IRF replication report

Raimondas Zemblys, Diederick Niehorster, Kenneth Holmqvist

1. Introduction

In this report we replicate the main results of Zemblys et al. [3] using an updated version of the IRF algorithm featuring updated probabilistic post-processing code and an additional hard post-processing step (see Section 2.4). We perform this replication using newly collected clean eye-tracking data recorded with a high-end eye-tracker. As in Zemblys et al. [3], we systematically resample this data and add increasing levels of noise to simulate recordings from other eye-trackers. We then train a random forest classifier to predict eye movement events from features used by existing event detection algorithms, in conjunction with descriptors of data quality. All code and input data needed for this replication are available alongside this report at `https://github.com/r-zemblys/irf`. The trained model (`irf_2018-03-26_20-46-41`) used in this replication report is furthermore available at `https://doi.org/10.5281/zenodo.1343920`.

2. Method

2.1. Data

Data were recorded from 5 volunteers and author DN. Data from one participant were discarded due to excessive blinks. The participants provided informed consent. Data were recorded binocularly at 1000Hz with the SR Research EyeLink 1000Plus in remote mode from participants stabilized on a chinrest. The default setting level 2 heuristic filter [2] was applied to the data. After a 13-point calibration, participants were shown a series of 53 points distributed across a 37 deg by 21.6 deg area of the screen in a 8x7 hexagonal grid. Each dot was shown for 1500 ms, and the dots were shown in random order.

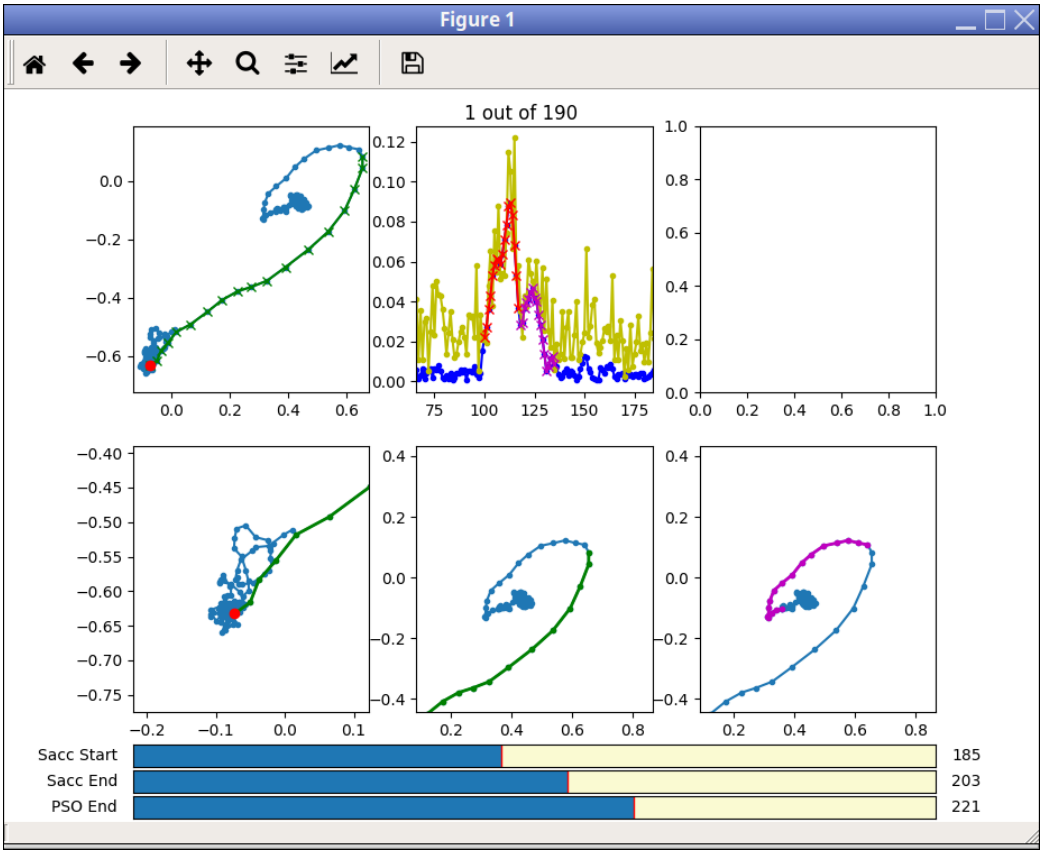

Figure 1: GUI for manual tagging

2.1.1. Manual event classification

Author RZ manually tagged raw data of the left eye into fixations, saccades, post-saccadic oscillations (PSOs), blinks and undefined. A two stage approach was used, where first approximate saccade intervals, blinks and undefined data were identified using the MATLAB software from Andersson et al. [1][1], and then a custom Python interface was used to adjust onsets and offsets of saccades and PSOs (see Figure 1). The remaining samples of the data are assumed to comprise fixations. The coding interface showed 2D positional data of the selected saccade with 100 samples of data before

[1]Available for download at https://github.com/richardandersson/EyeMovementDetectorEvaluation

and after (Figure 1, top left). Next to the 2D saccade plot is a velocity (or more specifically: a displacement per time unit) plot, featuring raw (yellow) and filtered (blue; Savitzky-Golay differentiation filter, window size 7 samples, polynomial order 2) traces (Figure 1, top middle). The bottom row of the interface shows zoomed in 2D positional data, focused on saccade onset, saccade offset and PSO offset. For the purpose of visualization in this GUI, all positional data was filtered with a Savitzky-Golay smoothing filter with a window size of 3 samples and a polynomial order of 1. Onset and offset of the saccade, as well as the offset of the PSO can be adjusted with slide bars at the bottom of the interface.

The manually coded events comprise the input events used to train and evaluate the algorithm. Because the aim of this work is to develop a framework for reproducing this coding on new data, whether the manual coding reproduces the exact definition of what fixations, saccades and PSOs is out of the scope of this report as well as of the original paper [3]. Nonetheless, here is an approximate description of how author RZ classified events:

- In general, I tended not to use the velocity plot, but relied only on the 2D gaze traces.
- Saccade onset is a sample that starts a sequence of directed samples heading out of the cluster
- Saccade offset is a sample that ends directed sequence of samples and either brings gaze to a new cluster, or starts changing direction.
- PSO offset is a sample that brings gaze to a new cluster. PSO samples show an oscillating pattern, which is likely caused by movement of the crystalline lens.
- A cluster is sequence of samples that do not show any directional behavior; i.e. that look in a sense "chaotic".

However these "rules" were not strictly followed, because when it comes to manual tagging, we hold the belief that experts do not need to be constrained by strict rules. As part of acquiring their expertise, experts have automatized part of the detection of event on- and offsets, which they can apply in context. As is often the case with expertise, it is not easily expressed in strict rules and therefore can described as tacit knowledge or know-how. Manual classification and development of algorithms that can capture and mimic expert classification patterns enable the computer to behave like an expert coder. That is the over-arching goal of this line of research.

2.1.2. Baseline dataset

Figure 2 shows distributions of manually tagged event durations. Compared to the baseline dataset originally used in Zemblys et al. [3], the dataset used in this report is very similar. Overall, the new dataset has more events, because subjects looked at 53 instead of 49 targets and because each target was shown for a longer duration (1.5s instead of 1s). Fixations range from 15 to 1489ms, saccades from 6 to 124ms and PSOs from 5 to 61ms. In Zemblys et al. [3] fixation durations ranged from 21 to 1044 ms, saccades had durations from 6 to 92 ms while PSO durations were 2 to 60ms.

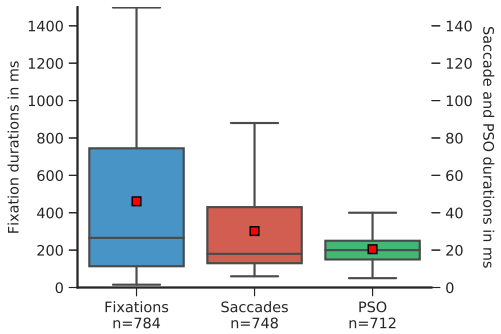

Figure 2: Manually tagged event durations. Red squares indicate means. This result closely replicates Figure 1 in Zemblys et al. [3]

Figure 3 shows the distribution of saccade amplitudes, which range from 0.17° to 41.54°. Compared to the range in Zemblys et al. [3], 0.1° to 29.8°, the new dataset has a similar lower bound, while the upper bound is higher and reflects a larger monitor size. The dataset used in this report has more short-amplitude saccades, which possibly are microsaccades brought on by the longer target durations.

The manually labeled dataset was split into development and testing sets by randomly selecting data from one subject (i.e. 20% of the data) to belong to the testing set. Further, the development set was split into training and validation sets by randomly assigning 25% of each trial to the validation sets and leaving the remaining 75% of the data in the training set. This is exactly what was done in Zemblys et al. [3].

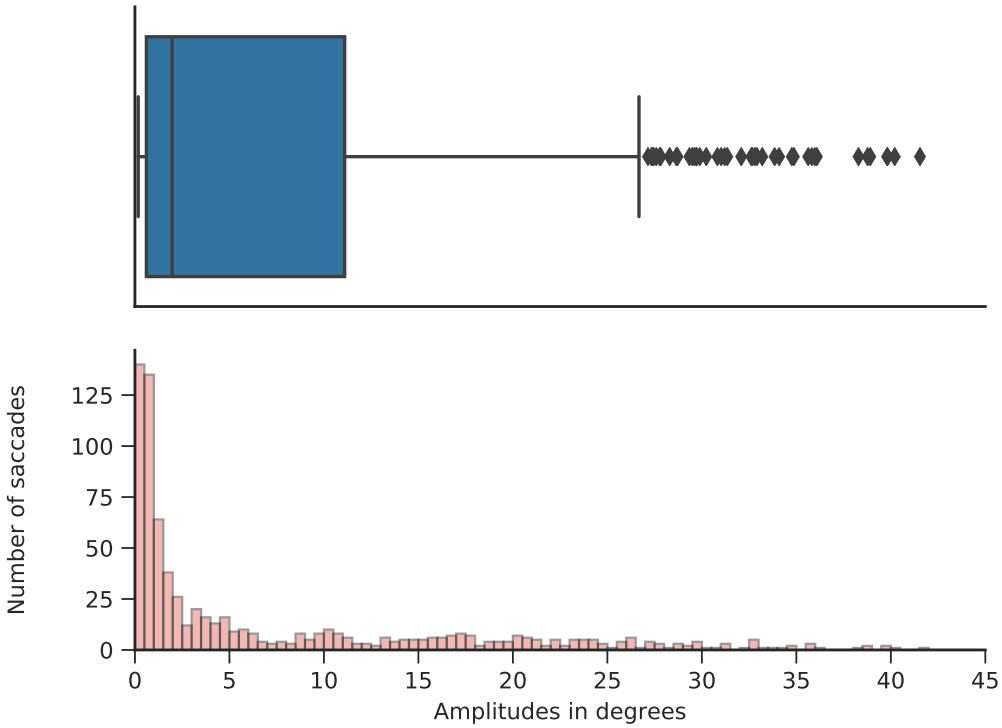

Figure 3: Distribution of amplitudes of manually tagged saccades. Bin size 0.5°. This replicates Figure 2 in Zemblys et al. [3]. Because of more small saccades, bin size was set to 0.25° instead of 1 deg used in original paper.

2.1.3. Data augmentation

First order spline interpolation is used to resample data to 30, 60, 120, 200, 250, 300, 500 and 1250 Hz. Before resampling, the data were low-pass filtered using a Butterworth filter with a cutoff frequency of 0.8 times the Nyquist frequency of the new data rate and a window size of 20 ms. Event data were resampled using "nearest" interpolation. Before resampling, missing data were interpolated to avoid artifacts, however interpolated data were again marked as missing after resampling.

Next, we systematically added white Gaussian noise to the resampled dataset at each sampling frequency. More specifically, for each recording in each data set, we added a white Gaussian noise signal generated using the Box-Muller method, separately for the horizontal and vertical gaze components. In the noise we add to datasets, we simulate increasing noise with distance from the middle of the screen by using a 2D Gaussian noise mapping function, which is minimal in the middle and maximal in the corners. The standard deviation of this Gaussian mapping function was chosen such that the noise level at the corners of the stimulus plane at $\pm 20^\circ$ was 3 times higher than in the middle of the screen. We further scale the variance of the generated noise signal to 10 levels starting from 0.005° RMS, where each subsequent noise level is double the previous one. This results in additive noise levels ranging from 0.005° to 2.56° RMS in the middle of the screen and 3 times higher noise at the corners. This again mimics Zemblys et al. [3].

2.2. Feature extraction

The 14 features that used in this report are the same as in Zemblys et al. [3]. In the original study [3] right before feature extraction, we interpolated missing data using a Piecewise Cubic Hermite Interpolating Polynomial. In this report however no interpolation is used. Because of that, 100ms of data before and after each period of track loss are labeled as undefined and are not used when training the random forest model and evaluating performance. This approach also handles saccade-like events, resulting from blinks, therefore no post-processing is needed to remove these.

Figure 4 shows Spearman's rank correlation between the features in the training dataset. Compared to Figure 4 in Zemblys et al. [3], feature correlations in the new dataset are nearly identical.

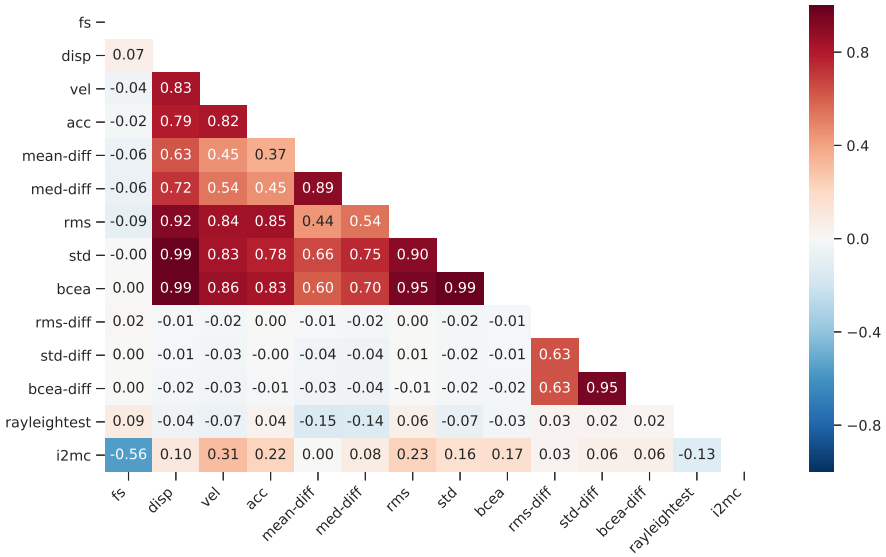

Figure 4: Spearman's rank correlation between the features in the training dataset. Replicates Figure 4 in Zemblys et al. [3]

2.3. Random forest model

In the original study [3], we started with a random forest model with 200 trees, however the results showed that there is no need for such a large model. Therefore in this report we train a model with 32 trees, and all 14 features and all other parameters remained the same as in Zemblys et al. [3].

2.4. Post-processing

For each of the data samples, the random forest classifier outputs three probabilities (summing to 1), indicating how likely it is that the sample is part of a fixation, a saccade or a PSO. The next step is to produce meaningful eye-tracking events. Following Zemblys et al. [3], we first apply a Gaussian smoother ($\sigma = 1$ sample) over time for each of the three probabilities and then apply heuristic steps to determine the final event labels.

In the original study post-processing was performed by downvoting and upvoting probabilities of samples that violate heuristic rules (which we will term *probabilistic post-processing*). This approach was selected so as to retain classification even if a detected event does not meet one of the defined post-processing criteria. For example, just removing short saccades would result in

a larger numbers of short undefined events, however if probabilities of "short saccade" samples are downvoted, these samples can become other classes. In our case that was only fixation class, but in future uses of this approach, someone may want to train the algorithm with e.g. smooth pursuit. The "short saccade" samples could then be downvoted to either become fixation or pursuit samples or any other event, depending on which of these has the next highest probability. However, it has since come to our attention that in the original work[3], some of the events that violated the heuristic post-processing rules in *unseen data and unseen scenarios* were not removed or not reclassified as other events. We have therefore updated the post-processing code to handle such erroneous behavior, and in addition implemented a routine that checks the output of probabilistic post-processing and, if required, as a final step performs "hard" post-processing with deterministic rules that remove samples that violate the post-processing rules. Below we list the heuristics applied in the original study and explain what was used in this report:

1. mark events that contain more than 75ms of interpolated data as undefined. *Not used in current report, because no interpolation of missing data was performed during feature extraction.*
2. merge fixations which are less than 75ms and 0.5° apart. *The thresholds used in this study are 75ms and 0.2°, because the new dataset includes many small saccades.*
3. make sure that all saccades have a duration of at least 3 samples, expand if required.
4. merge saccades that are closer together than 25ms.
5. remove saccades that are too short (<6ms) or too long (>150ms). *In this report only short saccades (<6ms) are detected and the probability of these samples being saccades is downvoted.*
6. remove PSOs that occur in other places than directly after a saccade and preceeding a fixation. *In this report PSOs not after saccades are removed by downvoting the probability that these samples are PSOs.*
7. remove fixations shorter than 50ms. *This step is left for "hard" post-processing step, see further.*
8. remove saccades and following PSO events that surround episodes of missing data as these are likely blink events. *Not used in current study, because these spurious events are dealt with by removing 100 ms of data at either side of an epoch of track loss (see above).*

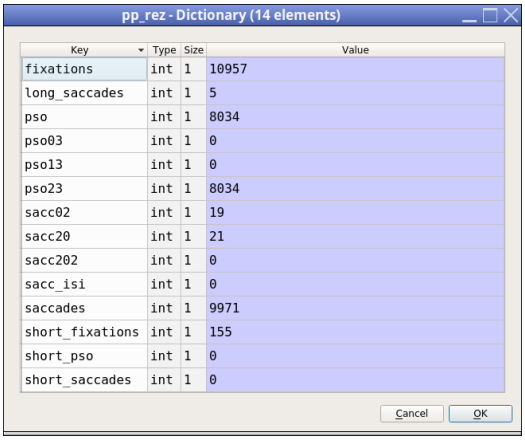

Figure 5: Number of events and violations of the post-processing rules after performing probabilistic post-processing on the validation set. Note that violations are just long saccades and short fixations, that by design were not handled by probabilistic post-processing step, while no too short saccades were found. In addition to reporting number of events found after the probabilistic post-processing step, this routine also reports number of PSOs after undefined (pso03), fixation (pso13) and saccade (pso23) events, saccades before (sacc20), after (sacc02) and surrounding (sacc202) undefined events and number of saccades closer than defined minimum intersaccadic interval (sacc_isi). Number of too short PSOs is not currently calculated.

An example of the output after this probabilistic post-processing step, obtained from the validation set, is shown in Figure 5. Note how 5 too long saccades and 155 too short fixations remain. These events were (by design) not handled in probabilistic post-processing (just because it would be a redundant procedure). Therefore, after the probabilistic post-processing step, a *hard post-processing* step is performed. It checks whether there are any events that violate post-processing rules and performs reclassification or removal if required. The *Hard post-processing* routine:

1. marks saccades with a duration less than 6 ms and more than 150 ms as undefined.
2. reclassifies PSOs after undefined to undefined and PSOs after fixations to fixations.
3. marks fixations with a duration lower than 50 ms as undefined.

For both post-processing steps, all thresholds in ms are converted to number of samples (rounded up to the nearest integer) and are available as configurable settings.

Finally, by running the hard post-processing routine again, the user can get a report like in Figure 5, detailing any remaining potential issues after both post-processing steps. The hard post-processing routine for instance does not remove saccades preceding (*sacc02*) and succeeding (*sacc02*) undefined, or undefined events surrounded by saccades (*sacc202*). It is up to the user to decide whether these should be removed or not. The report ensures that the users are aware of the occurrence of such situations in their output.

3. Results

Figure 6 shows the performance of the trained classifier on the validation set. We evaluate data using sample-to-sample comparison (measured using Cohen's Kappa). Agreement is only evaluated for samples that were manually tagged as fixations, saccades or PSOs and no post-processing is applied to the output of IRF for this evaluation. In Zemblys et al. [3], Figure 7 showed the performance of classifiers with different number of features across different sampling rates. As we do not perform feature selection in this report, we instead show performance for different sampling rates and different levels of additive noise. The last column shows average K over all

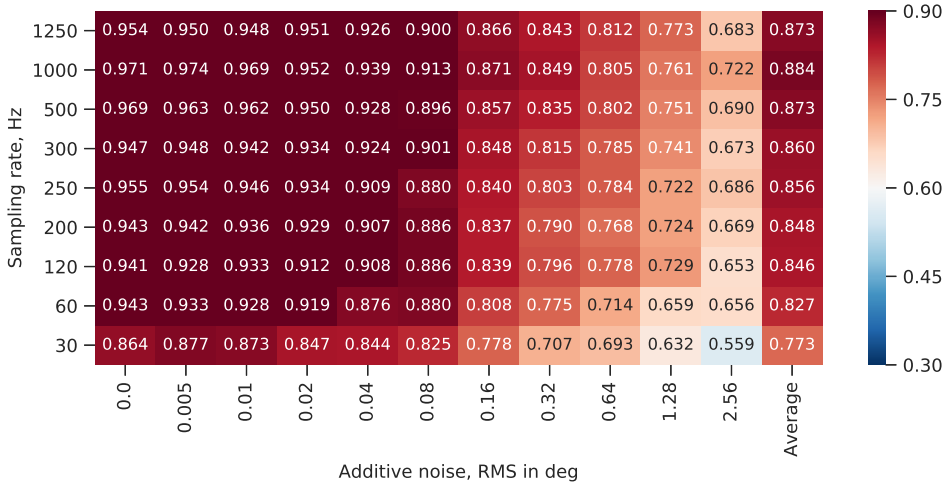

Figure 6: Performance of trained classifier on validation set. Performance is measured using Cohen's Kappa K and the presented figures are averages of K over all subjects in the validation set. Note that this is only the performance of the classifier without post-processing step. Last column shows average K over all noise levels.

noise levels and can be compared to column 1 in Figure 7 in Zemblys et al. [3]. The new classifier performs better compared to the results presented in Zemblys et al. [3]. Here the average K ranges from 0.88 in 1000Hz data to 0.77 in data sampled at 30 Hz, compared to K ranging from 0.86 to 0.48 in the original report (when 14 features and 200 trees were used, see Figure 7 in Zemblys et al. [3]).

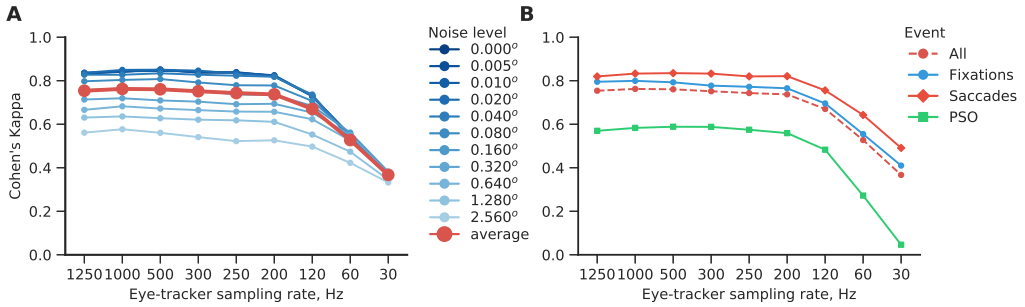

Figure 7: Performance on the testing dataset. Left side (A) shows performance for all events, with blue lines showing performance at different additive noise levels, while the red line depicts performance averaged across all noise levels. On the right side (B), we separately report performance for the three events our algorithm supports. Replicates Figure 10 in Zemblys et al. [3].

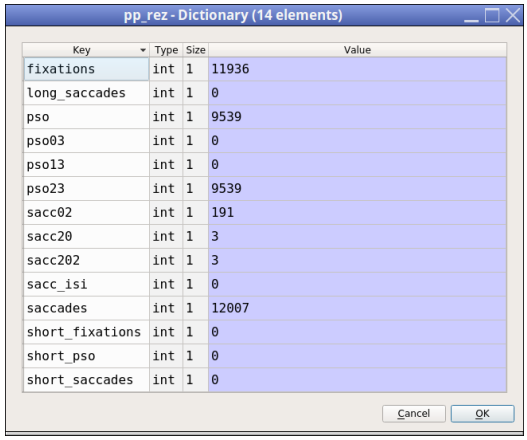

Figure 8: Output of post-processing check routine.

Figure 7 shows the performance of the improved IRF (including post-processing) on the testing dataset. Compared to Figure 10 in Zemblys

et al. [3], the new classifier, trained on the new dataset, and using updated post-processing shows nearly identical performance, as measured by Cohen's Kappa (K). Just like in the original study, performance is stable down to 200Hz and the algorithm performs quite well even at lower sampling frequencies. The classification performance slowly degrades with increasing noise. K, averaged over all noise levels, for all three events (figure 7, A) is around 0.75 in data down to 200Hz and gradually drops from 0.66 to 0.52 and to 0.36 in data sampled at 120Hz, 60Hz and 30Hz, respectively. For comparison, in the original study we report an average of 0.77 for data down to 200Hz and 0.72, 0.51 and 0.36 for 120Hz, 60Hz and 30Hz respectively (see Zemblys et al. [3, sec. 3.2.1]). Just like in the original study, the IRF classifier is best at correctly labeling saccades (see Figure. 7 B and Figure. 10 B in Zemblys et al. [3]).

In figure 8 we also show the output of the post-processing check on the testing set, for which we report results in this section. Note that all post-processing steps have been applied correctly: there are no too short or too long saccades or too short fixations, or PSOs not after saccades.

4. Conclusions

In this report, we have shown that the approach described in Zemblys et al. [3] is replicable and sound. Taking an entirely new, similar dataset and training a new classifier resulted in nearly identical results.

As described in detail above, for this report the post-processing routines have been improved. It should be clearly noted however, that different researchers do not necessarily need to use the same approach for post-processing. As is customary in the eye-movement community, final post-processing is the responsibility of the user, whether it means removing all saccades below a certain amplitude, or all fixations below a certain duration. For instance, the EyeLink manual of 2006 recommend users of their software that "Post-processing or data cleanup may be needed to prepare data during analysis. For example, short fixations may need to be discarded or merged with adjacent fixations, or artifacts around blinks may have to be eliminated." Similarly, the output of IRF may need further post-processing, but it should be tailored to the needs of the specific user.

5. Literature

[1] Andersson, R., Larsson, L., Holmqvist, K., Stridh, M., and Nyström, M. (2017). One algorithm to rule them all? an evaluation and discussion of ten eye movement event-detection algorithms. *Behavior research methods*, 49(2):616–637.

[2] Stampe, D. M. (1993). Heuristic filtering and reliable calibration methods for video-based pupil-tracking systems. *Behavior Research Methods, Instruments, & Computers*, 25(2):137–142.

[3] Zemblys, R., Niehorster, D. C., Komogortsev, O., and Holmqvist, K. (2017). Using machine learning to detect events in eye-tracking data. *Behavior Research Methods*, pages 1–22.