Biosignal Processing and Modeling
Tutorial – Eye Blink Detection

Course Instructors: Alireza Malekmohammadi `alireza.malekmohammadi@tum.de`,
Nicolas Berberich, `n.berberich@tum.de`
Teaching Assistant: Karahan Yilmazer, `karahan.yilmazer@tum.de`

Summer Semester 2023

Introduction

In the previous tutorial you have seen how raw and filtered EEG data streams look like. Furthermore, you have seen how prominent of an artifact eye blinks are in EEG recordings. In this tutorial, we will exploit their significance to use eye blinks as a control signal. Since this can be achieved with any EEG device you use without training the BCI pilot, you will have your first control signal by the end of this tutorial.

In contrast to previous tutorials, you will have to write the code yourself in this one. A skeleton script is provided for convenience. Feel free to change it to your liking.

Info: These tutorials are designed keeping in mind that the class comprises of people from different backgrounds. Do not be overwhelmed, the topics will be covered in as much detail as required to enable you to start using the technologies presented in the tutorials.

1 Detecting Eye Blinks

In the `T2-eye-blink-detection` folder, you will find a script called `eeg_eye_blink_detection.py`. This script should accepts raw EEG data through an LSL stream, filter it and compute a dynamic threshold for an eye blink detection. In the following one possible approach will be explained. You can either translate the explained algorithm step by step to Python code or come up with your own algorithm.

1.1 Explanation of the Algorithm

As discussed, eye blinks appear as prominent peaks in the EEG signal. This prominence led to the idea that a simple threshold model could detect eye blinks in the signal. In other words, when the signal amplitude exceeds a certain threshold, it should be classified as an eye blink.

Setting the threshold too low would result in random signal fluctuations to get falsely detected as eye blinks. On the other hand, setting it too high would result in no blinks getting detected. Fixing the threshold to a suitable value before each recording seemed to have solved this issue.

However, the signal amplitude changes over time caused another issue. Sometimes after the recording started, a higher threshold was required because, for example, the electrodes could move due to too strong blinking, and thus the overall signal amplitude would increase. Alternatively, sometimes the opposite held true. After the temperature of the mastoid reference electrodes matched the person's body temperature, the noise in the signal would decrease, making the overall amplitude slightly smaller than what the threshold was initially set for.

In the sample solution (which is not provided at the moment) these issues were addressed by implementing an adaptive threshold. The threshold would get updated in real-time based on the signal in a baseline window. In each iteration, the mean and the standard deviation of this baseline were calculated. A factor c was then multiplied with the standard deviation, and the result was added to the mean. This value would give the height of the current threshold.

More concretely, let T denote the threshold value, $\overline{x}_{BL}$ the mean of the baseline and σ_{BL} the standard deviation of the signal in the baseline window. Then the update rule for the threshold was defined as:

$$T = \overline{x}_{BL} + c\sigma_{BL} \tag{1}$$

Furthermore, let $D \in \{0, 1\}$ denote a Boolean variable specifying whether a blink was detected (1) or not (0) and x the amplitude of an arbitrary sample. Then the eye blink detection rule was defined as:

$$D = \begin{cases} 0 & x \leq T \\ 1 & x > T \end{cases} \tag{2}$$

Since the system worked in real-time, only the most recent samples were of interest when looking for an eye blink. Concretely, this meant that it was enough to apply equation 2 to a small window at the very end of the signal buffer, called the activity window. Lastly, since the goal was to detect single eye blinks and such eye blinks appeared as singular peaks, the maximum value in the activity window was enough to use as the sample amplitude x in equation 2.

These steps were done after band-pass filtering the signal between 1-30 Hz and applying a moving average filter to smoothen the signal. For band-pass filtering, a one-pass, zero-phase, non-causal filter was used. As the filtering was done on data buffers, using a non-causal filter was possible.

As moving average filters act as low-pass filters, it was rather an optional step to apply it for better visualization. It has to be noted that other smoothing filters, e.g., Gaussian filter, could be used for this purpose as well, but they were not tested out in this study.

Although it was shown that high-pass filters above 0.1 Hz deteriorate the waveform of EEG signals [1] [2], as mentioned, 1 Hz was selected for this implementation. There were three main reasons for this decision.

The first reason was that eye blink signals are muscle artifacts and not brain signals. This made it irrelevant whether the filtering distorted the waveform, as long as the filtered signal was used only for eye blink detection. While the system was running, raw EEG data could still be accessed. This meant that the signal could be filtered again with a different cutoff frequency for another application, e.g., for MI.

Second, it was intended that the system only detected intense, voluntary eye blinks and not the spontaneous ones that the user cannot prevent. This phenomenon can be seen in Figure 1. The small peaks around the samples 800, 2100, 2900, 4100, and 4800 were spontaneous eye blinks. They did not get detected because they were sub-threshold peaks. On the other hand, the peaks around 1100, 1800, and 3300 were voluntary single blinks with large amplitudes exceeding the detection threshold, thus getting detected as eye blinks. A 1 Hz high-pass filter was suitable for this task, as it "squished" these small amplitude peaks and kept only the voluntary peaks intact.

Lastly, a high-pass filter with a lower cutoff rate requires a longer filter, i.e. more samples to operate on [3]. Concretely, a 0.1-30 Hz band-pass filter would have the length of 8251 samples, which was larger than the buffer size of 5000 samples. The buffer size might have been increased to use this filter but this would result in an overall slower system. This was because the filtering would require more computing power and also there would be more samples to filter. Since this was an online system that operated with signal buffers in quasi real-time, keeping the system as light as possible was desirable.

One of the biggest disadvantages of using a buffer was the existence of edge artifacts due to filtering [4]. They can be seen at the start and end of recordings. These artifacts occur because filters try to access samples that are not available. To solve this issue it is recommended that recordings should start earlier and end later than planned, so that only these empty periods get affected by edge artifacts and can later get cropped out [2]. But this cannot be done in an online setting where only fixed signal buffers are available.

These artifacts could be ignored but unfortunately the random fluctuations that they cause would sometimes result in a false blink detection. Therefore, a small delay of 20 samples (0.08 s for 250 Hz sampling rate) was introduced to the activity window. This delay was short enough that the users would not even recognize it but sufficient to solve the problem of edge artifacts.

The introduced delay can be seen in Figure 2. This figure shows the zoomed in end of the signal. As it can be seen, filtering caused an edge artifact which persisted even after smoothing the function. The short delay accounted for this small residual. Due to this delay, the most recent samples reached the red activity window, starting around sample 4850, only after 0.08 s.

The last part of this detection algorithm was the implementation of a "refractory period" for blink detection. Without this period, a blink that the system detected would keep on getting detected as a new blink until it exited the activity window. This was because the threshold would not yet get updated until the detected peak entered the baseline period and increased the standard deviation

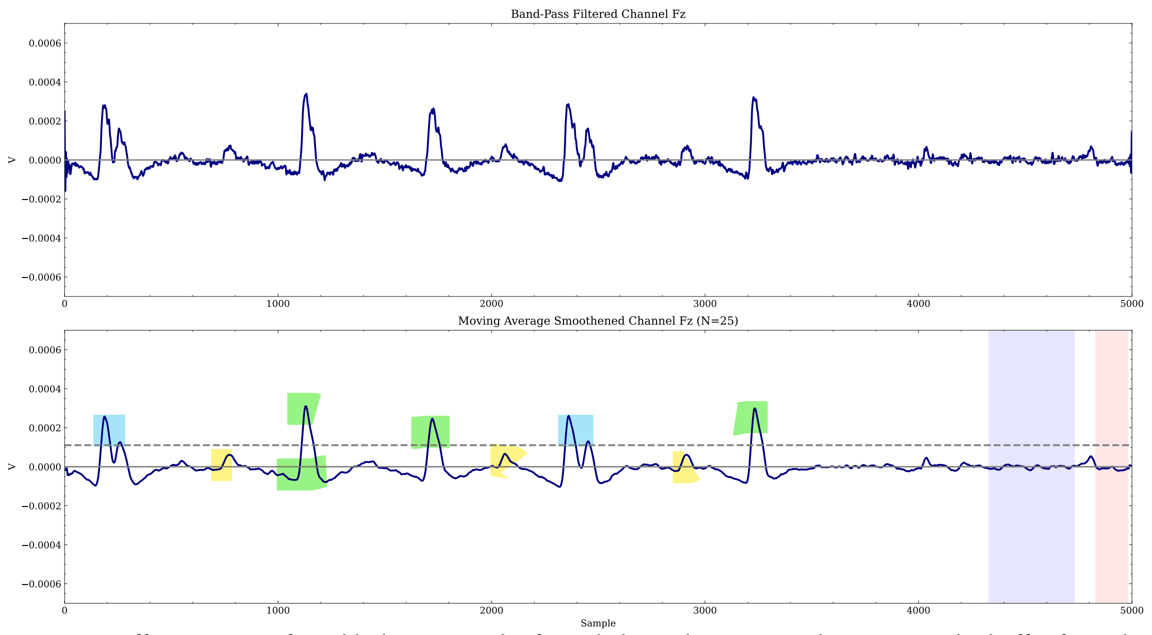

Figure 1: Different types of eye blinks seen in the frontal channel. New samples arrive in the buffer from the right. The blue window corresponds to the baseline and the red to the activity window. Small peaks around samples 800, 2100, 2900, 4100, and 4800 are sub-threshold spontaneous blinks, i.e., they do not get detected as eye blinks. Peaks around the samples 1100, 1800, and 3300 are voluntary single blinks with large amplitudes exceeding the detection threshold, thus getting detected as eye blinks. The peaks around the samples 200 and 2400 are double blinks. It can be seen that the second blink has slightly less amplitude than the first.

of the baseline signal. By making the system go into a refractory period until the peak exited the activity window, this issue could get resolved.

Concretely, the program would run a predefined number of iterations doing nothing and waiting for the refractory period to be over. This part was crucial for later, when the system would send control outputs. Without the refractory period, the control signal would not be sent only once, but as many times as the blink would get detected until it exited the activity window.

1.2 Intended Usage

After connecting your EEG headset and streaming data through LSL, run the `eeg_eye_blink_detection.py` script. After a brief calibration, you should see the filtered signal from a single frontal channel (Fz). When you produce a strong blink, it should get detected by the script and you should see a console output of "PEAK #X DETECTED".

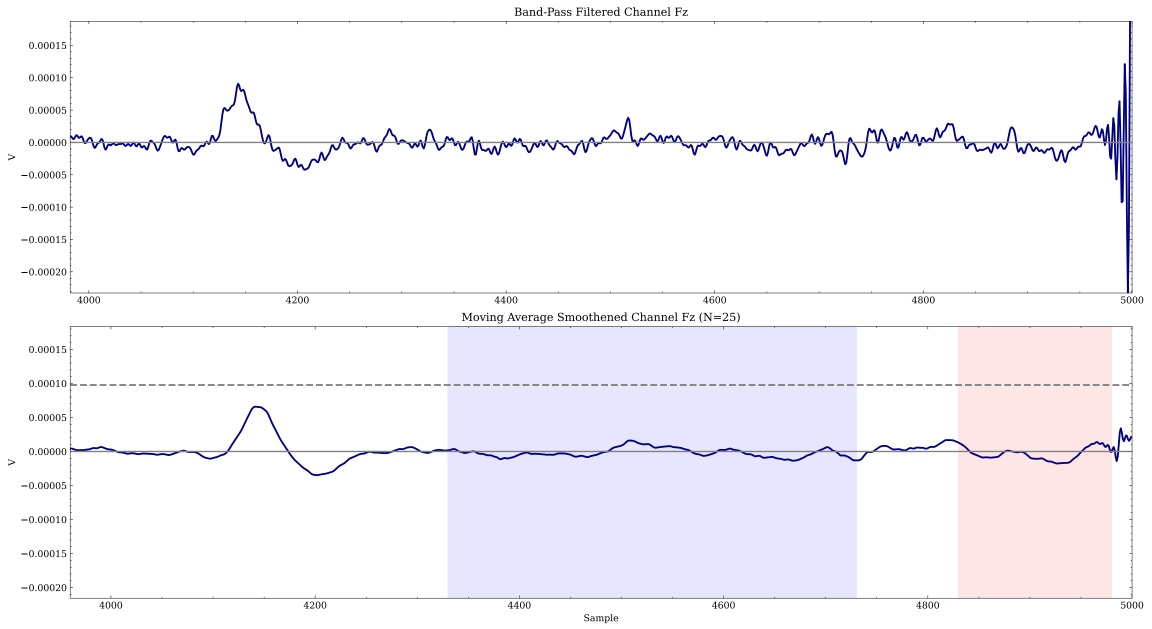

Figure 2: Edge artifacts due to filtering. At the edge of the buffer, filtering artifacts can be seen due to the filters trying to reach for samples that are not yet available. A short delay was introduced to the system to overcome the issue of false blink detections due to such artifacts. This delay causes the activity window to be shifted slightly to the left.

References

[1] Andreas Widmann, Erich Schröger, and Burkhard Maess, "Digital filter design for electrophysiological data – a practical approach," vol. 250, pp. 34–46.

[2] Steven J. Luck, *An Introduction to the Event-Related Potential Technique*, The MIT Press, second edition edition.

[3] Andreas Widmann and Erich Schröger, "Filter Effects and Filter Artifacts in the Analysis of Electrophysiological Data," vol. 3, pp. 233.

[4] Alain de Cheveigné and Israel Nelken, "Filters: When, Why, and How (Not) to Use Them," vol. 102, no. 2, pp. 280–293.