MWF toolbox for EEG artifact removal - Manual

Ben Somers, Tom Francart, Alexander Bertrand

November 2017

Contents

1 Introduction

This document is the user manual for the MATLAB toolbox for EEG artifact removal based on the Multi-channel Wiener Filter (MWF). The MWF algorithm for EEG artifact removal is described and validated in detail in [1].

The toolbox is available online at www.github.com/exporl/mwf-artifact-removal. This manual is written for the 1.0 release version. New features may be added later.

2 Repository contents

The GitHub repository contains the following folders and files:

- **mwf folder**: contains all required MATLAB code for using the MWF.
- **doc folder**: contains manual and demo files.
- **paper folder**: contains code for generating the results and figures described in the paper [1]. These are only included for reproducibility of the paper, they are not needed for your own use of the MWF.
- **.gitignore**, **README** and **LICENSE** files. Note that the licence (online link) under which this code is distributed does not allow usage for commercial purposes.

3 Quick start guide

All functions needed to perform MWF-based EEG artifact removal are in the mwf folder. Before starting, make sure that this folder is added to the MATLAB path.

The MWF first requires examples of EEG with and EEG without artifacts. Based on this segmentation of the EEG data, the MWF can be computed and applied in order to remove the artifacts. This two-step approach is fully implemented in the toolbox.

Step 1: EEG segmentation. In the toolbox, this step is performed by manual marking of the data using a GUI[1]. If you have your EEG data matrix in the the MATLAB workspace (channels x samples), you can obtain the artifact mask by calling

```
mask = mwf_getmask(EEG, samplerate);
```

This pops up the GUI in which artifacts can be marked by clicking and dragging over them. When done, clicking the 'Save Marks' button will close the GUI and the function returns a binary (1 x samples) mask. In this mask, ones correspond to artifact segments, and zeros correspond to clean data. Optionally, the mask may contain NaNs which indicate segments to be ignored from the MWF computation (i.e. they belong neither to the artifact nor the clean segments). Section 4.1.4 contains extra tips on annotating artifacts.

Important note: all segments that are not marked and come BEFORE the lasted marked segment will be treated as artifact-free samples in the training of the filter. The samples that come after the last marked segment are not used in the filter design. In other words: the filter design assumes that *all* artifacts before the last marked artifact are marked.

Because samples after the lasted marked segment are ignored, it is not necessary to go through the entire signal to mark all the artifacts. It is sufficient to annotate only the first few seconds or minutes of the signal. However, the more artifacts are marked, the better the filter design will be. Additionally, there should be enough clean (unmarked) segments before the last marked artifact for a good filter design.

The artifact detection step is not inherently a part of the MWF algorithm: if you prefer, you can also use a different method for acquiring the artifact mask (e.g. an automatic method, for example based on thresholding,...). The mask needs to consist of ones, zeros and NaNs, and have the same length as the EEG data.

Step 2: MWF artifact removal is performed by calling the mwf_process function. It requires the EEG data, the mask indicating which segments are artifacts, and optionally a delay parameter:

```
clean_EEG = mwf_process(EEG, mask, delay);
```

This will return the artifact-free EEG in the clean_EEG variable. Using the optional delay parameter includes temporal information into the filter, leading to better artifact removal but may increase processing time. If omitted, the default value is zero. See [1] for more details.

[1]The GUI makes use of EEGLAB functionality. In order to visualize the EEG for manual marking, it is required to download the EEGLAB MATLAB toolbox www.sccn.ucsd.edu/eeglab/index.php. Make sure the EEGLAB toolbox is added to the MATLAB path as well.

4 Function documentation

The MATLAB m-files of the MWF functions contain detailed documentation about the functions usage, inputs/outputs, and some examples. In addition, the mwf_demo.m file in the doc folder contains a step-by-step demonstration of MWF usage on artifact removal in some example EEG data. For basic use, the in-file documentation, the demo, and the quick start guide in section 3 should suffice. The following subsections contain additional documentation on some advanced usage of the MWF functions.

4.1 mwf_getmask

```
mask = mwf_getmask(EEG, Fs, cacheID, cachepath, redo, mode);
```

4.1.1 Input arguments

The mwf_getmask function implements the artifact selection GUI based on EEGLAB's plotting functionality. For basic usage, it is sufficient to provide the **EEG** data (channels x samples) and the EEG sample rate **Fs**. The other arguments are related to caching (i.e. saving) the created mask for later reuse:

- **cacheID** is a character string which will allow identification of a cached mask, e.g. "subject1_trial2". The mask will be saved in a matfile with the cacheID in the name, in the folder specified by cachepath (see next point). If no cacheID is given, the mask will not be saved.
- **cachepath** is a string specifying the path where to save the cache. It is recommended to create a dedicated cache folder and store all cached masks in there. If unspecified, the current directory will be used.
- **redo**. Normally, if you specify a cacheID and cachepath, the function will retrieve a saved mask if it exists. Setting the redo flag to true will allow you to redo the artifact marking regardless of whether there exists a cache already. The existing cache entry matching the cacheID and cachepath will be overwritten. By default, redo is false.

Caching is very handy as it allows you to easily re-run the analysis without having to mark the artifacts again. For example usage of the caching, see the documentation in mwf_getmask.m.

The last argument **mode** specifies the GUI to use. By default, EEGLAB is used (mode 0). It is possible to use the EyeBallGUI[2] (mode 1), a different MATLAB toolbox, but its usage is no longer supported by the MWF toolbox and therefore not recommended.

4.1.2 Output

The **mask** variable contains the user's markings in a vector of the same length as the input EEG data. At time points where an artifact was marked, the vector contains ones, at unmarked time points the mask vector contains zeros. All samples after the last marked artifact segment are set to NaN, which will exclude them from the data used to train the MWF. This allows the user to only mark the first part of the signals instead of having to mark everything.

[2]This MATLAB toolbox can be found at http://eyeballgui.sourceforge.net

4.1.3 Using the GUI for artifact marking

Running this function will pop up an EEGLAB plot as depicted in figure 1. The EEG display can be customized (e.g. number of channels in view, time range to show, scale,...). The user can scroll through channels and time points to inspect the EEG for undesired artifacts. Artifacts can be marked by click-and-dragging over segments, which will highlight the segments as shown in the figure. When finished marking, clicking the "SAVE MARKS" button will close the window and build the mask.

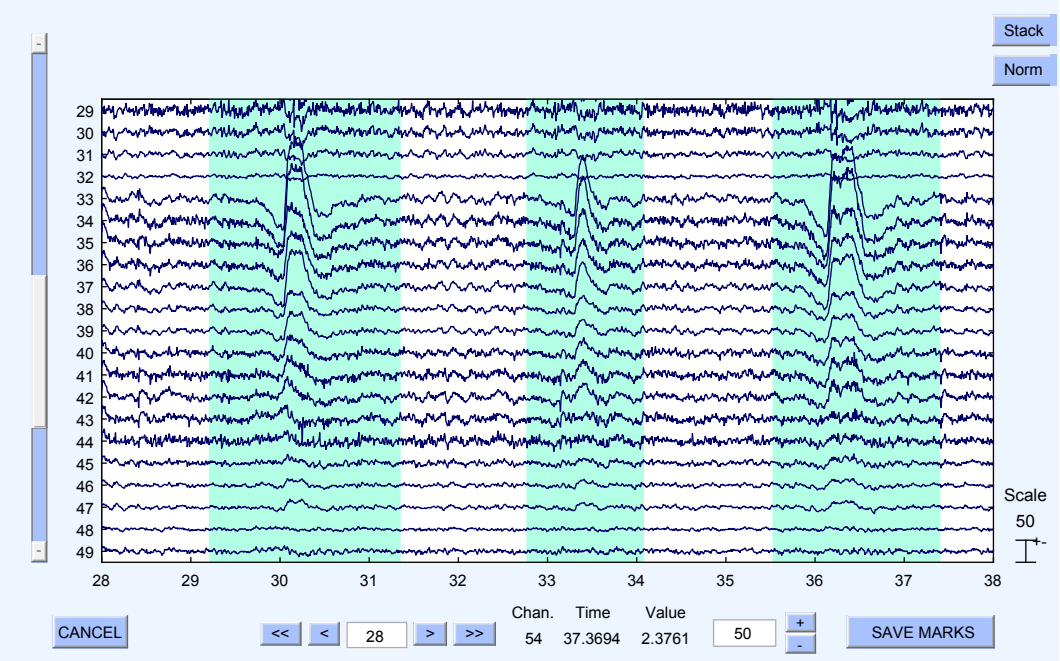

Figure 1: Example of marking some eye blink artifacts in the GUI.

4.1.4 Notes

- All segments that are not marked and come BEFORE the lasted marked segment will be treated as artifact-free samples in the training of the filter. The samples that come after the last marked segment are not used in the filter design. In other words: the filter design assumes that ALL artifacts before the last marked artifact are marked.
- The more examples of artifacts are marked, the better the MWF design will be.
- In addition to marking enough examples of artifacts, it is also important to have enough segments of clean (unmarked) EEG data in between the marked artifact segments.
- MWF performance decreases quickly if artifacts are "missed" and are included in the clean data. However, it is no problem when clean data is included in the artifact markings. Therefore, when marking artifact segments, it is recommended to draw broad windows around the artifacts. If you are unsure whether a segment is artifact or not, it is better to mark it anyway.
- If you alter the mask after obtaining it with this function, any value other than 1 and 0 will be ignored for the computation of the MWF. You can use this to exclude EEG segments from the training of the MWF (e.g. by setting them to NaN).

4.2 mwf_process

```
[n, d, W, SER, ARR, p] = mwf_process(EEG, mask, delay);
```

The mwf_process function processes the input **EEG** with the MWF to remove the artifacts as marked in the input **mask**. An option **delay** input argument can be specified to include temporal information in the MWF, which will improve artifact removal but increase computation time. The delay should be a positive scalar (samples) and is 0 by default. All other MWF parameters are set to their default values which should work well in general.

The function returns the clean, artifact-free EEG data **n** and the artifact estimate **d**. Other outputs are the filter matrix **W**, the performance measures **SER** and **ARR**, and the struct **p** containing the parameters used for the processing.

The mwf_process function is a high-level, user-friendly function and should generally yield good results for a well-chosen mask. If the user wants more control over the filter parameters, the functions in the next few subsections can be used.

4.3 mwf_params

```
p = mwf_params('parameter1', 'value1', 'parameter2', 'value2', ...);
```

4.3.1 Specification of MWF parameters

This function creates a MATLAB structure **p** containing key-value pairs of parameters defining the MWF processing. The struct should be passed on to the mwf_compute function (see section 4.4). Parameters can be set by providing their keyword followed by the desired value as an input, for example:

```
p = mwf_params('delay', 5, 'rank', 'poseig');
```

Unspecified parameters will be kept at their default value.

The following parameters are able to be set:

- **delay**: expects a non-negative integer. Specifies the amount of channel lags, expressed in samples, to incorporate in the filter, e.g. if delay is 5, then all delays from -5, ..., 5 will be included. The default value is 0 (i.e. no delays will be inlcuded).
- **rank**: expects one of the following strings (see [1] for more details):
  - 'full': creates MWF filter where all eigenvalues are retained
  - 'poseig': creates MWF filter where only positive eigenvalues are retained (default)
  - 'first': creates MWF where only the X largest eigenvalues are retained
  - 'pct': creates MWF where only X% of largest eigenvalues are retained
- **rankopt**: specify X if 'first' or 'pct' were selected for the rank parameter. Expects a positive integer ('first') or a percentage ('pct').
- **treatnans**: expects one of the following strings :
  - 'ignore': any NaNs in the mask are excluded from MWF training (default)
  - 'artifact': any NaNs in the mask are considered as artifact for MWF training
  - 'clean': any NaNs in the mask are considered as clean data for MWF training
- **mu**: noiseweighting factor. By default, mu is 1, which results in a regular MWF. If mu > 1, a so-called Distortion Weighted MWF (DW-MWF) [2] is computed, which allows better artifact estimation, but introduces more noise in the artifact-free segments (i.e. higher ARR but lower SER). The opposite trade-off is true when mu < 1 (but greater than 0).

4.3.2 Notes

- In [1], it was shown that there is a benefit of including some delays (e.g. 5 samples) in the MWF design. However, there is a trade-off with computation time.
- In [1], it was shown that the poseig option for rank generally selects the optimal rank for the MWF, which is why it is the default option.
- The factor mu allows to trade off artifact estimation quality for clean EEG distortion. It is recommended to leave it at the default value of 1.

4.4 mwf_compute

```
[W, Lambda] = mwf_compute(EEG, mask, p);
```

This function computes the MWF based on the input **EEG**. The EEG is split into artifacted and clean segments using the **mask**. By default, any samples corresponding to NaNs in the mask are ignored from the MWF computation. Additional parameters involved in the processing are set in the parameter struct **p**.

The function returns the MWF matrix **W**, as well as an ordered diagonal matrix containing the generalized eigenvalues in **Lambda**. The sizes of W and Lambda will be M x M, where M is the number of channels *including delayed channels*.

4.5 mwf_apply

```
[n, d] = mwf_apply(EEG, W);
```

This function applies a precomputed MWF matrix **W** to the input **EEG** data. The filtered, artifact-free EEG data **n** is returned, as well as the estimate of the artifacts **d**. It may be informative to inspect the pure artifact signal d to see what the artifacts look like without the rest of the EEG.

Because the compute and apply functionality if split in two different functions, it is possible to compute an MWF on one set of EEG signals with mwf_compute, and apply it to a different set of EEG signals with mwf_apply. One restriction is that the number of channels in both EEG sets need to be the same, otherwise the precomputed filter W is incompatible with the EEG test set.

4.6 mwf_performance

```
[SER, ARR] = mwf_performance(EEG, d, mask, d_real);
```

This function computes two performance measures SER and ARR, which are defined in [1]. The **SER** is a measure for distortion of clean EEG caused by the filter. The **ARR** is a measure for artifact estimation accuracy by the filter. Good performance is indicated by both high SER and high ARR. Both values are expressed on a decibel scale.

In order to compute these measures, the function requires the **EEG** data, as well as the artifact signal **d** estimated by the MWF, and the **mask** used to split EEG in artifact and clean segments (NaNs in the mask are ignored). An optional fourth input **d_real** can be provided, which should contain the real, ground-truth artifact signal that is estimated by **d**. Of course, this is only possible to provide for synthetic or simulated EEG data. If d_real is provided, a more accurate ARR can be computed.

5 Closing remarks

The MWF-EEG code is distributed freely online in the hope that it may provide a useful method for other researchers in dealing with artifacts in their EEG data. See the license file distributed along with the toolbox for license rights and limitations. If this toolbox is useful for your data analysis, please cite [1] in any publications that may follow from it.

We are open to any suggestions for interesting features or ways to further improve the toolbox. Communication regarding the MWF-EEG toolbox can be addressed through the GitHub website or directly to ben.somers@med.kuleuven.be

References

[1] B. Somers, Ben, T. Francart and A. Bertrand, "A generic EEG artifact removal algorithm based on the multi-channel Wiener filter", *Journal of Neural Engineering*, (2018).

[2] R. Serizel, M. Moonen, J. Wouters and S.H. Jensen, "Integrated active noise control and noise reduction in hearing aids", *IEEE Transactions on Audio, Speech, and Language Processing*, 18.6 (2018):1137-1146.