Computational Testing for Automated Preprocessing: a Matlab toolbox for better electroencephalography data processing

Benjamin U Cowley Corresp., [1, 2] , **Jussi Korpela** [1] , **Jari Torniainen** [1]

[1] BrainWork Research Centre, Finnish Institute of Occupational Health, Helsinki, Finland
[2] Cognitive Brain Research Unit, Institute of Behavioural Sciences, University of Helsinki

Corresponding Author: Benjamin U Cowley
Email address: benjamin.cowley@ttl.fi

EEG is a rich source of information regarding brain functioning, and is the most lightweight and affordable method of brain imaging. However, the pre-processing of EEG data is quite complicated and most existing tools present the experimenter with a large choice of methods for analysis, but no framework for method comparison to choose an optimal approach. Additionally, many tools still require a high degree of manual decision making for, e.g. the classification of artefacts in channels, epochs or segments. This introduces excessive subjectivity, is slow, and is not reproducible. Batching and well-designed automation can help to regularise EEG preprocessing, and thus minimise human effort, subjectivity, and consequent error. The Computational Testing for Automated Preprocessing (CTAP) toolbox facilitates: i) batch processing that is easy for experts and novices alike; ii) testing and comparison of automated methods. CTAP uses the existing data structure and functions from the well-known EEGLAB tool, based on Matlab, and produces extensive quality control outputs.

Computational Testing for Automated Preprocessing: a Matlab toolbox for better electroencephalography data processing

Benjamin Cowley[1,2,*], Jussi Korpela[1], and Jari Torniainen[1]

[1]BrainWork Research Centre, Finnish Institute of Occupational Health, Topeliuksenkatu 41 b, 00250 Helsinki, Finland
[2]Cognitive Brain Research Unit, Institute of Behavioural Sciences, University of Helsinki
[*]Corresponding author:
Benjamin Cowley
ben.cowley@helsinki.fi

ABSTRACT

EEG is a rich source of information regarding brain functioning, and is the most lightweight and affordable method of brain imaging. However, the pre-processing of EEG data is quite complicated and most existing tools present the experimenter with a large choice of methods for analysis, but no framework for method comparison to choose an optimal approach. Additionally, many tools still require a high degree of manual decision making for, e.g. the classification of artefacts in channels, epochs or segments. This introduces excessive subjectivity, is slow, and is not reproducible. Batching and well-designed automation can help to regularise EEG preprocessing, and thus minimise human effort, subjectivity, and consequent error. The Computational Testing for Automated Preprocessing (CTAP) toolbox facilitates: i) batch processing that is easy for experts and novices alike; ii) testing and comparison of automated methods. CTAP uses the existing data structure and functions from the well-known EEGLAB tool, based on Matlab, and produces extensive quality control outputs.

Keywords: electroencephalography, computation, testing, automation, preprocessing, EEGLAB, signal processing

INTRODUCTION

Measurement of human electroencephalography (EEG) is a rich source of information regarding certain aspects of brain functioning, and is the most lightweight and affordable method of brain imaging. However, among those types of human electrophysiology data recorded from surface electrodes (to which EEG is most similar in terms of recording methods, see e.g. Cowley et al. (2016) for a review), EEG data is comparatively difficult to pre-process. The qualities which cause difficulty for EEG analysis come in two classes: A) number and complexity of operations, and B) size and indeterminacy of the data.

Specifically in class A, normally many operations are required, which is time-consuming and therefore costly. Many of these operations require repeated human judgements, leading to subjectivity, non-reproducibility of outcomes, and non-uniformity of decisions. Compared to, e.g., counting peaks in an electrocardiogram signal, most variables of interest are relatively complicated derivations from the raw signal, implying more room for error in analysis. Related to that, the relatively complex 'standard' EEG processing operations are harder to debug.

In class B, for most research applications we can see that EEG data is high-bandwidth, systems which consist of 256+ channels are available. Due to the *inverse problem* it is not possible to precisely determine a 'ground truth' for the signal, i.e. a unique relationship to neural sources. The signal is also highly variable between individuals, and even between intra-individual recording sessions Dandekar et al. (2007).

Bigdely-Shamlo et al. (2015) emphasise the need for tools to process EEG data in an efficient manner, and have pointed out that "artifact removal and validation of processing approaches remain a long-standing open problem for EEG". While some standards have been suggested Keil et al. (2014), there remains a deficit of tools and methods to support standardisation. These issues illustrate the need for a software tool that helps to minimise human effort, subjectivity, and consequent error. Batching and careful automation can help to regularise and streamline EEG pre-processing, for which we present a solution in this paper.

Approach

We present the Computational Testing Automated Preprocessing (CTAP) toolbox, available from GitHub [1]. CTAP is built on Matlab (r2015a and higher) and EEGLAB v13.4.4b (Delorme and Makeig, 2004). The main aim of CTAP is to regularise and streamline EEG preprocessing. We regularise with a scripted data-processing pipeline that ensures the treatment of each file is the same. We streamline by separating the process of building functions from that of exploring and tuning the data. These features improve reproducibility, and separate the menial and important tasks, respectively.

In practice, the CTAP toolbox provides functionality for i) batch processing using scripted EEGLAB-compatible functions; ii) testing and comparison of automated methods based on extensive quality control outputs. In more detail, the core code supports scripted specification of a pipeline of diagnostic and artefact correction steps, with robust looping execution of steps and automated output of 'quality control' (QC) logs and imagery (more details provided below in Results). This helps to maintain transparency and traceability of all operations performed for every EEG file. After specification the pipeline can be adjusted to respond to QC issues.

Although CTAP works as a batch processing pipeline, it supports seamless integration of manual operations. This works such that the user can define a pipeline of operations, insert save points at appropriate steps, and work manually on that data before passing it back to the pipe.

CTAP is focused on leveraging existing methods that are compatible with EEGLAB-structured data, and on providing a simple interface to plug in different styles of EEG signal processing. The contribution we present is to extend EEGLAB for automated batch processing. This facilitates reproducible brain imaging research with the following features:

- script based automated batch processing (no GUI)
- time consuming bookkeeping of intermediate analysis files and script execution is automated
- existing EEGLAB based analysis methods can be easily integrated
- automated QC output help to spot problems (see section Peeks)
- includes ready made tools for feature storage and export (see section CTAP outcomes)

In summary, CTAP lets the user focus on content, instead of time-consuming implementation of foundation functionality. In the rest of the paper, we will describe how CTAP toolbox does this, and provide a motivating example of its application.

After we address related work, section Materials & Methods details the architecture and usage of CTAP. Section Results then describes the technical details and outcomes of a motivating example application. In section Discussion we set out the philosophy and possible uses of CTAP toolbox, including development as well as preprocessing; and describe issues and potential directions for future work.

RELATED WORK

Many methods are available from the literature to facilitate automated preprocessing (for a review see, e.g. Barua and Begum (2014)), and the rate of new contributions is also high. For example, we conducted a search of the SCOPUS database for articles published after 1999, with "EEG"

[1] `https://github.com/bwrc/ctap`

and ”electroencephalography” in the title, abstract, or keywords, plus ”Signal Processing” or ”Signal Processing, Computer-Assisted” in keywords, and restricted to subject areas ”Neuroscience”, ”Engineering” or ”Computer Science”. The search returned over 300 hits, growing year-by-year from 5 in 2000 up to a mean value of 36 between 2010 and 2015. Non-systematic reviews of the software tools available have been made by Agapov et al. (2016); Baillet et al. (2010), and in a milestone special issue, Baillet et al. (2011) gathered a large number of the academic contributions available at that time. This special issue is quite skewed toward tools for feature extraction, which illustrates again the need for better/more up-to-date solutions for the fundamental stages of EEG processing.

Among tools dedicated to EEG processing, EEGLAB (Delorme and Makeig, 2004) stands out for popularity and high number of third-party contributors, to the degree that it is considered by some to be a *de facto* standard. However EEGLAB is a graphical user interface (GUI)-based tool, which limits the scale at which it can be used.

Other popular tools focus on a more diverse set of signals, especially including magnetoencephalography (MEG). Brainstorm Tadel et al. (2011), Fieldtrip (Oostenveld et al., 2011), and EMEGS (ElectroMagnetic EncaphaloGraphy Software) Peyk et al. (2011) are all open source tools for EEG and MEG data analysis. Like EEGLAB, these tools are all *free* and *open source*, but based on the commercial platform Matlab (Natick, MA), which can be a limitation in some contexts due to high licence cost. Brainstorm in particular, but also the others, have originated with an emphasis on cortical source estimation techniques and their integration with anatomical data. More recently, Bigdely-Shamlo et al. (2015) released the PREP pipeline for Matlab, which also uses the EEGLAB data structure but is aimed only at experiment-induced artefacts and not those deriving from subject-activity such as, e.g. blinks.

The most notable commercial tool is Brainanalyzer (Brain Products GmbH, Munich, Germany), a graphical programming interface with a large number of features. NeuroPype is a commercial Python-based graphical programming environment for biosignal processing. It is only available as a closed beta and, to the authors' knowledge, has not been documented in a peer reviewed publication.

Tools which are completely *free* and *open source* are fewer in number and have received much less supplemental input from third parties. Python tools include MNE-Python for processing MEG and EEG data (Gramfort et al., 2013), and PyEEG (Bao et al., 2011), a module for EEG feature extraction. MNE, like Brainstorm and Fieldtrip, is primarily aimed at integrating EEG and MEG data. Several packages exist for the R computing environment, e.g. Tremblay and Newman (2015), however these do not seem to be intended as general-purpose tools.

We have chosen to extend EEGLAB because it has received many contributions to the core functionality, and is thus compatible with a good portion of the methods of EEG processing from the literature. Some compatible tools from the creators of EEGLAB at the Swartz Centre for Computational Neuroscience (SCCN) are detailed in Delorme et al. (2011), including tools for forward head modelling, estimating source connectivity, and online signal processing. Other key third-party preprocessing contributions to EEGLAB include SASICA (Chaumon et al., 2015), FASTER (Nolan et al., 2010), and ADJUST (Mognon et al., 2011), all semi-automated solutions for selection of artefactual data. The latter two are featured in CTAP as options for detecting bad data.

This integration of existing solutions illustrates the key difference of CTAP: it aims to extend an existing rich ecosystem of EEG-specific methods, by meeting a clear need within that ecosystem.

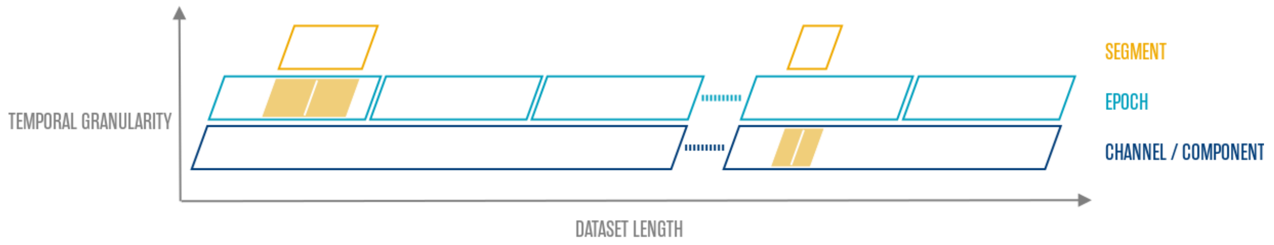


Figure 1. Relationship of the time domain data constructs dealt with in CTAP.

MATERIALS & METHODS

The core activity of CTAP is preprocessing EEG data by cleaning artefacts, i.e. detection and either correction or removal of 'bad' data, that is not likely to be attributable to neural sources. CTAP is able to operate on three different temporal granularities: channel, epoch and segment. Channel operations affect the entire time series at one spatial location. Epoch operations are performed on one or several epochs produced by EEGLAB epoching function. Finally, segments are fixed time-windows around specfic events which can be extracted from both channel and epoch levels, see Figure 1. An example of a typical segment could be a blink artefact with a window wide enough to include the entire blink waveform. Further functionality is provided for independent component analysis (ICA)-based methods. Artefact-detection methods based on some flavour of ICA algorithm have been shown to outperform temporal approaches Delorme et al. (2007). It was also shown that independent components (ICs) are valid representations of neural sources (Delorme et al., 2012). CTAP can thus help to combine the existing methods for EEG signal processing.

Outline of usage

Figure 2 shows the core components of CTAP. The colored boxes represent entities that the user has to specify in order to use CTAP. These are:

- what analysis functions to apply and in which order (analysis-pipe)
- parameters for the analysis functions (parameters)
- which EEG measurements/files to process (what-to-analyze)

Typically the analysis is run by calling a single script that defines all of the above and passes these on to a function that performs all requested analysis steps on all specified measurements. In the following, we describe in more detail how the configurations are made, how the pipe is executed, what outputs it provides and what options the user has to control the pipe. The complete details of all these aspects of CTAP are provided in the wiki pages of the GitHub repository, which will be referenced below as 'the wiki' [2].

Configuration

Let us assume that the main analysis script is stored in a file called `runctap_projectX.m`. First of all, this file specifies which *analysis steps* are performed in which order. An example of an analysis step is e.g. filtering or bad channel detection. Analysis steps can further be grouped into sets of steps referred to as *step sets*. An intermediate save is done after each step set, providing

[2] `https://github.com/bwrc/ctap/wiki`

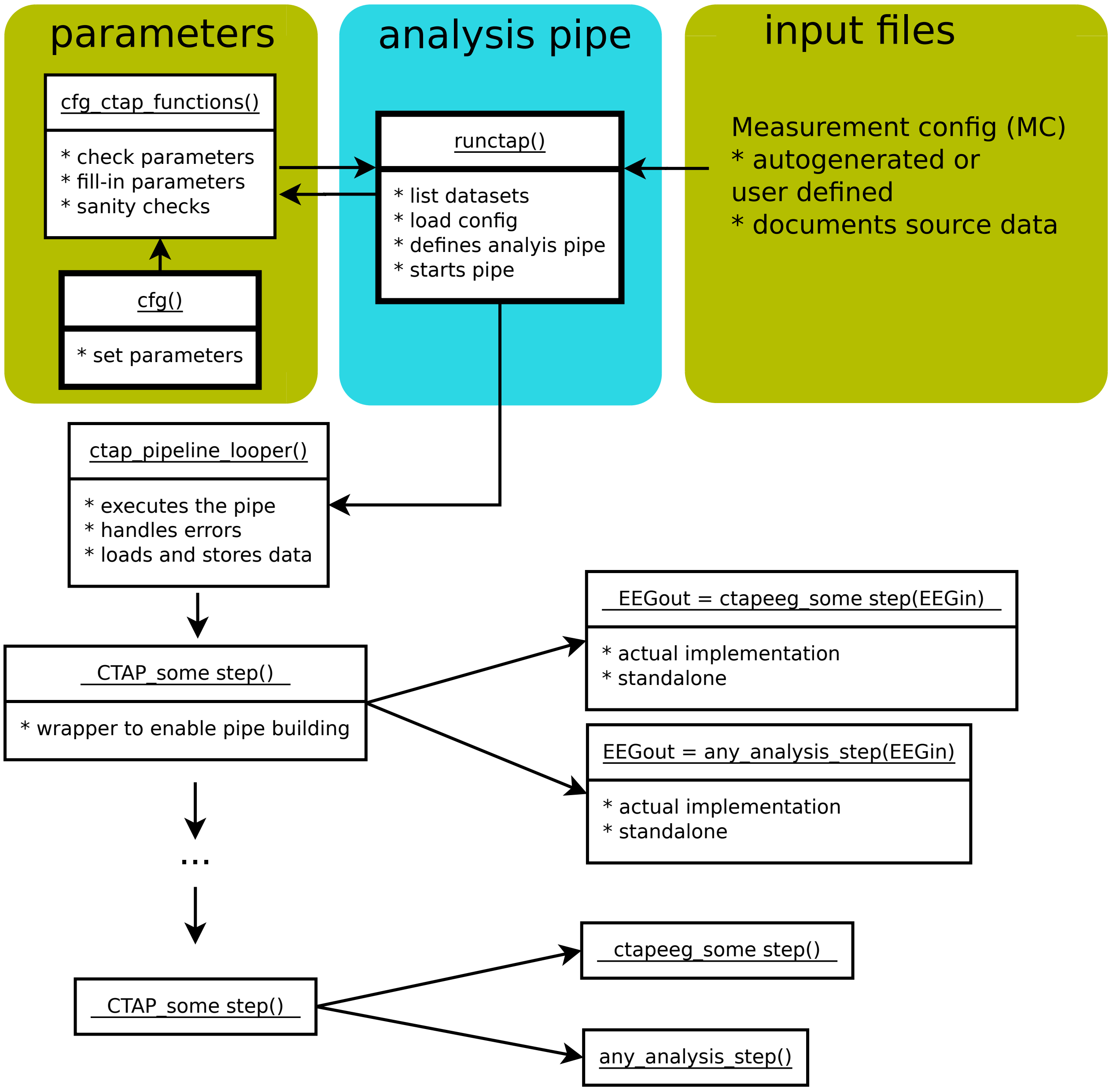


Figure 2. An overview of the core logic of CTAP. 'parameters', 'analysis pipe' and 'input files' illustrate the parts user must specify. White boxes represent Matlab functions, with the function-name on top and bulleted notes below. The functions which require user attention, `cfg()` and `runctap()`, have bold borders and can be freely named by the user. For good practice, these two functions take the name of the analysis pipe they implement, e.g. `runctap_mypipe()` .

a possibility run the whole pipe in smaller chunks. Below is a sample definition of a very small analysis pipe with two step sets:

```
i = 1; %stepSet 1
stepSet(i).funH = { @CTAP_load_data,...
                    @CTAP_load_chanlocs,...
                    @CTAP_tidy_chanlocs,...
                    @CTAP_reref_data,...
                    @CTAP_blink2event};
stepSet(i).id = [num2str(i) '_load_WCST'];
stepSet(i).srcID = '';

i = i+1;  %stepSet 2
stepSet(i).funH = { @CTAP_filter_data};
stepSet(i).id = [num2str(i) '_filter'];
stepSet(i).srcID = '';
```

The core function for pipe processing is designed to use functions named `CTAP_*()`, as these are defined to have a fixed interface. They take two arguments: data (`EEG`) and configuration struct (`Cfg`); and they return the same after any operations. Some `CTAP_*()` perform all operations (e.g. call EEGLAB functions) directly, while others call a corresponding `ctapeeg_()` function that actually implements the task. Hence `CTAP_*()` functions can be regarded as wrappers that facilitate batch processing (uniform interface) and also implement e.g. the plotting of quality control figures. Since they are quite simple, new `CTAP_*()` functions can easily be added by the user to include new analysis steps, according to the provided `CTAP_template_function()`. Users can also call the `ctapeeg_()` functions directly as part of their own custom scripts, since these are meant to be used like e.g. any EEGLAB analysis function.

Once the analysis functions have been defined the next step is to define parameters for the whole pipe, and for each analysis function. Default parameters are provided for most cases, but it is optimal to fine tune the behaviour. Like before, parameter information is passed to CTAP using a struct. It is usually practical to store the struct in a separete m-file, lets say `ctapcfg_projectX.m`. A typical minimal contents of this file might be:

```
function [Cfg, FP] = cfg_minimal(dataRoot, branchID)

%% Analysis output (data, quality control) storage location
Cfg.env.paths.analysisRoot = fullfile(dataRoot,'ctap',branchID);

%% Channel specifications
Cfg.eeg.chanlocs = fullfile(dataRoot,'channel_locations_acticap_32.ced');
Cfg.eeg.reference = {'TP10' 'TP9'}; % EEG reference channels to use

%% Configure analysis functions

% Load data
FP.load_data = struct('type', 'neurone');

% Amplitude thresholding from continuous data (bad segments)
FP.detect_bad_segments = struct('amplitudeTh', [-100, 100]); %in muV
```

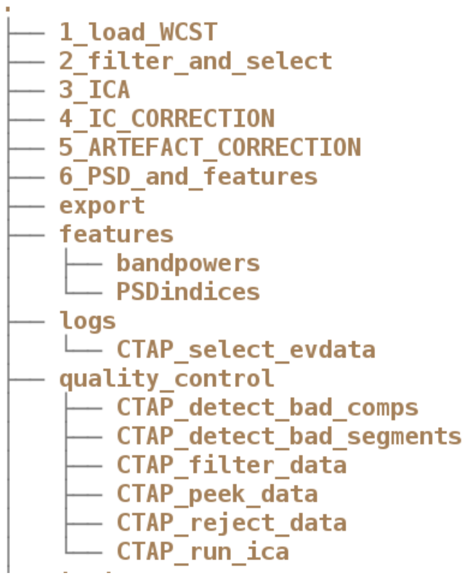
The third requirement to run the pipe is input data. In CTAP the input data are specified using a table-like structure called *measurement config* that lists all available measurements, the corresponding raw EEG files etc. The reason for using a dedicated data structure is that it allows for an easy selection of what should be analysed and it also helps to document the project. The measurement config structure can be either created manually, or it can be auto-generated based on a list of files or a directory (for details see the wiki). The former allows for full control and enforces project documentation whereas the latter is intended for effortless one-off analyses. Both spreadsheet and SQLite formats are supported.

In the last required step before pipeline execution, the configuration struct and the parameter struct are checked, finalised and integrated by `cfg_ctap_functions()`.

Pipe execution

Once all the prequisites listed above have been specified, the pipe is run using `CTAP_pipeline_looper()`. This function takes care of loading the correct (initial or intermediate) data set, applying the specified functions from each step set, and intermediate saving of the data. The looper manages error handling such that it is robust to crashing (unless in Debug mode), and will simply skip the remaining steps for a crashed file. Other settings determine how to handle crashed files at later runs of the pipe (see Documentation).

Analysis results are saved into the `Cfg.env.paths.analysisRoot` directory. A typical structure of this directory is shown in Figure 3.



```
.
├── 1_load_WCST
├── 2_filter_and_select
├── 3_ICA
├── 4_IC_CORRECTION
├── 5_ARTEFACT_CORRECTION
├── 6_PSD_and_features
├── export
├── features
│   ├── bandpowers
│   └── PSDindices
├── logs
│   └── CTAP_select_evdata
├── quality_control
│   ├── CTAP_detect_bad_comps
│   ├── CTAP_detect_bad_segments
│   ├── CTAP_filter_data
│   ├── CTAP_peek_data
│   ├── CTAP_reject_data
│   └── CTAP_run_ica
```

Figure 3. Typical directory tree.

First five directories contain intermediate results; their names are taken from the step set IDs as defined by the user. Directory `export` contains exported feature data (txt, csv or SQLite format), `features` computed EEG features in Matlab format, `logs` log files from each run, and `quality_control` contains quality control plots. The quality control plots reflect the analysis steps chosen by the user.

Apart from running the complete pipe at once the user has many options to run just a subset of the pipe, analyse only certain measurements, or otherwise adjust usage. Table 1 gives some examples.

CTAP outcomes

Visual evaluation CTAP automatically produces plots that help the user to answer questions such as: what has been done, what the data looks like, and was an analysis step successful or not. A full

Table 1. Some advanced ways to use the pipe.

Usage Options	**Possible Reasons**	**How**
Subset step sets	investigate a bug; recompute only intermediate results	set run sets to subset index, e.g. `Cfg.pipe.runSets = 3:5`
Run *test* step set	test new feature before including in pipe	add step set with id 'test', then set `Cfg.pipe.runSets = 'test'`
'Rewire' the pipe	test an alternative ordering of existing steps or temporarily change the input of some step	set the `.srcID` of a given step set equal to the `id` of another
Measurement configuration filter	run pipe for: subset of test subjects, or: measurements classes with separate configurations, e.g. pilots	use function `struct_filter()`
Run in debug mode	develop new method in CTAP	set `CTAP_pipeline_looper` parameter `'debug', true`
Overwrite obsolete results	update part of pipe: write new step set output over existing files	set `CTAP_pipeline_looper` parameter `'overwrite', true`
Write files from failed step sets	check partial outcome of step set	set `CTAP_pipeline_looper` parameter `'trackfail', true`
Turn off intermediate saves	extract numerical/visual analytics without producing updated files	set `stepSet(x).save = false;` set `stepSet(x+1).srcID = stepSet(x-1).id`

list of features can be found in the wiki documentation. Selected visualizations are described in more detail in Section Results. Examples include:

- blinks: detection quality, blink ERP
- bad segments: snippets of raw EEG showing detections
- EEG amplitudes: amplitude histograms, peeks
- filtering: PSD comparison
- ICA: IC scalpmap contact sheets, zoom-ins of bad components

Quantitative evaluation Every major pipe operation writes a record to the main log file. Data rejections, including channels, epochs, ICs or segments, are summarised here and also tabulated in a separate 'rejections' log. Values are given for how much data was marked as bad, and what percentage of the total was bad. If more than 10% of data is marked bad by a single detection, a warning is given in the main log.

In addition, useful statistics of each channel are logged at every call to `CTAP_peek_data()`, based on the output of the EEGLAB function `signalstat()`. Datapoints include trimmed and untrimmed versions of mean, median, standard deviation as well as skewness, kurtosis and normality testing.

Feature export Extracted EEG features are stored internally as Matlab structs that fully document all aspects of the data. These can be used to do statistical analysis inside Matlab. However, often users like to do feature processing in some other environment such as R or similar. For this, CTAP provides export functionality that transforms the EEG feature mat files into txt/csv text files, and/or an SQLite database. For small projects (for example, up to 10 subjects and 16 channels) txt/csv export is feasible but for larger datasets SQLite is more practical.

Data formats CTAP uses a number of custom data formats for configuration and data export purposes. Examples include the *measurement config* file for storing information related to available raw EEG files, measurement structure specifications and internal feature storage format used for storing features prior to export. All these are documented in detail in the wiki pages. However, as demonstrated by the motivating example (section Results), a basic analysis using CTAP does not require any knowledge of these.

RESULTS

We describe a motivating example that can also be used as a starting point for one's own analysis pipe.

Data. We use synthetically generated data with blink, myogenic (EMG), and channel variance artefacts to demonstrate the usage and output of CTAP. The example is part the repository (see function `runctap_manuscript.m`) and the details of the synthetic data generation process are documented in the wiki[3].

Pipeline. In the following sections, we show some example output of CTAP applied to the synthetic dataset, based on the analysis-pipe step sets shown below. The analysis steps marked red are next discussed in more detail. For each step, we will first step through the data processing outcomes, then illustrate the QC output generated.

```
stepSet(1).funH = { @CTAP_load_data,...
                    @CTAP_load_chanlocs,...
                    @CTAP_tidy_chanlocs,...
                    @CTAP_reref_data,...
                    @CTAP_blink2event,...
                    @CTAP_peek_data};

stepSet(2).funH = { @CTAP_filter_data,...
                    @CTAP_select_evdata,...
                    @CTAP_run_ica};
```

[3]https://github.com/bwrc/ctap/wiki/syndata-generation

```
stepSet(3).funH = { @CTAP_detect_bad_comps,...
                    @CTAP_reject_data,...
                    @CTAP_peek_data};

stepSet(4).funH = { @CTAP_detect_bad_channels,...
                    @CTAP_reject_data,...
                    @CTAP_detect_bad_segments,...
                    @CTAP_reject_data,...
                    @CTAP_interp_chan,...
                    @CTAP_peek_data};

stepSet(5).funH = { @CTAP_compute_psd,...
                    @CTAP_extract_bandpowers,...
                    @CTAP_extract_PSDindices};
```

Blinks

Detection The function `CTAP_blink2event()` creates a set of new events with latencies and durations matched to detected blinks.

The current blink detection implementation is based on the EOGERT algorithm by Toivanen et al. (2015) [4]. The algorithm finds all local peaks in the data, constructs a criterion measure and classifies peaks into blinks and non-blinks based on this measure.

Blinks can either be rejected or corrected using a method combining blink template matching and Empirical Mode Decomposition (EMD) based on high-pass filtering of ICs by Lindsen and Bhattacharya (2010). Thus, in this pipe the blink events are used by `CTAP_detect_bad_comps()` and `CTAP_reject_data()` to detect and correct the blinks.

Visualization The EOGERT detection process visualizes the classification result for QC purposes, as shown in Figure 4. Such figures make it easy to spot possible misclassifications.

The success of the blink correction is evaluated using blink Evoked Response Potentials (ERPs) (see e.g. Frank and Frishkoff (2007)) as shown in Figure 5. The correction method clearly removes the blink but leaves the underlying EEG largely intact. Some baseline shifts are visible but distortions to oscillatory activity are minor.

Peeks

Peek logic The `CTAP_peek_data()` function helps to regularize data inspection. Visual inspection of raw data is a fundamental step in EEG evaluation. A logical approach is to compare the raw data from before and after any correction operations. If ICA-based corrections are made, the same approach can also be used on the raw plot of IC data. CTAP aims to expedite this step. `CTAP_peek_data()` will generate raw data plots of randomly-chosen time-points [5]. These 'peek-points' are embedded as events which can then be plotted at a later stage in the pipe, generating true before-after comparisons even if the data time course changes (due to, e.g., removal of segments). If no peek-point data remains at the 'after' stage, no comparison can be made; however if peek-points are randomly chosen, then such an outcome is itself a strong indication that the data is very bad, or the detection methods are too strict.

[4]See code repository at `https://github.com/bwrc/eogert`
[5]Points can also be set by the user, e.g. chosen from existing events.

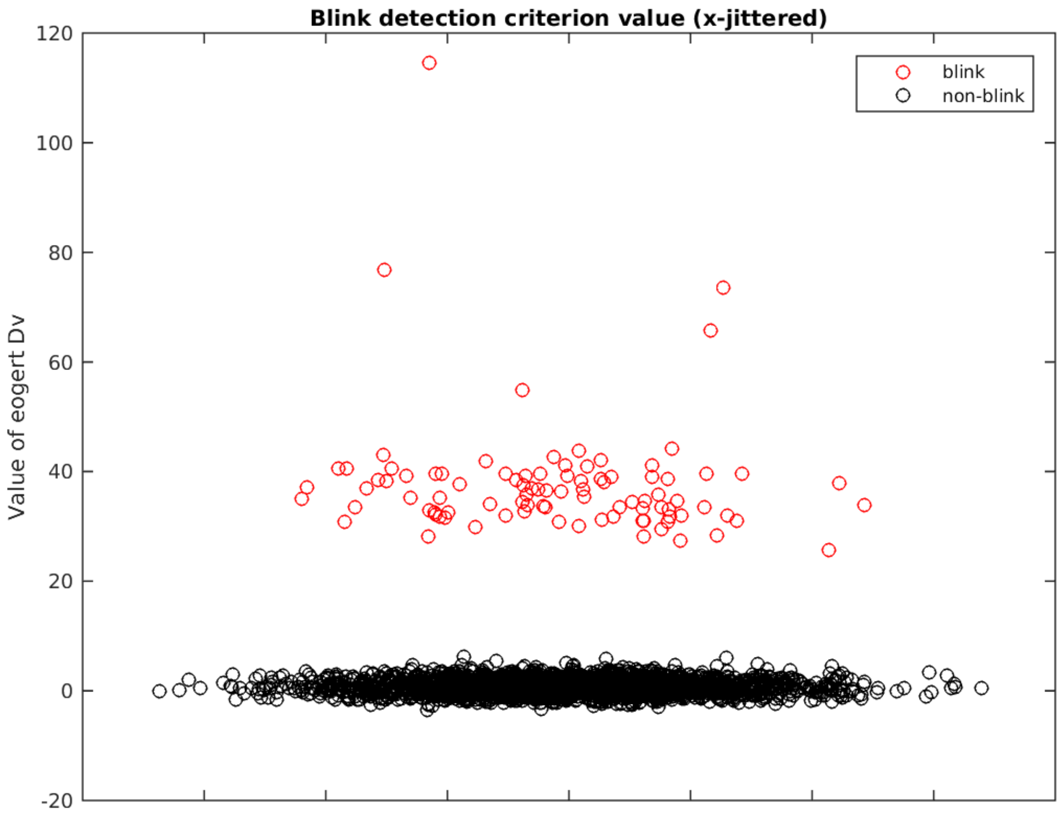


Figure 4. An example of the criterion used to detect blinks. Vertical axis shows the criterion value while horzontal axis is random data to avoid overplotting. The classification is done by fitting two Gaussians using the EM algorithm and assigning labels based on likelihoods.

`CTAP_peek_data()` will also estimate a set of statistics for every data channel, saved in Matlab table format and also aggregated to a log file.

EEG amplitudes Many EEG artefacts cause large changes in signal amplitudes and consequently several basic yet effective EEG artefact detection methods are based on amplitude thresholding. On the other hand measurement conditions and test subject can affect the amplitude of the recorded signal. Hence accurate knowledge of the average signal amplitude is often important. CTAP includes plotting routines for signal amplitude histograms as well as for raw EEG data.

A sample signal amplitude histogram produced using CTAP is show in Figure 6. It can be used e.g. to detect loose electrodes or in finding a suitable threshold for bad segment detection.

Bad IC detection & rejection

CTAP usage logic suggests that one or more detect operations for a given data type, e.g. channels, *or* epochs, *or* components, should be followed by a reject operation. It is bad practice to detect bad data across modalities, e.g. channels and epochs, before rejecting any of it, because artefacts of one type may affect the other. Thus we describe `CTAP_detect_bad_comps()` and `CTAP_reject_data()` together.

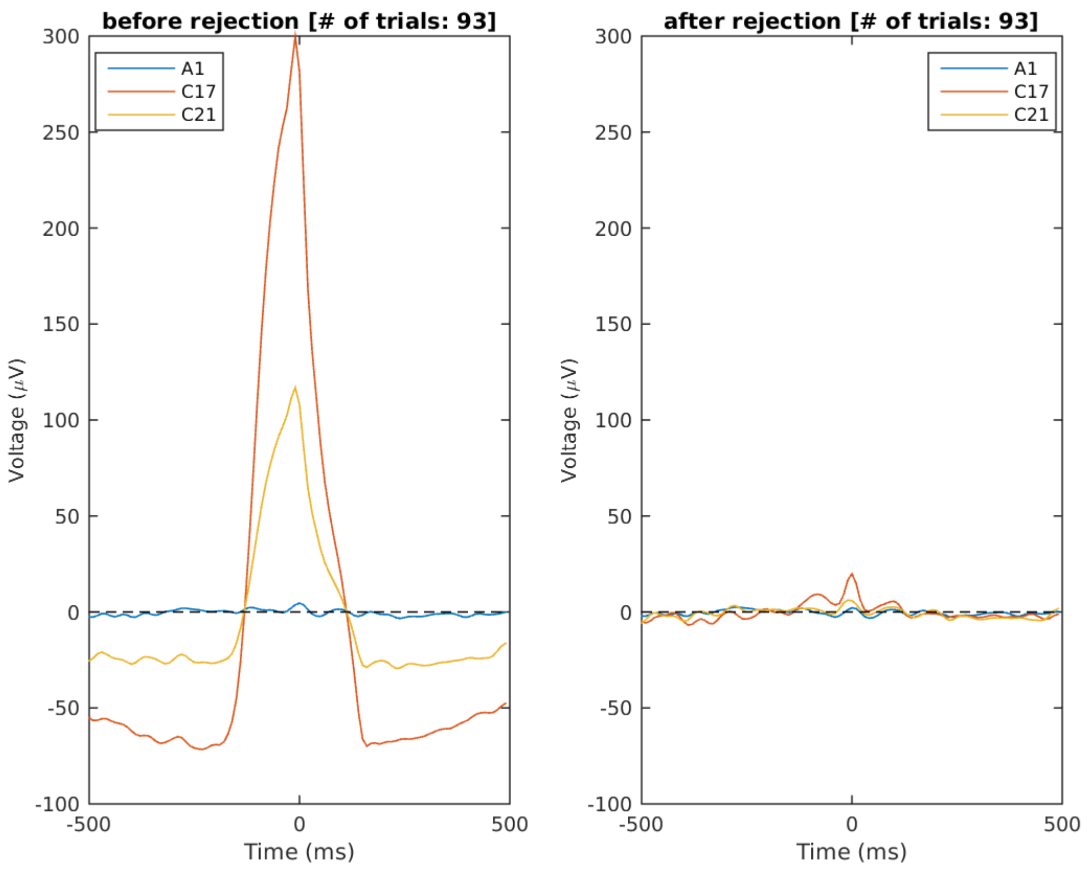


Figure 5. An example of the blink ERP. The left panel shows the blink centered ERP before correction with a clearly visible blink signal. The right panel shows the same plot after correction. The blink is clearly removed but the underlying EEG remains largely unaffected because the correction was done in IC base.

Detection CTAP contains several options to detect artefactual ICs (for details, see function documentation), similarly to other `CTAP_detect_bad_*()` functions. Whichever option is used, a field is created in the EEG struct to store the results. Another field collects all detections, pointing to the results. The logic that is then available to the user is to call one or many detection functions, possibly pooling the results of several approaches to bad data detection, and then pass the aggregate results to the `CTAP_reject_data()` function.

Rejection `CTAP_reject_data()` checks the detect field to determine which data type is due for rejection, unless explicitly instructed otherwise. Based on the data labelled by prior calls to detection functions, `CTAP_reject_data()` will call some internal EEGLAB function such as `pop_select()`, to remove the bad data.

Visualization Upon rejection EEGLAB plotting tools are used to produce plots that characterize the rejected components. An example of such plot is given in Figure 7.

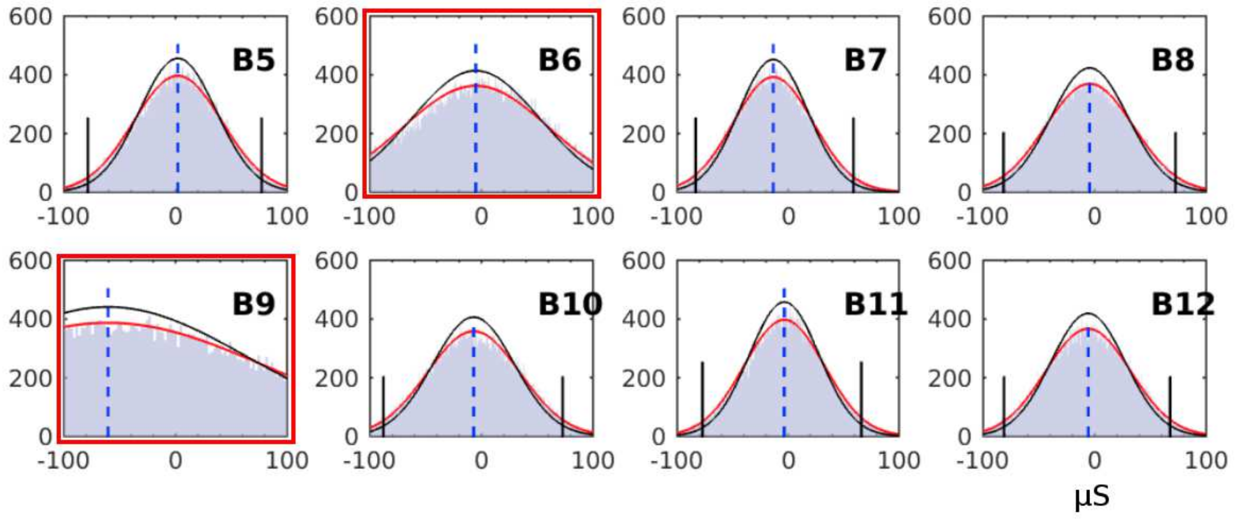


Figure 6. EEG amplitude histograms for eight channels. Fitted normal pdf is shown in red solid line, fitted normal pdf using trimmed sd in black solid line, upper and lower 2.5 % quantiles are vertical black solid lines, and distribution mean is vertical dashed blue line. Bad channels B6, B9 are highlighted red.

DISCUSSION

Philosophy, Benefits and Issues

The overall goal of CTAP is to improve on typical ways of preprocessing high-dimensional EEG data through a structured framework for automation. Automation both reduces the work load of the user and also removes subjectivity from the analysis. CTAP output can also be more easily reproduced because manual processing steps have been minimized. This enables the user to perform multiple comparative analyses for testing the robustness of the results against different preprocessing methods.

CTAP provides many default parameters, and streamlines many features into a handful of wrapper functions. This is in order to facilitate rapid build and testing of analysis pipes. The philosophy is to prevent users becoming 'stuck' in a single approach to the data because they have invested much in building the preprocessing code for it from scratch; or worse, because they have completed a laborious manual processing task and cannot afford to repeat it. However, the user should not usually rely on defaults, because the optimal choice often depends on the data. This is also one reason to have separate files for pipeline and parameters. Separating these by files is convenient for e.g. testing multiple parameter configurations.

As different analysis strategies and methods can vary greatly the best approach was to implement CTAP as a modular system. Each analysis can be constructed from discrete steps which can be implemented as standalone functions. The only requirement is to supress all types of pop-ups or GUI-elements which would prevent the automatic execution of the analysis pipe. It is also up to the user to call the functions in the right order (e.g., not calling averaging before epoching). As CTAP is meant to be extended with custom analysis functions the interface between core CTAP features and external scripts is also well defined in the documentation.

CTAP never overrides the user's configuration options, even when these might break the pipe. For example, `CTAP_reject_data()` contains code to autodetect the data to reject. However the user can set this option explicitly, and can do so without having first called any corresponding

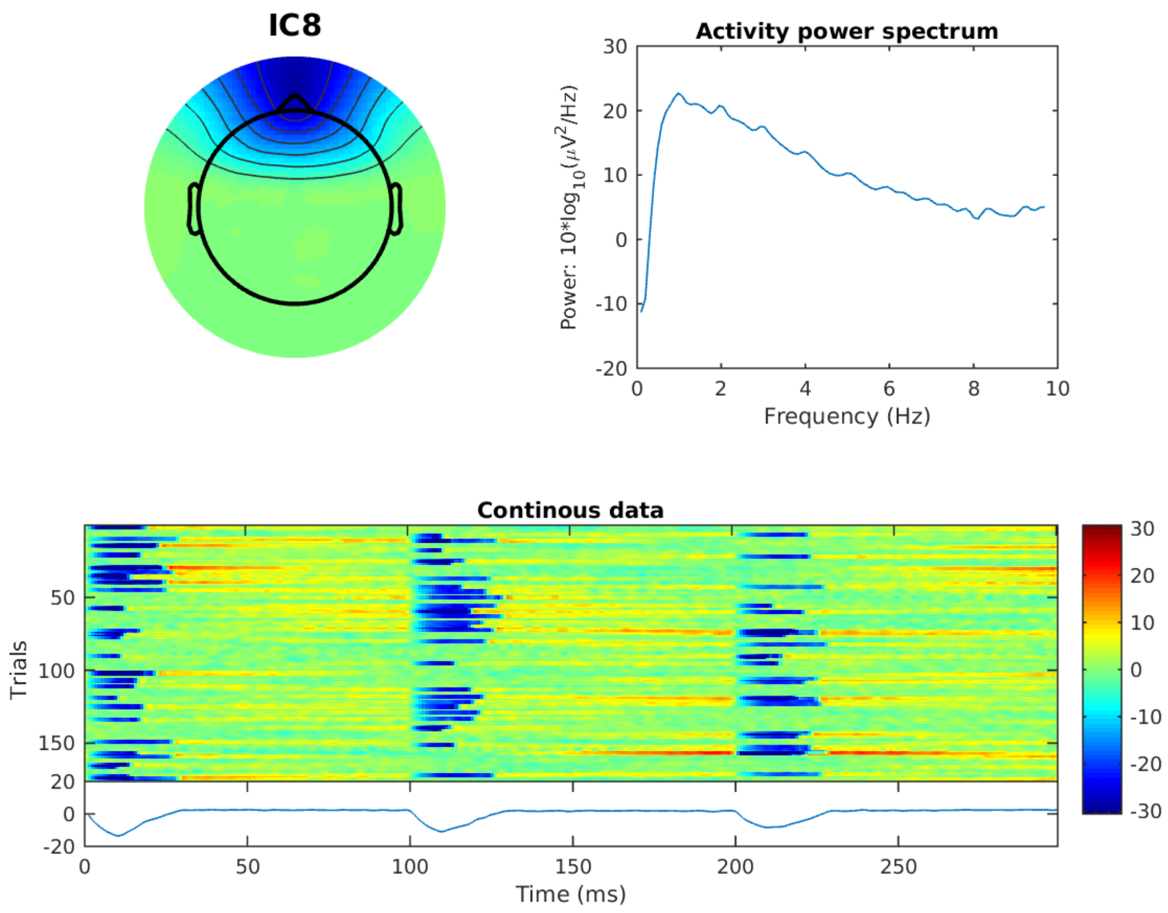


Figure 7. An example of a plot describing a bad ICA component - in this case the algorithm used to find the bad data was the blink detection procedure described above. Shown are component scalp map (top left), component power spectrum (top right), event-related spectral potential (ERSP) plot (bottom).

detection function, which will cause preprocessing on that file to fail. Allowing this failure to happen is the most straightforward approach. Combined with an informative error message the user gets immediate feedback on what is wrong with the pipe.

On the other hand, CTAP does provide several features to handle failure gracefully. As noted, the pipe will not crash if a single file has an unrecoverable error, although that file will not be processed further. This allows a batch to run unsupervised. Then, because no existing outputs are overwritten automatically, you can easily 'mop up' the files that failed without redoing all those that succeeded, if the fault is identified. Because pipes can be divided to step sets, the 'tricky' processes that are prone to failure can be isolated and reduce the overall time spent on crash recovery. CTAP also allows crashed files to be saved at the point of failure, permitting closer analysis of the problematic data.

In contrast to most analysis plugins built on top of EEGLAB, no graphical user interface (GUI) was included in CTAP. While GUIs have their advantages (more intuitive data exploration, easier for novice users, etc) it would have been very difficult and time consuming to add one to a complex a system like CTAP. The GUI also sets limits to configurability and can cause problems for automation if analysis pipes is executed on a hardware with no graphical capabilities. The absence of GUI also makes the development of extensions easier as there are fewer dependencies to worry about.

In contrast to many other physiological data analysis tools, CTAP is designed to meet a very focused goal with a specific approach. This does however create some drawbacks. Compared to scripting ones own pipeline from scratch, there are usage constraints imposed by the heavy use of struct passing interfaces. There is a learning curve for using some non-obvious features and it can be difficult to understand some the more complex underlying processes.

CTAP is also built to enable easy further development by third parties, by using standardised interfaces and structures. This was a feature of original EEGLAB code, but contrasts with many of the EEGLAB-compatible tools released since, where functionality was often built in an *ad hoc* manner. The main requirement for development is to understand the content and purpose of the `EEG.CTAP` field (which is extensively documented in the wiki), and the general logic of CTAP.

Developers can easily extend the toolbox by using (or emulating) the existing `ctapeeg_*()` functions, especially the `ctapeeg_detect_*()` functions, which are simply interfaces to external tools for detecting artefacts. Existing `CTAP_*()` functions can be relatively more complex to understand, but the existing template provides a guideline for development with the correct interface.

Future work

CTAP is far from finalized, and development will continue after the initial release of the software.

Many publications have described methods for processing EEG for different purposes, such as removing artefacts, estimating signal sources, analysing event-related potentials (ERPs), and so on. However despite the wealth of methodological work done, there is a lack of benchmarking, or tools for comparison of such methods. The outcome is that the most reliable way to assess each method is to learn how to it works, apply it, and test the outcome on one's own data: this is a highly time-consuming process which is not competitive with simply performing the bulk of preprocessing in a manual way, as seems to remain the 'gold standard'. The effect of each method on the data is also not commonly characterised, such that methods to correct artefacts can often introduce noise to the data, especially where there was no artefact (false positives).

In section Approach we described the aim to test and compare automated methods for preprocessing. This is still work in progress, as we are building an extension for CTAP that improves testing and comparison of preprocessing methods by repeated analyses on synthetic data. This extension, tentatively titled Handler for sYnthetic Data and Repeated Analyses (HYDRA), will use synthetic data to generate ground truth controlled tests of preprocessing methods. It will have capability to generate new synthetic data matching the parameters of the lab's own data. This allows experimenters to select the best methods for their purpose, or developers to flexibly test and benchmark their novel methods.

Another desirable but not vital future task is to expand the QC output, to include functionality such as statistical testing of detected bad data, for the experimenter to make a more informed decision. Although statistical testing is already implied in many methods of bad data detection, it is not visible to users.

We will also provide an automated tool to compare output from two (or more) peeks, to help visualize both changes in baseline level as well as changes in local waveforms.

CONCLUSIONS

The ultimate aims of CTAP are: a) to facilitate processing of large quantities of EEG data; b) to improve reliability and objectivity of such processing; c) to support development of 'smart' algorithms to tune the thresholds of statistical selection methods (for bad channels, epochs, segments or components) to provide results which are robust enough to minimise manual intervention. With this contribution, we have addressed aim a), partly also b), and laid the groundwork to continue developing solutions for c). CTAP will thereby help to minimise human effort, subjectivity and error; and facilitate easy, reliable batch processing for experts and novices alike.

ACKNOWLEDGEMENTS

The authors would like to thank Andreas Henelius, Miika Toivanen, and Lauri Ahonen for fruitful discussions on the CTAP toolbox and this paper.

REFERENCES

Agapov, S. N., Bulanov, V. A., Zakharov, A. V., and Sergeeva, M. S. (2016). Review of analytical instruments for EEG analysis. *arXiv*, 1605.01381.

Baillet, S., Friston, K., and Oostenveld, R. (2011). Academic Software Applications for Electromagnetic Brain Mapping Using MEG and EEG. *Computational Intelligence and Neuroscience*, 2011:4.

Baillet, S., Tadel, F., Leahy, R. M., Mosher, J. C., Delorme, A., Makeig, S., Oostenveld, R., Hämäläinen, M., Dalal, S. S., Zumer, J., Clerc, M., Wolters, C. H., Kiebel, S., and Jensen, O. (2010). *Academic Software Toolboxes for the Analysis of MEG Data*, chapter Academic S, pages 101–104. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bao, F. S., Liu, X., and Zhang, C. (2011). PyEEG: an open source Python module for EEG/MEG feature extraction. *Computational Intelligence and Neuroscience*, 2011(Article ID 406391):7.

Barua, S. and Begum, S. (2014). A Review on Machine Learning Algorithms in Handling EEG Artifacts. In Jensfelt, P., editor, *The Swedish AI Society (SAIS) Workshop SAIS, 14, 22-23 May 2014, Stockholm, Sweden*.

Bigdely-Shamlo, N., Mullen, T., Kothe, C., Su, K.-M., and Robbins, K. A. (2015). The PREP pipeline: standardized preprocessing for large-scale EEG analysis. *Frontiers in Neuroinformatics*, 9:16.

Chaumon, M., Bishop, D. V. M., and Busch, N. A. (2015). A practical guide to the selection of independent components of the electroencephalogram for artifact correction. *Journal of neuroscience methods*, 250:47–63.

Cowley, B., Filetti, M., Lukander, K., Torniainen, J., Henelius, A., Ahonen, L., Barral, O., Kosunen, I., Valtonen, T., Huotilainen, M., Ravaja, N., and Jaccuci, G. (2016). The Psychophysiology Primer: a guide to methods and a broad review with a focus on human computer interaction. *Foundations and Trends in HCI*, forthcoming.

Dandekar, S., Ales, J., Carney, T., and Klein, S. A. (2007). Methods for quantifying intra- and inter-subject variability of evoked potential data applied to the multifocal visual evoked potential. *Journal of neuroscience methods*, 165(2):270–86.

Delorme, A. and Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of neuroscience methods*, 134(1):9–21.

Delorme, A., Mullen, T., Kothe, C., Akalin Acar, Z., Bigdely-Shamlo, N., Vankov, A., and Makeig, S. (2011). EEGLAB, SIFT, NFT, BCILAB, and ERICA: new tools for advanced EEG processing. *Computational intelligence and neuroscience*, 2011:130714.

Delorme, A., Palmer, J., Onton, J., Oostenveld, R., and Makeig, S. (2012). Independent EEG sources are dipolar. *PloS one*, 7(2):e30135.

Delorme, A., Sejnowski, T., and Makeig, S. (2007). Enhanced detection of artifacts in EEG data using higher-order statistics and independent component analysis. *NeuroImage*, 34(4):1443–9.

Frank, R. M. and Frishkoff, G. A. (2007). Automated protocol for evaluation of electromagnetic component separation (APECS): Application of a framework for evaluating statistical methods of blink extraction from multichannel EEG. *Clinical Neurophysiology*, 118(1):80–97.

Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., and Hämäläinen, M. (2013). MEG and EEG data analysis with MNE-Python. *Frontiers in neuroscience*, 7:267.

Keil, A., Debener, S., Gratton, G., Junghöfer, M., Kappenman, E. S., Luck, S. J., Luu, P., Miller, G. A., and Yee, C. M. (2014). Committee report: publication guidelines and recommendations for studies using electroencephalography and magnetoencephalography. *Psychophysiology*, 51(1):1–21.

Lindsen, J. P. and Bhattacharya, J. (2010). Correction of blink artifacts using independent component analysis and empirical mode decomposition. *Psychophysiology*, 47(5):955–960.

Mognon, A., Jovicich, J., Bruzzone, L., and Buiatti, M. (2011). ADJUST: An automatic EEG artifact detector based on the joint use of spatial and temporal features. *Psychophysiology*, 48(2):229–40.

Nolan, H., Whelan, R., and Reilly, R. B. (2010). FASTER: Fully Automated Statistical Thresholding for EEG artifact Rejection. *Journal of neuroscience methods*, 192(1):152–62.

Oostenveld, R., Fries, P., Maris, E., and Schoffelen, J.-M. (2011). FieldTrip: Open source software for advanced analysis of MEG, EEG, and invasive electrophysiological data. *Computational intelligence and neuroscience*, 2011:156869.

Peyk, P., De Cesarei, A., and Junghöfer, M. (2011). ElectroMagnetoEncephalography software: overview and integration with other EEG/MEG toolboxes. *Computational intelligence and neuroscience*, 2011:861705.

Tadel, F., Baillet, S., Mosher, J. C., Pantazis, D., and Leahy, R. M. (2011). Brainstorm: a user-friendly application for MEG/EEG analysis. *Computational intelligence and neuroscience*, 2011:879716.

Toivanen, M., Pettersson, K., and Lukander, K. (2015). A probabilistic real-time algorithm for detecting blinks , saccades , and fixations from EOG data. *Journal of Eye Movement Research*, 8(2):1–14.

Tremblay, A. and Newman, A. J. (2015). Modeling nonlinear relationships in ERP data using mixed-effects regression with R examples. *Psychophysiology*, 52(1):124–39.