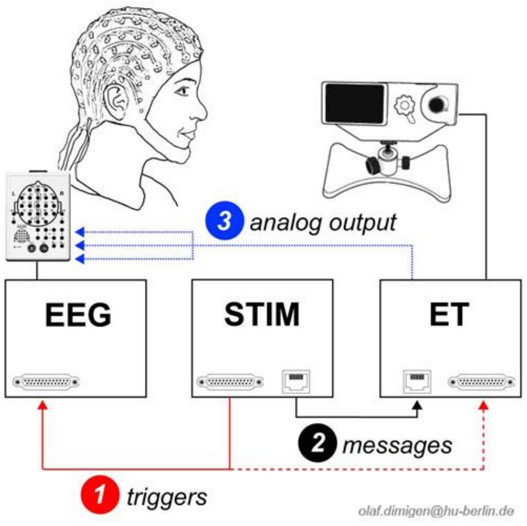
Toolbox for Simultaneous Eye tracking & EEG: Tutorial

olaf.dimigen@hu-berlin.de, version: April 22, 2013
http://www2.hu-berlin.de/eyetracking-eeg

Contents

Basics: Connecting eye tracker & EEG

Meaningful analyses of simultaneously recorded eye tracking (ET) and EEG data requires that both data streams are synchronized with millisecond precision. There are at least three ways to synchronize both systems (Dimigen et al., 2011). The plugin is compatible with all three, although we recommend the first one:



1: Shared triggers

Common trigger pulses ("triggers") are sent frequently from the stimulation computer to both ET computer and EEG recording computer. This is achieved via a Y-shaped cable that is attached to the parallel port of the stimulation computer and splits up the pulse so it is looped through to EEG and ET. We recommend to send triggers with a sufficient duration (e.g. at least 5 ms at 500 Hz sampling rate) to avoid the loss of some of the triggers (seen with both SR Research and SMI hardware). The advantage of this method is that the same physical signal is used for synchronization (although this does not guarantee that the trigger is inserted into the ET and EEG data streams without delays). The disadvantage is the need for an extra cable.

2: Messages + triggers

Messages are short text strings that can be inserted into the eye tracking data. While triggers are still sent to the EEG, messages are used as the corresponding events for the ET. Here, the ET computer is given a command to insert an ASCII text message (containing a keyword and the value of the corresponding EEG trigger) into the eye tracking data. In the stimulation software, the commands to send a trigger (to the EEG) and a message (to the ET) are given in immediate succession (see example code below).

3: Analogue output

A copy of the eye track is fed directly into the EEG. A digital-to-analogue converter card in the ET outputs (some of) the data as an analogue signal. With SMI, this signal can be fed directly into the EEG headbox. This requires a custom cable and resistors to scale the output voltage of the D/A converter to the EEG amplifier's recording range. While this method affords easy synchronization, there are disadvantages: First, voltages need to be rescaled to pixels for analysis. Second, the ET signal may exceed the amplifier's recording range and electrical interference with the EEG is possible. Third, additional information from the ET (messages, eye movements detected online) is not available. Finally, fewer channels remain to record the EEG (recording binocular gaze position and pupil diameter occupies six channels).

We recommended to test the timing accuracy of each method. For example this could mean to send triggers (and/or messages) to the eye tracker at known time intervals and to check for jitter in the recorded inter-trigger (inter-message) intervals.

Basics: Synchronization signals

Send triggers and/or messages to align the recordings

The plugin requires that there are at least two shared events present in the ET and EEG: One near the beginning and one near the end of the recording. These events will be called *start-event* and *end-event* in the following. Eye tracking data in between the start-event and end-event will be linearly interpolated to match the sampling frequency of the EEG. We recommend to use a unique event value (e.g. "100") to mark the start-event and another unique event-value (e.g., "200") for the end-event. The remaining shared events (triggers or messages) sent during the experiment (between start-event and end-event) are used to evaluate the quality of synchronization. Synchronization is possible even if some intermediate events were lost during transmission.

Original sampling rates of EEG and ET do not need to be the same. The ET will be resampled to the sampling frequency of the EEG. For example, if the EEG was sampled at 500 Hz, and eye movements were recorded at 1000 Hz, the plugin will downsample the eye track to 500 Hz. Since ET data outside of the synchronization range (before start-event, after end-event) cannot be interpolated, it is replaced by zeros. Please note that EEG and ET recordings *should not be paused* during experiments.

If synchronization method 2 (messages plus trigger) is used, synchronization messages sent to the eye tracker need to have a specified format. This format consists of an arbitrary user-defined keyword (e.g., "MYKEYWORD") followed by an integer value ("MYKEYWORD 100"). The integer value needs to be the same as that of the corresponding trigger pulse sent to the EEG (usually an 8-bit number between 1 and 255). An example is given in the code below. The EYELAB parser (Step 2: Preprocess eye track and store as MATLAB) will recognize messages with the keyword and treat them as synchronization events. A keyword-synchronization messages should be sent together with every trigger sent to the EEG, so intermediate events in-between start-event and end-event can be used to assess synchronization quality. Additional messages (that do not contain the keyword) may be sent to code other aspects of the experimental design. They are ignored by the plugin.

The following code is an example for an experimental runtime file containing the necessary synchronization signals. The example is for the software Presentation™, but similar commands exist in other software (e.g., Psychtoolbox, EPrime™):

```
# increase default duration of trigger pulses (to 6 ms)

pulse_width = 6;

# define parallel port

output_port myparallelport = output_port_manager.get_port(1);

# create eyetracker object (hex code is vendor-specific)

eye_tracker myET = new eye_tracker("{FF2F86B9-6C75-47B7-944F-2B6DECA92F48}");

myET.set_recording(true);

# beginning of experiment: send unique start trigger (or message) to ET & EEG

myparallelport.send_code(100);

myET.send_string("MYKEYWORD " + string(100)); # only needed for method 2

# [ …code of actual experiment… ]

# end of experiment: send unique end trigger (or message) to ET & EEG

myparallelport.send_code(200);

myET.send_string("MYKEYWORD " + string(200)); # only needed for method 2

myET.set_recording(false);
```

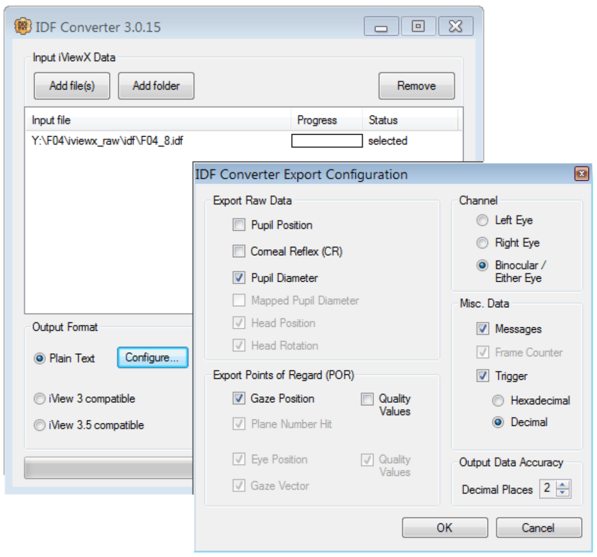
More information about synchronization is found in Step 3: Load and synchronize eye track.

Step 1: Convert eye tracking data to text

Convert eye tracking raw data from binary to text

For use with the plugin, the raw ET data (in binary format, *.edf or *.idf) needs to be converted to plain text (ASCII). Please use the following settings in the converter software provided by the manufacturer:

Sensomotoric Instruments
Conversion settings with "IDF converter":



Output Format: [✓ Plain Text] iView 3 compatible, iView 3.5 compatible

Export Raw Data: Pupil Position [ ], Corneal Reflex (CR) [ ], Pupil Diameter [✓], Mapped Pupil Diameter [ ], Head Position [✓], Head Rotation [✓]

Export Points of Regard (POR): Gaze Position [✓], Quality Values [ ], Plane Number Hit [✓], Eye Position [✓], Quality Values [✓], Gaze Vector [✓]

Channel: Left Eye, Right Eye, [✓ Binocular / Either Eye]

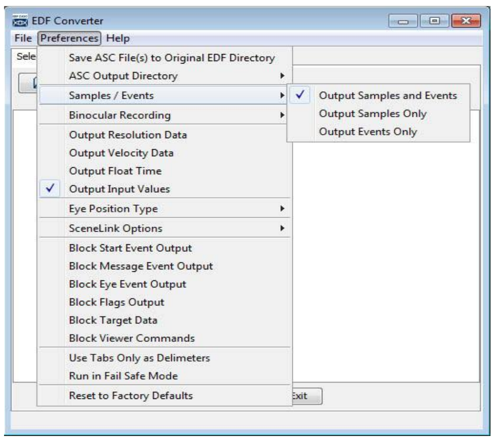
Misc. Data: Messages [✓], Frame Counter [✓], Trigger [✓]; Hexadecimal, [✓ Decimal]

Output Data Accuracy: Decimal Places 2

Make sure to export messages & trigger values:

- *Configure* > *Misc. data* > *Messages*
- *Configure* > *Misc. data* > *Trigger (Decimal)* (if sent)

SR Research
Conversion settings with "EDF converter":



Samples / Events: [✓ Output Samples and Events]

Output Input Values [✓]

Make sure to export messages & trigger values:

- *Preferences* > *Samples / Events* > *Output Samples and Events*
- *Preferences* > *Output Input Values*

Command line version of EDF converter

When using the command line tool *edf2asc*, write numerical data and messages into a single text file. This is achieved with the following command:

```
> edf2asc myeyetrackerdata.edf -input # flag -input writes triggers

# do NOT use flags -e -s -ns -ne -nse
```

If synchronization method 1 is used, make sure to use the flag "-input". This writes triggers to the output file. Depending on your settings in the Eyelink recording software, parallel port inputs are represented in two ways in the text-converted data:

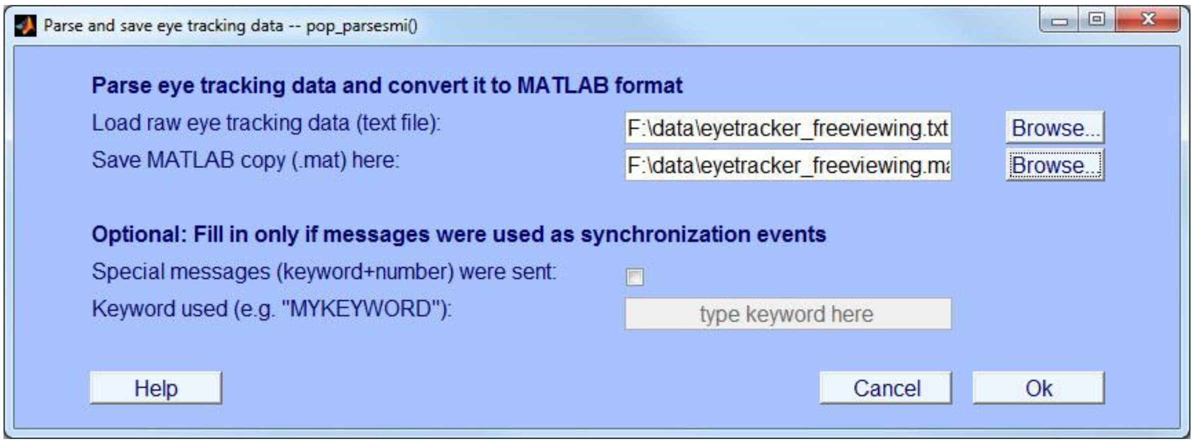
- extra messages starting with the word "INPUT" and containing the trigger value
- an extra data column, which contains either zeros (default) or the trigger value (whenever active)

  The parser of the plugin recognizes both formats.

Step 2: Preprocess eye track & save as a MATLAB structure

Convert eye track into MATLAB format

This function reads and preprocesses the text-converted eye tracking data and stores it as a structured array in MATLAB format (.mat). Click *Eyetracker > Parse eyetracker raw data* and select your ET model. The following dialogue appears:


Parse and save eye tracking data -- pop_parsesmi()

Parse eye tracking data and convert it to MATLAB format

Load raw eye tracking data (text file): F:\data\eyetracker_freeviewing.txt

Save MATLAB copy (.mat) here: F:\data\eyetracker_freeviewing.ma

Optional: Fill in only if messages were used as synchronization events

Special messages (keyword+number) were sent: ☐

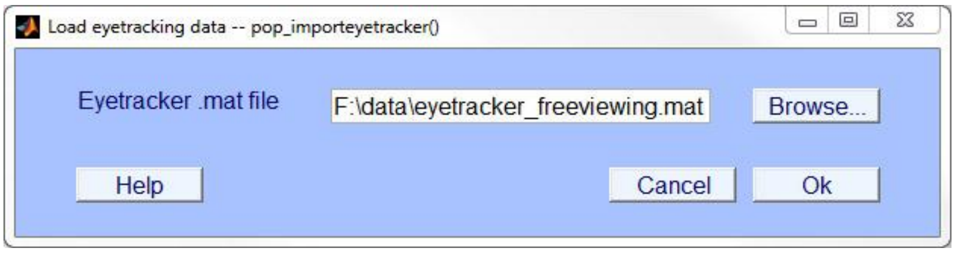
Keyword used (e.g. "MYKEYWORD"): type keyword here

- Select the file path of the text-converted ET data (.txt, .asc) generated in the previous step.
- Choose a file path to save the preprocesses data in MATLAB (.mat) format.
- If synchronization method 2 is used, select the checkbox and specify the keyword contained in your synch. messages. (e.g. "MYKEYWORD")
- Press **OK**. Parsing a large file may take a moment.

Step 3: Load and synchronize eye track

Load MATLAB copy of raw eye track

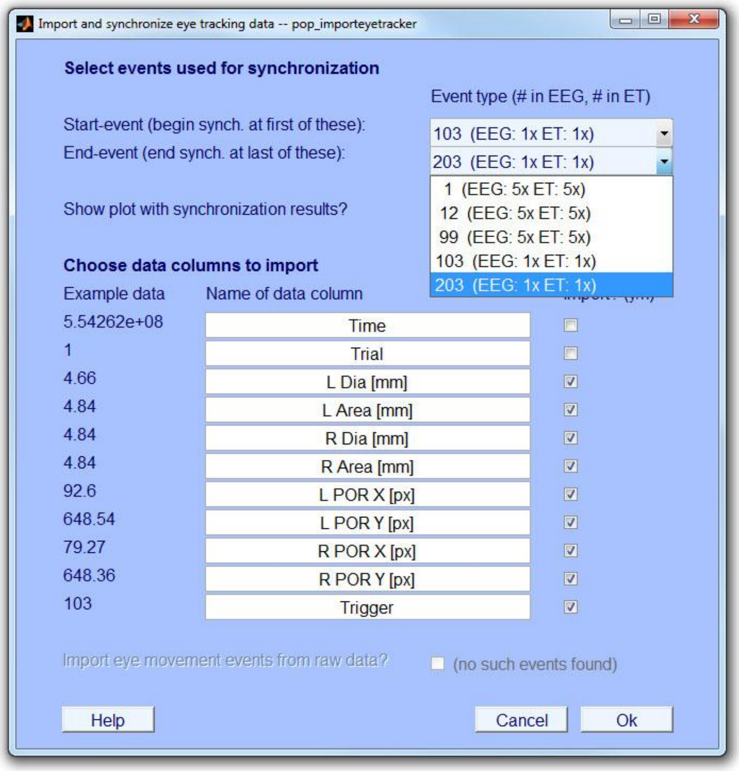
Use the normal EEGLAB functions to load a continuous EEG raw data file. Do not reject (cut out) bad EEG intervals from this file before you have aligned the EEG with the eye track. To add the eye tracking data, select *Import & synchronize eye track*. This pops up a first dialogue window:



Load eyetracking data -- pop_importeyetracker()

Eyetracker .mat file: F:\data\eyetracker_freeviewing.mat

Select the corresponding .mat file created in the previous step. Press **OK** to continue.
Shortly after, a second window pops up:



Import and synchronize eye tracking data -- pop_importeyetracker

Select events used for synchronization

Event type (# in EEG, # in ET)

Start-event (begin synch. at first of these): 103 (EEG: 1x ET: 1x)
End-event (end synch. at last of these): 203 (EEG: 1x ET: 1x)

1 (EEG: 5x ET: 5x)
12 (EEG: 5x ET: 5x)
99 (EEG: 5x ET: 5x)
103 (EEG: 1x ET: 1x)
203 (EEG: 1x ET: 1x)

Show plot with synchronization results?

Choose data columns to import

Example data	Name of data column	Import? (y/n)
5.54262e+08	Time	
1	Trial	
4.66	L Dia [mm]	✓
4.84	L Area [mm]	✓
4.84	R Dia [mm]	✓
4.84	R Area [mm]	✓
92.6	L POR X [px]	✓
648.54	L POR Y [px]	✓
79.27	R POR X [px]	✓
648.36	R POR Y [px]	✓
103	Trigger	✓

Import eye movement events from raw data? (no such events found)

Settings for synchronization and import

Select events used for synchronization
Select the start-event and end-event for synchronization. The dropdown menus contain a list of all event types found in both the EEG (in *EEG.event*) and in the eye tracking data. The dropdown menu shows the type of the event (e.g., "100") and the number of times it was found in the EEG and eye track. Events that were found in only one of the recordings (e.g. due to trigger loss) are not shown.

Note: When designing the experiment, we recommend using a unique event value (e.g. "100") as the start event and another unique event value (e.g., "200") as the end event. If there are *several instances* of the event selected as the start-event, the plugin will use the *first* of them for synchronization. Likewise, if the event selected as the end-event is found multiple times, the plugin will use the *last* one for synchronization.

Show plot with synchronization results?
Checking this option will pop up a figure showing synchronization quality (recommended).

Choose data columns to import and name them
A list of the data columns found in the ET data. Use the checkboxes to select the data channels you would like to import. You can also change the channel names. Avoid special characters incompatible with EEGLAB.

Import eye movement information found in raw data
This option is only active for Eyelink data, if the online event detection was switched on during recording. If checked, saccades, fixation, and blinks detected online by the Eyelink algorithm will be imported and added as new events to *EEG.event*. This option is greyed out for SMI data or if no eye movement events were found in the data.

Click **OK** to start synchronization.

Notes on synchronization: Synchronization is performed in two steps. First, ET data between the start- and end-event is linearly interpolated to match the number of EEG sampling points recorded during the same time interval. Afterwards, "shared" events are identified. This is important, because it is not uncommon that some events are lost in one of the recordings. An event is considered to be the same event if it is of the same type (e.g. "123") and less than 5 samples apart in both recordings after interpolation. In an optional step two, a linear function is fit to the shared event latencies in order to refine the estimation of the latency of the start-event and end-event in the ET recording. This can improve synchronization, especially if start-event or end-event were transmitted with a large delay or jitter.

Check synchronization quality

If the checkbox *Show plot with synchronization results* was selected (recommended), a figure with the synchronization quality appears:

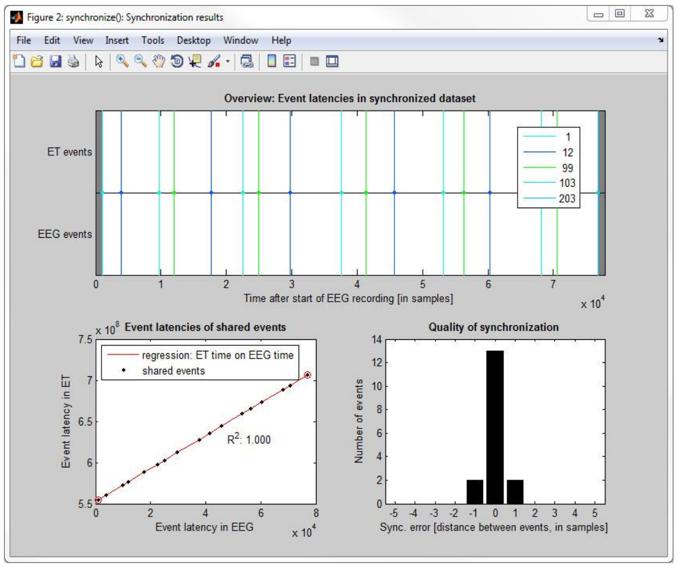


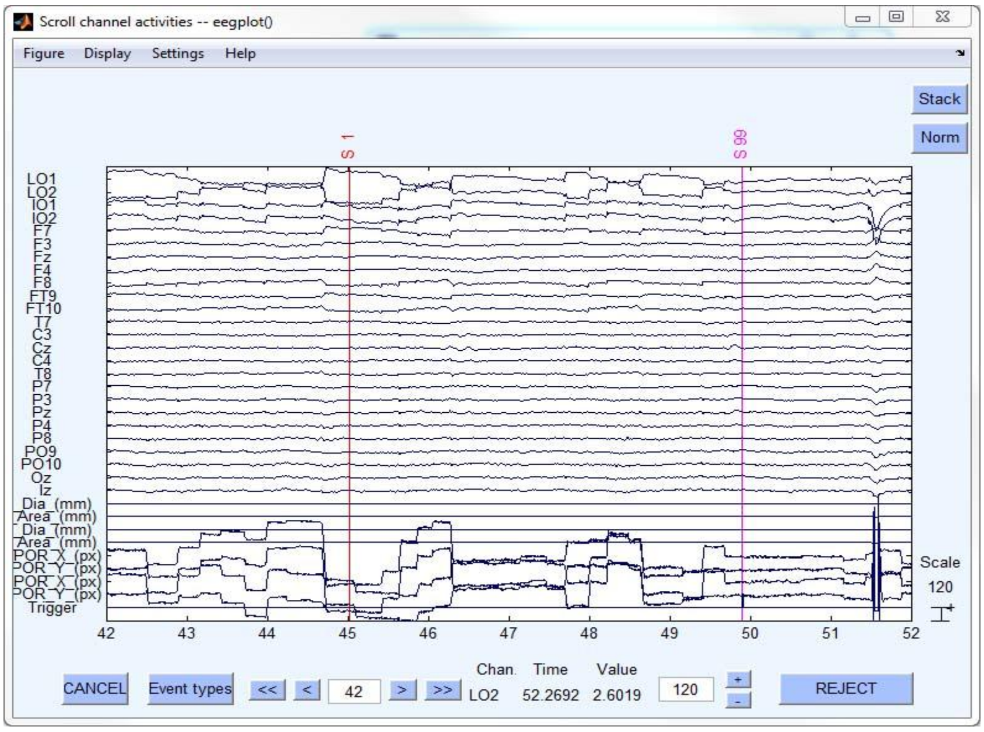
Figure 2: synchronize(): Synchronization results
File Edit View Insert Tools Desktop Window Help
Overview: Event latencies in synchronized dataset
ET events
EEG events
1
12
99
103
203
Time after start of EEG recording [in samples]
x 10^4
Event latencies of shared events
x 10^5
regression: ET time on EEG time
shared events
R^2: 1.000
Event latency in ET
Event latency in EEG
Quality of synchronization
Number of events
Sync. error [distance between events, in samples]

In the upper plot, each vertical line represents one event in the EEG (upper part) and ET (lower part). After synchronization, vertical lines should be neatly aligned. Intervals prior to the start-event or after the end-event are plotted in dark grey.

The lower left plot displays the latencies of "shared" events in the ET and EEG. It also displays the linear regression line for the regression of the latencies of ET events on the latencies of EEG events as well as the coefficient of determination (R^2). This fit is used to improve estimation of the latency of the start-event and end-event. The lower right plot shows a histogram of the synchronization error (in samples), that is, the latency difference between corresponding events in the EEG and the ET after synchronization. An error of zero means that the two triggers fall onto the same sample (i.e., are perfectly aligned) after linear interpolation of the eye tracking data. The mode of the distribution should therefore be at zero. More information about synchronization quality is provided as an output in the MATLAB command window.

Plot the synchronized data

Now take a look at the merged data with the EEGLAB function *Plot > Channel data (scroll)*.



Eye tracking data usually has high offset values (e.g., the screen center is at 512 pixels). To properly display the data, activate *Display > Remove DC offset* in the EEGLAB plotting window. If your eye tracking channels appear "flat" and only show zeros, this is because data outside the two synchronization events is not interpolated. Scroll forward to the first synchronization event (e.g. "100").

Step 4: Reject data based on eye tracker

Remove intervals with blinks or out-of-range data in the eye tracker

The menu *Eyetracker > Reject data based on eye track* allows to reject continuous intervals or epochs of data during which the participant's gaze was not within specified limits. This function serves two purposes:

1. Pruning the eye track for bad data

It is crucial that bad eye tracking data is removed prior to saccade & fixation detection. There are two sources of bad ET data: The first are eye blinks. Most eye trackers record zeros while the eye is closed. Moreover, strong outliers are often seen at the beginning and end of blinks, while the pupil is partially occluded by the eyelid. The second source of bad data are intervals during which the system lost tracking of one or both eyes. If kept in the data, blinks and other bad data are detected as abrupt changes in eye velocity - and therefore as saccades. One way to exclude bad data is to remove intervals or epochs of data during which the ET recorded a gaze position "outside the monitor" - that is, pixel values smaller than 1 pixel or higher than the screen resolution (e.g., > 1024 horizontal pixels).

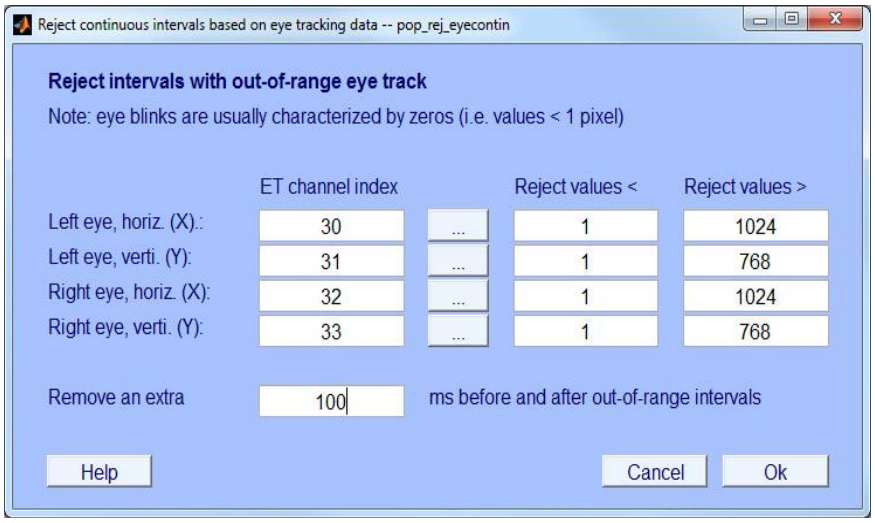
2. Controlling fixation

The menu is also useful to reject intervals or epochs during which the participant was not fixating some specified area of the screen. Let's say you want to exclude all trials in which gaze position deviated more than 50 pixels from a central fixation cross (e.g., shown at pixel values x=512 and y=384). To exclude large gaze shifts, you could enter limits of 462 and 562 pixels for the horizontal x-component of both eyes. This removes intervals or epochs in which the subject was not looking at the screen center.

This function comes in two variants: one for continuous and one for epoched data:

- *Reject bad continuous data:* Continuous data intervals with out-of-range eye tracking data are discarded. Boundary events are added to *EEG.event* at each resulting data break.
- *Reject bad epochs:* Whole epochs are discarded that contain any out-of-range eye tracking data.

To reject bad intervals in a continuous dataset, click *Eyetracker > Reject data based on eye track > Reject bad cont. data*.
The following menu will appear:


Reject continuous intervals based on eye tracking data -- pop_rej_eyecontin

Reject intervals with out-of-range eye track

Note: eye blinks are usually characterized by zeros (i.e. values < 1 pixel)

	ET channel index		Reject values <	Reject values >
Left eye, horiz. (X).:	30	...	1	1024
Left eye, verti. (Y):	31	...	1	768
Right eye, horiz. (X):	32	...	1	1024
Right eye, verti. (Y):	33	...	1	768

Remove an extra 100 ms before and after out-of-range intervals

Settings to remove out-of-range data

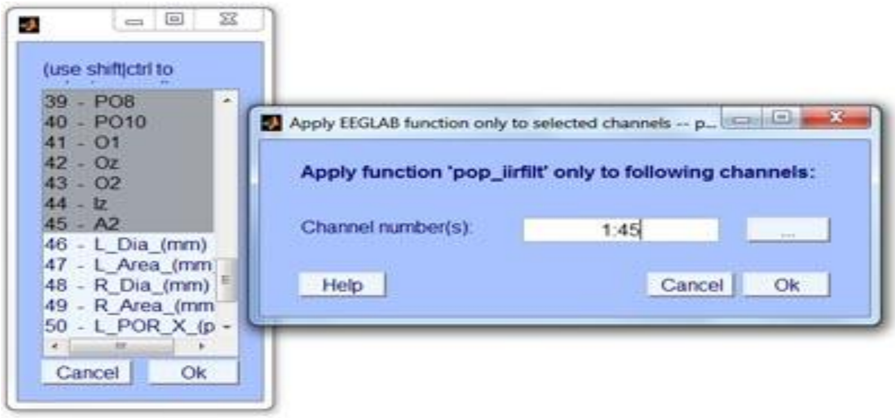
Enter the minimum and maximum allowed pixel values for each component of the eye track (left and right eye, x and y). These values could be the dimensions of your monitor (in pixels). If the recording was only monocular, leave the fields for the other eye empty. You do not need to fill out all fields.
As explained above, corrupted data which may not exceed the rejection thresholds is often found at the beginning and end of blinks. For continuous data, you have the option to reject some extra data samples before and after each interval with out-of-range values. For example, if you enter 100 in the input field, then data is excluded from 100 ms before until 100 ms after the threshold for bad data is exceeded. This option is not available for epoched data. For epoched data, the whole epoch is discarded if it contains a single out-of-range value.

Click **OK** to discard bad data.

Step 5: Apply EEGLAB function to selected channels

Apply existing EEGLAB functions only to EEG or eye track

This menu allows the application of existing EEGLAB functions to a subset of data channels.



(use shift|ctrl to

39 - PO8
40 - PO10
41 - O1
42 - Oz
43 - O2
44 - Iz
45 - A2
46 - L_Dia_(mm)
47 - L_Area_(mm
48 - R_Dia_(mm)
49 - R_Area_(mm
50 - L_POR_X_(p

Apply EEGLAB function only to selected channels -- p...

Apply function 'pop_iirfilt' only to following channels:

Channel number(s): 1:45

While a few EEGLAB functions can be applied to a subset of the channels (e.g., *Tools > Run ICA*), others are applied to all channels per default (e.g., *Tools > Remove baseline*, *Filter the data*). In many cases, it is not sensible to use the same settings for EEG and ET data. For example, pupil diameter data is usually filtered differently from EEG data. Similarly, when epochs are extracted via the GUI (*Tools > Extract epochs*), EEGLAB automatically calls pop_rmbase() and subtracts a baseline from all channels. However, it is often important to retain absolute values in the ET data (absolute gaze position on the screen, absolute pupil diameter).

The menu *Eyetracker > Apply function to selected channels* simply calls existing EEGLAB functions in a way that is compatible with ET data channels. No changes were made to the underlying EEGLAB functions. The following functions can be executed:

- *Filter the data*: Only filter some of the channels (e.g. only EEG/only ET channels)
- *Extract epochs (without baseline correction)*: Extract epochs without calling a baseline correction for all channels. Do not forget to select *Remove baseline (selected channels)* afterwards to baseline-correct the appropriate channels (i.e., the EEG)!
- *Remove baseline (selected channels)*: Remove baseline from selected channels only (e.g., only EEG).
- *Channel ERPs - with scalp maps*: Plot ERP & scalp maps. Use only the EEG channels.

Step 6: Detect saccades & fixations

Velocity-based saccade detection

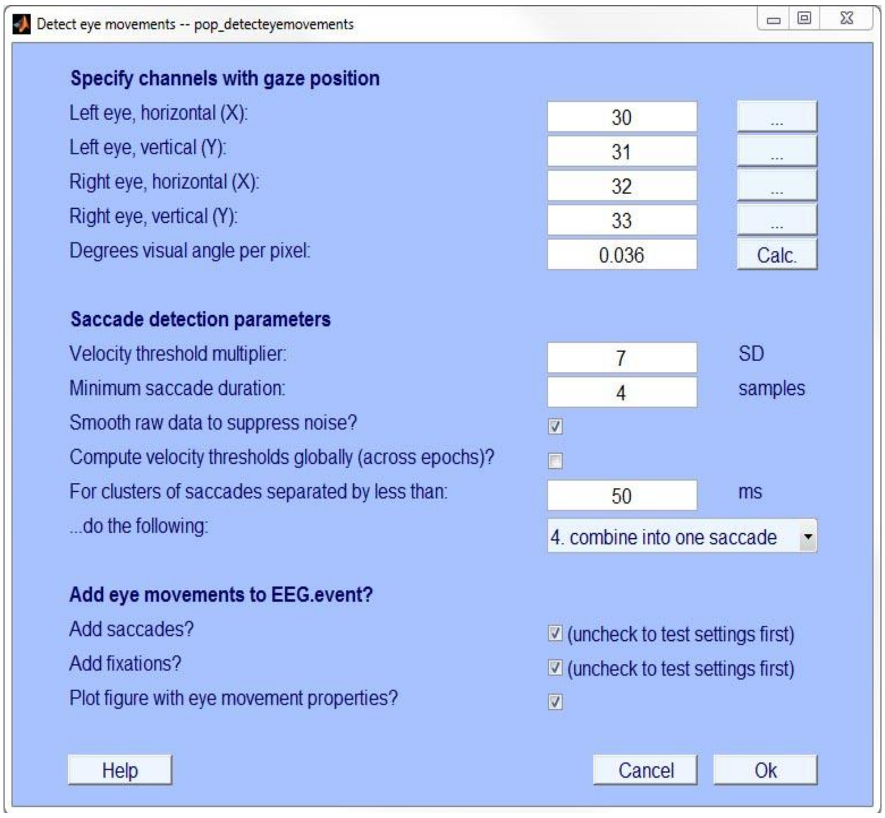
The plugin implements an updated version of the algorithm for (micro)saccade detection proposed by Ralf Engbert and colleagues (see Engbert & Kliegl, 2003; Engbert & Mergenthaler 2006, with permission).

Saccades are defined as monocular or binocular outliers in two-dimensional velocity space. The velocity thresholds for saccades are determined relative to the noise level of the data and separately for the horizontal and the vertical movement component of each eye. Fixations are defined as the intervals in-between saccades during which the eyes are relatively static. For compatibility with the current plugin, a few modifications were made to the original MATLAB functions. These changes concern user feedback as well as the option to apply the same velocity thresholds to all data epochs. Furthermore, the plugin adds an optional processing step to treat temporal clusters of saccades (function *mergesacc()*).

Detect eye movement onsets and add them as new events to *EEG.event*

Saccades and fixations can be detected in continuous data or epoched data (e.g., epochs locked to a stimulus onset). Which option makes more sense depends on the specifics of your experiment. Important: Make sure that intervals with bad data are rejected before detecting eye movements.

Click *Detect saccades & fixations*. This will pop a window where you can set the parameters for eye movement detection:



Specify channels with gaze position

Left eye, horizontal (X):	30	
Left eye, vertical (Y):	31	
Right eye, horizontal (X):	32	
Right eye, vertical (Y):	33	
Degrees visual angle per pixel:	0.036	

Saccade detection parameters

Velocity threshold multiplier:	7	SD
Minimum saccade duration:	4	samples
Smooth raw data to suppress noise?	✓	
Compute velocity thresholds globally (across epochs)?	☐	
For clusters of saccades separated by less than:	50	ms
...do the following:	4. combine into one saccade	

Add eye movements to EEG.event?

Add saccades?	✓ (uncheck to test settings first)
Add fixations?	✓ (uncheck to test settings first)
Plot figure with eye movement properties?	✓

Settings for saccade & fixation detection

Specify channels with gaze position

Left/right eye, horizontal and vertical channel:
Enter the channels containing horizontal and vertical gaze position data. If your recording was only monocular, leave both fields for the other eye empty. Saccade and fixation detection will then be performed in a monocular variant. In this case, the binocular overlap criterion (saccades need to overlap temporally in both eyes) is not applied.

Degrees visual angle per pixel
If this value is set (recommended), saccade amplitudes are reported and stored in degrees of visual angle. Otherwise, they are stored and reported in the original metric (most likely screen pixels). Click the button *[Calc.]* to calculate the visual angle per screen pixel for your laboratory setup.

Saccade detection parameters

Please read Engbert & Mergenthaler (2006) for details about the following settings.

Velocity threshold multiplier
Threshold multiplier (λ) for saccade detection. The velocity threshold for each component (x,y) of the eye track is set at λ multiples of a median-based estimator for the standard deviation of all samples in the data epoch. The optimal value for λ depends on your measurement setup and eye tracking hardware. For microsaccade detection, values between λ=4 and λ=6 have been used in the literature.

Minimum saccade duration
Minimum duration of saccades in sampling points. Movements lasting less than MINDUR samples will not be detected as saccades.

Smooth raw data to suppress noise?
If the checkbox is set, eye position data is smoothed before saccade detection (recommended for ET with high sampling rates).

Compute velocity thresholds globally (across epochs)?
If the checkbox is set, the same detection thresholds are applied to all data epochs. Velocity thresholds will not be computed individually for each epoch, but globally across all data epochs. If the dataset is still continuous, this setting has no effect (since there is only one "epoch" anyway). *Note:* This option has been added for the current plugin. It is still experimental.

For clusters separated by less than... do the following
Without additional processing, parts of the same eye movement trajectory may be detected as more than one saccade. For example, there is often an overshoot component seen at the end of saccades ("glissades", cf. Nyström & Holmqvist, BRM, 2010). If eye velocity drops temporarily below the threshold, the glissade would be detected as a new saccade event, even though it is probably better classified as part of the same movement.

To cluster saccades, enter a value (in milliseconds) in the input field. This value defines the minimum plausible interval (fixation duration) between successive saccades (e.g., 50 ms). The dropdown menu then offers four options on how to treat clusters of saccades that are closer together than this value:

- *1. keep all saccades*: Do nothing. Keep all saccades (even if they occur in temporal proximity)
- *2. keep first saccade*: Of each temporal cluster of saccades, keep only the first saccade
- *3. keep largest saccade*: Of each temporal cluster of saccades, keep only the saccade with the largest distance
- *4. combine into one saccade*: Combine all movements of a cluster into one saccade

Note: This function has been added for the current plugin. While we believe that it is useful, it is still experimental.

Add eye movements to EEG.event?

Add saccades
If checked, saccades will be written as new events to *EEG.event*. We strongly recommend to test your detection parameters before writing saccade events! Currently, there is no automated way to "undo" saccade detection without reloading the dataset.

Add fixations
If checked, fixations will be written as new events to *EEG.event*. We strongly recommend to test your detection parameters before writing fixation events! Currently, there is no automated way to "undo" fixation detection without reloading the dataset.

Plot figure with eye movement properties
If checked (recommended), a figure is produced that summarizes properties of detected saccades and fixations.

Make all settings and click **OK**.
In the MATLAB command window, you will now see detailed feedback on eye movement detection.

Show eye movement properties

If you activated the checkbox *Plot figure with eye movement properties* the detected eye movements are summarized:

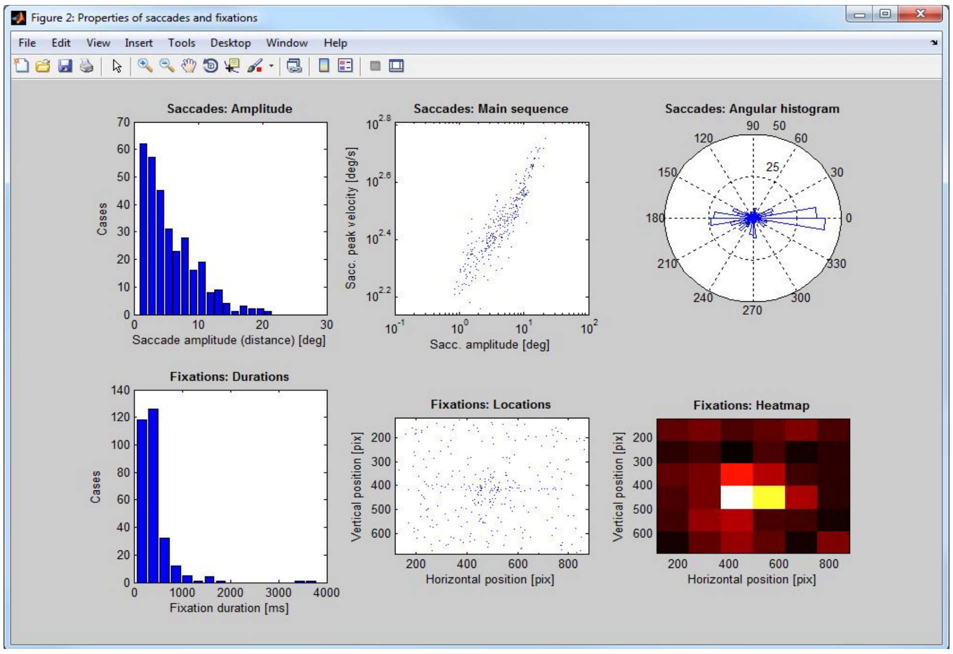


Figure 2: Properties of saccades and fixations
File Edit View Insert Tools Desktop Window Help
Saccades: Amplitude
Cases
Saccade amplitude (distance) [deg]
Saccades: Main sequence
Sacc. peak velocity [deg/s]
Sacc. amplitude [deg]
Saccades: Angular histogram
Fixations: Durations
Fixation duration [ms]
Fixations: Locations
Vertical position [pix]
Horizontal position [pix]
Fixations: Heatmap

- *Saccades: Amplitude*:
  shows the distribution of saccade distances (Euclidian distance between the starting position and the landing point of a saccade). If *Degrees visual angle per pixel* was set in the previous dialogue, amplitudes are given in degree visual angle
- *Saccades: Main Sequence*:
  saccade amplitude is plotted against saccade peak velocity on a logarithmic scale ("saccadic main sequence")
- *Saccades: Angular histogram*:
  shows distribution of saccade orientations (e.g. leftwards vs. rightwards). This plot does not convey information about saccade amplitudes. Note: This plot assumes that the origin of the ET coordinate systems is in the *upper* left screen corner. Otherwise saccade directions will be flipped vertically.
- *Fixations: Durations*:
  distribution of fixation durations [in ms]
- *Fixations: Locations*:
  scatter plot of fixation locations (on the stimulus)
- *Fixations: Heatmap*:
  same data as in previous plot, shown as a "heatmap" (2D histogram).

How are eye movements stored in *EEG.event*?

If selected, saccades and/or fixation will be written as new events to *EEG.event*. This may greatly increase the size of the *EEG.events* structure. The following properties are stored:

Saccades:

- **type**: "saccade"
- **latency**: latency of saccade onset [sampling points]
- **duration**: duration of saccade [sampling points!]
- **sac_vmax**: peak velocity of saccade [degree/pixel per second]
- **sac_amplitude**: amplitude of saccade [degree/pixels]
- **sac_angle**: angular orientation of saccade [in degree]. Note that the origin of the ET coordinate system can be in the *upper* left screen corner.
- **sac_startpos_x**: horizontal (X) launch position of saccade (pixel) [fixation position immediately before saccade onset]. Averaged across eyes for binocular recordings.
- **sac_startpos_y**: vertical (Y) launch position of saccade (pixel) [fixation position immediately before saccade onset]. Averaged across eyes for binocular recordings.
- **sac_endpos_x**: horizontal (X) landing position of saccade (pixel) [fixation position immediately after saccade offset]. Averaged across eyes for binocular recordings.
- **sac_endpos_y**: vertical (Y) landing position of saccade (pixel) [fixation position immediately after saccade offset]. Averaged across eyes for binocular recordings.
- **epoch**: number of corresponding data epoch ("1" if data is continuous)

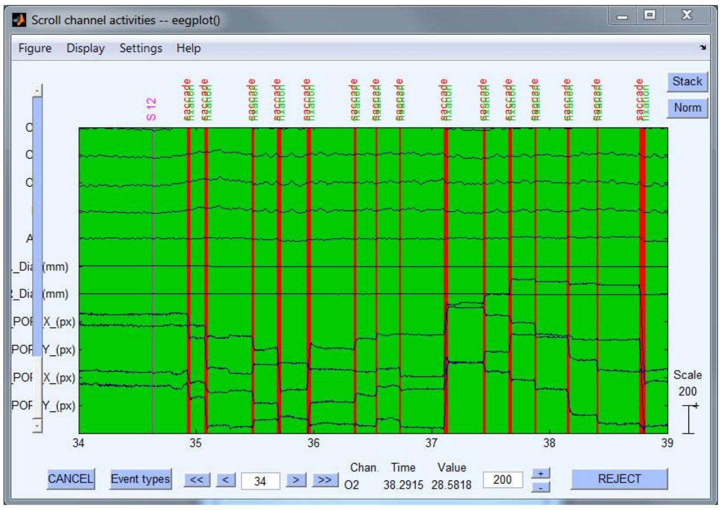
Fixations:

- **type**: "fixation"
- **latency**: latency of fixation onset [sampling points]
- **duration**: duration of fixation [sampling points!]
- **fix_avgpos_x**: horizontal (X) fixation position (pixels). Averaged across eyes for binocular recordings.
- **fix_avgpos_y**: vertical (Y) fixation position (pixels). Averaged across eyes for binocular recordings.
- **epoch**: number of corresponding data epoch ("1" if data is continuous)

Note on EEG.urevents: The new eye movement events are also written to the urevent structure (*EEG.urevent*). Although eye movements are strictly speaking not *ur*-events (because they are only added later on) this has several advantages. Furthermore, EEGLAB does not tolerate if *EEG.event* contains entries that are not also present in *EEG.urevent*. Eye movement events are also written to *EEG.urevent* if they are detected in epoched data.

Look at new eye movement events

In a channel data plot (*Plot > Channel data (scroll)*) you will now see the new saccade and/or fixation events. In the plotting window, you can switch on *Display > Plot event duration* to highlight their durations in color:



Step 7: Reject independent components with eye tracker

Objectively identify ocular ICs based on the electrically independent eye track

Eye movements and blinks generate different types of measurement artifacts in the EEG: Artifacts from the movement of the corneoretinal dipole, a muscle spike potential at saccade onset, and the "rider" artifact of the eye lid during blinks and upward saccades. In addition, saccades are preceded and followed by several types of saccade-related brain potentials.

Independent component analysis (ICA) is an established method to remove artifacts from EEG data. However, the decision whether a particular independent component (IC) reflects a non-cerebral artifact source or genuine brain activity is often made subjectively. Furthermore, the criteria for this classification are often neither reported nor replicable.

Simultaneous eye tracking provides a potentially effective and unbiased way to identify signal components that reflect ocular artifacts. The reason is that *"the eye track provides a measure of eye position that is electrically independent of the EEG. Correlations between eye track and EEG after correction are therefore likely to result from (...) corneoretinal or myogenic artifacts (or, less likely, from saccade-related brain activity occurring in synchrony with the saccade). (...) Considering the relationship between the components' time series and gaze position (i.e., eye tracker-informed ICA) should greatly improve the reliability of component selection."* (Dimigen et al., 2011, p. 567). The saccadic spike potential, in particular, often decomposes into multiple ICs, whose scalp maps may be difficult to identify otherwise.

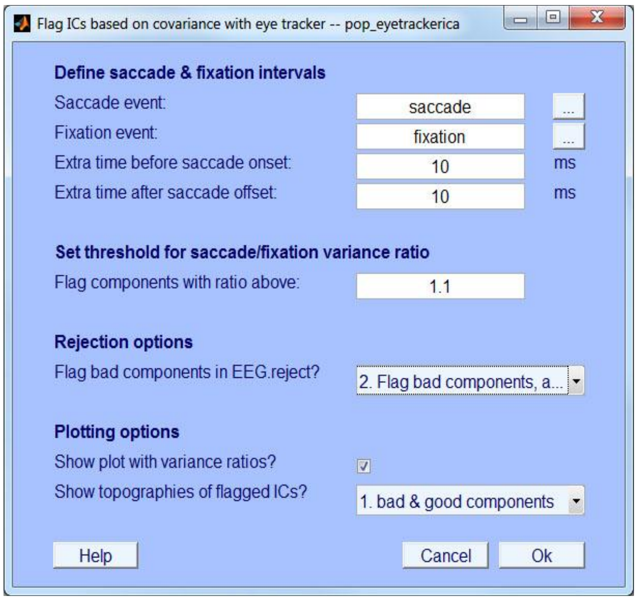
Plöchl et al. (2012) recently proposed and evaluated an elegant criterion to identify ocular ICs using concurrent eye tracking. According to their variance-ratio criterion, an IC is likely to reflect ocular artifact if it shows more activity in its time course during saccades than during fixations. Because ocular artifacts (the spike potential, in particular) may begin slightly before saccade onset and may outlast the duration of the saccade, some additional time can be added to saccade intervals to make this measure more robust.

The EYELAB plugin implements the variance-ratio criterion by Plöchl et al. using their settings as the default. Please note that this procedure relies strongly on a good ICA decomposition, which is not trivial to achieve for data recorded during natural, multi-saccadic vision. Spatial filtering methods that define artifact components empirically (see Dimigen et al., 2011) may provide a more robust alternative under these circumstances.

Prerequisites for using this function:

- A (good) ICA decomposition has been achieved (*Tools > Run ICA*)
- Saccades and fixations are present in *EEG.event* (see *Eyetracker > Detect saccades & fixations*)

Click *Eyetracker > Reject components with eye tracker*. The following dialogue will pop up:


Flag ICs based on covariance with eye tracker -- pop_eyetrackerica

Define saccade & fixation intervals

Saccade event:	saccade	...
Fixation event:	fixation	...
Extra time before saccade onset:	10	ms
Extra time after saccade offset:	10	ms

Set threshold for saccade/fixation variance ratio

Flag components with ratio above: 1.1

Rejection options

Flag bad components in EEG.reject? 2. Flag bad components, a...

Plotting options

Show plot with variance ratios? ✓

Show topographies of flagged ICs? 1. bad & good components

Settings for component rejection

Define saccade & fixation intervals

- *Saccade event*: Name of saccade event in *EEG.event* (default: "saccade").
- *Fixation event*: Name of fixation event in *EEG.event* (default: "fixation")
- *Extra time before saccade onset*: Saccade intervals will be extended to include this interval before saccade onset.
- *Extra time after saccade offset*: Saccade intervals will be extended to include this interval after saccade offset.

Note: When component variance during saccades and fixations is computed, the extra time assigned to saccade intervals will be subtracted from the corresponding fixation intervals.

Set threshold for saccade/fixation variance ratio

Flag components with ratio above: Critical variance ratio. Components that exceed this ratio are flagged for rejection.

Rejection options

Flag bad components in EEG.reject?

- *1. Test mode, flag nothing*: Variance ratios are shown in the MATLAB console. No components are flagged for rejection.
- *2. Flag bad components, add to existing flags*: Components that exceed the ratio are set to "reject" in EEG.reject.gcompreject. Existing rejection ("bad") flags in EEG.reject.gcompreject are not undone.
- *3. Flag bad components, overwrite existing flags*: Components that exceed the ratio are set to "reject" in EEG.reject.gcompreject. ICs that *do not* exceed the threshold will be set to "good" in EEG.reject.gcompreject.

Plotting options

Show plot with variance ratios?: Plot variance ratio for all components

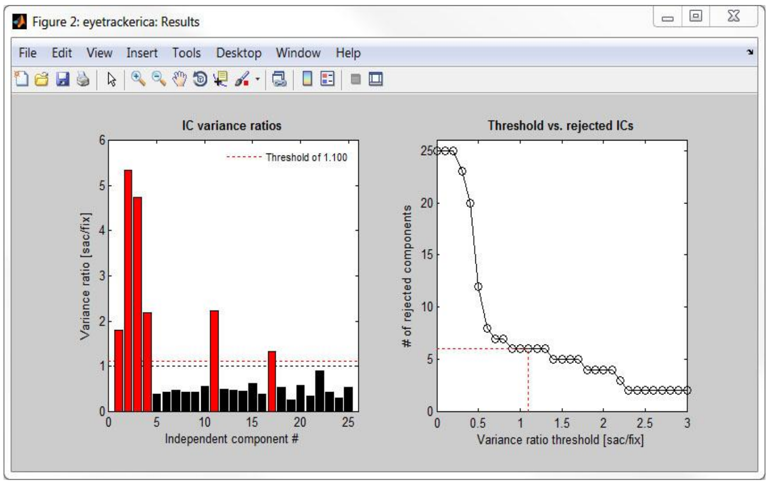
Show topographies of flagged ICs?

- *1. bad & good ICs*: Plot two figures showing the "bad" and "good" IC topographies, respectively.
- *2. bad ICs*: Plot only topographies of ICs flagged for rejection.
- *3. good ICs*: Plot only topographies of ICs *not* flagged for rejection.
- *4. do not plot topos*: Do not plot any IC topographies.

Make all settings and click **OK**.

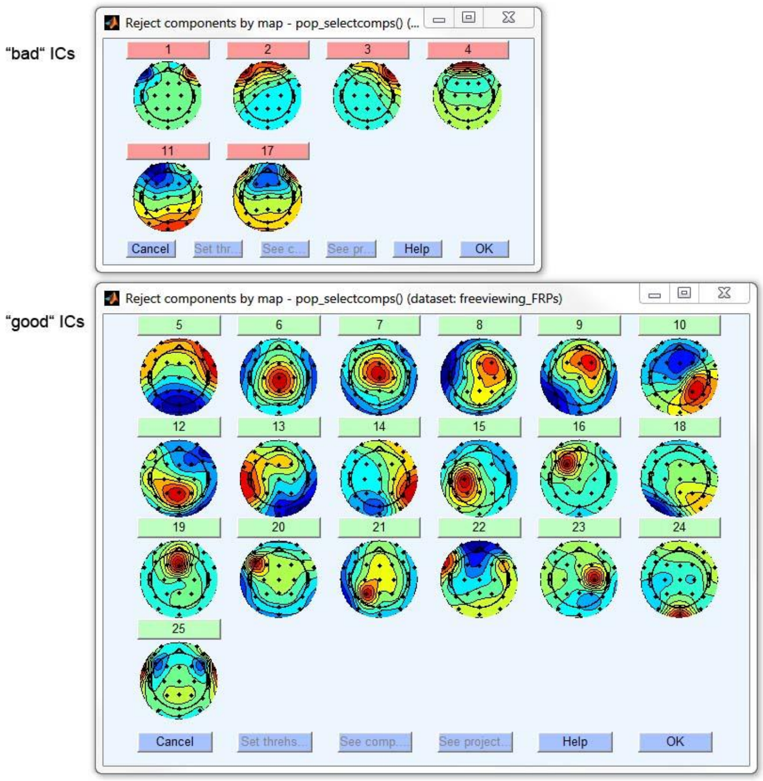
Display the results of component rejection

In the MATLAB command prompt, you will now see the variance of component activity time courses during saccade and fixation intervals and the resulting variance ratio. Components that exceed the threshold ratio are marked with an asterisk. If you selected *Show plot with variance ratios*, an extra figure pops up that displays these results graphically:



The left plots shows you the variance ratio for all components together with the choosen threshold (here: ratio of 1.1) . Components that exceed the threshold are highlighted in red (here: 6 components). The right plot illustrates you how many components would have been rejected at different thresholds. The choosen threshold (here: 1.1) and the number of components that exceed it (here: 6) are indicated by the dashed red lines.

If you have selected *Show topographies of flagged ICs*, one or two additional figures will display the maps of all components flagged as "bad" and/or "good". These plots (*EEGLAB function pop_selectcomps*) can also be used to change the rejection status of the components manually. *Note*: Topographies are not shown in "Test mode" or if there are no electrodes coordinates for the dataset.



Remove components

To actually remove the flagged ("bad") components from your dataset, click *Tools > Remove components*.

Writing scripts

Using custom scripts

With EEGLAB's history scripting functionality, MATLAB code is automatically generated whenever the graphical user interface is used. The command history can be retrieved with the eegh() command and easily adapted into custom scripts (e.g., to loop over participants). Below are the names of the underlying main functions for steps 2 to 7 above:

```
>> eegh % get EEGLAB command history

parsesmi(...); parse raw data from SMI

parseeyelink(...); parse raw data from EyeLink

EEG = pop_importeyetracker(...); import & synchronize EEG & eye tracker

EEG = pop_applytochannels(...); use EEGLAB function on some channels only

EEG = pop_rej_eyecontin(...); discard cont. intervals with out-of-range eye track

EEG = pop_rej_eyeepochs(...); discard epochs with out-of-range eye track

EEG = pop_detecteyemovements(...); detect saccades & fixations

EEG = pop_ploteyemovements(...); visualize eye movement properties

EEG = pop_eyetrackerica(...); remove ICs that covary with gaze
```