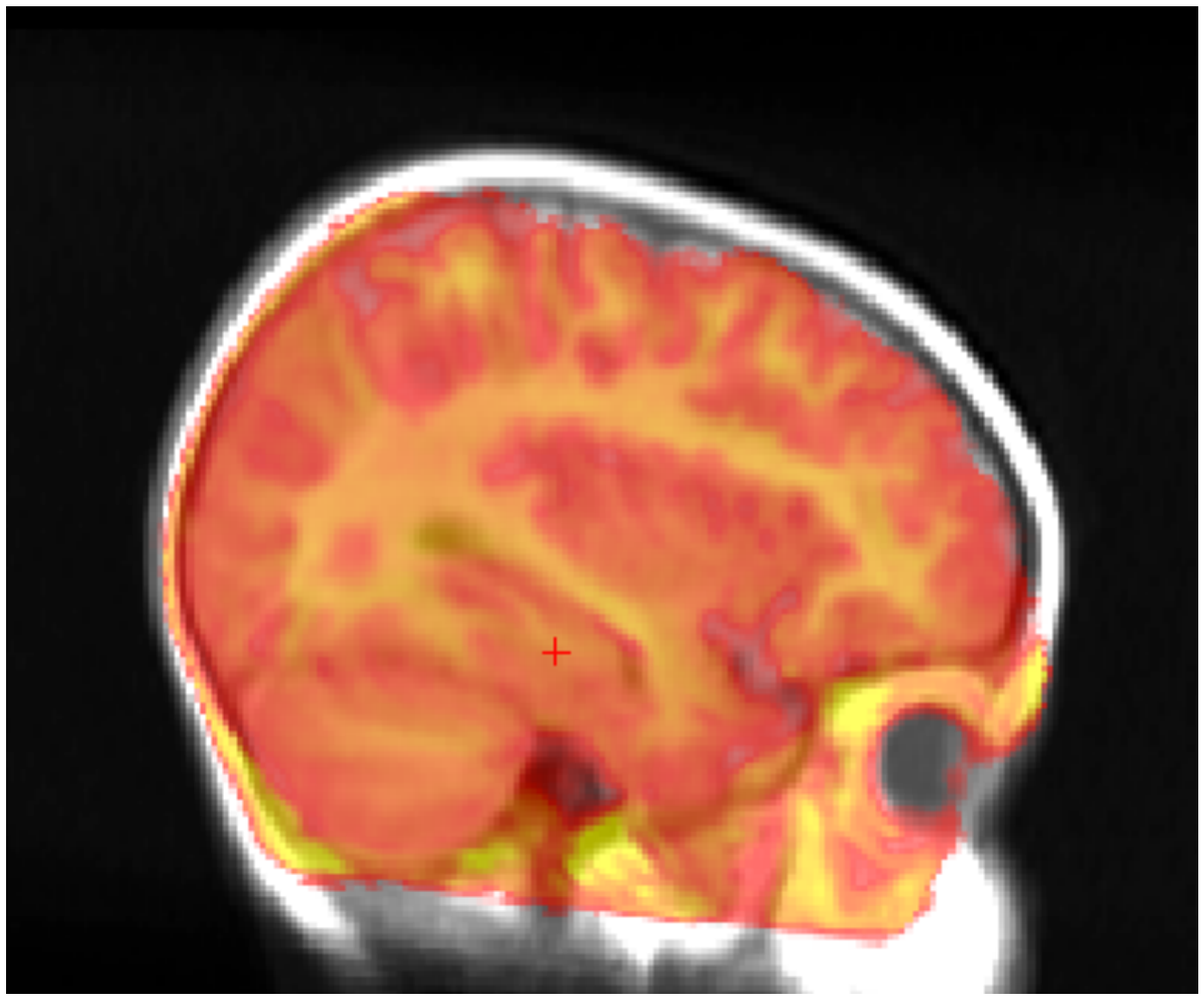
For linear alignment methods, we recommend supplementing the automatic registration with manual registration, which is scaffolded in a step by step fashion by *$SUBJ_DIR/scripts/ manual_registration.sh*

Registration to an age appropriate infant standard is performed in Freeview since 9 DOF is necessary. When saving the matrix from freeview we need to flip any values that have a sided effect since this assumes the wrong spatial orientation of the data

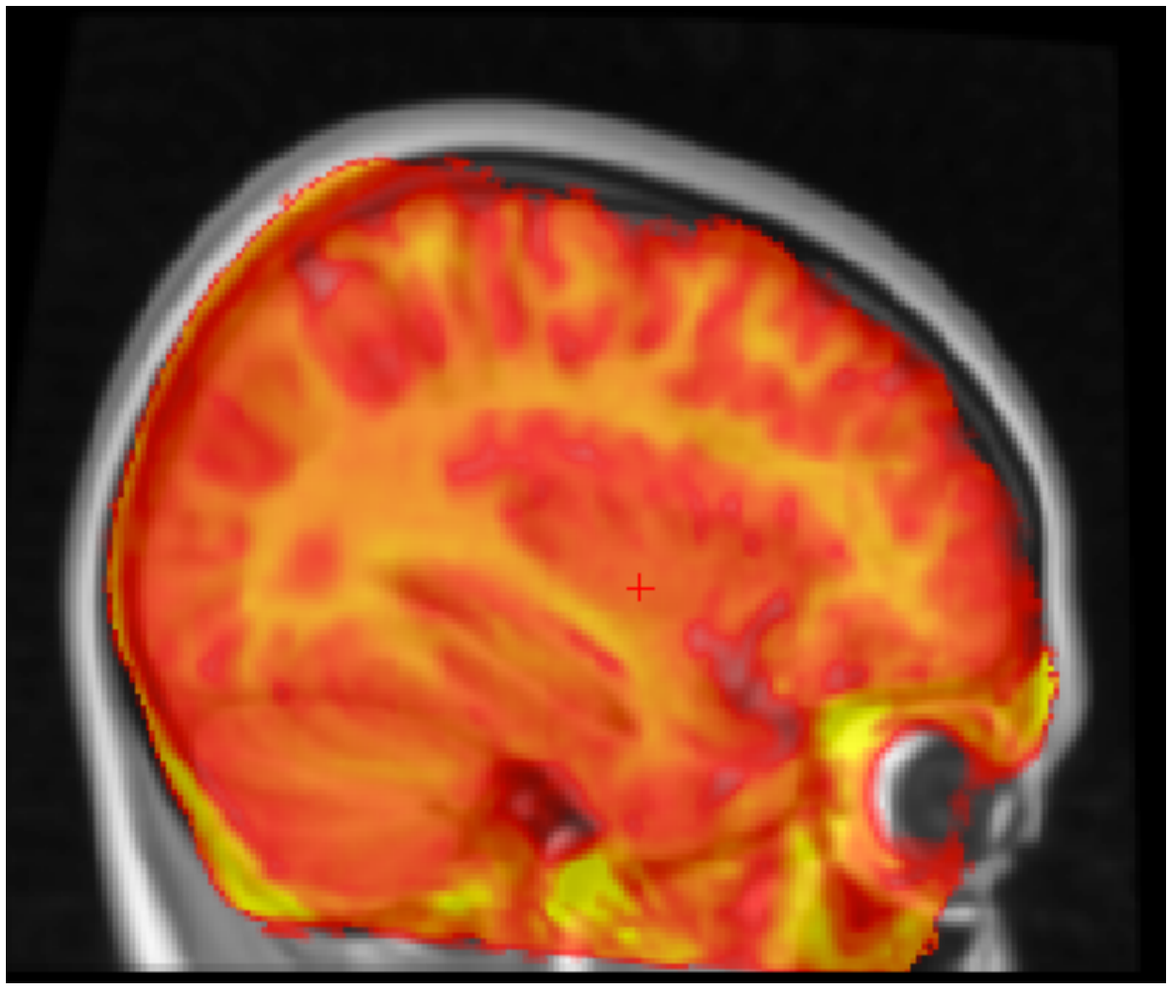
Note, even if you do your best, standard registration is mediocre when only using 9 DOF

Manual registration (second level)

Anat to Infant Standard

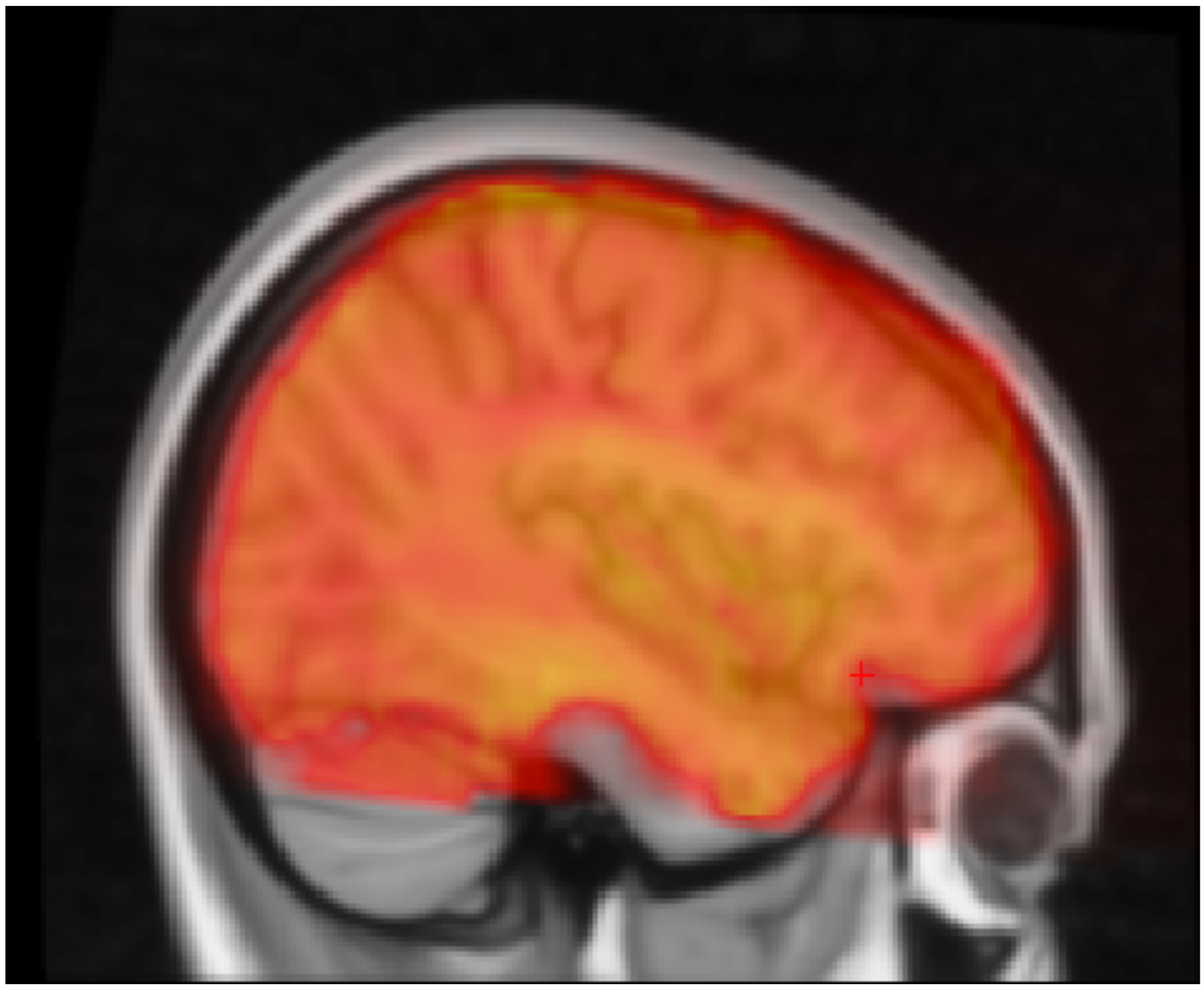


Anat to Adult Standard



Manual registration (second level)

example_func2standard



16. Nonlinear alignment

run_ANTs_highres2standard.sh

Even with somewhat low-quality anatomical images, ANTs can be amazingly effective at aligning infant data to standard space.

Critical to success is that the skullstripping is very accurate, so it is critical that an accurate brain mask is used.

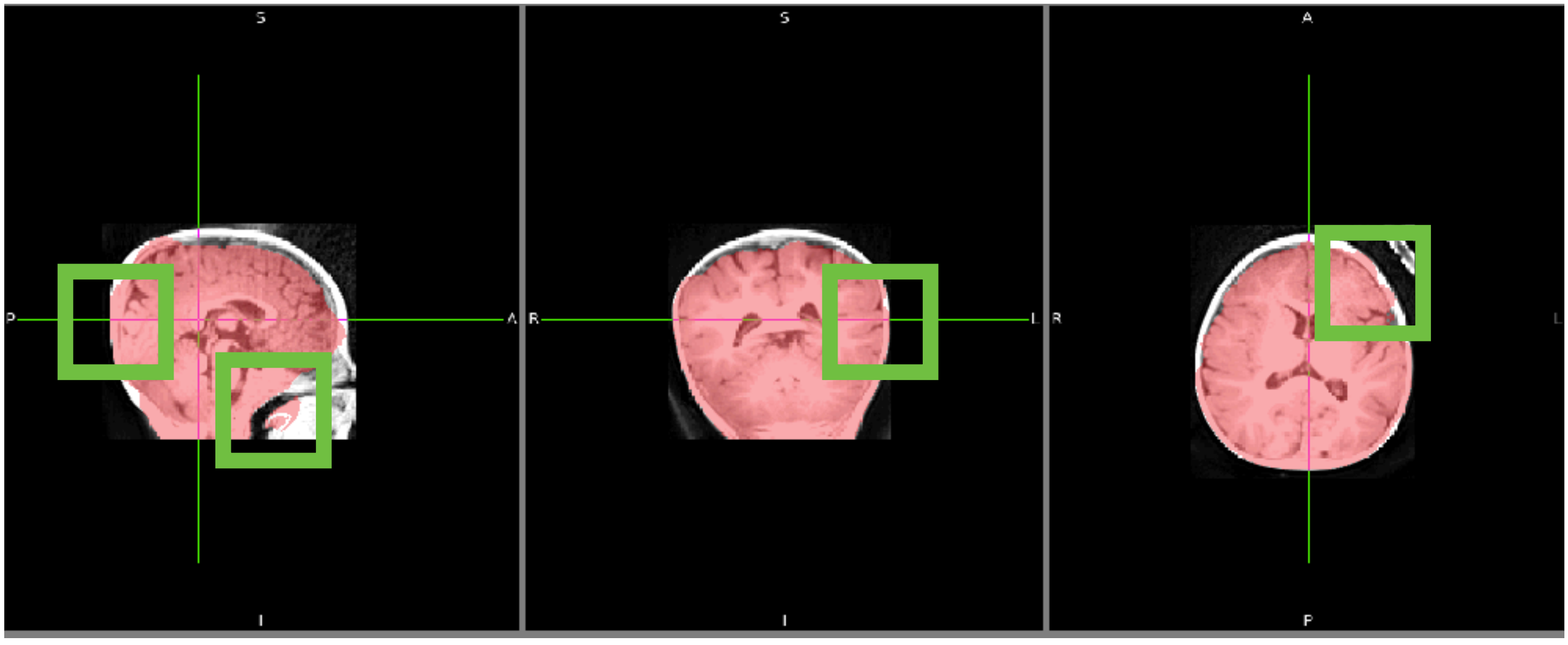
In our script, we use FreeSurfer for pulling out the brain mask, and then manually edit it (examples to follow)

Functional alignment to standard with the nonlinear warping might not look as good but that is because the warping exacerbates the functional to highness alignment

16. Nonlinear alignment

run_ANTs_highres2standard.sh

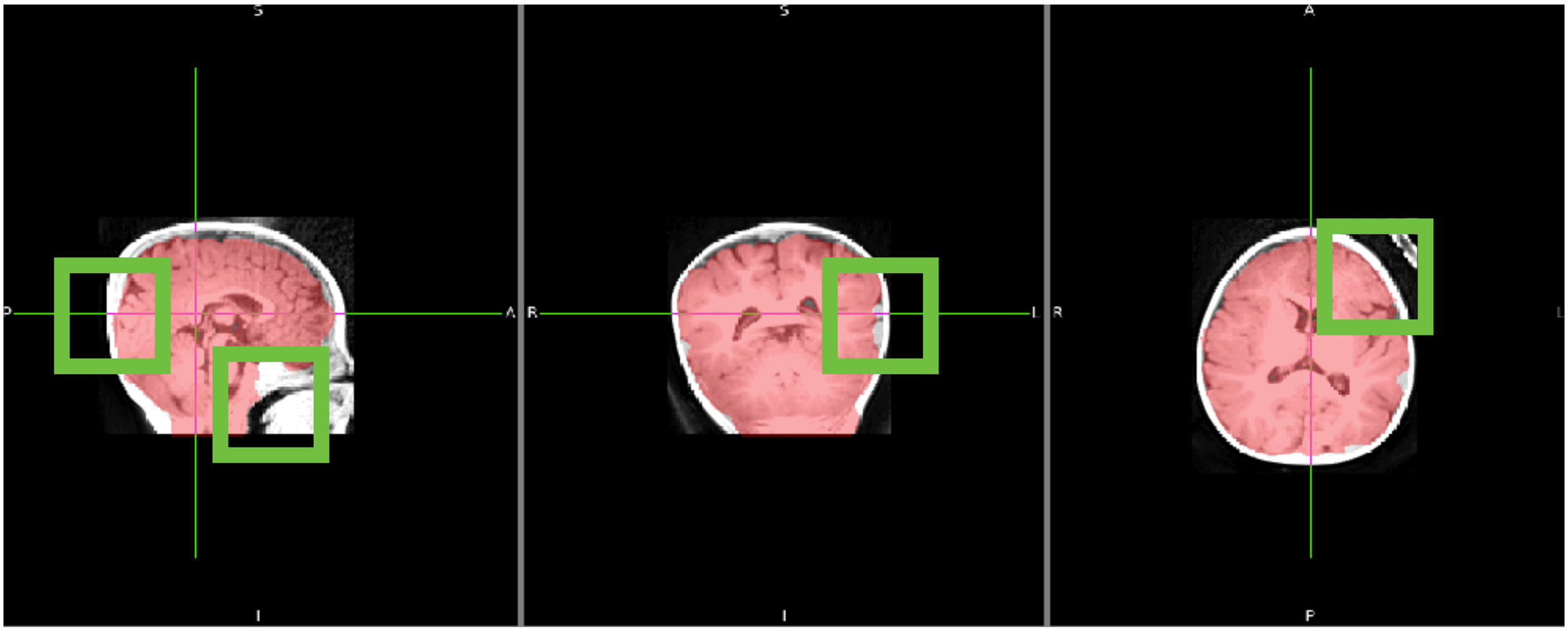
3dSkullStrip mask



16. Nonlinear alignment

run_ANTs_highres2standard.sh

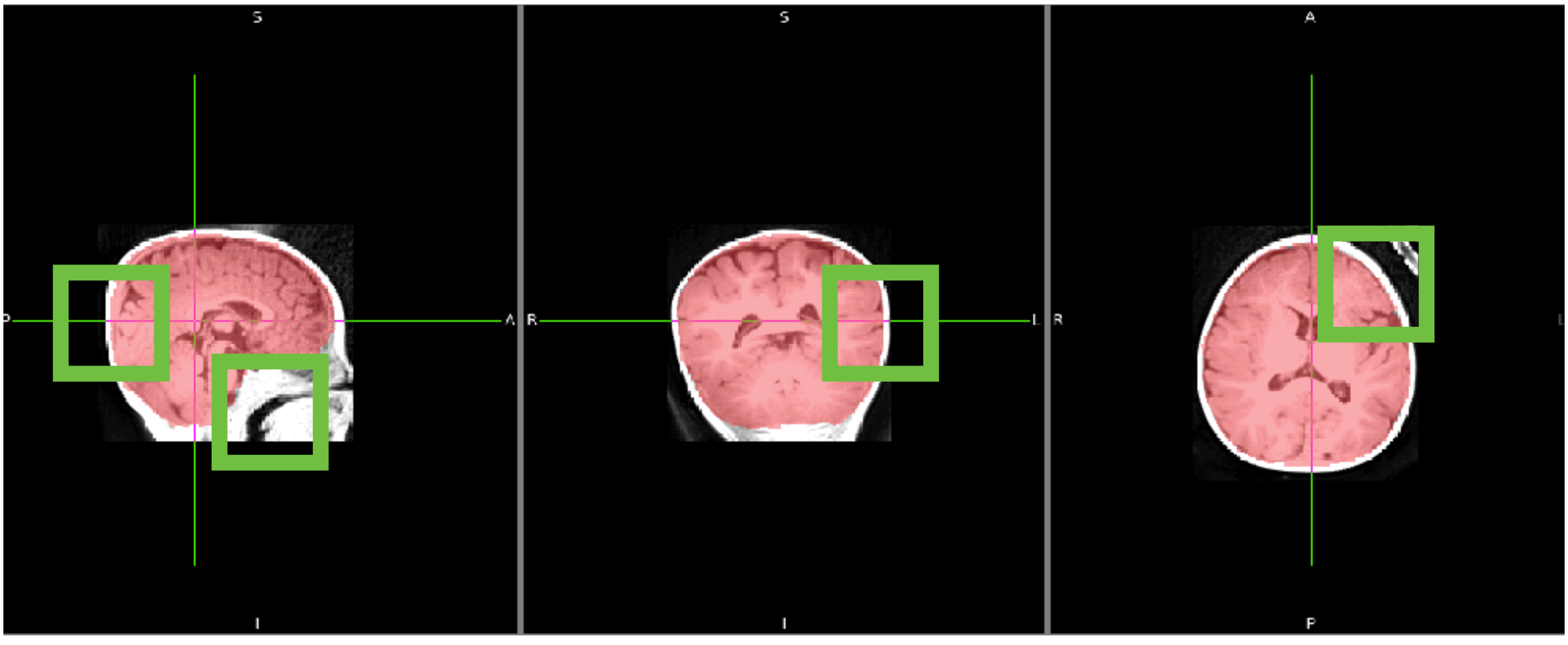
Freesurfer mask



16. Nonlinear alignment

run_ANTs_highres2standard.sh

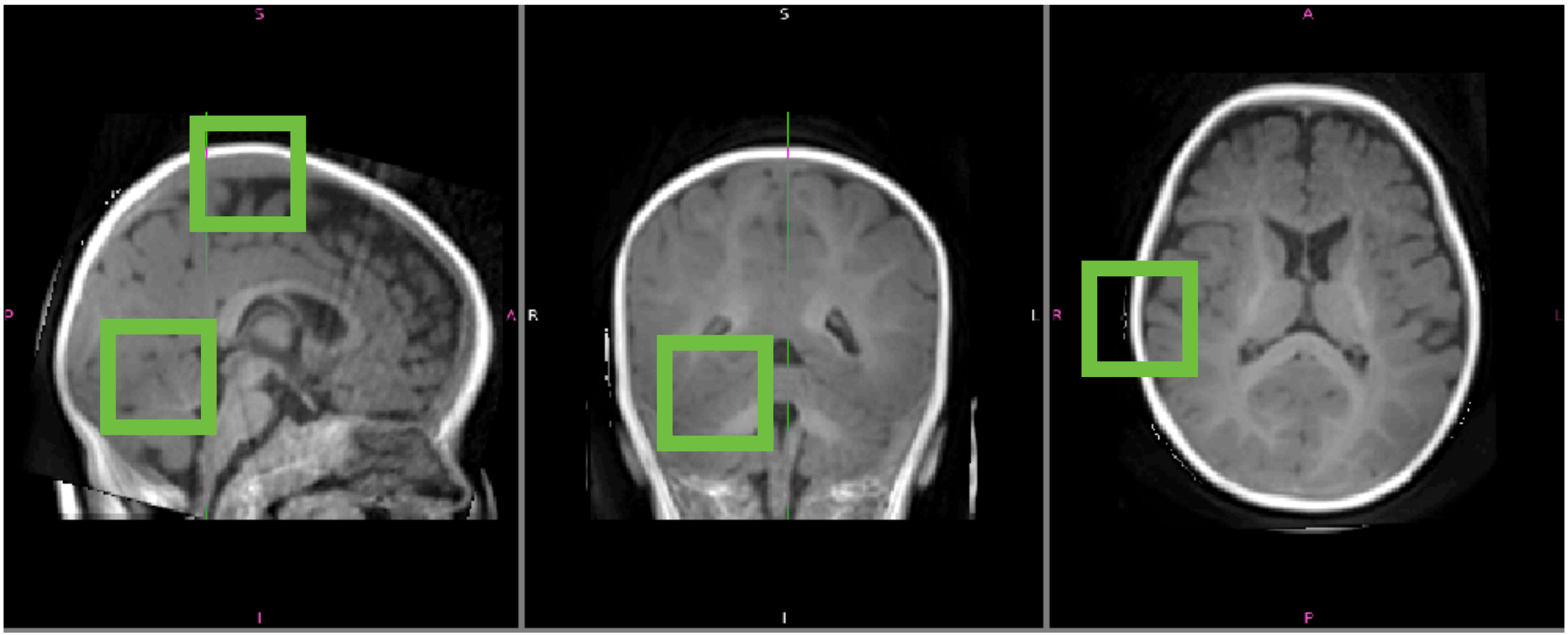
Manually masked (based on Freesurfer)



16. Nonlinear alignment

run_ANTs_highres2standard.sh

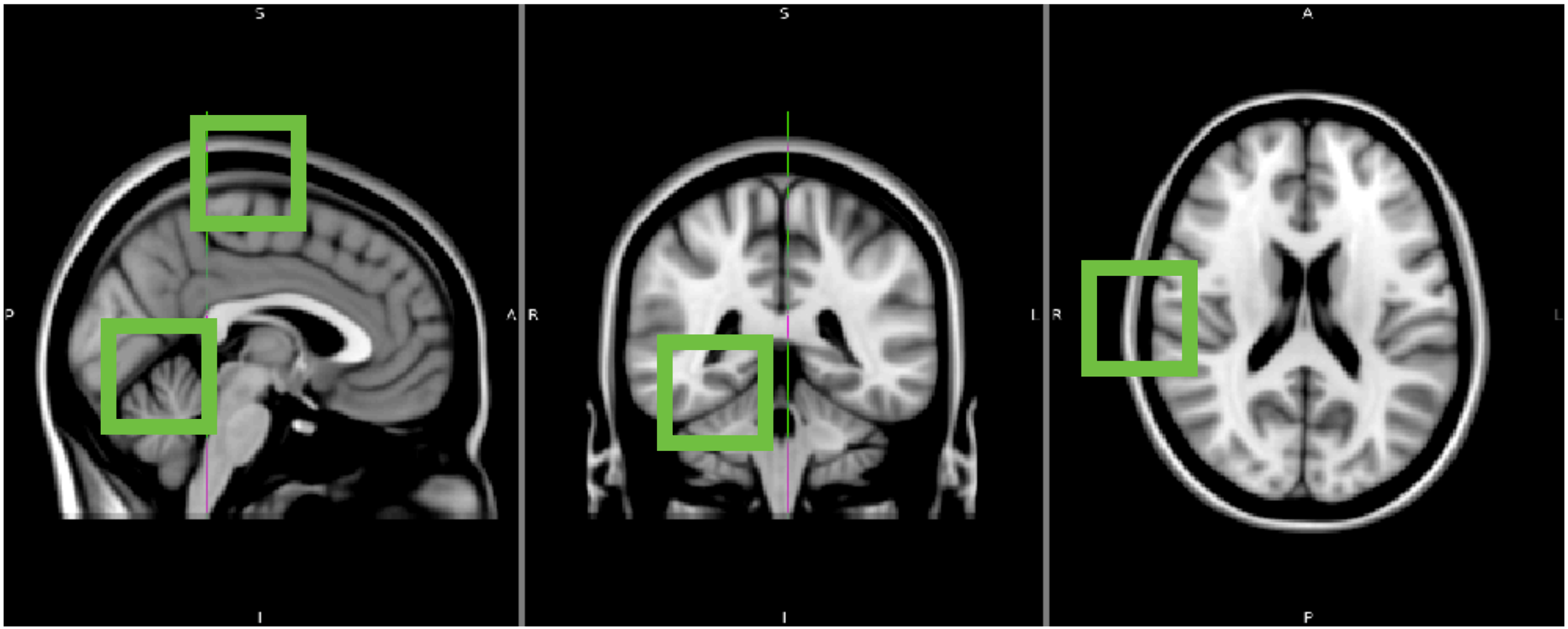
Linear alignment



16. Nonlinear alignment

run_ANTs_highres2standard.sh

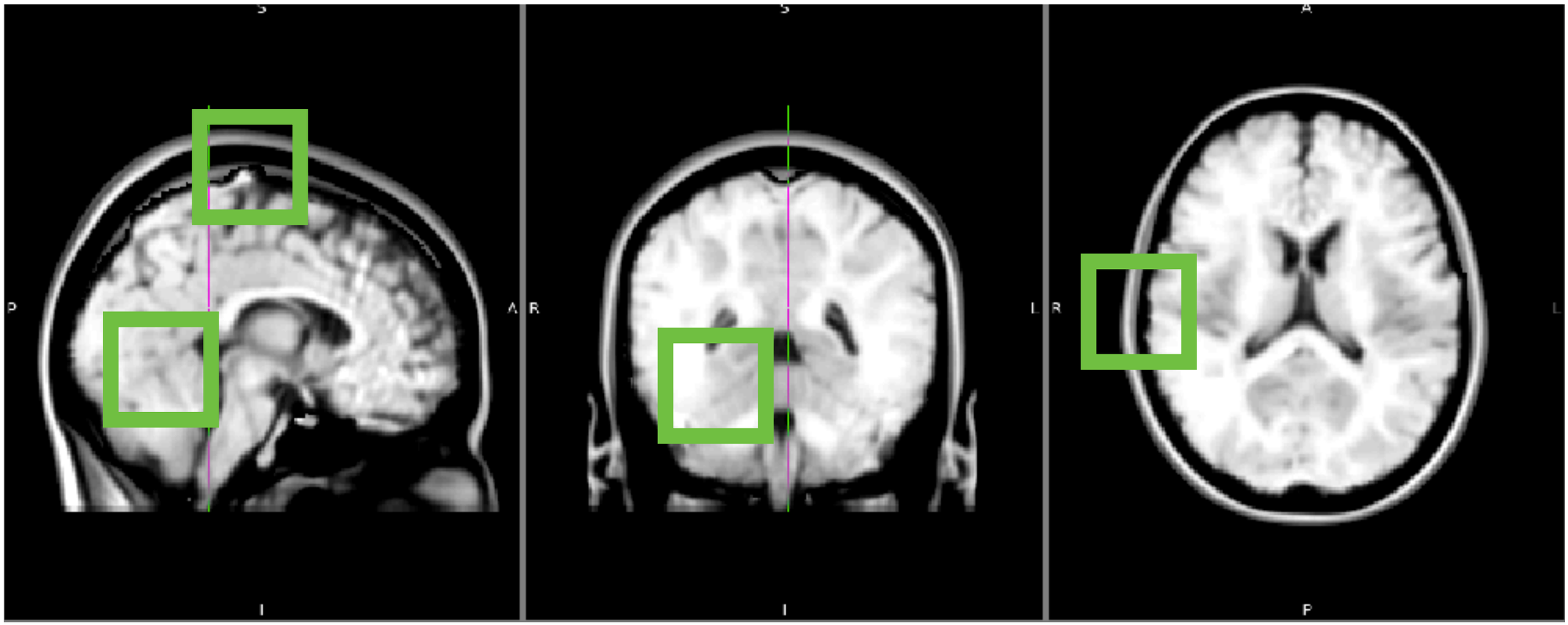
Standard



16. Nonlinear alignment

run_ANTs_highres2standard.sh

Nonlinear alignment



17. Run task-evoked analyses

FEAT_univariate.sh

This step creates a task versus rest contrast for each run ignoring different conditions or even experiment differences

This also produces an analysis comparing the task design matrix with the motion parameters and reports how motion may be correlated with task

This step is not essential but it is useful for showing the magnitude of evoked response and is essential for running the summary in step 19

18. Run FIR model

run_FIR.sh

This step creates a task versus rest FIR model to potentially show the shape of the HRF for each participant

This step is not essential but it is useful for showing the magnitude of evoked response and is essential for running the summary in step 19

19. Summarise results

participant_summary.ipynb

To see a summary of the participant's data, including motion, alignment, the effects of temporal filtering/z scoring and any behavioral plots, run this jupiter notebook.

To run notebooks interactively on a cluster you can use '*$SUBJ_DIR/scripts/launch_jupyter.sh*' but it is likely that this will need to be edited to run on your cluster setup

20. Second level analyses

Once the data is through the pipeline, it should be ready for experiment specific analysis as if it was all collected in a single run

Supervisor scripts can be set up to automate the analyses. These scripts might make fsf files.

In order to make the feats use the z scored data, these scripts run the feat analyses twice, once without the z scoring (thus setting up an appropriate mask) and once with z scoring.

There are tools for aligning functional data to standard space, like *align_stats.sh* and *align_functionals.sh*

Free parameters in the pipeline

What parameters are chosen and could be changed:

Movement criteria for exclusion? 3mm relative motion, frame wise displacement

Thresholds for including coders? <50% intraframe or interframe

Proportion of eye tracker calibration epoch to consider? >50% of time

Threshold for excluding epoch? >50% inappropriate fixation

Threshold for excluding epoch? >50% motion TRs

Decorrelate motion parameters? >0.95 correlation

Which blocks to ignore in a run if there are balancing issues? First

Misc tools

$SUBJ_DIR/scripts/align_stats.sh takes in a statistics map, overlays it on highres and standard and takes an image.

$SUBJ_DIR/scripts/align_functionals.sh same as above but for align

$PROJ_DIR/scripts/generate_report.sh finds all images across participants that match a certain name and creates an html file with all those images and names

$PROJ_DIR/scripts/Participant_Index.m can be used for selecting a subset of participants that meet specified criteria (e.g. age)

$PROJ_DIR/scripts/ScanTimeAnalysis.m summarizes the amount of usable and unusable functional and anatomical data collected from all participants.