FieldTrip Tutorial (DRAFT)

April 25, 2015

Giorgio Arcara [1]

[1]*Department of Neuroscience, Padua, Italy*

Contents

1 Introduction

This is a short tutorial on the use of FieldTrip to analyze MEG data collected with CTF system. All explanations provided assume some knowledge on MATLAB and on MEG/EEG pre-processing and data analysis. The materials contained in this document come from the freely available online FieldTrip tutorials, or from personal experience and considerations. The tutorial is focused on Fieldtrip, but some functions from other toolboxes (as EEGLAB) are also used. The tutorial is distributed without warranty of any kind.

2 Key ideas for using Fieldtrip

There are some key ideas related to the use of Fieltrip. Knowing them in advance will make the life a lot easier.

- FieldTrip works only trough the command line
- The *cfg* structure
- The same function does several things
- Fieldtrip *doesn't* make trial bookkeeping
- If you have several conditions, you have to load them separately

2.1 FieldTrip works only trough the command line

There is no graphical interface in Filedtrip. No buttons, almost no windows. All operations (e.g., filtering, epoching, calling graphics), are performed via command lines. In other words, working with fieldtrip means writing lines of code, and knowing some basic concepts of MATLAB (and of programming in general). If you don't know what is an object, or a directory, it is better you drop this tutorial and look for a basic tutorial on MATLAB or programming.

2.2 The *cfg* structure

Almost all functions in FieldTrip require to specify a **cfg** structure. For example:

```
data=ft_preprocessing(cfg);
```

Technically speaking, the **cfg** is a MATLAB struct with several fields that can (or must) be specified. Each field specify some arguments of the function that is called. For example:

```
cfg=[];
cfg.headerformat='ctf_ds'
cfg.dataset='prova_VisualOddBall_20141110_01.ds'
cfg.viewmode = 'butterfly';
cfg.continuous='yes'
cfg.ylim = [-2.20e-11 2.20e-11]


cfg=ft_databrowser(cfg)
```

In this example a **cfg** structure is first created. Then several fields are added by specifying some arguments of the call that is made afterward (with `ft_databrowser`). There are some important things related to the cfg structure to be noted here:

1. Typically in a fieldtrip analysis pipeline cfg structures are created (and overwritten) several times. Basically, a temporary cfg is created every time there is a call to a function.

2. Typically in the cfg structure only a limited number of arguments of the function call are specified. All other arguments are left to the default values.

3. The output of a call of a function can yield as output a data structure (see later) or a cfg structure.

2.3 The same function does several things

There is relatively a small number of functions in FieldTrip, but some of them perform different things. By changing some fields of the cfg structure

the same function can be used for several different purposes.This is because several FieldTrip functions are basically wrappers for other functions (that very often can be called individually). For example, in the box below it is shown how the function `ft_preprocessing` can be used for different purposes:

```
% filter data
cfg=[];
cfg.channel='MEG';
cfg.lpfilter='yes';
cfg.lpfreq=30;
cfg.hpfilter='yes';
cfg.hpfreq=1;

data_filt=ft_preprocessing(cfg, data)

% baseline correction
cfg=[];
cfg.demean = 'yes';
cfg.baselinewindow = [-0.2 0];

data_bc=ft_preprocessing(cfg, data_filt)
```

What happens in this case is that the call to `ft_preprocessing` imply a call to several other more basic functions, as `ft_read_data` and `ft_channelselection`. Importantly, with such system it is possible to perform several actions with a simple call to a function. In this case however, it is important to understand what is the order of the actions performed (for example, first resampling and then filtering?). If the order with which the actions be will performed is not clear, it is recommended to breaks the actions in different call to a function.

2.4 Fieldtrip can work on objects or on files

Fieldtrip allows to work on objects but also directly on files (without creating an object first). To my knowledge loading an object should allow faster computations in some cases, but it could overload the RAM when too many

objects are loaded in the workspace (thus slowing down MATLAB). If a function work directly on a files, the filename is specified in the cfg structure. If the funciton work on an objects, the function should be called by specifying two arguments, the cfg structure and the object.

```
% filter data directly from a file
cfg=[];
cfg.dataset='GR_VisualOddBall_20141114_01.ds'
cfg.channel='MEG';
cfg.lpfilter='yes';
cfg.lpfreq=30;
cfg.hpfilter='yes';
cfg.hpfreq=1;

data_filt=ft_preprocessing(cfg);

% filter data from an object
cfg=[];
cfg.channel='MEG';
cfg.lpfilter='yes';
cfg.lpfreq=30;
cfg.hpfilter='yes';
cfg.hpfreq=1;

data_filt=ft_preprocessing(cfg, data);
```

2.5 Fieldtrip *doesn't* make trial bookkeeping

It is up to the user to keep track on the trials, and which condition they belong. This is particularly relevant given the following point, explained in section 2.6.

2.6 If you have several conditions, you have to load them separately

Unlike other software it appears that if you have more than one condition (for example marked with different events name), you have to load them separately if you want to extract epochs. Afterwards, you can append the separate datasets to create a single dataset of epoched data. This can be useful if you want to perform an ICA on all the epoched data. **Note** that in this case you lose the information of the trial original ordinal position, cause the trials will be stacked in the order of data appending. If you append two datasets, respectively with Target and NonTarget trials, the resulting dataset will contain first all Target trials and then all NonTarget trials. The original position of target can be retrieved from the `sampleinfo` field of the resulting dataset. The field `sampleinfo` records the starting and ending samples (timepoints) of the epoch.

3 A pipeline for FieldTrip analysis (single subject level)

The following section explains a pipeline for data analysis with FieldTrip. The pipeline considered will take into account the following steps, that will be explained in details in different subsections:

1. Loading the data in MATLAB
2. Giving a first look at the data
3. Filtering the data
4. Epoching the data and retrieve trial information
5. Removing bad channels and bad trials
6. Performing ICA
7. Removing artefactual ICA components
8. baseline correction and trial rejection
9. correcting head movements

10. timelock analysis

3.1 Loading data with FieldTrip

This section explains how to load data with FieldTrip. The section focuses on CTF data files. First of all you have to change the directory to the one containing the `.db` folder of the data to be imported. The import is than made by means of the `ft_preprocessing` function:

```
cd('/Users/giorgioarcara/Desktop/MEG/Prove MEG fieldtrip')

cfg=[];
cfg.dataset='prova_VisualOddBall_20141110_01.ds'
cfg.channel='MEG';

data_meg=ft_preprocessing(cfg)
```

The field `cfg.dataset` telles the name of the data to be imported, whereas `cfg.channel='MEG'` tells to import all MEG channels. The flag 'MEG' is understood by the system as a shortcut to import all MEG channels. You can see these options by looking at the help of `ft_channelselection`. Alternatively you can specify the channel labels to be imported as a cell `{'BG1' 'BG2' BG3'}`. The resulting struct, that in this example is called `data_meg`, should be similar to the following:

```
data_meg =

hdr: [1x1 struct]
label: {273x1 cell}
time: {[1x240000 double]}
trial: {[273x240000 double]}
fsample: 600
sampleinfo: [1 240000]
grad: [1x1 struct]
cfg: [1x1 struct]
```

The output of our call to `ft_preprocessing` is a struct (`data_meg`) with several fields. One of this fields (i.e., `trial`) will be the matrix containing the MEG data, but also other information will be stored.

- `hdr` is a struct containg several informations on the dataset.
- `time` contains the time (in seconds) of the recording.
- `trial` contains the data as a *electrodes* $\times$ *timepoints* matrix.
- `fsample` indicates the sampling rate.
- `sampleinfo` indicates information of the sampling points of the epochs (in this case just one epoch, because data are still stored as continuous).
- `grad` contains informations on the gradiometers.
- `cfg` is the complete cfg structure used for the call.

3.2 Giving a first look at the data

After importing the data, and after checking that everything makes sense (by inspecting the loaded structure), it is a good habit to give a quick look at the data. The basic function for browsing data in FieldTrip is `ft_databrowser`.

```
cfg=[];
ft_databrowser(cfg, data_meg);
```

If you run the code above, probably you won't see anything and you will be disappointed. This is just a matter of y-axis range. The code above calls the function for plotting the graph without any further parameters, that are left to default. You can manually adjust the y-axis range by clicking on the + and - buttons beside the label *verticals* on the right bottom. After this modificaiton you should see a graphic similar to that in Figure 1

Figure 1: A butterfly plot

The code below shows how to add some parameters before calling the graph.

```
cfg=[];
cfg.viewmode = 'butterfly';
cfg.continuous='yes'
cfg.ylim = [-2.20e-11 2.20e-11]

ft_databrowser(cfg, data_meg)
```

Alternativly the code below can be used for a plot of stacked electrodes (see Figure 2).

```
cfg=[];
cfg.viewmode = 'vertical';
cfg.continuous='yes'
cfg.ylim = [-2.20e-14 2.20e-14] % note the different scale

ft_databrowser(cfg, data_meg)
```

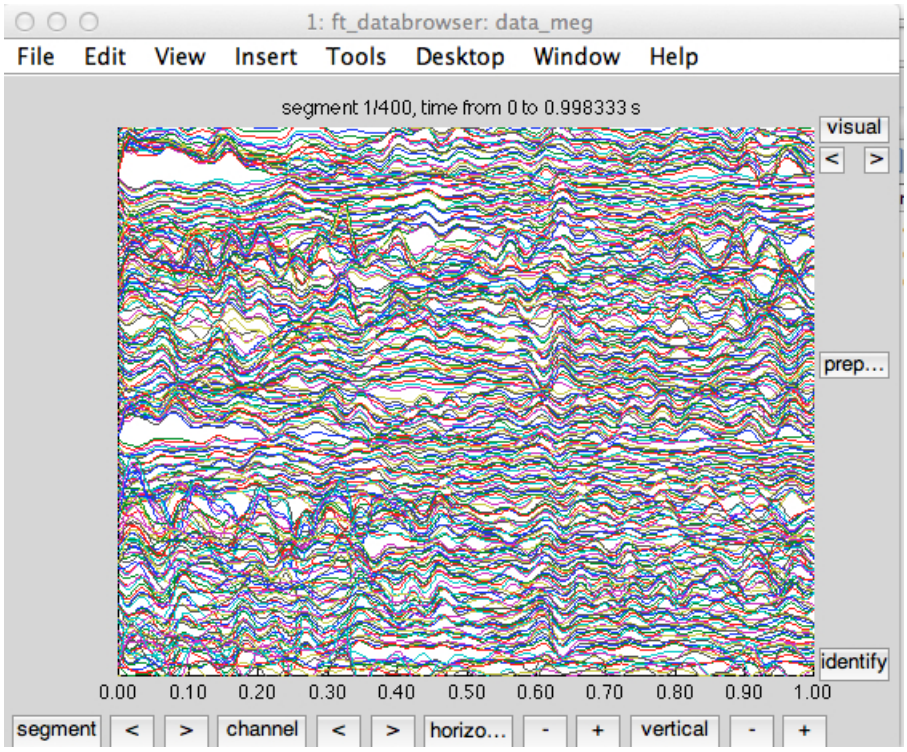

Figure 2: A scroll plots of all channels

Note that in these lines of code the `cfg.ylim` specifies a slightly different limits for the y-axis. I determined this range empirically to obtain a satisfying visualization.

A nice way to visualize stacked (or grouped) electrodes is to codify the colors to group left, right and central sensors, with three different colors. To achieve it is possible to specify the parameter `colorgroups`, specifying `'labelchar2'`. This indicates to take the second letter of channel name to create a group (and the second letter specify the laterality):

```
cfg=[];
cfg.viewmode = 'vertical';
cfg.continuous='yes';
cfg.colorgroups='labelchar2';
cfg.ylim = [-2.20e-14 2.20e-14] % note the different scale

ft_databrowser(cfg, data_meg)
```

Finally, as already explained, if with the result of the call to the function is put in an object, this latter will be a `cfg` object containing all the parameters of the call: the one specified by the users, plus all the defaults (In the code below only part of the output is displayed).

```
cfg=[];
cfg.viewmode = 'vertical';
cfg.continuous='yes'
cfg.ylim = [-2.20e-14 2.20e-14] % note the different scale

cfg=ft_databrowser(cfg, data_meg)

cfg =

viewmode: 'vertical'
continuous: 'yes'
ylim: [-2.2000e-14 2.2000e-14]
callinfo: [1x1 struct]
version: [1x1 struct]
trackconfig: 'off'
checkconfig: 'loose'
checksize: 100000
showcallinfo: 'yes'
debug: 'no'
trackcallinfo: 'yes'
trackdatainfo: 'no'
trackparaminfo: 'no'
preproc: []
artfctdef: [1x1 struct]
selectfeature: {'visual'}
selectmode: 'markartifact'
blocksize: 1
selfun: {1x5 cell}
selcfg: {1x5 cell}
colorgroups: 'sequential'
channelcolormap: [15x3 double]
...
```

It is also possible to call a graphic directly from the file, without loading the object before. In this case, there are some other parameters to be specified in the cfg structure, but in the call of `ft_preprocessing` only the cfg

structure has to be specified.

```
cfg.headerformat='ctf_ds'
cfg.dataset='GR_VisualOddBall_20141114_01.ds'
cfg.viewmode = 'butterfly';
cfg.continuous='yes'
cfg.ylim = [-2.20e-11 2.20e-11]

ft_databrowser(cfg)
```

3.3 Filtering the data

One of the first step of data pre-processing is filtering. The following files shows how to do it with FieldTrip. Again the filtering is made with the function `ft_preprocessing`. Since this is a quite automatic step I find useful to combine the filtering with the data import. However, as already explained is possible to divide these two steps, first importing and then filtering. In this latter case, the function will require two arguments, the cfg and the data structure.

```
cfg=[];
cfg.dataset='GR_VisualOddBall_20141114_01.ds'
cfg.channel='MEG';
cfg.lpfilter='yes';
cfg.lpfreq=30;
cfg.hpfilter='yes';
cfg.hpfreq=1;

data_meg=ft_preprocessing(cfg)
```

To check if the filtering worked properly I find useful to use a function from EEGLAB. So the following code will work only if EEGLAB is installed.

```
spectopo(data_meg.trial{1}, 0, data_meg.fsample, ...
'percent', 5, 'limits', [0 70 NaN NaN NaN NaN] );
```

The arguments of the call to spectopo are the following. The first indicates the data on which calculate the spectra, the second indicates the frames per epoch (0 tells to use the default, taht is the length for the data), then it is indicated to calcualted the spectra only on 5% of the data to reduce the time needed. The last arguments indicates the limits. The first two are the limits of the frequency (in Hz) all the others indicates the other limits, with NaN the limits are automatically calculated.

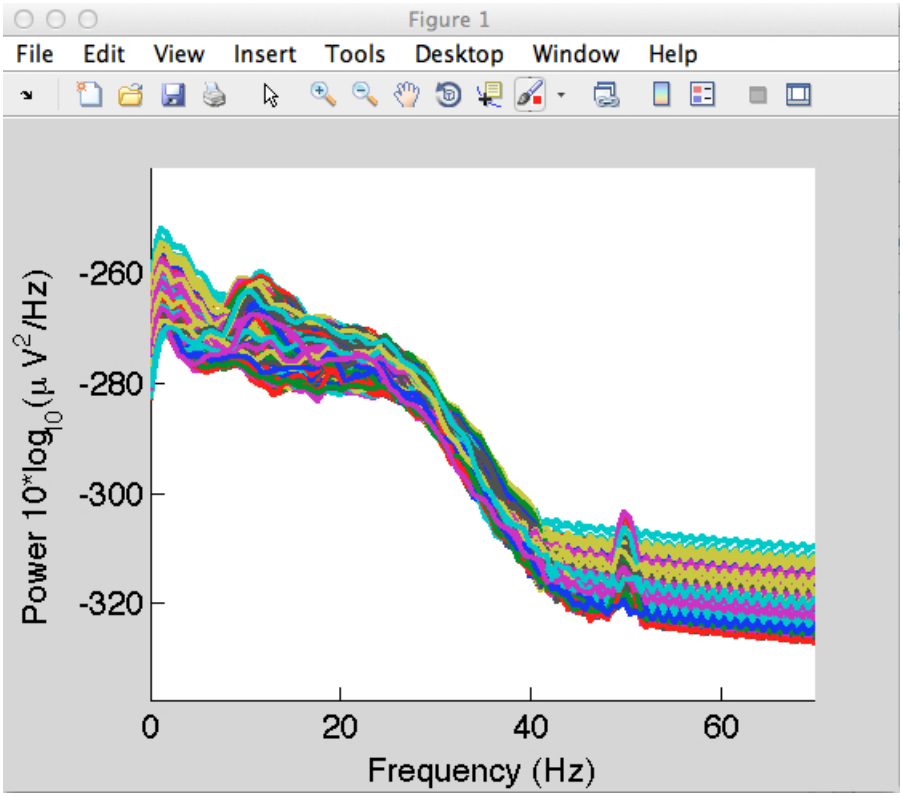

Figure 3: Channel spectra after filtering

3.4 Epoching the data and retrieving trial information

A crucial aspect of epoching data with FieldTrip is that **every condition has to be processed separately**. Epoching data consists of two steps:

1. retrieving trial information from the original raw file (by means of the function `ft_definetrial`)

2. epoching the data according to the trial information (by means of `ft_redefinetrial`).

For example in the case of two experimental conditions (Target and NonTarget), the following steps are required.

```
% defining Target epochs
cfg = [];
cfg.dataset='GR_VisualOddBall_20141114_01.ds' %
cfg.trialdef.eventtype = 'Target';
cfg.trialdef.prestim = 0.2;
cfg.trialdef.poststim = 1.2;

TargetTrials = ft_definetrial(cfg);

% create epoched data set for Target condition
data_megTarget=ft_redefinetrial(TargetTrials, data_meg)

% defining NonTarget epochs
cfg = [];
cfg.dataset='GR_VisualOddBall_20141114_01.ds' %
cfg.trialdef.eventtype = 'NonTarget';
cfg.trialdef.prestim = 0.2;
cfg.trialdef.poststim = 1.2;

NonTargetTrials = ft_definetrial(cfg);

% create epoched data set for NonTarget condition
data_megNonTarget=ft_redefinetrial(NonTargetTrials, ...
data_meg)
```

After the data have been epoched is possible to combine them again in a single data set. Appending the data in a single data set can be useful for the purpose of running a single ICA on all the conditions for artifact rejection (this is desirable, rather than performing separate ICA on each condition). It is important to note that after appending it's up to the user to keep and retrieve the information on the trials (e.g. the condition to which they belong). In this case trial information can be desumed from the number of trials of the original datasets (before appending) and from the fact that the data are appended sequentially.

```
% defining Target epochs
cfg = [];
data_megEpoch=ft_appenddata(cfg, data_megTarget, ...
data_megNonTarget)
```

For example in the case above if TargetTrials consist of 30 trials, in the resulting appending data the first 30 Trial will be TargetTrials and the remeaining will be NonTargetTrials.

IMPORTANT: In some cases you may want to append dataset from differents recording runs (for example if you splitted the recording session in two separate files). In such a case the resulting appended data will lose the coil locations. This is because the coil locations will be different and for protection fieldtrip will delete the locations. You may follow two routes to solve this issue: 1) If you are sure that the subject is approximately in the same position you could retrieve the original coil locations from one of the appended dataset; 2) You may run separate analysis for the two datasets (for example source analysis) and then average the results of the different runs.

3.5 Removing bad channels and bad trials

3.5.1 Manual rejection of channels and trials

After Epoching trials it is desirable to check if there are problematic (bad) channels or trials. To do this check it is possible to use te function `ft_rejectvisual`.

```
cfg=[];
data_megEpochclean= ft_rejectvisual(cfg,data_meg);
```

Calling this function open a GUI (see Figure 4) that allows to visualize potential bad channels and trials. The bottom left panel shows the variable currently selected to inspect channel and trials (default is variance). The upper right panel show the trials and channels, with hotter colors indicating

potentially bad channels or trials (a hot "column "indicates a bad trials, and a hot "row "indicates a bad channel). The upper right panel indicates the channel, with outlier values probably denoting channel to be removed. The bottom left panel indicates the trials with outlier values probably denoting trials to be removed. You may select (by drawing a window) the trials or the channels you want to remove. This modifications will be stored in the object if the call to `ft_rejectvisual` has been made creating an object (as in the example code above). By changing the measures it is easy to spot dead channels (if they are stuck to zero). Be careful with the selection of trials or channels to remove because there is no way of undoing a selection. There is are two main issues with the visual selection of channels (and/or trials). First, there is no optimal threshold for selecting if a channel/trial is an outlier as compared to the others. Second, you cannot base your selection on how far a channel/trial is, as compared to the others, since the figure automatically change the scales based on the range. For such reasons, I strongly prefer automatic rejection (see next paragraph). Nonetheless, the visual inspection allows to identify easily dead channle (with 0 values).

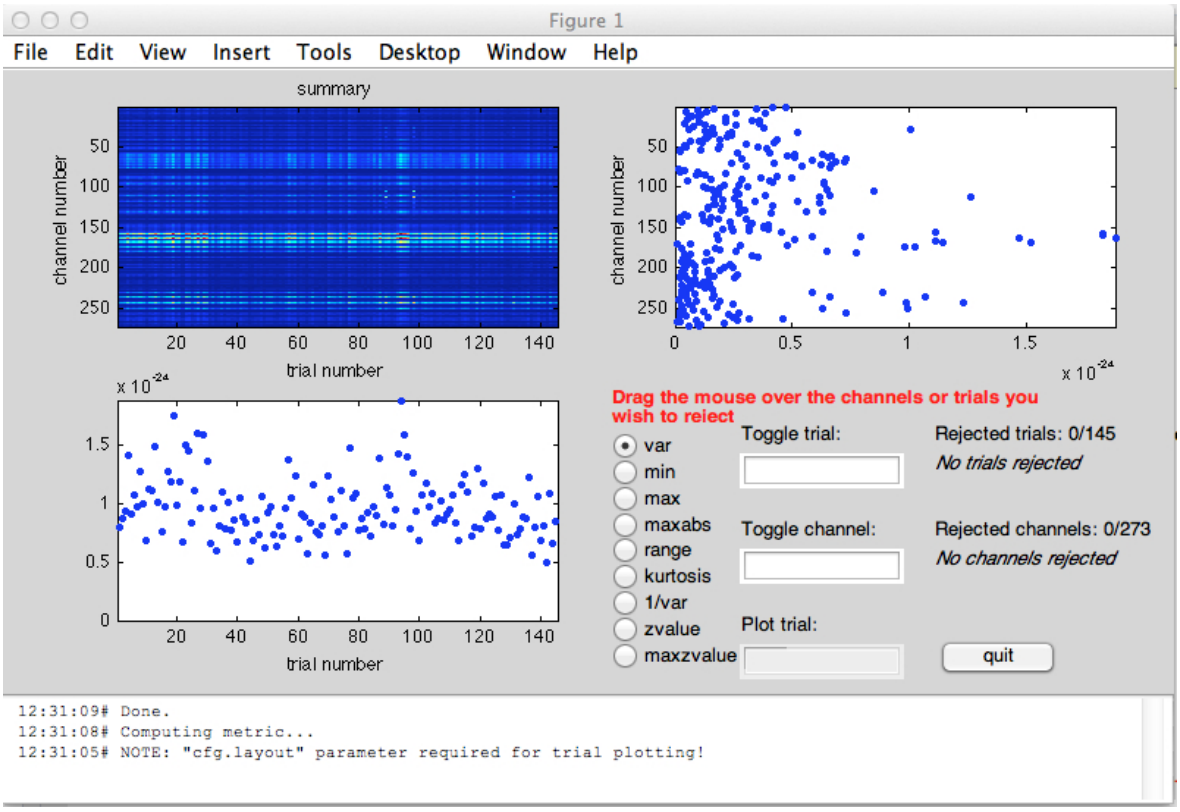

Figure 4: Fieldtrip GUI for visual artifact rejection

3.5.2 Authomatic rejection of channels and trials

Rather than make an manaul rejection it is possible to make an authomatic rejection of channels (and trials). To identify potentially bad channel it is possible to check if that channel is highly improbable given the data distribution on all the channels (the function jointprob is the same called for channel rejection in eeglab). An alternative is to use the function `rejkurt`, with the same parameters as in jointprob. The parameters of the code below are the same of the default parameters when making channel rejections in EEGLAB.

```
[prob indelec]=jointprob(cell2mat(data_megEpoch.trial), ...
5, [], 2);

find(indelec)
```

3.5.3 Interpolation of missing channels

Once you have deleted bad channel you can *interpolate* them with the function `ft_channelrepair`. To do this there are some (mechanic) steps to be performed.

```
% retrieve the labels of delete channels
rem_chans=setdiff(data_megEpochclean.cfg.channel, ...
data_megEpochclean.label)

cfg=[];
mylayout = ft_prepare_layout(cfg, data_megEpoch)

cfg=[];
neighbours=ft_prepare_neighbours(cfg)

cfg=[];
cfg.neighbours=neighbours
cfg.missingchannel=rm_chans

interp=ft_channelrepair(cfg, data_megEpoch)
```

3.6 Correcting head movements

. The main reason to use FieldTrip to analyze MEG data (rather than other software, like EEGLAB and BRAINSTORM) is that it allows to correct for

head motion. There are two main reference for head motion correction: the article by Stolk et al. (download it here `http://fieldtrip.fcdonders.nl/_media/faq/stolkneuroimage2013.pdf`) or this tutorial page on FieldTrip Wiki `http://fieldtrip.fcdonders.nl/example/how_to_incorporate_head_movements_in_meg_analysis`. There are two important things to keep in mind about head movement correction:

- head movement correction should be the last step before analysis (e.g. the last step before timelock analysis or source computation).
- head movement correction depends on the subsequent analysis. For example, if correction is made for timelock analysis (i.e., averaging) the corrected data **cannot** be used for source modeling.