Erasmus University Rotterdam
Erasmus School of Economics
Master Thesis Econometrics

Robust Detection of Fixations in Eye-Tracking Data

Boyd Voet (457559)

Supervisor:	Andreas Alfons
Second assessor:	Carlo Cavicchia
Date final version:	13th August 2023

Abstract

This research adapts the existing DBSCAN algorithm by Ester, Kriegel, Sander and Xu (1996) in three ways, to detect fixations in eye-tracking data from static stimuli. These adapted versions seek to emulate the results of hand-labelled fixations and to be robust when dealing with lower-quality eye-tracking data. Adaptation is done through the use of strict consecutivity (DBSCAN-CONSECUTIVE), relaxed consecutivity (DBSCAN-ROBUST), and a velocity-based post-processing step (DBSCAN-VELOCITY). DBSCAN-VELOCITY best predicts hand-labelled values; it outperforms the other DBSCAN algorithms and ten other algorithms for similarity in fixation duration properties to human labellers and has the highest agreement with human labellers on a point-by-point basis. When the eye-tracking data quality is lowered with the introduction of bias, decreased precision, data loss and decreased frequency, DBSCAN-VELOCITY and DBSCAN-ROBUST are found to be suitable algorithms as their labels stay in almost perfect agreement when compared to their original labels on the unaltered data. DBSCAN-CONSECUTIVE performs worse for lower quality eye-tracking data, due to its strict consecutivity requirement. Directions for further research center around comparing other algorithms to DBSCAN-VELOCITY, modifying the elbow-point detection method, combining outputs from several algorithms, implementing the possibility for binocular data, replicating results on other data sets, and adapting the algorithm for changes in the participant-to-screen distance during a recording.

Contents

Chapter 1

Introduction

Eye movements tell us more about the interaction of a human with the world around them. Different kinds of eye movements are associated with different levels of visual intake and can inform us what parts of the world interest an observer most. The eye-movement event of interest in this research is a fixation; it occurs when the observer fixes their gaze on a specific region for a prolonged period. The minimum duration of this fixation differs between research, however the mean duration value of fixations in a sample is often between 200ms and 300ms (Andersson, Larsson, Holmqvist, Stridh & Nyström, 2017)(van der Lans, Wedel & Pieters, 2011). Visual intake is the highest during a fixation and thus detecting fixations can tell us which parts of a stimulus interest an observer.

The other two main eye-movement events are saccades and smooth pursuits. Saccades occur when the eye covers a relatively large distance in a short amount of time; they typically occur to move the gaze from one part of the stimulus to another. They are of short duration and have been hand-labelled to be as short as 8ms(Andersson et al., 2017). Smooth pursuits occur when the eye moves at a limited speed to track a moving object, a moving fixation, one could say. For this event to occur, there must thus be a dynamic stimulus in which moving points can be tracked.

In the past, humans labelled eye movements by hand to assign the different kinds of movements. However, this task is very time-consuming; thus, it is beneficial to create algorithms that perform this labelling automatically in a fraction of the time. This research aims to build an algorithm that can detect fixations accurately in high-quality data for static stimuli whilst also performing well on eye-tracking data with a lower quality.

For this, we modify an existing clustering algorithm DBSCAN to be suitable for eye-tracking data. We compare three versions of it to other algorithms on high-quality, hand-labelled data and then lower the data quality artificially to see the effects on fixation detection. We discuss previous research on the subject in Chapter 2, describe the data sets we use in Chapter 3, consider the methodology in Chapter 4, present the results in Chapter 5 and conclude with our main findings and consider directions for further research in Chapter 6.

Chapter 2

Literature Review

There are two main types of eye-tracking event-detection algorithms (Veneri et al., 2011). Velocity-based algorithms use the velocities of points to label them; points with a low velocity are potential fixations, and points with a high velocity are potential saccades. Dispersion-based algorithms use the spacing of points to label them; points close together are potential fixations, and points far apart are potential saccades. Algorithms like that of Startsev, Agtzidis and Dorr (2019) combine these two approaches sequentially.

Andersson et al. (2017) compare the performance of ten eye-tracking algorithms on both static and dynamic stimuli. None of these algorithms can detect smooth pursuits; all algorithms detect one or a combination of fixations, saccades and post-saccadic oscillations. They note that no single algorithm is used amongst all research, nor are there clear instructions for which eye-tracking algorithms to use for which kind of stimulus, and that algorithms require parameters to be inputted, which require domain knowledge, something not all users have. Marcus Nyström and Richard Andersson hand-label data recorded earlier by Larsson, Nyström and Stridh (2013) and use these hand-labelled values as the ground truth to judge performance. Friedman, Prokopenko, Djanian, Katrychuck and Komogortsev (2023) note that Marcus Nyström and Richard Andersson are considered experts in the field of identifying eye-movement events in their paper where they consider the inter-rater reliability of hand-labelling by humans. Evaluation of the ten algorithms is performed using a Root Mean Square Deviation (RMSD) score compared to the benchmark of human-labellers for the event duration parameters, and they use a Kappa score for point-by-point comparisons, where they evaluate the agreement between the human-labelled values and the values from the algorithms. They conclude that the selected algorithms performed reasonably well for static stimuli but hardly better than chance for dynamic stimuli. The latter is not surprising as the algorithms did not support smooth pursuit labelling, which is an event that frequently accurs in dynamic stimuli. Furthermore, they found that the choice of algorithm created dramatic variations in event properties, highlighting the importance of choosing a classification algorithm that suits the application. Friedman(2020) notes that two of the fourteen samples for static stimuli used in this research are 200Hz, while the rest are 500Hz. Andersson et al. do not realise this, which leads to a potentially flawed comparison of algorithms.

DBSCAN by Ester et al. (1996) is a clustering algorithm that constructs circles around each data point with a radius equal to *eps*, the algorithm's first parameter, and then labels the points with circles containing a number of points greater than or equal to $MinPts$, the second parameter, as core points. Core points that lie within each others' circles are clustered together, and the other points in the circles of these core points, so-called boundary points, are also added to the same cluster. Points that are not part of clusters receive the noise label. These properties allow DBSCAN to be used for eye-tracking data, where the intuition is that the clusters are fixations, while the noise points capture other eye movements. A benefit of DBScan is that it allows for the construction of clusters of arbitrary shapes, making different shapes of fixations possible for different types of stimuli. A second benefit is that it does not require us to choose the number of clusters in advance, unlike the K-means algorithm, for example.

We can link DBSCAN's parameters to eye-tracking. The value for *eps* can be related to the visual cone of individuals, and we can postulate that $MinPts$ must be proportional to the sampling frequency of the eye-tracker. The fovea is the part of the eye which yields the highest resolution. It sees $2°$ of the visual field; thus, for tasks like reading, the eyes must constantly move their gaze to bring different parts into the fovea (Fairchild, 2005). It is hypothesised and checked whether the core points in the DBScan clustering lie in the centre of these two degrees of the visual field, leading to an *eps* parameter of $1°$.

Previous research has applied DBSCAN to detect fixations in eye-tracking data. The first by S. Li et al. (2015) implements DBSCAN without accounting for the temporal aspect of the data. This allows clustering of points into fixations that are separated further in time than expected for points that are part of the same fixation. B. Li et al. (2016) account for the temporal element by requiring the points in the neighbourhood of a core point to be consecutive and set fixed values for the parameters *eps* and $MinPts$ of $0.5°$ and 20, respectively. This approach does not consider differences between individuals and samples and requires the $MinPts$ value to be adjusted manually if one uses eye-tracking data of a different frequency. Thirdly Yu et al. (2016) apply this same consecutivity but do not set fixed values for *eps* and $MinPts$. They base the $MinPts$ value on a minimum time for a fixation, $150ms$ in their case, times the sampling frequency, with a minimum value of 6. They determine their value for *eps* by frequencies of the dispersion maxima of consecutive points exceeding possible dispersion thresholds; this results in values for *eps* between $0.5°$ and $1.5°$. They collected data at 30 and 60Hz sampling frequencies on static stimuli, hand-labelled points as fixations or saccades and compared their algorithms' performance to four others according to four performance metrics. Their algorithm was the best performer among all performance metrics for the two different frequencies and for data with lower and higher amounts of saccades. These frequencies are equal to those of low-quality data, indicating that modifying DBSCAN for low-quality eye-tracking data is promising.

It is, however, impossible to find the code used by Yu et al., they do not describe their data processing, and their data is not shared. Thus replicating their findings is impossible; we can only label them promising. Another point of concern for their findings is that they hand-label their own data, construct an algorithm to detect fixations and then compare the performance of their algorithm against others on their hand-labelled data. Friedman et al. show that the agreement between human labellers is not perfect and note that the definition of a fixation differs per labeller and depends on their background. This has the consequence that when someone labels fixations according to their definition and creates an algorithm to detect their definition of fixations, it is more likely to outperform other algorithms that others have constructed with different perceptions of fixations.

Hoffmann et al.(2021) write on the replicability of research across disciplines. They note that the academic community has recently faced the problem that findings from earlier research often do not replicate on independent data; this is now known as the replication crisis. This issue is not the same as fraud and does not imply malicious intent; the problem is that there are multiple ways to attempt to answer the same research question; thus, it is important to describe methods and choices made elaborately. Hoffmann et al. recommend reporting the results of alternative analyses, acknowledging points of uncertainty in the methodology, and publishing all code and data one uses.

The eye-tracking data recorded by Larsson et al. is of high quality. They record it using a high-end eye tracker in laboratory conditions at a high frequency of 500Hz, it has high precision and few missing measurements. This measurement setup sets a high barrier to entry to do measurements of this quality. Designing an eye-tracking event detection algorithm that is robust to data of a lower quality would allow for a broader application of fixation detection as more studies and companies can collect this lower-quality data. Brand, Diamond, Thomas and Diamond (2021) evaluate the data quality of the Gazepoint GP3 eye-tracker when used by different study participants in non-laboratory conditions. The Gazepoint GP3 eye tracker is cheap ($845) compared to other eye trackers, and the measurement conditions in this paper are less demanding. The study participants receive training on using the tracker and must then perform sampling independently a week later. We can use the results from this research to artificially lower the quality of the data by Larsson et al. and observe the effects this has on the detected fixations.

Research by Hessels, Niehorster, Kemner and Hooge (2017a) and Dar, Wagner and Hanke (2021) focuses on robust eye-detection algorithms. Hessels et al. work on data from static stimuli, and Dar et al. work on data from dynamic stimuli, noting the need for this after observing that none of the algorithms evaluated by Andersson et al. performed much better than chance for dynamic stimuli. Hessels et al. (2017a) construct an algorithm for eye-tracking data from infants and young children. They introduced an algorithm, identification by two-means clustering (I2MC), designed for data with a wide range of noise levels and capable of dealing with periods of data loss. To judge its performance, they introduced varying noise levels to reduce precision and varying levels of data loss. Here they compare its robustness to the Adaptive Velocity Algorithm for Low-Frequency Data by Hooge and CampsHooge and Camps, the Binocular-Individual Threshold (BIT) algorithm by van der Lans et al.(2011) and the CDT algorithm by Veneri et al.(2011). These last two algorithms are also used in the research by Andersson et al., where we see that the BIT algorithm performs relatively well for both RMSD and Kappa scores. Hessels et al. show that the fixation properties of their I2MC are robust for differing noise levels and data loss; the BIT algorithm is not robust for added noise and also struggles with high levels of data loss.

The BIT algorithm differs from the others mentioned here as it uses data from both eyes to detect fixations. Although we do not take this approach in this research, it could be a direction for further research. An important finding from the paper by van der Lans et al. that we do use is that they show the importance of different fixation thresholds to detect correct fixations for different individuals and samples. Our algorithm performs this individual-specific approach by tuning the dispersion threshold *eps* separately for every eye-tracking sample.

Chapter 3

Data

For this research, we use two data sets. The first is collected by Larsson et al. (2013), and events are hand-labelled by Richard Andersson and Marcus Nyström. We use the hand-labelled values as a ground truth, with which we evaluate the performance of our algorithms. These hand-labelled values are imperfect, as hand-labellers do not have perfect agreement and make mistakes. However, there are no formally agreed upon guidelines for the identification of eye-tracking events and thus the labelled values by humans experienced in the field of eye-tracking are the best possible benchmark. Friedman et al.(2023) evaluate the inter-rater agreement in hand-labelling. They find that the inter-rater agreement is higher for high-quality data. This increases the usefulness of the hand-labelled values by Richard Andersson and Marcus Nyström, as the quality of this eye-tracking data is high and thus there is a higher consistency between the hand-labelled values than if the data was of a lower quality. It is beneficial that the data is collected and labelled by others as it ensures we do not unfairly favour our algorithm through our subjective perception of fixations. It is publicly available at https://github.com/richardandersson/EyeMovementDetectorEvaluation, allowing for the replication of our results by others.

The second data set is recorded by Martinovici (2019); it has not been hand-labelled. It is not available publicly; however, its role in this research is relatively small compared to the role of the data by Larsson et al..

3.1 Data by Larsson et al.

Larsson et al. collect the eye-tracking data from 17 students at Lund University, Sweden and select a subset of it blindly from the larger set. Amongst the used stimuli are photographs; these are static stimuli, and participants received no specific instructions when viewing them. There are fourteen eye-tracking samples for these photographs, which are all roughly ten seconds in length; however, twelve have a frequency of 500Hz and two have a frequency of 200Hz. This is not discussed or mentioned by Andersson et al. (2017), however Friedman (2020) makes the same finding.

Stimuli are displayed on a screen with a resolution of 1024 by 768 pixels and a dimension of 380mm by 300mm. The setup to record the data is a tower-mounted Hi-Speed 1250 system from SensoMotoric Instruments GmbH, which uses a chin- and forehead rest to minimise head movements. It captures eye-tracking data at a frequency of 500Hz with participants' eyes located 670mm from the screen. The default "bilateral filter" (SensoMotoricInstruments, 2009) for this system is used; it "preserves the edges of large changes in the signal while averaging small changes caused by noise" and does not introduce latency in the signal. The average gaze offset of the data is 0.4° according to a 4-point validation procedure, and the precision of the data is 0.03° RMSD. Precision is estimated by calculating the root-mean-square deviation (RMSD) of samples that both human labellers identify as fixations. The angular precision of a set of n samples can be calculated as $\theta_{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\theta_i^2}$, where θ is the angular distance between samples. The recorded eye-tracking variables are listed and explained in Table 3.1.

Table 3.1: The recorded eye-tracking variables by Larsson et al.

Variable	Explanation
Timestamp	The timestamp of the measurement in microseconds
Horizontal pupil diameter	The horizontal pupil diameter in millimetres
Vertical pupil diameter	The vertical pupil diameter in millimetres
x-coordinate gaze	The x gaze location on the screen of the right eye, in pixel coordinates
y-coordinate gaze	The y gaze location on the screen of the right eye, in pixel coordinates

The eye-tracking data is hand labelled by two people with 10 and 11 years of experience in the field of eye movements. Coder MN (Marcus Nyström) has worked on the compression of videos and evaluated them using eye-tracking, and he has also been involved in the design of the Nyström & Holmqvist (NH) algorithm. Coder RA (Richard Andersson) has a background in psycholinguistics and has not designed any eye-tracking algorithms. They are recognised experts in eye movement classification (Friedman et al., 2023). The coders agree to identify the following events: fixations, saccades, post-saccadic oscillations, smooth pursuits, blinks and undefined. The labelling is done using only data from the right eye, as hand labelling is a time-consuming process and labelling more samples was prioritised over labelling a second eye. Furthermore, the coders are not informed of the kind of stimulus used for a particular data stream to ensure they do not have more information than the algorithms. An example of a photographic stimulus with overlaying eye-tracking data labelled by Richard Andersson is shown in Figure 3.1. Points that Richard Andersson labels as fixations are red; the remaining points are blue.

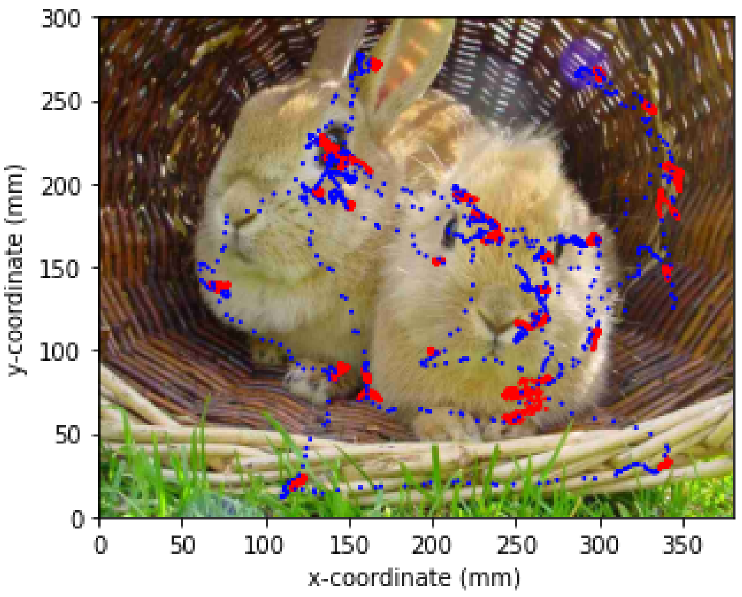


Figure 3.1: An example of a photograph stimulus from Larsson et al., with eye-tracking data overlaid. Points labelled as fixations by Richard Andersson are displayed in red; other points are displayed in blue.

In the subset of randomly sampled eye-tracking data for photographic stimuli, Marcus Nyström detects 380 fixations data with a mean duration of 248ms and a standard deviation of 271ms. Richard Andersson detects 369 fixations with a mean length of 242ms and a standard deviation of 273ms. We note other empirical human parameters in Table 3.2; these parameters are used to simulate human decision criteria in algorithms. For the fixation durations, there is a very small subset of fixations with durations between 8 and 54ms, with the nearest following fixation duration at 82ms. Therefore Andersson et al. (2017) choose to use a value of 55ms as the empirical human parameter for minimum fixation duration.

Table 3.2: The empirical human parameters from the hand-labelled values by Marcus Nyström and Richard Andersson. The used value for minimum fixation duration is chosen differently from the minimum value after inspecting the distribution of fixation durations.

Parameter	Min	Max	Used
Minimum fixation duration (ms)	8	4428	55
Minimum saccade velocity (°s)	45.4	1096	45.4
Minimum saccade duration (ms)	8	22	8

Lastly, we note two properties of the coders that we consider when creating our algorithms. The first is that both coders label fixations outside the screen's dimensions. Some of these fixations are marginally outside the screen and could be due to a gaze offset in the sample. There are however, also fixations detected far beyond these boundaries. The algorithms we construct can thus also detect fixations outside the screen. The second property is that when the coders see an erroneous measurement within a fixation, they label this point to be part of the fixation. The algorithms we construct thus do the same.

3.2 Data by Martinovici

Martinovici collects eye-tracking data from 443 students at a large university in the Netherlands. The stimuli are static and are slides with information for different brands on specific types of products. Participants have the task to choose a brand based on a certain goal, either the most environmentally friendly or the highest performance. The tasks contain an exercise with and without time pressure, two decision goals, and two brand orders.

The study uses a Tobii T60XL eye-tracker containing a 24 inch screen with a 1920 by 1200 pixels resolution and collects data at a rate of 60Hz. Participants are seated such that the centre of the screen is on the same level as their eyes, and the distance from the eyes to the screen is approximately 60cm. The mean distance to the screen between samples, however varies between 515mm and 745mm and thus we account for this when converting *eps* values from mm to °. There is no information on the precision and accuracy of the data; however, the manual notes an accuracy of 0.5° and a precision of 0.35° RMSD (TobiiTechnology, 2009). We do not use all recorded eye-tracking variables; we list those we do use in Table 3.3. The data contains measurements for both eyes, but to be consistent with the methodology from Andersson et al.(2017), we only use data from the right eye.

Table 3.3: The eye-tracking variables that we use from Martinovici's data.

Variable	Explanation
RecordingTimestamp	The timestamp of the measurement in milliseconds
ParticipantName	A value that is combined with RecordingName, to split up files into subsets of separate participants
RecordingName	A value that is combined with ParticipantName, to split up files into subsets of separate participants
GazePointRightX (ACDSpx)	The x gaze location on the screen of the right eye in pixel coordinates, measured from the left side of the screen
GazePointRightY (ACDSpx)	The y gaze location on the screen of the right eye in pixel coordinates measured from the top of the screen

In this research, we consider the data from one randomly selected file instead of all the samples, the only choice we make is that the data must be from a task without time pressure to get as many data points as possible for each individual. We use the file "2016 Nov Consumer preferences_Consumer preferences 1_Slide 9_noTP_ABCD_.jpg.tsv"; the stimulus and overlaid eye-tracking data from a participant is shown in Figure 3.2 . In total, the file contains eye-tracking data from seventy-three individuals, we exclude four of these as more than 25% of their data is missing. When this much data is missing, questions can be raised about the usefulness of performing analysis.

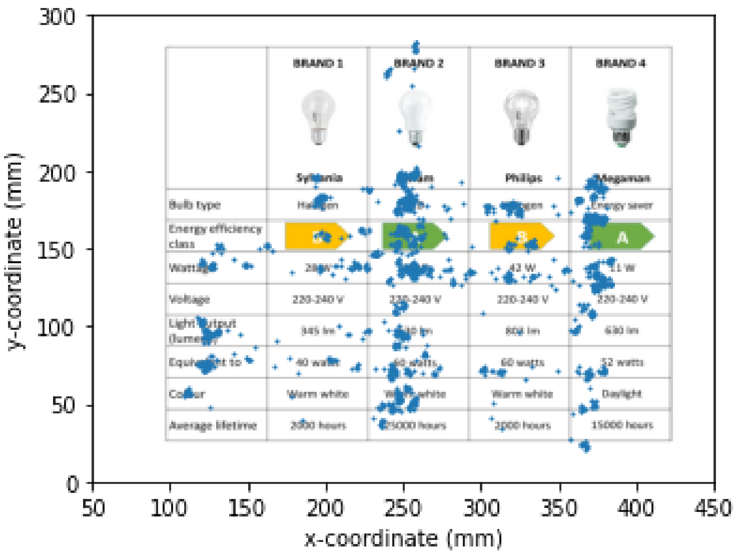


Figure 3.2: An example of a stimulus from Martinovici, with eye-tracking data overlaid.

Chapter 4

Methodology

In this chapter, we discuss the methodology for this research. Section 4.1 considers the three modified versions of DBSCAN, the tuning of parameters and its pseudocode. Section 4.2 discusses the aspects of lower-quality data that we consider in the scope of robustness and section 4.3 discusses the methods to evaluate the algorithms' outputs.

4.1 Modified DBSCAN

DBSCAN by Ester et al. (1996) is a clustering algorithm that constructs clusters of arbitrary shape, labels points that are not assigned to clusters as noise and does not require the number of clusters to be predefined. In the case of eye-tracking data, clusters can be considered fixations, while the noise label covers the other eye-tracking events. To consider the temporal aspect of eye-tracking data, we modify the algorithm. We present the modified algorithms below.

We first define the data form. The eye-tracking data must, at minimum, be of the form (t, x_{gaze}, y_{gaze}) to detect fixations.

Definition 1: A sample of eye-tracking data G is a list of points $[g_0 = (t_0, x_{0,gaze}, y_{0,gaze}), ..., g_{n-1} = (t_{n-1}, x_{n-1,gaze}, y_{n-1,gaze})]$. It must hold that $t_0 < t_1 < ... < t_{n-1}$, and the points must be equally spaced in time.

Next, we define the neighbourhood of a point as the set of consecutive points that contain the point itself and are located within a set dispersion threshold *eps*. The consecutive order requirement is loosened by allowing gaps in the sequence if they do not total to more than 8ms for DBSCAN-ROBUST. Firstly, this adjustment for DBSCAN-ROBUST allows for missing data and erroneous measurements as long as they do not take more time than the minimum duration of a saccade, the shortest eye movement event. This value is taken as then the points exceeding the dispersion threshold *eps* may be a separate eye-tracking event. Secondly, this adjustment allows for a noisy measurement without unjustly cutting off a neighbourhood. For DBSCAN-CONSECUTIVE, we do not loosen the consecutive order requirement, and thus neighbourhoods are split if there are gaps in the sequence.

Definition 2: The neighbourhood of a point g_i, denoted by $Neigh(g_i)$, is the collection of points g_j for which $dist(g_i, g_j) <= eps$ and $j = i - m, ..., i + k$. Not every point between $i - m$ and $i + k$ has to be included, and gaps equivalent to 8ms are allowed for DBSCAN-ROBUST; for DBSCAN-CONSECUTIVE every point between $i - m$ and $i + k$ has to be included. In our application of eye-tracking data, a point is part of its own neighbourhood. Note that a gap of one data point is always permitted for DBSCAN-ROBUST, regardless of eye-tracker frequency.

When the number of points from the starting to the end point time of a neighbourhood is equal to or greater than a defined threshold $MinPts$, a point is labelled as a core point. The threshold for this algorithm is set to a neighbourhood time length of 55ms or more, matching the minimum fixation duration in Chapter 3. Note that points not part of the neighbourhood, but between $(i-m)$ and $(i+k)$, are counted toward the $MinPts$ threshold for DBSCAN-ROBUST.

Definition 3: An eye-tracking point g_i is a core point if the condition $((i+k)-(i-m)+1) >= MinPts$ holds for its neighbourhood. Here $(i-m)$ is the starting and $(i+k)$ is the end point of the neighbourhood.

Points within the neighbourhood of a core point are all part of the same cluster as the core point; this includes erroneous or noisy measurements that fall within the starting and end point of the neighbourhood. This cluster can be expanded further if core points are present within the neighbourhood of the original core point. This process repeats sequentially. Points in the clusters of core points that are not core points themselves, are boundary points.

Definition 4: An eye-tracking point is a boundary point if it is in the neighbourhood of a core point, but the condition $((i+k)-(i-m)+1) < MinPts$ holds for its own neighbourhood.

After all points are evaluated, multiple clusters can be formed. Points that are not part of any cluster are labelled as noise points. Noise points is a collection term for all eye-tracking events other than fixations here.

Definition 5: Let the clusters (fixations) $C_1, ..., C_k$ be formed in the eye-tracking sample G, based on the values for *eps* and $MinPts$. Noise points are the points for which the condition $\forall i : p \notin C_i$ holds.

Figure 4.1 illustrates the combination of definitions presented above.

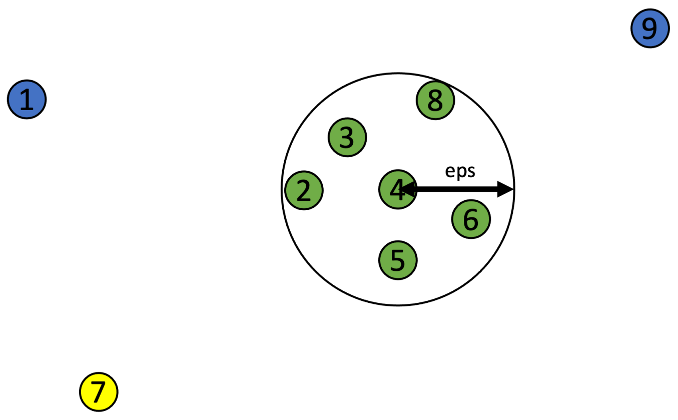


Figure 4.1: This figure illustrates how the DBSCAN-ROBUST algorithm works. Eye-tracking point 4 is a core point as there are more than $MinPts(=5)$ points from its starting to endpoint. The points in its neighbourhood (in green) are given the same cluster label as point 4. Point 7, which represents an erroneous measurement or very noisy point, is assigned the same cluster label but is not part of the neighbourhood. Points 1 and 9 are not part of the neighbourhood and do not qualify as potential erroneous measurements between points of the neighbourhood; thus they are assigned a noise label. In the case of DBSCAN-CONSECUTIVE, point 7 leads to a split in the neighbourhood of points around point 4; thus points 7 and 8 are not part of the same cluster as point 4.

4.1.1 Tuning the Dispersion Threshold

The clusters, equivalent to fixations for eye-tracking data, depend on the values for $MinPts$ and *eps*. Lower $MinPts$ or higher *eps* values can lead to more points being classified as parts of clusters; as we relax the requirements for core points and neighbourhoods can increase. The opposite is true for higher $MinPts$ or lower *eps* values.

It is clear that different pairs of $MinPts$ and *eps* can lead to similar results. Thus two approaches are possible: either we fix the value for *eps* and tune the value for $MinPts$, or we fix the value for $MinPts$ and tune the value for *eps*. Fixing an *eps* value can be linked to the fovea and the degrees of view a person can see with high clarity, as discussed in Chapter 2. Fixing $MinPts$, on the other hand, can be linked to the minimum fixation duration. We consider both options, however fixing a value for *eps* across all samples is not possible as their optimal *eps* values for a range of $MinPts$ values equivalent from 50ms to 400ms, did not all show overlap. Thus an approach is chosen to fix $MinPts$ to the equivalent of 55ms, the empirical human parameter for minimum fixation duration, and tune *eps* separately for each sample. We do this to allow for heterogeneity between individuals and tasks; earlier work by van der Lans et al. (2011) shows the need for this approach. The method to tune *eps* is similar to that of Yu et al. (2016) and is the same for DBSCAN-ROBUST and DBSCAN-CONSECUTIVE.

For each eye-tracking point, we find all other eye-tracking points within a timespan of 200ms on either side and calculate their distance to the original point. We sort the distances from small to large and note the distance of the $(MinPts - 1)$ nearest point. The timespan value of 200ms is chosen to allow for a blink to occur and there still be enough eye-tracking points to achieve $MinPts$ total points, whilst also ensuring that the points around the point are relevant to it temporally and could be part of a fixation together. Larger values for the timespan lead to the distances of points being considered that are less likely to be part of a fixation with the original point. We then plot the proportion of cases where the distance to the $(MinPts - 1)$ nearest point exceeds a given threshold against the corresponding thresholds. An example for one of the samples is shown in Figure 4.2 .

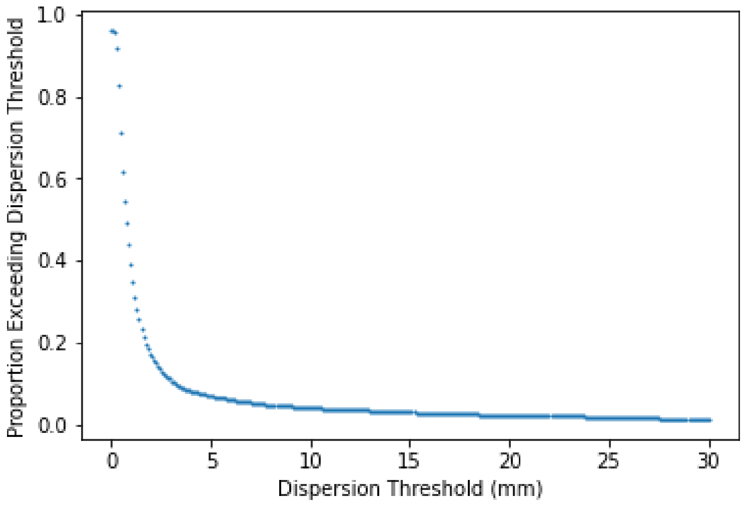


Figure 4.2: The proportion of points for which the $(MinPts - 1)$ nearest point exceeds varying dispersion thresholds.

In Figure 4.2, we see an elbow plot where the number of points exceeding an increasing threshold decreases rapidly for the lower threshold values, while this rate slows for higher thresholds. When the frequency of exceeding a threshold decreases rapidly, a relatively large portion of points remain for which the excess is strongly dependent on the threshold. When the frequency of exceeding a threshold decreases slowly, the excesses for the remaining points are not strongly dependent on the threshold. The logic is used is that these latter points have few points nearby and are thus saccades, while the former points have many points nearby and are thus fixations. We postulate that the elbow point of this plot is the optimum dispersion threshold that allows us to find the least dense cluster.

We find the elbow point using the Kneedle algorithm by Satopaa, Albrecht, Irwin and Raghavan (2011). It is an algorithm designed specifically for finding elbow points in plots of discrete data; this makes it suitable for this application. There are other options available, however, assessing performance for a variety of them goes beyond the scope of this paper. We choose this method as it outperforms Angle-based, Menger and EWMA algorithms on synthetic datasets where the knees are known. It fits a smoothing spline through the set of points and finds the point where the smoothed spline has the largest difference from the line going through (x_{min}, y_{min}) and (x_{max}, y_{max}). Figure 4.3 shows an illustration of the process. For settings, we use the polynomial interpolation method with its standard degree of 7, we set the curve to be convex with decreasing direction, and we set the Online setting to True. This last parameter allows the algorithm to find new knees instead of stopping once it has found one; the offline method finds the knee on the initial flat part ($eps = 0$) of the curve for some samples and is thus not robust.

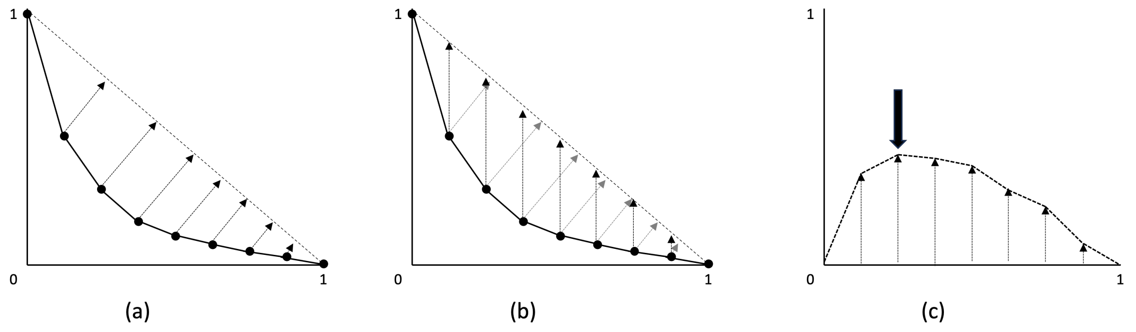


Figure 4.3: This figure illustrates the online elbow point detection method for the Kneedle algorithm. (a) shows the smoothed data with the dashed bars representing the perpendicular distance between the smoothed spline and the line y=x. (b) shows the smoothed data, with the dashed lines rotated 45 degrees; these lines are the differences used in the elbow point detection. (c) plots these difference values and shows that the elbow point is detected at the third data point.

Figure 4.4 plots a line through the data points in Figure 4.2 and includes a dashed line indicating where the Kneedle algorithm detects the elbow point.

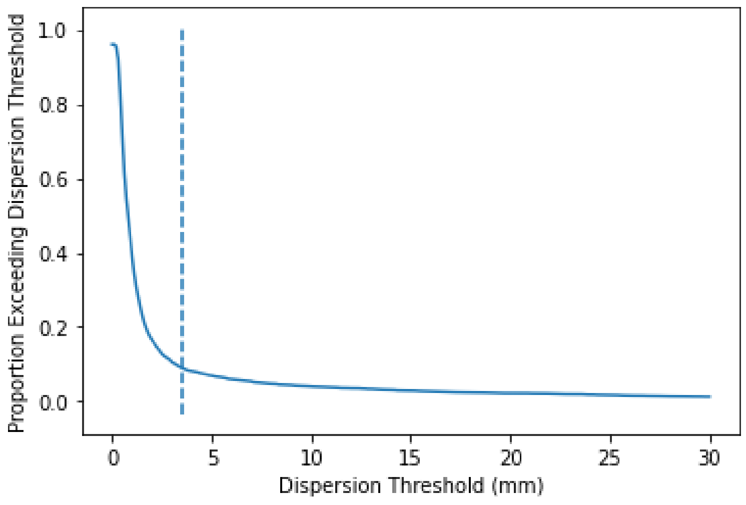


Figure 4.4: A line through the data points in Figure 4.2, together with a dashed vertical line representing the elbow point found by the Kneedle algorithm.

4.1.2 Algorithm Pseudocode

This section presents the pseudocode for the DBSCAN-ROBUST algorithm. First, we identify the inputs and outputs of the algorithm. Then we note the RetrieveNeighbours function, which finds the neighbourhood of a point according to Definition 2 in Section 4.1; the cutoffs ensure that there are no gaps in the neighbourhood larger than 8ms. The next section tunes the dispersion threshold for the value of $MinPts$ corresponding to the fixed value of 55ms, as described in Section 4.1.1. The last step of the algorithm assigns points to clusters using the tuned dispersion threshold *eps* and the RetrieveNeighbours function. This is done according to the steps in Section 4.1, where we exclude missing points and blinks from the core points check; these are noted as $(t_j, 0, 0)$ in the data. Missing points can be labelled as part of a cluster if their duration is less than 8ms; for blinks this is not possible as blinks take longer than 8ms. All points start off with the label 0 and noise-value 0. It must be noted that this pseudocode is a simplified version of the Python code; the Python code with comments can be found in Appendix C. To obtain DBSCAN-CONSECUTIVE one sets the CutoffStep to 0, this enforces strict consecutivity in neighbourhoods.

Algorithm 1 DBSCAN-ROBUST(G)

Inputs

- G: The eye-tracking data points, where $G = \{g_1, g_2, ..., g_n\}$ and $g_i = (t_i, x_i, y_i)$

Outputs

- C: collection of fixations clusters

```
*Calculating the sample frequency*
$f$ = round(1/(($t_{11} - t_1$)/10)))

*The RetrieveNeighbours function*
function RetrieveNeighbours(G, i, eps, f)
    Neighbours = [ ]
    for j in 1:n do
        if $Distance(i, j) \leq eps$ & $|t_j - t_i| \leq 0.2s$ then
            Neighbours.append(j)
        end if
    end for
    CutoffStep = RoundUpwards(8ms*f)
    Cutoffs = [ ]
    for k in range(length(Neighbours)-1) do
        if Neigbours(k+1)-Neighbors(k) > (CutoffStep+1) then
            Cutoffs.append(Neighbours(k))
        end if
    end for
    UpperBoundCoordinate = FindFirstAbove(Cutoffs, i)
    LowerBoundCoordinate = FindFirstBelow(Cutoffs, i)
    Neighbours = G[LowerBoundCoordinate+1, UpperBoundCoordinate]
return Neighbours
end function

*Tuning the dispersion threshold*
MinPts = RoundDownwards(55ms*f)
for j in 1:n do
    TimeRelevantPoints = [ ]
    for k in 1:n do
        if $|t_j - t_k| \leq 0.2s$ then
            TimeRelevantPoints.append(k)
        end if
    end for
    TimeRelevantPoints.sort()                              ▷ by distance to point j
    $\text{Dist}_j = \text{Dist}(j, TimeRelevantPoints(MinPts))$
end for
*Plot the frequencies of local dispersions exceeding a variable threshold (eps). The optimal value
for eps is the elbow point in this plot and found using the the Kneedle algorithm by Satopaa
et al..*

*Assigning the clusters*
ClusterNumber = 0
for i in 1:length(T) do
    if $\text{label}_i$ == 0 & $\text{noise}_i$ == 0 then
        Neighbourhood = RetrieveNeighbours(G, i, eps, f)
        if (max(Neighborhood) - min(Neighbourhood)+1) < $MinPts$ then
            $\text{noise}_i = 1$
        else
            ClusterNumber += 1
            $\text{label}_i$ = ClusterNumber
            PotentialCorePointsCheck = [ ]
            for point in Neighbourhood do
                $\text{label}_{point}$ = ClusterNumber
                if point > $i$ then
                    PotentialCorePointsCheck.append(point)
                end if
            end for
            while PotentialCorePointsCheck.isnotempty() do
                point = PotentialCorePointsCheck.pop()
                Neighbourhood = RetrieveNeighbours(G, point, eps, f)
                if (max(Neighborhood) - min(Neighbourhood)+1) >= $MinPts$ then
                    for For point2 in Neighbourhood do
                        $\text{label}_{point2}$ = ClusterNumber
                        if point2 > point then
                            PotentialCorePointsCheck.append(point2)
                        end if
                    end for
                end if
            end while
        end if
    end if
end for
```

4.1.3 Velocity-based Post-processing

DBSCAN-ROBUST and DBSCAN-CONSECUTIVE are density-based algorithms. We consider a velocity-based post-processing step as an extension to DBSCAN-ROBUST to create DBSCAN-VELOCITY. We take the labels DBSCAN-ROBUST assigns to eye-tracking points, identify the different fixations and analyze the velocities of points at the start and end of fixations. If the velocities of these points exceed a given threshold, we remove them from the fixation.

The reasoning for this post-processing step is that at the start and end of the labelled fixations by DBSCAN-ROBUST, there are potentially points that are not part of the fixation but are labelled as such as they fall within the dispersion threshold of one of the core points. We attempt to decrease this flaw of dispersion-based clustering algorithms by adding a velocity-based step. We take points totalling 20ms at each fixation's start and end and calculate these points' velocity with $v_i = \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{t_i - t_{i-1}}$. For points considered as erroneous measurements due to either missing data or being too noisy, we set their velocity to 0 and calculate the velocity for the point that follows them using the first point before it that is not considered an erroneous measurement. This ensures that erroneous measurements do not unjustly cut off fixations due to this post-processing step, as not accounting for these erroneous measurements leads to incorrect velocities. We then check the point with the highest index for the start of the fixation and the point with the lowest index for the end of the fixation first and move respectively down and up in index until we find a point with a velocity that exceeds the threshold. That point and all points after it are cut off from the fixation. An illustration of the process is shown in Figure 4.5

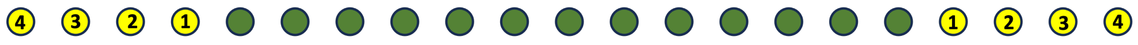


Figure 4.5: This figure illustrates the velocity-based post-processing process. The points totalling to 20ms at the start and end of each fixation are checked for their velocities. These points are shown in yellow here. Each yellow point has a number, which indicates the sequence in which they are checked. When a point for which the velocity is too high is found, it and all points with a higher index are excluded from its end of the fixation.

The velocity threshold is set to the empirical human parameter of 45.4°/s. It can be converted to a speed in mm/s with $v_{mm} = \frac{\pi * R * v_{angl}}{180}$, where v_{mm} is the velocity in mm/s, R is the distance to the screen in mm and v_{angl} is the velocity in °/s. The choice for a span of 20ms is based on the hypothetical situation where a fixation stops at one end of the dispersion threshold circle around a core point, and the saccade that follows it ends at the opposite side of the circle whilst travelling at the minimum speed of saccades. This leads to the maximum possible duration in time. When we assume a dispersion threshold of 0.6°, which is near the upper end of values we see in practice and a velocity of 45.4°/s, this returns a maximum duration of 26ms. We round this down to 20ms, as we have taken the longest possible path for the saccade and the lowest possible velocity. If one notes that their dispersion thresholds differ greatly from 0.6°/s, this part of the algorithm should be revised.

4.2 Robustness

The quality of eye-tracking data varies due to measurement apparatuses and conditions. Less advanced measurement systems and worse measurement conditions can lead to lower quality and frequency data. Typical properties of lower-quality data are data loss, decreased data precision and biased measurements as noted by Holmqvist, Nyström and Mulvey (2012). Brand et al. (2021) evaluate the data quality of the Gazepoint GP3 eye-tracker when used by different study participants in non-laboratory conditions. Participants receive training on using the tracker and must then perform sampling independently a week later.

We worsen the high-quality data by Andersson et al. by simulating the data properties found by Brand et al.. We first apply the inaccuracies for different aspects separately to analyze their individual effects and thereafter combine them to simulate real-world data of lower quality. We simulate the data quality properties for this eye-tracker as it is cheap ($845) compared to other eye-trackers and there is research that investigates the quality of its recordings.

4.2.1 Lowered Frequency

The frequency of the GP3 eye-tracker is 60Hz. To lower the frequency of the data by Larsson et al., we take $\frac{1}{8}$ of the points in the data. This leads to new data with a frequency of 62.5Hz. This is not exactly equal to 60Hz, however, this method does not require interpolation of points and allows us to get very close to the frequency of the GP3 eye-tracker.

4.2.2 Data Loss

Data loss can occur for several reasons; examples are faulty measurements or conditions that prevent the eye-tracker from observing the eye well. Data loss is simulated by removing a percentage of points randomly in a uniform manner over the eye samples. Research by Brand et al. recorded 3% of data to be missing. In this research, we randomly replace 3% of the points in samples by Larsson et al. to simulate this, as almost no data points were labelled as data loss. This trial is done once. We note that this method of adding data loss is unrealistic if we expect data loss to occur in bunched groups.

4.2.3 Precision and Accuracy

Figure 4.6 displays the relation and difference between precision and accuracy. The centres of the circles are the true locations of the eye-tracking point, and the red dots are different realisations from the eye-tracker. A high accuracy means the average location of the different realisations is equal to the true location of the points; a low accuracy means the average location of the different realisations is not equal to the true location and thus has an offset. A high precision means the different realisations are closely bunched together, and a low precision means the different realisations are spread out more.

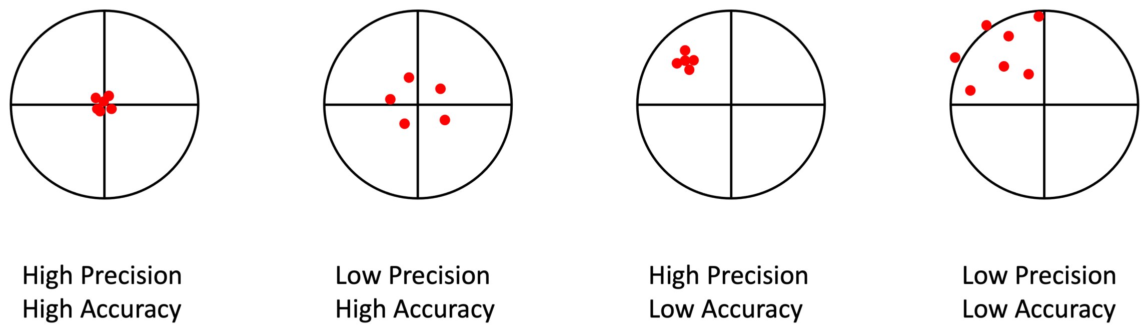


Figure 4.6: This figure displays the relation and difference between accuracy and precision. The centres of the circles are the true location of the eye gaze.

The research by Brand et al. (2021) returns average horizontal and vertical accuracy and precision values across participants for different dot locations on the screen. We square these values for each dot separately and take the combination of horizontal and vertical accuracy or precision with the highest combined squared value to simulate the worst data quality. The worst combination of horizontal and vertical accuracy is 1.05° and 0.65°. To emulate this, we shift all eye-tracker recordings 1.05° to the left and 0.65° down; other directions are also possible, but changing this does not have an effect as long as the algorithms can detect fixations outside the screen. This is because the relative locations of points do not change, but points can be moved out of the screen's dimensions. The least precise combination of horizontal and vertical precision is 0.8° and 1.13 °; we find this by squaring and summing horizontal and vertical precision for all combinations by Brand et al. and choosing the combination for which the summed squared values are maximized. These precision values are obtained by finding the ellipse within which 95% of points around the fixation location are located, where the width and height of the ellipses represent the horizontal and vertical precision. From literature, we know that, assuming multivariate normality, the width of a 95% ellipse is equal to $2\sigma_x\sqrt{5.991}$, and the height is equal to $2\sigma_y\sqrt{5.991}$. Here we use that $chi^2(df = 2) = 5.991$, for a p value of 0.05. This leads to a value of 0.163° for σ_x and a value of 0.231° for σ_y. We use these standard deviations to generate noise for each point individually.

This method of adding a bias and randomised noise to measurements leads to data of worse quality than that of the GP3 eye-tracker as there is already a bias and noise present in the data by Larsson et al..

4.3 Evaluation

We evaluate DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY in three ways. The first is how similar they are to the human experts in labelling fixations in the data by Larsson et al.. This tells us how well they classify fixations compared to other algorithms when considering human experts as the ground truth and allows us to uncover their tendencies in classifying fixations. The second is how lower data quality affects the labelling of fixations, as described in Section 4.2. Thirdly, we detect fixations in the data by Martinovici to see if the mean duration and standard deviation of fixations are within acceptable ranges. This tells us whether the algorithms lead to plausible results in other data for which we do not know the empirical human parameters.

4.3.1 Similarity to Human Experts

We evaluate the similarity to humans in the same way as in the paper by Andersson et al., to allow for comparison between our algorithms and the others they consider in their research. The first way is to compare the properties of the fixations detected by the algorithms to that of the hand labellers. The mean fixation duration, the standard deviation of the fixation durations and the number of fixations are combined into a single measurement, the root-mean-squared deviation (RMSD). To calculate the RMSD, we first re-scale all three variables to a [0, 1] range, using the minimum and maximum values for each variable across all algorithms. We enter these values into a matrix of k rows and l columns; the rows are the different humans and algorithms and the columns are the normalised fixation duration variables.

We then separate the matrix M into a matrix A for the algorithms and a matrix H for the humans. The summed RMSD for $algorithm_i$ is then calculated by

$$RMSD_i = \sum_{\forall j} \sqrt{(A_{ij} - \sum_m \frac{H_{mj}}{h})^2}. \tag{4.1}$$

Here i is the index of the algorithm, j is the index of the different event duration variables, m is the index of the human labellers, and h is the number of human experts. The algorithm with event properties most similar to human experts is the algorithm with the lowest RMSD value. These values are relative, do not have an absolute meaning and cannot be compared between data sets.

The second method to compare similarity to human labellers is the comparison on a point-by-point basis. This ensures that we not only consider the properties of detected fixations but also that eye-tracking points labelled as fixations by the humans are also labelled as fixations by the algorithm. We do this using Cohen's Kappa(Cohen, 1960), Warrens(2008) proves that it is equivalent to the Hubert-Arabie adjusted Rand index. The metric is calculated by

$$K = \frac{P_o - P_c}{1 - P_c}, \tag{4.2}$$

where P_o is the observed proportion of agreement between two labellers (algorithm or human) for all points and P_c is the proportion of chance agreement between the two labellers given their proportion of labelling fixations. P_c is calculated through $P_c = \frac{1}{N^2} \sum_k n_{k1} * n_{k2}$, where N is the total amount of observations, k are the predicted categories (0 (not a fixation) or 1 (fixation) in this case), and n_{ki} is the number of times labeller i labelled category k. K ranges from -1 to 1; negative numbers indicate that the observed agreement is worse than the chance agreement, zero indicates that the two perform identically, and values above 0 indicate that the labellers agree more than chance. The maximum value of 1 occurs when the two coders agree on all points. The higher this value, the more the algorithm and humans agree on a point-by-point basis. We compare algorithms against both humans and average the two Kappa scores, the humans are compared against the other human and themselves and these scores are also averaged.

Cohen(1960) suggests that Kappa values can be interpreted as: $\leq$0 no agreement, 0.01 to

0.20 no to slight agreement, 0.21-0.40 fair agreement, 0.41-0.60 moderate agreement, 0.61-0.80 substantial agreement and 0.81-1.00 almost perfect agreement.

Statistical Analysis

Andersson et al. (2017) paste the labels for all fourteen samples into one vector for each algorithm and human and then calculate the Kappa scores between these vectors of labels. We replicate this to allow comparison of our algorithms to those in their paper. We also add a statistical analysis, to determine whether an algorithm outperforms other algorithms for the individual ten second samples.

For this, we calculate the performances of the algorithms on the twelve individual 500Hz samples. This is only possible for the algorithms we run ourselves; the DBSCAN algorithms and the BIT algorithm. For each algorithm we obtain twelve kappa scores and compare these through a paired test. A paired test calculates the differences between the kappa scores for each of the twelve samples and tests a null hypothesis using these difference values. This comparison is done for the similarity to the benchmark of the two human labellers on unaltered data, and for the robustness in labelling to different methods of lowering the data quality.

The suitable test depends on the assumptions that are made. The most strict assumption is that the differences are normally distributed. Plotting the difference values revealed mostly non-normal distributions. A less strict assumption is that the differences are symmetrically distributed. However, the plotted difference values do not reveal symmetric distributions either. In this case, the paired samples sign test, also known as the sign test, is used. For a pair of, in this case 12, observations, it determines whether one member of the pair tends to be greather than the other. The null hypothesis is that the difference between a paired draw of X and Y, where X and Y are Kappa scores for two different algorithms in this case, has a median of zero. The alternative hypothesis is one-sided; the difference between a paired draw of X and Y, has a median larger than zero in favor of X. Here X is chosen as the Kappa scores from the outperforming algorithm in the comparison.

If the null hypothesis is true, we expect that for the set of twelve differences, six are positive and six are negative. This leads to a binomial distribution with P=0.5. To test our null, we first count the number of positive differences, n_{pos}, and the number of negative differences, n_{neg}. We take the smallest of the two, k, and calculate the probability of this outcome or a more extreme one occurring

$$p = \sum_{i=0}^{k} Pr(Outcome = i) = \sum_{i=0}^{k} \binom{n}{i} * 0.5^{i} * (1-0.5)^{12-i} = \sum_{i=0}^{k} \binom{n}{i} * 0.5^{12}. \tag{4.3}$$

In this research, we assume a significance level of α=0.025.

The Kappa scores in Chapter 5 are for the case where all labels for the separate ten second samples are pasted into one vector and the vectors are compared between labellers. The Kappa scores for each individual sample are listed for reference and reproduce-ability in Appendix B, in Chapter 5 we solely state the resulting p-values from the sign tests.

4.3.2 Robustness

We worsen the data quality and rerun the DBSCAN algorithms to detect fixations. We first worsen the four different aspects separately to see their individual effects and then combine them to simulate data from a GP3 eye-tracker.

To judge robustness, we assess the stability of assigning labels when the data quality is lowered. Hessels, Niehorster, Kemner and Hooge (2017b) perform a similar lowering of data quality and check how the fixation duration properties change. We do this, and we calculate Cohen's Kappa between the labels on the original data and the new labels on the data for which the quality has been lowered. This approach checks the robustness of the fixation duration properties and whether the individual labels remain the same.

4.3.3 Non-labelled Data

We detect fixations in the samples by Martinovici using the three DBSCAN algorithms. It is not possible to do an evaluation as in Section 4.3.1 because the eye-tracking points have not been assigned labels. Therefore, we judge the plausibility of the fixation duration properties and use this to conclude whether the algorithm leads to plausible results for data in which we do not know the empirical human parameters. We can apply a robustness check by worsening the data in the same way we did for the data by Larsson et al., however that is not the goal of this part of the methodology.

The criterion we judge is whether the fixations' mean duration is similar to those in the data by Larsson et al.. Both stimuli are namely static, and thus we can expect similar fixation properties. It must, however be noted that the stimuli by Martinovici contain text, and participants had specific instructions when viewing the stimuli; this is not the case for the data by Larsson et al..

This method does not lead to a test statistic or performance metric, and thus no strong conclusions can be taken. We answer whether the algorithms return plausible results on data other than that from which it uses empirical human parameters. For this, the evaluation method is sufficient.

Chapter 5

Results

We present the results in three parts. First, we evaluate our algorithms' performance compared to the others in research by Andersson et al. (2017). Secondly, we evaluate how lower-quality data affects fixation labelling by our algorithms and the BIT algorithm. Lastly, we check fixation duration properties on eye-tracking data for which we do not know the empirical human parameters.

5.1 Similarity to Human Experts

We detect fixations in the data by Larsson et al. and calculate the fixation duration properties and Kappa scores of our DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY algorithms and the BIT algorithm. The properties of our algorithms, the BIT algorithm, the other algorithms in the work by Andersson et al. and the hand-labellers, are shown in Table 5.3. Apart from the BIT algorithm, we did not run the algorithms in their paper ourselves. However, our results for the BIT algorithm and the human labellers are identical to those found by Andersson et al., indicating that our scoring methodology aligns with Andersson et al.'s, enabling a comparison of algorithms.

The p-values for the sign test on the differences between Kappa values on individual samples for $CoderMN$, the DBSCAN algorithms and the BIT algorithm are shown in Table 5.2. Values that are significant at a p=0.025 level are indicated with a $*$; in the case of significance, the outperforming labeller is also marked with a *. For all algorithms, there are statistically significant differences between their Kappa scores and those of human labeller $CoderMN$, where $CoderMN$ outperforms the algorithms. The same holds for $CoderRA$, as he has identical Kappa scores to $CoderMN$.

Table 5.1: The fixation duration properties, RMSD values and Kappa scores for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY, the human labellers and the algorithms in the research by Andersson et al.

Algorithm	Mean (ms)	Std Dev (ms)	Number	RMSD	Kappa	Total Fix (ms)
CoderMN	248	271	380	0.02	0.92	94,240
CoderRA	242	273	369	0.02	0.92	89,298
DBSCAN-VEL	267	289	370	0.13	0.77	98,620
NH	258	299	292	0.25	0.52	75,336
DBSCAN-CONS	278	343	364	0.30	0.68	101,378
IMST	304	293	333	0.33	0.38	101,232
DBSCAN-ROB	303	361	342	0.47	0.66	103,814
BIT	209	136	423	0.53	0.67	88,407
IKF	174	239	513	0.56	0.63	89,262
IDT	399	328	242	0.90	0.36	96,558
IHMM	133	216	701	1.08	0.67	93,233
IVT	114	204	827	1.39	0.67	94,278
CDT	397	559	251	1.42	0.38	99,647

Table 5.2: The p-values from the sign test on the differences in paired Kappa scores for the unaltered data between CoderMN(M), DBSCAN-CONSECUTIVE(C), DBSCAN-ROBUST(R), DBSCAN-VELOCITY(V) and BIT(B). Significant values at a p-level of 0.025 are indicated with a * next to the value and a * for the labeller that outperforms.

	M*-V	M*-C	M*-R	M*-B	V*-C	V*-R	V*-B	C-R	C-B*	R-B
p-value	<0.001*	<0.001*	<0.001*	<0.001*	<0.001*	<0.001*	0.003*	0.073	0.019*	0.073

Before we analyze the results in Table 5.3, we first note that the mean fixation duration values and Kappa scores in Table 5.3 are likely flawed as Andersson et al. do not realise two of their fourteen samples are 200Hz instead of 500Hz. This leads to too low mean values for fixation durations, affects output for algorithms that require the frequency of the data to be entered by hand and leads to a comparison through Kappa of sequences that are not monotone in frequency. We refer to Appendix A.2 for more information. It is not possible to fix this issue in the scope of this thesis; thus, we continue with this slightly flawed analysis here and consider all samples to be 500Hz, to be able to compare the results of our algorithms to that of the other algorithms. The Kappa scores for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on only the 500Hz data are 0.66, 0.68, 0.78 and 0.68. DBSCAN-VELOCITY's and BIT's Kappa score increase by 0.01, the others remain the same when rounded to two decimals.

A second thing that must be noted, but is not mentioned by Andersson et al., is that neighbouring fixations are merged when calculating fixation duration properties. This came to light when replicating results for the BIT algorithm. There is merit to this, as the hand-labellers coded fixations using the label 1, and thus neighbouring fixations can also not be distinguished for them. We refer to Appendix A.1 for more information.

The third and last difference we note is that this research leads to different absolute RMSD values for the algorithms in the paper by Andersson et al.. The ranking of the algorithms according to RMSD, however, remains the same. As RMSD is a relative scoring metric, we do not consider this a problem. This difference likely stems from a step in the re-scaling to [0,1] method by Andersson et al. that is not mentioned. See Appendix A.3 for more information.

For most algorithms, their total fixation duration is similar to that of the human coders, but the combinations of mean duration and number of fixations differ. It shows that some tend to merge fixations (CDT for example), while others tend to split up fixations (IHMM for example). The algorithms are sorted by RMSD score; the lower the RMSD score, the better, as it measures the deviation from the mean values of the human coders for the mean fixation duration, the standard deviation of these fixation durations and the number of fixations. The NH algorithm performs best by this metric out of the algorithms in the paper by Andersson et al., but its total fixation duration differs most from the human coders out of all algorithms. This algorithm detects 22% fewer fixations than the mean of the human coders, while its mean fixation duration is only 5% higher. Questions can be raised about whether a metric that does not account for the correlation between the mean duration of fixations and the number of fixations is suitable. A way to account for this could be to include the total fixation duration in the calculation of the RMSD, as this penalises algorithms that do not have a strong inverse correlation between the deviation of the mean fixation duration and the number of fixations compared to the benchmark.

The Kappa score measures agreement between labellers on a point-by-point basis; the higher this score, the better. The NH algorithm ranks seventh for this metric whilst performing best for the RMSD metric. This shows the importance of evaluating algorithms on a point-by-point basis and indicates that NH performs relatively poorly on a point-by-point basis whilst having the most similar fixation duration properties out of the algorithms in the work by Andersson et al. compared to the benchmark.

Figure 5.1 shows the x_{gaze} coordinates for a sample, with bars below indicating detected fixations by CoderMN, DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and the BIT algorithm.

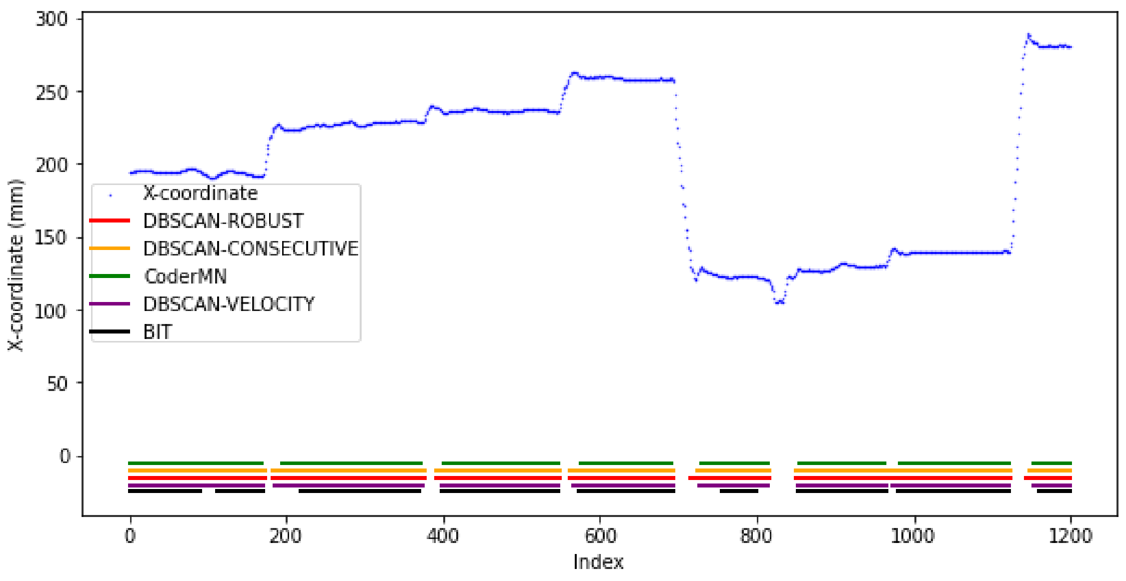


Figure 5.1: The x$_{gaze}$ coordinates and the detected fixations for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY, BIT, and CoderMN, for a sample of eye-tracking data.

5.1.1 DBSCAN-ROBUST

DBSCAN-ROBUST ranks fifth out of the algorithms for RMSD, with a value of 0.47. It detects 9% less fixations than the benchmark and its mean fixation duration is 24% higher. It assigns more points than the benchmark, to be labelled as fixations and either misses or merges some fixations. DBSCAN-ROBUST has a Kappa score of 0.66 and is surpassed on a point-by-point basis by the BIT, IHMM, IVT, DBSCAN-CONS and DBSCAN-VELOCITY algorithms. The p-values for the sign test in table 5.2 show that the differences in Kappa scores on individual samples are statistically significant for DBSCAN-VELOCITY.

In Figure 5.1, DBSCAN-ROBUST, the middle line indicated in red, tends to start fixations too early and end them too late compared to hand-labeller CoderMN. The early start and late end are attributed to how a clustering algorithm works. The points before and after fixations are often saccades or post-saccadic oscillations, according to the human labellers. These points have a higher velocity than expected from points that are part of the fixation. Density-based clustering algorithms, however, do not consider this; they check whether points fall within a certain distance from each other. A second point we notice is that fixations six and seven by the human labeller appear to be merged into one fixation by the DBSCAN-ROBUST algorithm. This is not the case, as these are neighbouring fixations. However, when we do not account for neighbouring fixations, we label this as a merged fixation.

5.1.2 DBSCAN-CONSECUTIVE

DBSCAN-CONSECUTIVE ranks second out of the algorithms regarding RMSD, with a value of 0.30. It detects 3% fewer fixations than the benchmark and its mean fixation duration is 13% higher. It assigns more points than the benchmark to be labelled as fixations and either misses or merges some fixations. DBSCAN-CONSECUTIVE has a Kappa score of 0.68 and is only surpassed on a point-by-point basis by the DBSCAN-VELOCITY algorithm. The p-values for the sign test in table 5.2 show that the differences in Kappa scores on individual samples are statistically significant for DBSCAN-VELOCITY and BIT. This is noteworthy, as the Kappa scores for the entire sample are equal for BIT and DBSCAN-CONSECUTIVE, and is due to BIT scoring higher on more individual samples.

In Figure 5.1, there are similar properties for DBSCAN-CONSECUTIVE, the second line in orange, as DBSCAN-ROBUST, but for some fixations the effect of starting early and ending late is less strong. The reason for this same flaw is the earlier mentioned property of density-based clustering algorithms. The improvement is due to some points at the end of fixations being labelled as erroneous measurements and thus being cut off by DBSCAN-CONSECUTIVE. A second improvement to DBSCAN-ROBUST is that fixation six and seven according to CoderMN, are not merged by DBSCAN-CONSECUTIVE. This is, however, not visible in the graph as there is only one point separating the fixations.

5.1.3 DBSCAN-VELOCITY

We address the shortcomings of an algorithm that is only density-based by introducing the DBSCAN-VELOCITY algorithm. DBSCAN-VELOCITY ranks best out of all the algorithms regarding RMSD with a value of 0.13. It detects 1% fewer fixations than the benchmark, its mean fixation duration is 9% higher and it assigns 7% more points than the benchmark to be labelled as fixations. DBSCAN-VELOCITY also performs best for Kappa, with a score of 0.77. This is a very impressive score as the kappa score between human labellers is 0.84; as they are both part of the benchmark, this becomes 0.92 in Table 5.3. The p-values for the sign test in table 5.2 show that the differences in Kappa scores on individual samples are statistically significant for DBSCAN-VELOCITY compared to DBSCAN-CONSECUTIVE, DBSCAN-ROBUST and BIT, where DBSCAN-VELOCITY is the outperforming algorithm.

In Figure 5.1, there are improvements by DBSCAN-VELOCITY, the fourth line in purple, relative to DBSCAN-ROBUST and DBSCAN-CONSECUTIVE. The flaw of starting fixations early and ending them late is decreased, and the split between fixations six and seven is present and more defined.

5.1.4 BIT

The BIT algorithm follows a similar approach to DBSCAN-ROBUST, where a sequence of candidate fixation points is classified as a fixation if at least a number of subsequent points equalling 55ms are classified as fixation points, where they can be interrupted by a maximum number of consecutive points equalling 6ms. These values stem from the fact that Andersson et al. only adjusted the *min_samples* parameter for the change in frequency and kept *n_lost* at the default value of 3.

The BIT algorithm ranks sixth out of all the algorithms for RMSD, with a value of 0.53. It detects 13% more fixations than the benchmark and its mean fixation duration is 15% lower. It assigns fewer points than the benchmark to be labelled as fixations and likely splits some fixations. It is ranked jointly third in terms of Kappa together with IHMM and IVT, with a score of 0.67. The p-values for the sign test in table 5.2 show that the differences in Kappa scores on individual samples are statistically significant for BIT compared to DBSCAN-CONSECUTIVE, where BIT is the outperforming algorithm.

In Figure 5.1, BIT, the bottom line in black, deviates from the labelling by CoderMN. The first fixation, as labelled by CoderMN, is split into two fixations by the BIT algorithm, confirming that some fixations are split. Furthermore the BIT algorithm cuts some of the fixations short by starting them late or ending them early. These tendencies are the opposite of those of DBSCAN-ROBUST; DBSCAN-ROBUST starts fixations early, ends them late and merges fixations when we do not account for neighbouring fixations.

5.1.5 Intermediate Conclusion

From this analysis, DBSCAN-VELOCITY outperforms all other algorithms for the data by Larsson et al. for both the RMSD measure and the Kappa score. Furthermore, the differences in Kappa scores for individual 500Hz samples between DBSCAN-VELOCITY and the other algorithms are statistically significant This impressive result indicates DBSCAN-VELOCITY should be considered for detecting fixations in high quality eye-tracking data for static stimuli.

This analysis also shows that the BIT algorithm has RMSD and Kappa scores near those of DBSCAN-ROBUST and DBSCAN-CONSECUTIVE. The similarity of RMSD and Kappa scores leads us to add BIT to the robustness comparison in Section 5.2.

5.1.6 Addressing the Use of Empirical Human Parameters

Using empirical human parameters from the same sample for which we assess labelling can lead to unfair advantages for algorithms that use more empirical human parameters as inputs.

A method to decrease this advantage could be splitting up the sample, obtaining empirical human parameters from one sample and judging performance on the other. We choose not do this approach as only twelve samples of ten seconds are available to us. A second method to decrease this advantage could be to judge performance on a different data set labelled by the same human labellers. This is, however, not available at the current moment in time. A third method would be to take data hand-labelled by others and judge performance on that sample. This method, however, has the drawback that human-labellers are different, as shown by Friedman et al. (2023). Thus performance of algorithms relative to each other could be different for different human labellers.

In this research, we propose levelling the playing field by assessing how changes in the empirical human parameters affect the scoring of the DBSCAN-VELOCITY algorithm. For the BIT algorithm, the settings are set to a minimum fixation duration of 55ms and the maximum time a fixation can be interrupted for is 6ms. There is no velocity threshold set.

To give the DBSCAN-VELOCITY algorithm similar information as the BIT algorithm, we keep the minimum fixation duration at 55ms, reduce the possible interruption time from 8ms to 6ms and test saccade velocity thresholds of 22.7°/s and 68.1°/s. These saccade velocity thresholds are respectively 50% lower and higher than the true empirical human parameter, emulating values from a sample from the same human labellers, that by chance has a very different empirical minimum velocity threshold. The results are shown in Table 5.3. An increased velocity threshold decreases the Kappa score from 0.77 to 0.75 and keeps the RMSD score constant. A decreased velocity threshold increases the Kappa score from 0.77 to 0.78 and improves the RMSD score from 0.13 to 0.06.

We see that in both cases, DBSCAN-VELOCITY is still, by a large margin, the best algorithm for both the RMSD and Kappa score. Thus we conclude that the large outperformance by DBSCAN-VELOCITY is not due to unfair use of empirical human parameters.

Table 5.3: The fixation duration properties, RMSD values and Kappa scores for DBSCAN-VELOCITY, with altered velocity threshold and a maximum interruption of 6ms.

Velocity Threshold	Mean (ms)	Std Dev (ms)	Number	RMSD	Kappa	Total Fix (ms)
22.7°/s	250	285	382	0.06	0.78	95,294
68.1°/s	270	288	371	0.13	0.75	100,322

5.2 Robustness

Table 5.4 shows the fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN VELOCITY and the BIT algorithm for the unaltered 500Hz data by Larsson et al.. Note that the properties for the unchanged data differ from Section 4.3.1. This is due to the exclusion of 200Hz samples in this analysis and the fact that we now do account for neighbouring fixations, as our algorithms allow for this.

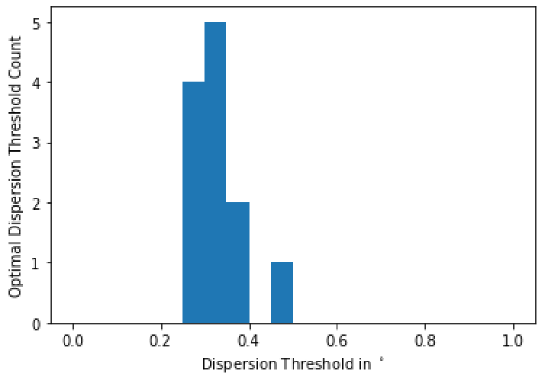
For the BIT algorithm, we adjust the settings to what is recommended in the work by van der Lans et al.(2011). We set the pixel dimensions to 1024x768 (the dimensions of the screen used by Larsson et al.), *minimum_samples* and *n_lost* to 28 for the 500Hz data and 3 for the 62.5Hz data (as they are both set to 3 (60ms) in the paper by van der Lans et al.) and leave the rest of the parameters at the default settings.

All variations of DBSCAN have similar fixation duration properties, whilst the BIT algorithm assigns fewer points as part of fixations, with more fixations in total and a lower mean duration per fixation.

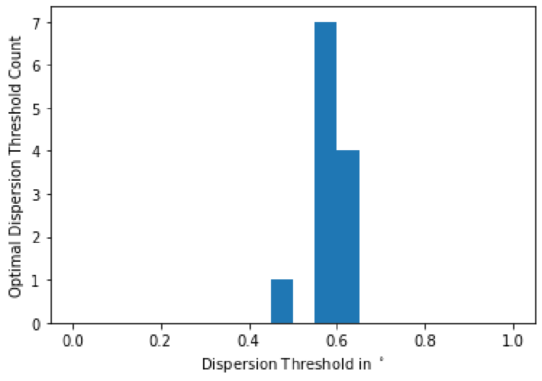
Table 5.4: The fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on the unalterded 500Hz data by Larsson et al..

Algorithm	Mean (ms)	Std Dev (ms)	Number	Total Fix (ms)
DBSCAN-ROB	303	304	320	96,902
DBSCAN-CONS	281	287	337	94,644
DBSCAN-VEL	287	304	320	91,876
BIT	179	106	472	84,488

We now lower the quality of the data in five different ways and evaluate how they affect the labelling of fixations by each algorithm. In three cases we include the same sample as in Figure 5.1, to visualize the differences in fixation labelling. The calculation of the dispersion threshold *eps* is identical for all DBSCAN algorithms. The distributions of optimal dispersion thresholds are very similar for the unaltered data, biased data, data with 3% data loss and data with a lowered frequency, with mean values of 0.33°, 0.33°, 0.34° and 0.34° respectively. The distribution of optimal dispersion thresholds is also similar for data with a lowered precision and data worsened in all aspects, with respective mean values of 0.58° and 0.57°. The distribution of the optimal *eps* values for the unaltered data is shown in Figure 5.2a, and the distribution for data with a lowered precision is shown in Figure 5.2b.



(a) The optimal *eps* distribution for the unaltered data.



(b) The optimal *eps* distribution for the data with a lowered precision.

Figure 5.2: The optimal eps distributions for the unaltered data and the data with a lowered precision.

5.2.1 Biased Data

The first way we lower the data quality is by introducing a bias; we shift all eye-tracking points 1.05° to the left and 0.65° down. The fixation duration properties and Kappa scores for the algorithms on biased data are shown in Figure 5.5. The labelling of fixations by DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY does not change, as indicated by the perfect kappa score of 1 and the identical fixation duration properties. The bias shifts all data and thus has no effect for any clustering algorithm that depends on the relative distances between points. The BIT algorithm displays a slight change in the detected fixation properties and a near-perfect kappa score of 0.98. This is likely due to some points being pushed outside the screen by introducing the bias and the BIT algorithm not detecting fixations outside of the screen. This hypothesis is supported by the fact that the BIT algorithm has a perfect Kappa score of 1 for ten out of the twelve samples, as shown in Appendix B.2.

Table 5.5: The fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on the 500Hz data by Larsson et al. with an added bias.

Algorithm	Mean (ms)	Std Dev (ms)	Number	Total Fix (ms)	Kappa
DBSCAN-ROB	303	304	320	96,902	1
DBSCAN-CONS	281	287	337	94,644	1
DBSCAN-VEL	287	304	320	91,786	1
BIT	180	107	464	83,520	0.98

The p-values for the sign test on the differences between Kappa scores on individual samples for the different algorithms are shown in Table 5.6. Here the Kappa scores measure agreement with the original labels of each algorithm on the unaltered data. In the case of biased data, there are no significant p-values at a 0.025 level.

Table 5.6: The p-values from the sign test on the differences in paired Kappa scores for the data with an added bias between CoderMN(M), DBSCAN-CONSECUTIVE(C), DBSCAN-ROBUST(R), DBSCAN-VELOCITY(V) and BIT(B). There are no significant values at a 0.025 level.

	V-C	V-R	V-B	C-R	C-B	R-B
p-value	0.5	0.5	0.063	0.5	0.063	0.063

5.2.2 Data Loss

We now introduce a data loss of 3% to the unaltered data. The fixation duration properties and Kappa scores for the algorithms on data with and introduced data loss are shown in Table 5.7. We see that DBSCAN-ROBUST, DBSCAN-VELOCITY and BIT are all not affected strongly, with very similar fixation duration properties to the unaltered data and near-perfect Kappa scores. Data loss can lead to problems for DBSCAN-ROBUST and DBSCAN-VELOCITY if the points occur after one another, leading to gaps larger than the minimum saccade duration of 8ms and thus, the unjust splitting of fixations. They also impact fixation detection if they are at the start or end of fixations, then the detected fixations are started too late or end too early. This latter point will also impact hand-labellers as they cannot tell whether the point is part of the fixation or the bordering eye-tracking events. In all other cases, the relaxed consecutivity allows for these errors without affecting fixation detection.

Table 5.7: The fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on the 500Hz data by Larsson et al., with an introduced data loss of 3%.

Algorithm	Mean (ms)	Std Dev (ms)	Number	Total Fix (ms)	Kappa
DBSCAN-ROB	305	304	317	96,766	0.99
DBSCAN-CONS	104	51	689	71,760	0.43
DBSCAN-VEL	290	304	317	91,840	0.99
BIT	173	104	478	82,694	0.93

DBSCAN-CONSECUTIVE, however, has very poor robustness to data loss. When 3% of measurements are deleted, the mean fixation duration drops by 63%, the number of fixations increases by 104%, the total fixation duration drops by 24% and the Kappa score is only 0.43. This shows that fewer points are labelled as fixations because neighbourhoods are split up and the threshold for $MinPts$ is no longer reached, resulting in fewer points being labelled as core points. Existing fixations are also split up when an erroneous measurement occurs in a fixation.

These results are more extreme when we remove 15% of the points. The total fixation duration drops 94% and the Kappa score is 0.02, indicating that the agreement is hardly better than chance. This is due to the earlier-mentioned effects becoming more extreme, so much so that core point neighbourhoods are capped to values close to $MinPts$.

Figure 5.3 shows how erroneous measurements affect the detection of fixations for DBSCAN-CONSECUTIVE, DBSCAN-ROBUST, DBSCAN-VELOCITY and BIT; it concerns the same sample as Figure 5.1 but with an artificial data loss of 3%. There are no big changes in fixation detection by DBSCAN-ROBUST and DBSCAN-VELOCITY. For DBSCAN-CONSECUTIVE, however, there are big changes. What are originally its second, fifth, sixth and seventh fixations are now split in two; not every split is visible due to the small gaps relative to the scale of the figure. Furthermore, missing measurements cut off starts and ends of fixations. The points that are cut off do not total to enough points to achieve the $MinPts$ threshold and are thus labelled as noise (non-fixation). For BIT, there are two main changes in the detected fixations, the first is that what is originally the sixth fixation for the unchanged data, no longer exists. Second of all, what was originally the seventh fixation, has now been split into two fixations. Overall, the Kappa score for BIT is however 0.93, indicating that this randomly selected sample has more trouble with the introduced data loss than the rest of the data.

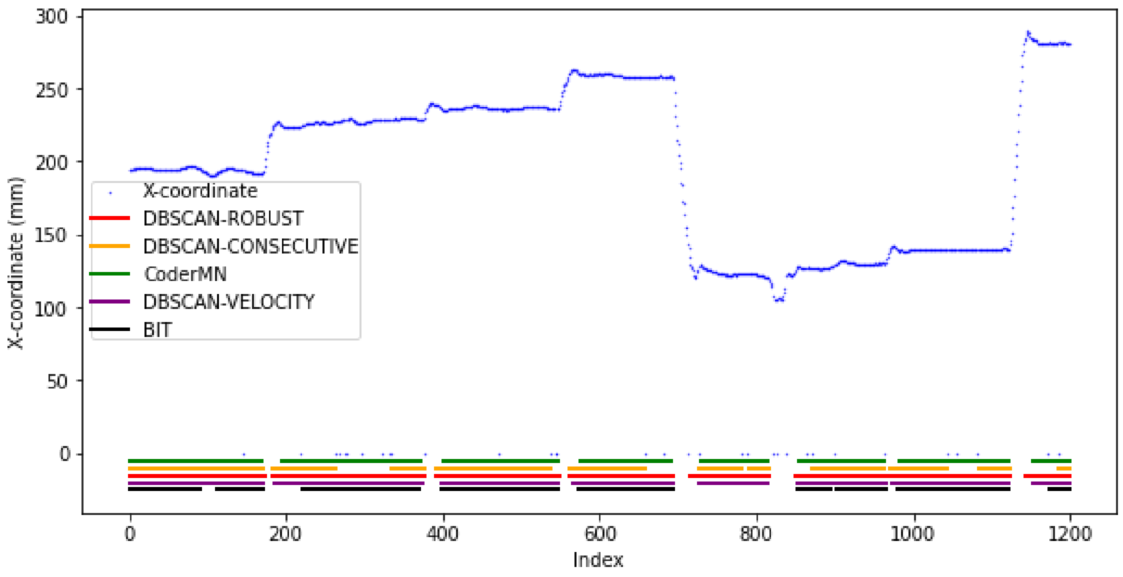


Figure 5.3: The x_{gaze} coordinates and the detected fixations for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY, and CoderMN, for a sample of eye-tracking data for which 3% data loss is introduced. Missing data points are plotted as x-coordinate 0.

The p-values for the sign test on the differences between Kappa scores on individual samples for the different algorithms are shown in Table 5.8. There are statistically significant differences between the Kappa scores of the algorithms. DBSCAN-VELOCITY and DBSCAN-ROBUST outperform DBSCAN-CONSECUTIVE and BIT. BIT outperforms DBSCAN-CONSECUTIVE.

Table 5.8: The p-values from the sign test on the differences in paired Kappa scores for the data with introduced data loss between CoderMN(M), DBSCAN-CONSECUTIVE(C), DBSCAN-ROBUST(R), DBSCAN-VELOCITY(V) and BIT(B). Significant values are indicated with a * next to the value and a * for the algorithm that outperforms.

	V*-C	V-R	V*-B	C-R*	C-B*	R*-B
p-value	<0.001*	0.387	<0.001*	<0.001*	<0.001*	0.003*

5.2.3 Lowered Precision

The fixation duration properties and Kappa scores for the algorithms on data with a lowered precision are shown in Table 5.9. In contrast to earlier, all algorithms are now affected strongly, with DBSCAN-ROBUST and DBSCAN-VELOCITY performing best with Kappa scores of 0.80 and 0.78, respectively. The similarity in robustness performance between DBSCAN-ROBUST and DBSCAN-VELOCITY is to be expected, as DBSCAN-VELOCITY is simply DBSCAN-ROBUST with a velocity post-processing step applied.

Table 5.9: The fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on the 500Hz data by Larsson et al. with a lowered precision.

Algorithm	Mean (ms)	Std Dev (ms)	Number	Total Fix (ms)	Kappa
DBSCAN-ROB	340	380	296	100,786	0.80
DBSCAN-CONS	160	104	572	91,702	0.71
DBSCAN-VEL	307	393	315	96,804	0.78
BIT	285	205	349	99,465	0.41

For DBSCAN-ROBUST, the mean detected fixation duration increases by 12%, the number of detected fixations decreases by 8%, and the total fixation duration increases by 4%. It is to be expected that noisy data strongly influences the detection of fixations by an algorithm dependent on the densities of points. Lowering the precision leads to the relative distances between points increasing or decreasing and affects the densities of points around points, as indicated by the change in optimal *eps* distribution. This leads to the differences between points that are and are not part of fixations becoming less clear and affects the fixation detection of a dispersion-based algorithm. DBSCAN-VELOCITY shows similar behavior, where the mean fixation duration increases by 7%, the number of detected fixations decreases by 2%, and the total fixation duration increases by 5%.

For DBSCAN-CONSECUTIVE, the fixation duration properties become much less stable when precision is lowered as the mean fixation duration drops by 43%, the number of fixations increases by 70%, the total fixation duration drops by 3%, and the Kappa score is 0.71. This indicates that a lowered precision leads to fixations being split up into shorter ones. It is noteworthy that a lowered precision leads to fixations being merged for DBSCAN-ROBUST.

Figure 5.4 illustrates the reason for this; it concerns the same sample as Figure 5.1, but with an artificially lowered precision. The most significant changes compared to Figure 5.1, are that the first fixation detected by CoderMN is split into two fixations by DBSCAN-CONSECUTIVE, and that the third and fourth fixations detected by CoderMN are merged into one fixation by DBSCAN-ROBUST. This leads to more fixations with a shorter mean duration for DBSCAN-CONSECUTIVE and fewer fixations with a higher mean duration for DBSCAN-ROBUST.

For the BIT algorithm, the mean fixation duration increases 59% and the number of fixations, decreases by 26%. 18% more points are labelled as fixations and the Kappa score is only 0.41. Figure 5.4 illustrates this change. What were originally the second to fourth fixations for the BIT algorithm, are now merged into one and what were originally the first and fifth fixations, are neighbouring fixations to this one merged fixation. What were originally the seventh and eighth fixation are now also neighbouring and all fixations now have a tendency to start early and end late, in contrast to what happened first, where they started late and ended early. These changes explain the increase in mean fixation duration, the decrease in the number of fixations, the increase in the total fixation duration and the poor Kappa score of 0.41.

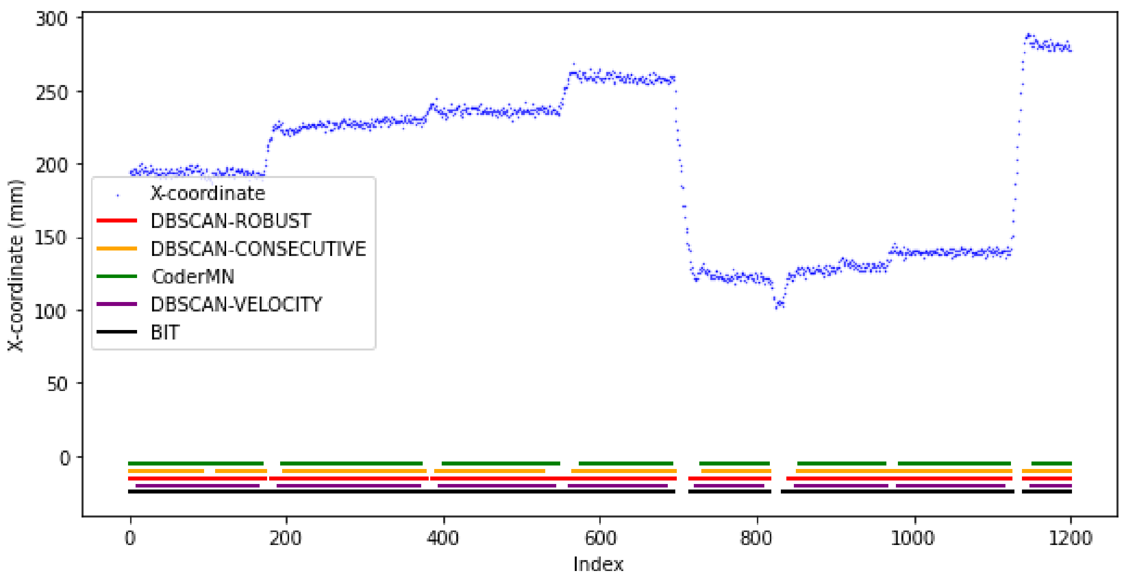


Figure 5.4: The x_{gaze} coordinates and the detected fixations for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY, BIT and CoderMN, for a sample of eye-tracking data for which the precision has been artificially lowered.

The p-values for the sign test on the differences between Kappa scores on individual samples for the different algorithms are shown in Table 5.10. There are statistically significant differences between the Kappa scores of the algorithms. DBSCAN-VELOCITY, DBSCAN-ROBUST and DBSCAN-CONSECUTIVE outperform BIT.

Table 5.10: The p-values from the sign test on the differences in paired Kappa scores for the lowered precision data between CoderMN(M), DBSCAN-CONSECUTIVE(C), DBSCAN-ROBUST(R), DBSCAN-VELOCITY(V) and BIT(B). Significant values are indicated with a * next to the value and a * for the algorithm that outperforms.

	V-C	V-R	V*-B	C-R	C*-B	R*-B
p-value	0.387	0.073	<0.001*	0.387	0.003*	<0.001*

5.2.4 Lowered Frequency

The fixation duration properties and Kappa scores for the algorithms on data with a lowered frequency are shown in Table 5.11. DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY are all not affected strongly with very similar fixation properties to the unaltered data and all have Kappa scores of 0.88 or higher. This indicates a near-perfect point-by-point agreement that is even better than the agreement between hand labellers on the unchanged data. The BIT algorithm, on the other hand, is affected more strongly, the mean fixation duration increases by 39%, the total number of fixations decreases by 20%, and the Kappa score is 0.69.

Table 5.11: The fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on the 500Hz data by Larsson et al. with a lowered frequency.

Algorithm	Mean (ms)	Std Dev (ms)	Number	Total Fix (ms)	Kappa
DBSCAN-ROB	290	288	334	96,880	0.92
DBSCAN-CONS	277	269	349	96,592	0.92
DBSCAN-VEL	280	289	334	93,600	0.88
BIT	248	176	378	93,744	0.69

The p-values for the sign test on the differences between Kappa scores on individual samples for the different algorithms are shown in Table 5.12. There are statistically significant differences between the Kappa scores of the algorithms. DBSCAN-ROBUST and DBSCAN-CONSECUTIVE outperform DBSCAN-VELOCITY and BIT. DBSCAN-VELOCITY outperforms BIT.

Table 5.12: The p-values from the sign test on the differences in paired Kappa scores for the data for which the frequency has been lowered, between CoderMN(M), DBSCAN-CONSECUTIVE(C), DBSCAN-ROBUST(R), DBSCAN-VELOCITY(V) and BIT(B). Significant values are indicated with a * next to the value and a * for the algorithm that outperforms.

	V-C*	V-R*	V*-B	C-R	C*-B	R*-B
p-value	0.019*	0.019*	<0.001*	0.5	<0.001*	<0.001*

5.2.5 Everything Applied

The fixation duration properties and Kappa scores for the algorithms on data with a lowered quality in all aspects are shown in Table 5.13. DBSCAN-ROBUST and DBSCAN-VELOCITY perform best, with Kappa scores of 0.83 and 0.81 respectively. This is similar to the agreement rate of the human labellers on the unaltered data, which have a Kappa score of 0.84 relative to each other. For DBSCAN-ROBUST, the mean fixation duration decreases by 4%, the number of fixations increases by 8% and the total fixation duration increases by 4%. For DBSCAN-VELOCITY the mean fixation duration decreases by 5%, the number of fixations increases by 8% and the total fixation duration increases by 3%.

Table 5.13: The fixation duration properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT on the 500Hz data by Larsson et al. with a lowered quality in all aspects.

Algorithm	Mean (ms)	Std Dev (ms)	Number	Total Fix (ms)	Kappa
DBSCAN-ROB	290	229	347	100,496	0.83
DBSCAN-CONS	172	130	559	96,240	0.68
DBSCAN-VEL	274	229	345	94,544	0.81
BIT	340	328	293	99,620	0.45

When all properties of lowered data quality are applied together, DBSCAN-CONSECUTIVE performs better than when introducing data loss or lowering precision individually. This is an interesting observation and is likely due to a lowered frequency decreasing the effects of other properties of lower-quality data. Data loss in lower frequency data namely represents data loss of consecutive points in higher frequency data and the effects of a lowered precision on the relative distances between points decreases when frequency decreases, as the distances between points increase and thus the relative magnitude of a lowered precision decreases. When the data quality is lowered in all aspects, the mean fixation duration drops by 39%, the number of fixations increases by 66%, the total fixation duration increases by 2%, and we obtain a Kappa score of 0.68.

The BIT algorithm performs worst with a Kappa score of 0.45. The mean fixation duration increases by 90%, the number of fixations decreases by 38% and the total fixation duration increases by 18%. For BIT, the robustness on data with a lowered quality in all aspects is slightly improved compared to data for which only the precision has been lowered, with respective Kappa scores of 0.45 and 0.41.

Figure 5.5 shows how lowering the quality of data in all aspects affects the detection of fixations for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY and BIT; it concerns the same sample as Figure 5.1 but with a lowered quality in all aspects. The index now runs from 0 to 149 instead of from 0 to 1199, as the frequency has decreased by a factor of 8. As expected from the Kappa score, there are small changes for DBSCAN-ROBUST and DBSCAN-VELOCITY. The biggest change for DBSCAN-CONSECUTIVE is that what was originally its second fixation is now split in two by the missing measurement at index 33. Furthermore, some fixations start earlier or end later than was first the case. This explains the increase in total fixation duration, paired with the decreased mean fixation duration and increased number of fixations. For the BIT algorithm, what were originally its first and second and its seventh and eight fixation, have now been merged, and the fixations now have a tendency to start too early and end too late. In the unaltered data, there was a tendency to start late and end too early. These changes explain the increase in mean fixation duration, the decrease in the number of fixations, the increase in total fixation duration and the poor Kappa score of 0.45.

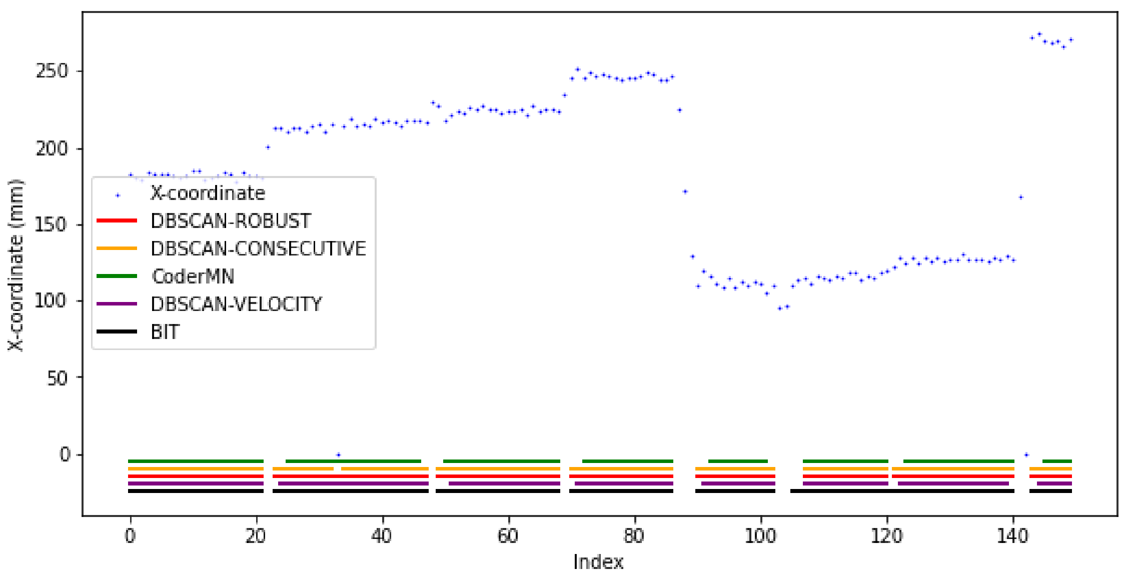


Figure 5.5: The x_{gaze} coordinates and the detected fixations for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE, DBSCAN-VELOCITY, BIT and CoderMN, for a sample of eye-tracking data for which the quality has been artificially lowered in all aspects.

The p-values for the sign test on the differences between Kappa scores on individual samples for the different algorithms are shown in Table 5.14. There are statistically significant differences between the Kappa scores of the algorithms. DBSCAN-VELOCITY and DBSCAN-ROBUST outperform DBSCAN-CONSECUTIVE and BIT. DBSCAN-CONSECUTIVE outperforms BIT.

Table 5.14: The p-values from the sign test on the differences in paired Kappa scores for the data for which the quality has been lowered in all aspects, between CoderMN(M), DBSCAN-CONSECUTIVE(C), DBSCAN-ROBUST(R), DBSCAN-VELOCITY(V) and BIT(B). Significant values are indicated with a * next to the value and a * for the algorithm that outperforms.

	V*-C	V-R	V*-B	C-R*	C*-B	R*-B
p-value	<0.001*	0.194	<0.001*	<0.001*	<0.001*	<0.001*

5.2.6 Intermediate Conclusion

We conclude that out of the considered algorithms, DBSCAN-ROBUST is the most stable in terms of Kappa score for all the ways in which the quality of the data is artificially lowered. DBSCAN-VELOCITY performs very similarly, which is to be expected as it is DBSCAN-ROBUST with a velocity-based post-processing step. They are both least stable for data of a lowered precision; a lowered precision strongly affects the optimal dispersion threshold and leads to the most different fixation duration properties and worst Kappa score. It is noteworthy that the algorithms are more stable for data that has been worsened on all aspects, as precision is one of these aspects.

DBSCAN-CONSECUTIVE is least stable for data with missing measurements. Data loss strongly decreases the number of points that are labelled as core points and cuts off fixations by breaking up neighbourhoods. It confirms earlier reasons named for introducing a relaxed consecutivity requirement. It is notable that the algorithm is more stable for data that has been worsened on all aspects, as missing measurements is one of these aspects.

We acknowledge that the method of creating missing measurements in this research, may be different from what is observed in practice. We sample from a uniform distribution but it can be hypothesised that, in practice, missing measurements occur in short bursts of randomized size. When the data quality is lowered in all aspects, DBSCAN-CONSECUTIVE obtains a Kappa score of 0.68.

The BIT algorithm performs the worst on the data worsened in all aspects. It obtains a Kappa score of 0.45 and has an even lower Kappa score of 0.41 for data with a lowered precision, it also performs relatively poorly on data with a lowered frequency, with a Kappa score of 0.69. It is thus the least suitable out of the considered algorithms for detecting fixations in data with a lower quality. Hessels et al.(2017b) also find that the BIT algorithm struggles with the introduction of noise in data.

Lastly, we note that hand-labellers will also not obtain perfect kappa scores between high and lower-quality data as they will not be able to detect the same fixation when data gets very noisy or missing measurements occur at the start or end of fixations. Friedman et al. (2023) note that the agreement between hand labellers was lowest for low-quality data; this indicates the difficulty of hand-labelling low-quality data.

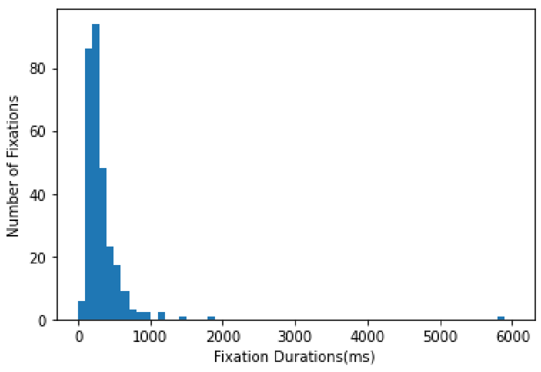
5.3 Non-labelled Data

We detect fixations in the data by Martinovici using DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY. We exclude samples with more than 25% data loss, as we question the point of analysing a sample of such poor quality. The results are shown in Table 5.15.

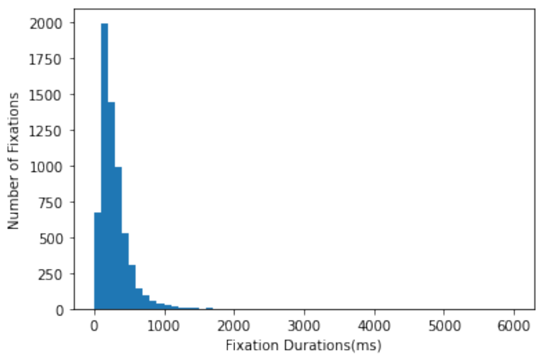
Table 5.15: The fixation properties for DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY in the data by Martinovici.

Algorithm	Mean (ms)	Std Dev (ms)	Number
DBSCAN-ROBUST	195	120	8,936
DBSCAN-CONSECUTIVE	173	107	9,646
DBSCAN-VELOCITY	182	119	8902

The mean duration of the fixations detected in the data by Larsson et al., is 303ms for DBSCAN-ROBUST, 281ms for DBSCAN-CONSECUTIVE and 287ms for DBSCAN-VELOCITY. This means detected fixations for data by Martinovici are 36% shorter for DBSCAN-ROBUST, 38% shorter for DBSCAN-CONSECUTIVE and 37% shorter for DBSCAN-VELOCITY. Differences in mean fixation duration could be due to the difference in stimulus contents and assignment, this effect could, however, also be inverted and would require further research to uncover. The mean values are plausible and similar to what other research mentions, but the standard deviations are much lower for this data than for the data by Larsson et al.. This could be due to larger and relatively more outliers in the fixation durations for the data by Larsson et al.. Figure 5.6 presents the distributions of fixation durations for the two different data sets; there is a higher relative amount of outliers, that, on top of that, have a larger magnitude in the data by Larsson et al..



(a) A histogram of the fixation durations for the data by Larsson et al. as detected by DBSCAN-ROBUST.



(b) A histogram of the fixation durations for the data by Martinovici as detected by the DBSCAN-ROBUST.

Figure 5.6: The fixation duration distributions as detected by DBSCAN-ROBUST in the data by Larsson et al. and Martinovici.

DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY have similar changes in mean fixation durations (36%, 38% and 37%) compared to the unaltered data by Larsson et al.. This possibly indicates that the data by Martinovici is of a higher quality than the data altered to simulate the properties of the GP3 eye-tracker, as the difference in changes between the algorithms is smaller. This thesis could be supported by the optimal dispersion thresholds for the data by Martinovici shown in Figure 5.7. The mean optimal dispersion threshold is 0.49° which is roughly halfway between the mean dispersion thresholds for the unchanged data and data with added noise by Larsson et al.. This could indicate that the data by Martinovici has a precision somewhere inbetween these two samples, which decreases the difference in changes of mean fixation duration. The difference in optimal eps distributions could however also be due to other reasons, like different tasks and different participants.

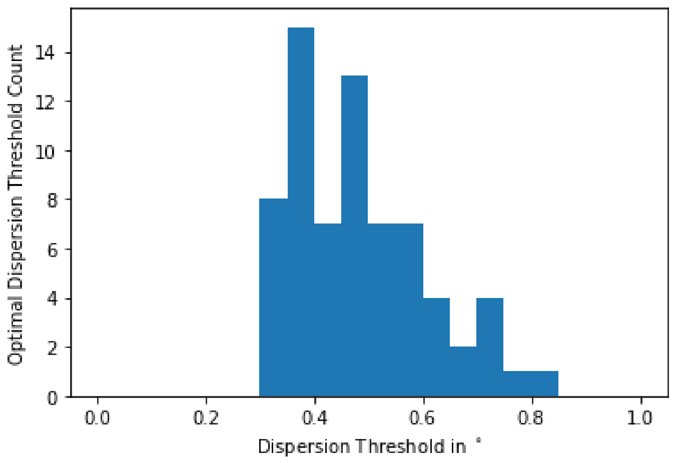


Figure 5.7: The optimal dispersion thresholds for the samples from Martinovici. The mean optimal dispersion threshold is 0.49°.

We conclude that DBSCAN-ROBUST, DBSCAN-CONSECUTIVE and DBSCAN-VELOCITY return plausible fixation duration properties on data for which they do not know the empirical human parameters.

Chapter 6

Conclusion

This research aimed to construct an algorithm that accurately detects fixations in high-quality data while being robust for eye-tracking data with a lowered quality. We modify the existing DBSCAN algorithm (Ester et al., 1996) to be suitable for detecting fixations in eye-tracking data and extend over three existing implementations (S. Li et al., 2015)(B. Li et al., 2016)(Yu et al., 2016) in two ways. The first is that we introduce a relaxed consecutivity constraint to make the algorithm more robust for lower-quality data, and the second is that we introduce a velocity-based post-processing step to tackle the limitations of eye-movement classification algorithms that are solely dispersion-based.

We construct three algorithms; DBSCAN with strict consecutivity (DBSCAN-CONSECUTIVE), DBSCAN with relaxed consecutivity (DBSCAN-ROBUST) and DBSCAN-ROBUST with an added velocity-based post-processing step (DBSCAN-VELOCITY). We judge the algorithms' performances by detecting fixations in the eye-tracking data collected by Larsson et al. and comparing the algorithms' labels to the hand-labelled values by Richard Andersson and Marcus Nyström, two experts in the field of eye-tracking. This comparison is done through RMSD, a measure that compares the similarity of the fixation duration properties, and the Kappa score, which measures the agreement between the algorithms and the hand-labellers on a point-by-point basis. Andersson et al. (2017) do the same for ten other algorithms, allowing a comparison between our algorithms and their algorithms. The results show that DBSCAN-VELOCITY is the best algorithm, with a Kappa score of 0.77 and an RMSD value of 0.13 compared to the benchmark of the two human labellers. The best Kappa score out of the algorithms considered by Andersson et al. is 0.67, the Kappa score between the human labellers is 0.84, and Cohen(1960) considers Kappa scores above 0.8 to indicate almost perfect agreement.

Thus, we have succeeded in constructing an algorithm that accurately detects fixations in high-quality data, according to the Kappa and RMSD metrics and have created an eye-tracking classification algorithm that outperforms the existing options considered by Andersson et al.. To address the second question of robustness, we artificially lower the quality of the eye-tracking data by Larsson et al. by introducing a bias, decreasing precision, introducing data loss and lowering the frequency. These are typical properties of lower-quality data, according to Holmqvist et al.(2012). We observe the stability of fixation duration properties and evaluate the change in labelling through the Kappa score when the quality of the data is lowered. We evaluate the robustness of the DBSCAN algorithms and the BIT algorithm; the latter is added as it is one of the three algorithms with the highest Kappa score in the work by Andersson et al. and has the best RMSD measure out of these.

DBSCAN-ROBUST and DBSCAN-VELOCITY return very similar Kappa scores of 0.83 and 0.81 when the quality of the data is lowered in all aspects, which is to be expected as DBSCAN-VELOCITY simply adds a velocity-based post-processing step to the output of DBSCAN-ROBUST. According to Cohen, both these values are part of the almost perfect agreement category. DBSCAN-CONSECUTIVE and BIT perform worse with Kappa scores of 0.68 and 0.45 and see 39% and 90% changes in their mean fixation duration.

DBSCAN-VELOCITY is therefore considered robust for eye-tracking data with a lowered quality and thus, we have succeeded in creating an algorithm that is both robust and has a high accuracy for high-quality eye-tracking data. However, we cannot conclude whether DBSCAN-VELOCITY is the best algorithm in terms of accuracy and robustness. The results for accuracy can differ between hand-labelled data sets, due to differing behaviour by hand labellers and varying levels of quality, and we cannot compare our algorithm against all existing algorithms. Certain eye-tracking algorithms are specialized, such as the Identification by two-means clustering (I2MC) algorithm by Hessels et al.(2017b), which is specialized in robustness for noisy data, and the MNH algorithm by Friedman, Rigas, Abdulin and Komogortsev(2018), which is specialized in accuracy on high-quality data. A comparison of DBSCAN-VELOCITY to these algorithms would be an interesting direction for further research.

There are several other directions for further research. The first is a different tuning method for the optimal dispersion threshold *eps*. The current method relies on finding an elbow point using the Kneedle algorithm by Satopaa et al.(2011). Other elbow point detection methods can be evaluated, and perhaps a machine-learning approach could be implemented if enough hand-labelled data is available. A second method would be to combine the output of DBSCAN-VELOCITY with the output of another eye-tracking classification algorithm. DBSCAN-VELOCITY namely only detects fixations; thus, its output could be combined with algorithms that do not detect fixations but do detect saccades, smooth pursuits or post-saccadic oscillations, like the EM (Engbert & Mergenthaler, 2006) or the LNS (Larsson et al., 2013) algorithm.

A third direction for further research is implementing the option for binocular data, DBSCAN-VELOCITY currently only considers data from one eye. This decision was made as Andersson et al. hand-labelled the data from only one eye, and hand-labelled data for two eyes would be required to implement this. The BIT algorithm uses data from both eyes when it is available; one could take a similar approach as the BIT algorithm does. This is also a limitation of this research, as we do not take full advantage of the possibilities of the BIT algorithm. Using data from both eyes will lead to an improved comparison. A last possible direction for further research, is allowing for the distance of a participant to the screen to vary during a sample. Currently, this is not possible; introducing it would allow one to use it in measurement conditions where there is no headrest for participants.

Lastly, we consider several potential shortfalls of this research. The first is that the method for introducing data loss may be flawed. We introduce single missing measurements randomly; in practice, they could occur differently. Hessels et al. introduce data loss into their sample by randomly introducing periods of data loss taken from a different data set. The second potential shortfall is that we compare the performance of our algorithm against those in the work by Andersson et al.. Andersson et al. state that they adjust the parameters of these algorithms as much as can be expected from someone without expert domain knowledge. Whilst our algorithm does not require domain knowledge to implement, this could lead to the other algorithms performing worse if their parameters were adjusted too little or incorrectly. This same point holds for BIT, as it currently seems to not be able to detect fixations outside of the screen, whilst DBSCAN algorithms can, increasing the performance of the DBSCAN algorithms relative to BIT, as both human labellers label fixations outside the screen. This in addition to the fact that BIT can use binocular data, a possibility that is not utilized in this research. A last shortfall is that all conclusions are based on one set of hand-labelled data. Ideally, the analysis would be repeated on other hand-labelled data sets. These are however hard to find and one would also need to implement the algorithms used by Andersson et al. to compare performance and robustness. Hopefully more hand-labelled (binocular) data sets become available in the future, allowing for addressing of this shortfall.

To summarize, we construct DBSCAN-VELOCITY, a modified version of DBSCAN, that is suitable for the detection of fixations in eye-tracking data from static stimuli. It is accurate for the high quality eye-tracking data by Larsson et al., outperforming all other considered algorithms and achieving Kappa scores on the individual eye-tracking samples that are significantly higher. Furthermore, its labelling is robust when the quality of the data is lowered, as the new labels are in almost perfect agreement with the original labels. Thus DBSCAN-VELOCITY is a suitable algorithm for detecting fixations in eye-tracking data, for which the quality is equal to or better than the quality of the modified data that is considered in this work.

References

Andersson, R., Larsson, L., Holmqvist, K., Stridh, M. & Nyström, M. (2017). One algorithm to rule them all? an evaluation and discussion of ten eye movement event-detection algorithms. *Behavior Research Methods*, *49*, 616–637.

Brand, J., Diamond, S., Thomas, N. & Diamond, D. (2021). Evaluating the data quality of the gazepoint gp3 low-cost eye tracker when used independently by study participants. *Behavior Research Methods*, *53*, 1502 - 1514.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*, 37 - 46.

Dar, A., Wagner, A. & Hanke, M. (2021). Remodnav: robust eye-movement classification for dynamic stimulation. *Behavior Research Methods*, *53*, 399–414.

Engbert, R. & Mergenthaler, K. (2006). Microsaccades are triggered by low retinal image slip. *Proceedings of the National Academy of Sciences*, *103*, 7192-7197. doi: 10.1073/pnas.0509557103

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the second international conference on knowledge discovery and data mining* (p. 226–231). AAAI Press.

Fairchild, M. D. (2005). *Color appearance models.* John Wiley Sons, Ltd.

Friedman, L. (2020). Brief communication: Three errors and two problems in a recent paper: gazenet: End-to-end eye-movement event detection with deep neural networks (zemblys, niehorster, and holmqvist, 2019). *Behavior Research Methods*, *52*, 1671-1680. doi: 10.3758/s13428-019-01342-x

Friedman, L., Prokopenko, V., Djanian, S., Katrychuck, D. & Komogortsev, O. (2023). Factors affecting inter-rater agreement in human classification of eye movements: a comparison of three datasets. *Behavior Research Methods*, *55*, 417-427. doi: 10.3758/s13428-021-01782-4

Friedman, L., Rigas, I., Abdulin, E. & Komogortsev, O. (2018). A novel evaluation of two related and two independent algorithms for eye movement classification during reading. *Behaviour Research Methods*, *50*, 1374-1397. doi: 10.3758/s13428-018-1050-7

Hessels, R., Niehorster, D., Kemner, C. & Hooge, I. (2017a). Noise-robust fixation detection in eye movement data: Identification by two-means clustering (i2mc). *Behavior Research Methods*, *49*, 1802–1823.

Hessels, R., Niehorster, D., Kemner, C. & Hooge, I. (2017b). Noise-robust fixation detection in eye movement data: Identification by two-means clustering (i2mc). *Behavior Research Methods*, *49*, 1802-1823. doi: 10.3758/s13428-016-0822-1

Hoffmann, S., Schönbrodt, F., Elsas, R., Wilson, R., Strasser, U. & Boulesteix, A.-L. (2021). The multiplicity of analysisstrategies jeopardizesreplicability: lessons learned across disciplines. *Royal Society Open Science*, *8*. doi: 0.1098/rsos.201925

Holmqvist, K., Nyström, M. & Mulvey, F. (2012). Eye tracker data quality: What it is and how to measure it. In *Proceedings of the symposium on eye tracking research and applications* (p. 45–52). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2168556.2168563` doi: 10.1145/2168556.2168563

Hooge, I. & Camps, G. (2013). Scan path entropy and arrow plots: capturing scanning behavior of multiple observers. *Frontiers in Psychology*, *4*. doi: 10.3389/fpsyg.2013.00996

Larsson, L., Nyström, M. & Stridh, M. (2013). Detection of saccades and postsaccadic oscillations in the presence of smooth pursuit. *IEEE Transactions on Biomedical Engineering*, *60*(9), 2484-2493. doi: 10.1109/TBME.2013.2258918

Li, B., Wang, Q., Barney, E., Hart, L., Wall, C., Chawarska, K., ... Shic, F. (2016). Modified dbscan algorithm on oculomotor fixation identification. In *Proceedings of the ninth biennial acm symposium on eye tracking research applications* (p. 337–338). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2857491.2888587` doi: 10.1145/2857491.2888587

Li, S., Xu, Y., Sun, W., Yang, Z., Wang, L., Chai, M. & Wei, X. (2015). Driver fixation region division–oriented clustering method based on the density-based spatial clustering of applications with noise and the mathematical morphology clustering. *Advances in Mechanical Engineering*, *7*(10).

Martinovici, A. (2019). *Revealing attention - how eye movements predict brand choice and moment of choice* (Doctoral dissertation, Tilburg University). (CentER Dissertation Series Volume: 616) doi: 10.26116/center-lis-1937

Satopaa, V., Albrecht, J., Irwin, D. & Raghavan, B. (2011). Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops* (p. 166-171). doi: 10.1109/ICDCSW.2011.20

SensoMotoricInstruments. (2009). iview x system manual [Computer software manual].

Startsev, M., Agtzidis, I. & Dorr, M. (2019). Characterizing and automatically detecting smooth pursuit in a large-scale ground-truth data set of dynamic natural scenes. *Journal of Vision*, *19*(14).

TobiiTechnology. (2009). Tobii t60 xl eye tracker [Computer software manual].

van der Lans, R., Wedel, M. & Pieters, R. (2011). Defining eye-fixation sequences across individuals and tasks: the binocular-individual threshold (bit) algorithm. *Behavior Research Methods*, *43*, 239–257.

Veneri, G., Piu, P., Rosini, F., Federighi, P., Federico, A. & Rufa, A. (2011). Automatic eye fixations identification based on analysis of variance and covariance. *Pattern Recognition Letters*, *32*, 1588-1593. doi: 10.1016/j.patrec.2011.06.012

Warrens, M. (2008). On the equivalence of cohen's kappa and the hubert-arabie adjusted rand index. *Journal of Classification*, *25*, 177-183. doi: 10.1007/s00357-008-9023-7

Yu, M., Lin, Y., Breugelmans, J., Wang, X., Wang, Y., Gao, G. & Tang, X. (2016). A spatial-temporal trajectory clustering algorithm for eye fixations identification. *Intelligent Data*

Analysis, *20*(2), 377–393.

Appendix A

Notes on Work by Andersson et al.

A.1 Fixation Duration Properties Calculation

From our work to process fixation point labels into fixation duration properties, we find that Andersson et al. make two choices that are not mentioned in their research.

The first is that they exclude fixations that begin at the start of a sample or stop at the end of a sample. This decision is one we agree with, as we do not know their true start and stop points respectively. Calculating fixation duration properties for the human labellers in this manner lead to the same fixation duration properties as in the paper.

The second is that they do not account for neighbouring fixations in the calculation of event duration properties. This means that when two fixations are not separated by non-fixation labels, they are treated as one fixation. The possible reason for this choice is that for the human labellers, it is not possible to identify neighbouring fixations, as all points belonging to fixations get label "1". For the BIT and DBSCAN algorithms this is different as they label points belonging to fixations with a number, where every fixation receives a different number. Thus, when comparing fixation duration properties to those of human-labellers, it makes sense to also not account for neighbouring fixations. Calculating fixation duration properties for the BIT algorithm in this manner lead to the same fixation duration properties as in the paper.

In the robustness comparison we do account for neighbouring fixations, as we compare algorithms that are capable of establishing that there are neighbouring fixations.

A.2 Frequencies

Andersson et al. do not realise that two out of their fourteen samples are recorded at 200Hz instead of 500Hz. This observation is also made by Friedman(2020). This leads to mean fixation durations that are too low and incorrect calculation of fixations for algorithms that require the frequency to be manually inputted.

In the robustness analysis, we exclude the 200Hz samples to prevent issues when artificially lowering the frequency of the data.

A.3 RMSD

When calculating the RMSD values, we obtain different values that Andersson et al.. The sequence of the algorithms in their ranking based on RSMD does however not change. As the RMSD is a relative metric and can only be used to compare results on one data set, we deem this to not be an issue.

It is likely that their description to rescale event duration parameters on a scale of [0,1] is incomplete. Equation 1 on page 625 seems to be incomplete. In this research we rescale to [0,1] using minimum and maximum values, where the minimum value resembles 0 and the maximum value resembles 1.

Appendix B

Kappa Scores

B.1 Unaltered Data

Table B.1: Kappa scores for the unaltered 500Hz samples.

Sample	MN	RA	CONS	ROB	VEL	BIT
UH33vy	0.90	0.90	0.62	0.58	0.75	0.66
UL43Rome	0.97	0.97	0.80	0.75	0.88	0.86
TL28Kon	0.87	0.87	0.54	0.62	0.71	0.56
UL23Eur	0.92	0.92	0.73	0.72	0.83	0.78
UL31Kon	0.92	0.92	0.81	0.79	0.87	0.87
UH27vy	0.96	0.96	0.67	0.58	0.83	0.77
UH21Rome	0.96	0.96	0.62	0.53	0.75	0.75
TL20Kon	0.87	0.87	0.62	0.66	0.75	0.39
TH34Eur	0.92	0.92	0.67	0.63	0.81	0.67
UH29Eur	0.94	0.94	0.67	0.64	0.81	0.78
TH34vy	0.61	0.61	0.35	0.34	0.44	0.16
UL39Kon	0.95	0.95	0.56	0.57	0.64	0.59

B.2 Biased Data

Table B.2: The Kappa scores for the biased 500Hz samples.

Sample	CONS	ROB	VEL	BIT
UH33vy	1	1	1	1
UL43Rome	1	1	1	1
TL28Kon	1	1	1	1
UL23Eur	1	1	1	1
UL31Kon	1	1	1	1
UH27vy	1	1	1	0.79
UH21Rome	1	1	1	1
TL20Kon	1	1	1	1
TH34Eur	1	1	1	1
UH29Eur	1	1	1	0.97
TH34vy	1	1	1	1
UL39Kon	1	1	1	1

B.3 Lowered Precision

Table B.3: The Kappa scores for the 500Hz samples with a lowered precision.

Sample	CONS	ROB	VEL	BIT
UH33vy	0.46	0.75	0.72	0.25
UL43Rome	0.82	0.80	0.79	0.48
TL28Kon	0.87	0.56	0.66	0.25
UL23Eur	0.84	0.74	0.73	0.47
UL31Kon	0.94	0.85	0.84	0.63
UH27vy	0.64	0.80	0.72	0.35
UH21Rome	0.61	0.71	0.68	0.26
TL20Kon	0.72	0.71	0.73	0.20
TH34Eur	0.56	0.75	0.74	0.26
UH29Eur	0.76	0.81	0.74	0.37
TH34vy	0.06	0.53	0.62	0.07
UL39Kon	0.81	0.91	0.89	0.57

B.4 Data Loss

Table B.4: The Kappa scores for the 500Hz samples with introduced data loss.

Sample	CONS	ROB	VEL	BIT
UH33vy	0.24	1	1	0.90
UL43Rome	0.35	0.99	0.98	0.89
TL28Kon	0.45	0.96	0.97	0.96
UL23Eur	0.48	1	1	0.96
UL31Kon	0.74	1	0.99	0.98
UH27vy	0.36	0.99	1	0.95
UH21Rome	0.25	0.98	0.98	0.90
TL20Kon	0.36	1	0.99	0.89
TH34Eur	0.34	0.99	1	0.88
UH29Eur	0.35	1	1	0.94
TH34vy	0.13	0.99	0.99	0.88
UL39Kon	0.58	1	0.99	0.97

B.5 Lowered Frequency

Table B.5: The Kappa scores for the samples with a lowered frequency of 62.5Hz.

Sample	CONS	ROB	VEL	BIT
UH33vy	0.95	0.92	0.88	0.63
UL43Rome	0.96	0.90	0.85	0.81
TL28Kon	0.82	0.90	0.80	0.61
UL23Eur	0.89	0.85	0.83	0.75
UL31Kon	0.95	0.95	0.91	0.88
UH27vy	0.94	0.88	0.90	0.70
UH21Rome	0.90	0.91	0.85	0.71
TL20Kon	0.87	0.90	0.83	0.47
TH34Eur	0.96	0.91	0.88	0.69
UH29Eur	0.91	0.93	0.89	0.71
TH34vy	0.89	0.89	0.91	0.28
UL39Kon	0.90	0.97	0.90	0.70

B.6 Everything Added

Table B.6: The Kappa scores for the samples, where the frequency has been lowered an the quality has been lowered in all aspects.

Sample	CONS	ROB	VEL	BIT
UH33vy	0.67	0.80	0.80	0.38
UL43Rome	0.66	0.84	0.79	0.59
TL28Kon	0.56	0.65	0.69	0.24
UL23Eur	0.67	0.78	0.77	0.45
UL31Kon	0.80	0.89	0.83	0.74
UH27vy	0.68	0.82	0.80	0.27
UH21Rome	0.59	0.81	0.78	0.28
TL20Kon	0.65	0.81	0.79	0.28
TH34Eur	0.63	0.80	0.81	0.37
UH29Eur	0.58	0.84	0.81	0.48
TH34vy	0.25	0.49	0.68	0.05
UL39Kon	0.75	0.89	0.87	0.59

Appendix C

Programming Code

C.1 Retrieve Neighbours Function

The code shown below is for DBSCAN-ROBUST, to obtain DBSCAN-CONSECUTIVE we set cutOffStep = 0.

```
def RetrieveNeighbours(G, i, eps, f):

    timespan = 0.2
    upper = math.floor((i+f*timespan))
    lower = math.ceil((i-f*timespan))
    if upper > (n-1): upper = (n-1)
    if lower < 0: lower = 0
    cutOffStep = math.ceil(f*0.008) # 8ms can contain up to this many
       points.
    Neighbours = list(range(lower, (upper+1)))
    Neighbours2 = list()

    for p in Neighbours:
        if (G["x-coordinate(gaze)"][p] != 0) & (G["y-coordinate(gaze)"][
           p] != 0) : #don't consider blinks and missing measurements
            dist = math.sqrt((G["x-coordinate(gaze)"][i] - G["x-
               coordinate(gaze)"][p])**2 + (G["y-coordinate(gaze)"][i] -
                G["y-coordinate(gaze)"][p])**2)
            if dist <= eps:
                Neighbours2.append(p)
    Neighbours = Neighbours2
    Neighbours.sort()

    #Finding where the neighbourhood has been left for more than 8ms
    cutoffs = []
    for q in range(len(Neighbours)-1):
        if Neighbours[q+1]-Neighbours[q] > (cutOffStep+1):
            cutoffs.append(Neighbours[q])
    #Finding the upper and lowerbound of the neighbourhood using the
        cutoffs
    upperBound = -1
    lowerBound = -1

    if len(cutoffs) == 0:
        return Neighbours

    if min(cutoffs) > i: lowerBound = 0

    if min(cutoffs) == i:
        lowerBound = 0
        upperBound = Neighbours.index(i)
        return Neighbours[lowerBound:(upperBound+1)]

    if max(cutoffs) < i: upperBound = len(Neighbours)-1

    if lowerBound == -1:
        cutoffs.sort(reverse=True)
        cutoffvalue = cutoffs[next(x for x, val in enumerate(cutoffs) if
            val < i)]
        lowerBound = Neighbours.index(cutoffvalue)+1

    if upperBound == -1:
        cutoffs.sort(reverse=False)
        cutoffvalue = cutoffs[next(x for x, val in enumerate(cutoffs) if
            val >= i)]
        upperBound = Neighbours.index(cutoffvalue)

    Neighbours = Neighbours[lowerBound:(upperBound+1)]

    return Neighbours
```

C.2 DBSCAN Algorithm

```
#The place where the eye-tracking data is stored, each sample must be in
    a separate csv file
path = "~/*labelled_MN.csv"

timespan = 0.2
#The possible range of values for eps in mm
eps = np.linspace(0, 30, 301)
files = glob.glob(path)

for fname in files:
    print(fname)
```

```
# Load in data
eye_track = pd.read_csv(fname, names = ['timestamp', '
    horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate(
    gaze)', 'y-coordinate(gaze)', 'label'], header = None)
eye_track['timestamp'] = eye_track['timestamp'].divide(10**6) #Time
    from microseconds to seconds
eye_track['x-coordinate␣(gaze)'] = eye_track['x-coordinate␣(gaze)'
    ]/1024*380 #Turn x and y into mm values, (0, 0) is lower left
    coordinate of screen
eye_track['y-coordinate␣(gaze)'] = eye_track['y-coordinate␣(gaze)'
    ]/768*300

n = len(eye_track)
f=round(10/(eye_track["timestamp"][10]-eye_track["timestamp"][0]))
print(f)

#The minimum fixation duration
RangeMinPts=[math.floor(0.055*f)]

#Determining eps for minpts
for MinPts in RangeMinPts:
    Dist = pd.Series(0.0, index=list(range(n)))
    for j in range(math.ceil(0+f*timespan),math.floor(n-f*timespan))
        :
        if (eye_track["x-coordinate(gaze)"][j] != 0) & (eye_track["y
            -coordinate(gaze)"][j] != 0): #Exclude blinks and missing
             measurements
            Neighbours = list(range(math.ceil((j-f*timespan)), math.
                floor((j+f*timespan+1))))
            for point in Neighbours:
                if (eye_track["x-coordinate(gaze)"][point] == 0) or
                    (eye_track["y-coordinate(gaze)"][point] == 0): #
                    Remove blinks and missing measurements from a
                    neighbourhood.
                     Neighbours.remove(point)
            Neighbours.sort(key = lambda p: (eye_track["x-coordinate
                (gaze)"][j] - eye_track["x-coordinate(gaze)"][p])**2
                + (eye_track["y-coordinate(gaze)"][j] - eye_track["y-
                coordinate(gaze)"][p])**2)
            Dist[j] = math.sqrt((eye_track["x-coordinate(gaze)"][j]
                - eye_track["x-coordinate(gaze)"][Neighbours[MinPts
                ]])**2 + (eye_track["y-coordinate(gaze)"][j] -
                eye_track["y-coordinate(gaze)"][Neighbours[MinPts]])
                **2)
        else: Dist[j] = -101

    DistCutOff = Dist[(math.ceil(0+f*timespan)):math.floor(n-f*
        timespan)]
```

```
DistCutOff = DistCutOff.sort_values()

excess = pd.Series(0, index = list(range(len(eps))))
for r in range(len(eps)):
    excess[r] = sum(dis > eps[r] for dis in DistCutOff if dis !=
        -101)

kn = kneed.KneeLocator(eps, excess, curve = "convex", direction
   = 'decreasing', interp_method="polynomial", polynomial_degree
   =7, online=True)

# The optimised value for the dispersion threshold eps in mm
opteps = kn.knee
print(opteps)

#Cluster the sample using opteps
initialiseNoiseLabel = [0]*n #0 is not noise, 1 is noise
initialiseTrajClusterLabel = [0]*n #0 is no cluster, other
   numbers are respective clusters
errorPointInFix = [0]*n #0 is not errorpoint in fix, 1 is
   errorpoint in fix
corePoint = [0]*n #0 is not core point, 1 is core point
eye_track['Noise'] = initialiseNoiseLabel
eye_track['Cluster'] = initialiseTrajClusterLabel
eye_track['ErrorInFix'] = errorPointInFix
eye_track['corePoint'] = corePoint

#The actual clustering
ClusterNo = 0
arc = opteps
for index in range(n):
    if (eye_track["Noise"][index] == 0) & (eye_track["Cluster"][
       index] == 0) & (eye_track["x-coordinate(gaze)"][index] !=
        0) & (eye_track["y-coordinate(gaze)"][index] != 0):
        Neighborhood = RetrieveNeighbours(G=eye_track, i = index
           , eps = arc, f= f)
        if len(Neighborhood) == 0:
            eye_track["Noise"][index] == 1
            continue

        minNeighborhood = min(Neighborhood)
        maxNeighborhood = max(Neighborhood)

        #Compare size of neighbourhood to MinPts
        if (maxNeighborhood-minNeighborhood+1) < MinPts:
            eye_track["Noise"][index] = 1
        else:
            fullNeighborHood = list(range(minNeighborhood,(
   maxNeighborhood+1)))
ClusterNo += 1
eye_track["Cluster"][index] = ClusterNo
eye_track['corePoint'][index] = 1
potentialClusterPointCheck = []
for point in fullNeighborHood:
    if eye_track["Cluster"][point] == 0:
        eye_track["Cluster"][point] = ClusterNo
        eye_track["Noise"][point] = 0
        if point > index:
            potentialClusterPointCheck.append(point)
                #The points before index have
                already been checked
        if (Neighborhood.__contains__(point) ==
           False):
            eye_track["ErrorInFix"][point] = 1
            if point > index:
                potentialClusterPointCheck.remove(
                   point)

#Checking all points in the neighbourhood of a core
   point to see if they are core points and do this
   iteratively till every point of (extended)
   neighbourhood has been checked.
while len(potentialClusterPointCheck) != 0:
    potClustPoint = potentialClusterPointCheck.pop
       (0)
    Neighborhood = RetrieveNeighbours(G=eye_track, i
        = potClustPoint, eps = arc, f= f)
    minNeighborhood = min(Neighborhood)
    maxNeighborhood = max(Neighborhood)

    if (maxNeighborhood-minNeighborhood+1) >= MinPts
       :
        eye_track['corePoint'][potClustPoint] = 1
        fullNeighborHood = list(range(
           minNeighborhood,(maxNeighborhood+1)))
        for point in fullNeighborHood:
            if eye_track["Cluster"][point] == 0:
                eye_track["Cluster"][point] =
                   ClusterNo
                eye_track["Noise"][point] = 0
                if point > potClustPoint:
                    potentialClusterPointCheck.
                       append(point) #The points
                       before index have already
                       been checked
                if (Neighborhood.__contains__(point)
                                                    == False):
                                                    eye_track["ErrorInFix"][point] =
                                                        1
                                                    if point > index:
                                                        potentialClusterPointCheck.
                                                            remove(point)
    #Save the csv with cluster labels
    docstring = fname + 'DBSCANNED.csv'
    eye_track.to_csv(docstring)
```

C.3 Velocity Post-Processing

```
path = "~/*DBSCANNED.csv"
files = glob.glob(path)

for fname in files:
    print(fname)

    # Load in data
    eye_track = pd.read_csv(fname, header = 0, index_col=0)
    n = len(eye_track)
    initVel = [0]*n
    eye_track["Velocity"] = initVel

    #Calculate velocities, whilst account for erroneous measurements
    for i in range(1,n):
        if (eye_track["ErrorInFix"][i] == 1) or (eye_track["x-coordinate
            (gaze)"][i] == 0) or (eye_track["y-coordinate(gaze)"][i] ==
            0):
             continue
        elif (eye_track["ErrorInFix"][i-1] == 1) or (eye_track["x-
            coordinate(gaze)"][i-1] == 0) or (eye_track["y-coordinate(
            gaze)"][i-1] == 0):
             firstGoodPoint = i-1
             while True:
                 firstGoodPoint = firstGoodPoint -1
                 if firstGoodPoint < 0:
                     break
                 if (eye_track["ErrorInFix"][firstGoodPoint] != 1) & (
                     eye_track["x-coordinate(gaze)"][firstGoodPoint] != 0)
                      & (eye_track["y-coordinate(gaze)"][firstGoodPoint]
                     != 0):
                      eye_track["Velocity"][i] = math.sqrt((eye_track["x-
                          coordinate(gaze)"][i]-eye_track["x-coordinate(
                          gaze)"][firstGoodPoint])**2+(eye_track["y-
                          coordinate(gaze)"][i]-eye_track["y-coordinate(
                          gaze)"][firstGoodPoint])**2)/((eye_track["
```

```
                    timestamp"][i]-eye_track["timestamp"][
                    firstGoodPoint]))
                break
    else:
        eye_track["Velocity"][i] = math.sqrt((eye_track["x-
            coordinate(gaze)"][i]-eye_track["x-coordinate(gaze)"][i
            -1])**2+(eye_track["y-coordinate(gaze)"][i]-eye_track["y-
            coordinate(gaze)"][i-1])**2)/((eye_track["timestamp"][i]-
            eye_track["timestamp"][i-1]))

f = round(10/(eye_track["timestamp"][10]-eye_track["timestamp"][0]))
print(f)

#Calculating the number of points to check on either side and the
   velocity threshold
pointsEitherSide = math.ceil(0.02*f)
distToScreen = 670
minSaccVelDegr = 45.4
minSaccVelmm = minSaccVelDegr*math.pi*distToScreen/180

fix_on = []
fix_off = []
n = len(eye_track)

#Find the individual fixations to post-process for velocity
for i in range(n):
    if i == 0:
        if eye_track['Cluster'][i] != 0:
            fix_on.append(i)
        continue
    if (eye_track['Cluster'][i] != 0) & (eye_track['Cluster'][i-1]
       == 0):
        fix_on.append(i)
    if (eye_track['Cluster'][i] == 0 ) & (eye_track['Cluster'][i-1]
       != 0):
        fix_off.append(i-1)

if len(fix_on) > len(fix_off):
    fix_off.append(n-1)

#This deals with neighbouring fixations
for i in range(1,n):
    if (eye_track['Cluster'][i] != eye_track['Cluster'][i-1]) & (
       eye_track['Cluster'][i] != 0) & (eye_track['Cluster'][i-1] !=
        0):
        fix_off.append(i-1)
        fix_on.append(i)
```

```
#Sort the fixations after appending the values to account for
    neighbouring fixations
fix_on.sort()
fix_off.sort()

newFix_on = []
newFix_off = []

#Check the points at the start and end of each fixation
for startPoint in fix_on:
    potentialsaccadepoints = list(range(startPoint, startPoint+
        pointsEitherSide))
    newFix_onPoint = startPoint
    while len(potentialsaccadepoints) > 0:
        potentialsaccadepoint = potentialsaccadepoints.pop()
        if eye_track["Velocity"][potentialsaccadepoint] >
            minSaccVelmm:
            newFix_onPoint = potentialsaccadepoint+1
            break
    newFix_on.append(newFix_onPoint)

    #Remove the saccade point and the points before it from the
        fixation
    if newFix_onPoint  != startPoint:
        eye_track["Cluster"][startPoint:newFix_onPoint] = 0

for endPoint in fix_off:
    potentialsaccadepoints = list(range(endPoint-pointsEitherSide+1,
        endPoint+1))
    newFix_offPoint = endPoint
    while len(potentialsaccadepoints) > 0:
        potentialsaccadepoint = potentialsaccadepoints.pop(0)
        if eye_track["Velocity"][potentialsaccadepoint] >
            minSaccVelmm:
            newFix_offPoint = potentialsaccadepoint-1
            break
    newFix_off.append(newFix_offPoint)

    #Remove the saccade point and the points after it from the
        fixation
    if newFix_offPoint != endPoint:
        eye_track["Cluster"][(newFix_offPoint+1):(endPoint+1)] = 0

#Save the csv with cluster labels
docstring = fname + 'VelocityPostProcessed.csv'
eye_track.to_csv(docstring)
```

C.4 Artificially Lowering the Data Quality

A seed must be set, to ensure reproducibility. Here the seed was set as random.seed(10).

C.4.1 Lowered Frequency

```
path = "~/*labelled_MN.csv"
files = glob.glob(path)
fix_durations_MN = []

for fname in files:
    print(fname)

    #Load in one set of data
    eye_track = pd.read_csv(fname, names = ['timestamp', '
       horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate(
       gaze)', 'y-coordinate(gaze)', 'label'], header = None)
    n = len(eye_track)

    #Include every eigth point
    nmod8 = int((n-n%8)/8)
    nmodindeces = []
    for i in range(nmod8):
        nmodindeces.append(i*8)
    eye_track_mod = eye_track.iloc[nmodindeces]

    #Save the csv with modified data
    docstring = fname + "62.5Hz.csv"
    eye_track_mod.to_csv(docstring, header=False, index=False)
```

C.4.2 Biased Data

```
path = "~/*labelled_MN.csv"
files = glob.glob(path)
fix_durations_MN = []

for fname in files:
    print(fname)

    #Load in one set of data
    eye_track = pd.read_csv(fname, names = ['timestamp', '
       horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate(
       gaze)', 'y-coordinate(gaze)', 'label'], header = None)

    #Shift the data 1.05degree to the left and 0.65 degree down
    xshift = 670*math.pi*1.05/180/380*1024
    yshift = 670*math.pi*0.65/180/300*768
    n = len(eye_track)
    for i in range(n):
        if (eye_track['x-coordinate(gaze)'][i] != 0) & (eye_track['y-
            coordinate(gaze)'][i] != 0): #Missing measurements have
            coordinate (0,0), do not move these
            eye_track['x-coordinate␣(gaze)'].loc[i] = eye_track['x-
                coordinate(gaze)'].loc[i]-xshift
            eye_track['y-coordinate␣(gaze)'].loc[i] = eye_track['y-
                coordinate(gaze)'].loc[i]-yshift

    #Save the csv with modified data
    docstring = fname + "Biased.csv"
    eye_track.to_csv(docstring, header=False, index=False)
```

C.4.3 Missing Measurements

```
path = "~/*labelled_MN.csv"
files = glob.glob(path)
fix_durations_MN = []

for fname in files:
    print(fname)

    #Load in one set of data
    eye_track = pd.read_csv(fname, names = ['timestamp', '
        horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate(
        gaze)', 'y-coordinate(gaze)', 'label'], header = None)
    n = len(eye_track)

    #Generate missing measurements from a uniform distribution with
        probability 0.03
    rand = np.random.uniform(size=n)
    binary = []
    for ran in rand:
        if ran <= 0.03:
            binary.append(1)
        else: binary.append(0)
    for i in range(n):
        if binary[i] == 1:
            eye_track['x-coordinate(gaze)'].loc[i] = 0
            eye_track['y-coordinate(gaze)'].loc[i] = 0

    #Save the csv with modified data
    docstring = fname + "Errors15Added.csv"
    eye_track.to_csv(docstring, header=False, index=False)
```

C.4.4 Lowered Precision

```
path = "~/*labelled_MN.csv"
files = glob.glob(path)
fix_durations_MN = []

#Standard deviations for the standard normal distributions we generate
   noise from
sigmax = 670*math.pi*0.163/180/380*1024
sigmay = 670*math.pi*0.231/180/300*768

for fname in files:
    print(fname)

    #Load in one set of data
    eye_track = pd.read_csv(fname, names = ['timestamp', '
       horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate(
       gaze)', 'y-coordinate(gaze)', 'label'], header = None)
    n = len(eye_track)

    #Generate and add noise to each measurement, that does not have the
       coordinates of a missing measurement
    for i in range(n):
        if (eye_track['x-coordinate(gaze)'][i] != 0) & (eye_track['y-
           coordinate(gaze)'][i] != 0):
            eye_track['x-coordinate(gaze)'].loc[i] = eye_track['x-
               coordinate(gaze)'].loc[i] + np.random.normal(loc=0.0,
               scale = sigmax)
            eye_track['y-coordinate(gaze)'].loc[i] = eye_track['y-
               coordinate(gaze)'].loc[i] + + np.random.normal(loc=0.0,
               scale = sigmay)

    #Save the csv with modified data
    docstring = fname + "NoiseAdded.csv"
    eye_track.to_csv(docstring, header=False, index=False)
```

C.4.5 Everything Together

```
path = "~/*labelled_MN.csv"
files = glob.glob(path)
fix_durations_MN = []

#Standard deviations for generating noise to lower precision
sigmax = 670*math.pi*0.163/180/380*1024
sigmay = 670*math.pi*0.231/180/300*768

for fname in files:
    print(fname)

    #Load in one set of data
```

```
eye_track = pd.read_csv(fname, names = ['timestamp', '
    horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate␣
    (gaze)', 'y-coordinate␣(gaze)', 'label'], header = None)
n = len(eye_track)

#Lower Freq First
nmod8 = int((n-n%8)/8)
nmodindeces = []
for i in range(nmod8):
    nmodindeces.append(i*8)
eye_track_mod = eye_track.iloc[nmodindeces]
eye_track_mod = eye_track_mod.reset_index(drop=True)
n = len(eye_track_mod)

#Bias Values
xshift = 670*math.pi*1.05/180/380*1024
yshift = 670*math.pi*0.65/180/300*768

#Missing measurements
rand = np.random.uniform(size=n)
binary = []
for ran in rand:
    if ran <= 0.03:
        binary.append(1)
    else: binary.append(0)

for i in range(n):
    #Lower Precision and Introduce Bias
    if (eye_track_mod['x-coordinate␣(gaze)'][i] != 0) & (
        eye_track_mod['y-coordinate␣(gaze)'][i] != 0):
        eye_track_mod['x-coordinate␣(gaze)'].loc[i] = eye_track_mod[
            'x-coordinate␣(gaze)'].loc[i] + np.random.normal(loc=0.0,
             scale = sigmax) - xshift
        eye_track_mod['y-coordinate␣(gaze)'].loc[i] = eye_track_mod[
            'y-coordinate␣(gaze)'].loc[i] + + np.random.normal(loc
            =0.0, scale = sigmay) - yshift

    #Create Missing Measurements
    if binary[i] == 1:
        eye_track_mod['x-coordinate␣(gaze)'].loc[i] = 0
        eye_track_mod['y-coordinate␣(gaze)'].loc[i] = 0

#Save the csv with modified data
docstring = fname + "EverythingAdded.csv"
eye_track_mod.to_csv(docstring, header=False, index=False)
```

C.5 Calculating Fixation Duration Properties

```
#Indicate the processed files you want to check the fixation duration
   properties for
path = "~/*labelled_MN.csvDBSCANNED.csv"
files = glob.glob(path)
fix_durations_algo = []

for fname in files:
    print(fname)

    # Load in one set of data
    eye_track = pd.read_csv(fname, header = 0, index_col=0)

    f = round(10/(eye_track["timestamp"][10]-eye_track["timestamp"][0]))

    #This can be used to filter out the samples with a too low frequency
    if f<250:
        print("f␣too␣small")
        continue

    fix_on = []
    fix_off = []
    n = len(eye_track)

    #Find the fixations, not accounting for neighbouring fixations
    for i in range(n):
        if i == 0:
            if eye_track['Cluster'][i] != 0:
                fix_on.append(i)
            continue
        if (eye_track['Cluster'][i] != 0) & (eye_track['Cluster'][i-1]
           == 0):
            fix_on.append(i)
        if (eye_track['Cluster'][i] == 0 ) & (eye_track['Cluster'][i-1]
           != 0):
            fix_off.append(i-1)

    #If there is a running fixation at the endpoint, create endpoint for
        that fixation at the end of the sample
    if len(fix_on) > len(fix_off):
        fix_off.append(n-1)

    # This deals with neighbouring fixations
    for i in range(1,n):
        if (eye_track['Cluster'][i] != eye_track['Cluster'][i-1]) & (
           eye_track['Cluster'][i] != 0) & (eye_track['Cluster'][i-1] !=
            0):
```

```
            fix_off.append(i-1)
            fix_on.append(i)
    fix_on.sort()
    fix_off.sort()

    for j in range(len(fix_on)):
        #Skip fixations that begin at the start or stop at the end
        if fix_on[j] == 0: continue
        if fix_off[j] == n-1: continue

        fix_dur = fix_off[j]-fix_on[j]+1
        totalfix_dur = totalfix_dur+fix_dur
        fix_durations_algo.append(fix_dur)

#Set frequency of the samples
f = 500
#Calculate mean, std dev and number of fixations
print(statistics.mean(fix_durations_algo)*1/f*1000)
print(statistics.stdev(fix_durations_algo)*1/f*1000)
print(len(fix_durations_algo))
```

C.6 Calculating Kappa

C.6.1 Compared to the Human Labellers

```
#Cohen's Kappa for unaltered data
import glob
import numpy as np
import pandas as pd
import sklearn
path = "/Users/boydvoet/Documents/Scriptie/*labelled_MN.csv"
eyeTrackingDataBIT = pd.read_csv('/Users/boydvoet/Documents/Scriptie/BIT
   /OneDrive_1_13-7-2023/BIT_fix_stampsNormalDataAnderssonParams.csv',
   sep=';', names=["identifier", "timestamp␣(ms)", "x-coord", "y-coord",
    "distance", "Cluster"])#, dtype={'ParticipantName': str, '
   RecordingName': str})

switch = [-1]
for i in range(len(eyeTrackingDataBIT)-1):
    if (eyeTrackingDataBIT['identifier'][i+1] != eyeTrackingDataBIT['
        identifier'][i]):
         switch.append(i)
switch.append(len(eyeTrackingDataBIT)-1)
mapper = {"/Users/boydvoet/Documents/Scriptie/
   UH47_img_Europe_labelled_MN.csv": 1,"/Users/boydvoet/Documents/
   Scriptie/TH34_img_Europe_labelled_MN.csv": 2, "/Users/boydvoet/
   Documents/Scriptie/TH34_img_vy_labelled_MN.csv": 3, "/Users/boydvoet/
```

```
    Documents/Scriptie/TL20_img_konijntjes_labelled_MN.csv": 4, "/Users/
    boydvoet/Documents/Scriptie/TL28_img_konijntjes_labelled_MN.csv": 5,
            "/Users/boydvoet/Documents/Scriptie/UH21_img_Rome_labelled_MN.
                csv": 6, "/Users/boydvoet/Documents/Scriptie/
                UH27_img_vy_labelled_MN.csv": 7, "/Users/boydvoet/Documents
                /Scriptie/UH29_img_Europe_labelled_MN.csv": 8, "/Users/
                boydvoet/Documents/Scriptie/UH33_img_vy_labelled_MN.csv":
                9,
            "/Users/boydvoet/Documents/Scriptie/
                UL23_img_Europe_labelled_MN.csv": 10, "/Users/boydvoet/
                Documents/Scriptie/UL31_img_konijntjes_labelled_MN.csv":
                11, "/Users/boydvoet/Documents/Scriptie/
                UL39_img_konijntjes_labelled_MN.csv": 12, "/Users/boydvoet/
                Documents/Scriptie/UL43_img_Rome_labelled_MN.csv": 13, "/
                Users/boydvoet/Documents/Scriptie/
                UL47_img_konijntjes_labelled_MN.csv": 14}
files = glob.glob(path)
labelsMNbinary = np.zeros(1)
lablesRAbinary = np.zeros(1)
lablesDBSCANROBbinary = np.zeros(1)
lablesDBSCANCONSbinary = np.zeros(1)
lablesDBSCANVELbinary = np.zeros(1)
lablesBITbinary = np.zeros(1)
kappas = np.zeros((6,12))
counter = 0

for fname in files:
    if (fname == "/Users/boydvoet/Documents/Scriptie/
        UL47_img_konijntjes_labelled_MN.csv") or (fname == "/Users/
        boydvoet/Documents/Scriptie/UH47_img_Europe_labelled_MN.csv"):
        continue
    print(fname)
    eye_track1 = pd.read_csv(fname, names = ['timestamp', '
        horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate␣
        (gaze)', 'y-coordinate␣(gaze)', 'label'], header = None)
    # print(eye_track1)
    y1 = eye_track1['label']
    for i in range(len(y1)):
        if y1[i] != 1: y1[i]=0
    fname2 = fname[:-6]+"RA.csv"
    # print(fname2)
    eye_track2 = pd.read_csv(fname2, names = ['timestamp', '
        horizontalPupilDiameter', 'verticalPupilDiameter', 'x-coordinate␣
        (gaze)', 'y-coordinate␣(gaze)', 'label'], header = None)
    y2 = eye_track2['label']
    for i in range(len(y2)):
        if y2[i] != 1: y2[i] = 0
    fname3 = fname+"DBSCANNEDCutoff=0.csv"
    eye_track3 = pd.read_csv(fname3, header = 0, index_col=0)
    y3 = eye_track3['Cluster']
    for i in range(len(y3)):
        if y3[i] != 0: y3[i] = 1
    fname4 = fname+"DBSCANNED.csv"
    eye_track4 = pd.read_csv(fname4, header = 0, index_col=0)
    y4 = eye_track4['Cluster']
    for i in range(len(y4)):
        if y4[i] != 0: y4[i] = 1
    fname5 = fname+"DBSCANNED.csvVelocityPostProcessed.csv"
    eye_track5 = pd.read_csv(fname5, header = 0, index_col=0)
    y5 = eye_track5['Cluster']
    for i in range(len(y5)):
        if y5[i] != 0: y5[i] = 1
    mapBIT = mapper[fname]-1
    eye_track6 = eyeTrackingDataBIT[(switch[mapBIT]+1):(switch[mapBIT
       +1]+1)]
    eye_track6 = eye_track6.reset_index()
    y6 = eye_track6["Cluster"]
    for i in range(len(y6)):
        if y6[i] != 0: y6[i] = 1

    #Kappa per sample
    kappas[0, counter] = (sklearn.metrics.cohen_kappa_score(y1,y2)+
       sklearn.metrics.cohen_kappa_score(y1,y1))/2
    kappas[1, counter] = (sklearn.metrics.cohen_kappa_score(y1,y2)+
       sklearn.metrics.cohen_kappa_score(y2,y2))/2
    kappas[2, counter] = (sklearn.metrics.cohen_kappa_score(y1,y3)+
       sklearn.metrics.cohen_kappa_score(y2,y3))/2
    kappas[3, counter] = (sklearn.metrics.cohen_kappa_score(y1,y4)+
       sklearn.metrics.cohen_kappa_score(y2,y4))/2
    kappas[4, counter] = (sklearn.metrics.cohen_kappa_score(y1,y5)+
       sklearn.metrics.cohen_kappa_score(y2,y5))/2
    kappas[5, counter] = (sklearn.metrics.cohen_kappa_score(y1,y6)+
       sklearn.metrics.cohen_kappa_score(y2,y6))/2

    #Kappa for all the samples together
    labelsMNbinary=np.append(labelsMNbinary, y1)
    lablesRAbinary=np.append(lablesRAbinary, y2)
    lablesDBSCANCONSbinary = np.append(lablesDBSCANCONSbinary, y3)
    lablesDBSCANROBbinary = np.append(lablesDBSCANROBbinary, y4)
    lablesDBSCANVELbinary = np.append(lablesDBSCANVELbinary, y5)
    lablesBITbinary = np.append(lablesBITbinary, y6)

    counter+=1

labelsMNbinary = labelsMNbinary[1:]
lablesRAbinary = lablesRAbinary[1:]
```

```
lablesDBSCANROBbinary = lablesDBSCANROBbinary[1:]
lablesDBSCANCONSbinary = lablesDBSCANCONSbinary[1:]
lablesDBSCANVELbinary = lablesDBSCANVELbinary[1:]
lablesBITbinary = lablesBITbinary[1:]

kappaMN = kappas[0, :]
kappaRA = kappas[1, :]
kappaCONS = kappas[2, :]
kappaROB =  kappas[3, :]
kappaVEL = kappas[4, :]
kappaBIT = kappas[5,:]

#Sign Test. Divide p-values by 2 to account for one-sided test
import statsmodels.stats.descriptivestats

print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaMN))
print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaVEL))
print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaCONS)
   )
print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaROB))

print(statsmodels.stats.descriptivestats.sign_test(kappaVEL - kappaMN))
print(statsmodels.stats.descriptivestats.sign_test(kappaVEL - kappaCONS)
   )
print(statsmodels.stats.descriptivestats.sign_test(kappaVEL - kappaROB))

print(statsmodels.stats.descriptivestats.sign_test(kappaROB - kappaMN))
print(statsmodels.stats.descriptivestats.sign_test(kappaROB - kappaCONS)
   )

print(statsmodels.stats.descriptivestats.sign_test(kappaCONS - kappaMN))

#Kappa scores for the entire sample
print((sklearn.metrics.cohen_kappa_score(labelsMNbinary,lablesRAbinary)+
   sklearn.metrics.cohen_kappa_score(labelsMNbinary,labelsMNbinary))/2)
print((sklearn.metrics.cohen_kappa_score(lablesRAbinary,lablesRAbinary)+
   sklearn.metrics.cohen_kappa_score(lablesRAbinary,labelsMNbinary))/2)
print((sklearn.metrics.cohen_kappa_score(lablesDBSCANCONSbinary,
   lablesRAbinary)+sklearn.metrics.cohen_kappa_score(
   lablesDBSCANCONSbinary,labelsMNbinary))/2)
print((sklearn.metrics.cohen_kappa_score(lablesDBSCANROBbinary,
   lablesRAbinary)+sklearn.metrics.cohen_kappa_score(
   lablesDBSCANROBbinary,labelsMNbinary))/2)
print((sklearn.metrics.cohen_kappa_score(lablesDBSCANVELbinary,
   lablesRAbinary)+sklearn.metrics.cohen_kappa_score(
   lablesDBSCANVELbinary,labelsMNbinary))/2)
print((sklearn.metrics.cohen_kappa_score(lablesBITbinary,lablesRAbinary)
   +sklearn.metrics.cohen_kappa_score(lablesBITbinary,labelsMNbinary))
   /2)
```

C.6.2 Compare Output on Worsened Data to Output on Normal Data

```
#Kappa for worsened data
import glob
import pandas as pd
import numpy as np
path = "/Users/boydvoet/Documents/Scriptie/*labelled_MN.csv"
worsenedtype = "EverythingAdded.csv"
eye_trackBITFull = pd.read_csv('/Users/boydvoet/Documents/Scriptie/BIT/
   OneDrive_1_13-7-2023/BIT_fix_stamps500HzNormalData.csv', sep=';',
   names=["identifier", "timestamp␣(ms)", "x-coord", "y-coord", "
   distance", "Cluster"])
#Needs manual adjustment to align with the worsened type of the other
   data
eye_trackBITFullWorsened = pd.read_csv('/Users/boydvoet/Documents/
   Scriptie/BIT/OneDrive_1_13-7-2023/BIT_fix_stamps_EverythingAdded.csv'
   , sep=';', names=["identifier", "timestamp␣(ms)", "x-coord", "y-coord
   ", "distance", "Cluster"])
switch = [-1]
for i in range(len(eye_trackBITFull)-1):
     if (eye_trackBITFull['identifier'][i+1] != eye_trackBITFull['
        identifier'][i]):
         switch.append(i)
switch.append(len(eye_trackBITFull)-1)
switchWorsened = [-1]
for i in range(len(eye_trackBITFullWorsened)-1):
     if (eye_trackBITFullWorsened['identifier'][i+1] !=
        eye_trackBITFullWorsened['identifier'][i]):
         switchWorsened.append(i)
switchWorsened.append(len(eye_trackBITFullWorsened)-1)

mapper = {"/Users/boydvoet/Documents/Scriptie/
   TH34_img_Europe_labelled_MN.csv": 1, "/Users/boydvoet/Documents/
   Scriptie/TH34_img_vy_labelled_MN.csv": 2, "/Users/boydvoet/Documents/
   Scriptie/TL20_img_konijntjes_labelled_MN.csv": 3, "/Users/boydvoet/
   Documents/Scriptie/TL28_img_konijntjes_labelled_MN.csv": 4,
          "/Users/boydvoet/Documents/Scriptie/UH21_img_Rome_labelled_MN.
             csv": 5, "/Users/boydvoet/Documents/Scriptie/
             UH27_img_vy_labelled_MN.csv": 6, "/Users/boydvoet/Documents
             /Scriptie/UH29_img_Europe_labelled_MN.csv": 7, "/Users/
             boydvoet/Documents/Scriptie/UH33_img_vy_labelled_MN.csv":
             8,
          "/Users/boydvoet/Documents/Scriptie/
             UL23_img_Europe_labelled_MN.csv": 9, "/Users/boydvoet/
             Documents/Scriptie/UL31_img_konijntjes_labelled_MN.csv":
             10, "/Users/boydvoet/Documents/Scriptie/
```

```
            UL39_img_konijntjes_labelled_MN.csv": 11, "/Users/boydvoet/
            Documents/Scriptie/UL43_img_Rome_labelled_MN.csv": 12}
files = glob.glob(path)

lablesDBSCANROBbinary = np.zeros(1)
lablesDBSCANROBbinaryWorsened = np.zeros(1)
lablesDBSCANCONSbinary = np.zeros(1)
lablesDBSCANCONSbinaryWorsened = np.zeros(1)
lablesDBSCANVELbinary = np.zeros(1)
lablesDBSCANVELbinaryWorsened = np.zeros(1)
lablesBITbinary = np.zeros(1)
lablesBITbinaryWorsened = np.zeros(1)

counter = 0

kappas = np.zeros((4,12))
for fname in files:
    #Exclude 200Hz files
    if (fname == "/Users/boydvoet/Documents/Scriptie/
       UH47_img_Europe_labelled_MN.csv") or (fname == "/Users/boydvoet/
       Documents/Scriptie/UL47_img_konijntjes_labelled_MN.csv"):
       continue

    #DBSCAN-Consecutive
    fnameCONS = fname+"DBSCANNEDCutoff=0.csv"
    fnameCONSWorsened = fname+worsenedtype+"DBSCANNEDCutoff=0.csv"
    eye_trackCONS = pd.read_csv(fnameCONS, header = 0, index_col=0)
    eye_trackCONSWorsened = pd.read_csv(fnameCONSWorsened, header = 0,
       index_col=0)
    eye_trackCONSmod = []

    for timestamp in eye_trackCONSWorsened['timestamp']:
        indexeyetrack = eye_trackCONS['timestamp'][eye_trackCONS['
           timestamp']==timestamp].index[0]
        eye_trackCONSmod.append(eye_trackCONS.iloc[indexeyetrack])

    eye_trackCONSmod = pd.DataFrame(eye_trackCONSmod, columns = ["
       timestamp", "horizontalPupilDiameter", "verticalPupilDiameter", "
       x-coordinate␣(gaze)", "y-coordinate␣(gaze)", "label", "Noise", "
       Cluster", "ErrorInFix", "corePoint"])
    eye_trackCONSmod = eye_trackCONSmod.reset_index(drop=True)
    eye_trackCONS = eye_trackCONSmod
    yCONS = eye_trackCONS['Cluster']
    yCONSWorsened = eye_trackCONSWorsened['Cluster']
    for i in range(len(yCONS)):
        if yCONS[i] != 0: yCONS[i] = 1
    for i in range(len(yCONSWorsened)):
        if yCONSWorsened[i] != 0: yCONSWorsened[i] = 1
```

```
#DBSCAN-Robust
fnameROB = fname+"DBSCANNED.csv"
fnameROBWorsened = fname+worsenedtype+"DBSCANNED.csv"
eye_trackROB = pd.read_csv(fnameROB, header = 0, index_col=0)
eye_trackROBWorsened = pd.read_csv(fnameROBWorsened, header = 0,
   index_col=0)
eye_trackROBmod = []

for timestamp in eye_trackROBWorsened['timestamp']:
    indexeyetrack = eye_trackROB['timestamp'][eye_trackROB['
       timestamp']==timestamp].index[0]
    eye_trackROBmod.append(eye_trackROB.iloc[indexeyetrack])

eye_trackROBmod = pd.DataFrame(eye_trackROBmod, columns = ["
   timestamp", "horizontalPupilDiameter", "verticalPupilDiameter", "
   x-coordinate␣(gaze)", "y-coordinate␣(gaze)", "label", "Noise", "
   Cluster", "ErrorInFix", "corePoint"])
eye_trackROBmod = eye_trackROBmod.reset_index(drop=True)
eye_trackROB = eye_trackROBmod
yROB = eye_trackROB['Cluster']
yROBWorsened = eye_trackROBWorsened['Cluster']
for i in range(len(yROB)):
    if yROB[i] != 0: yROB[i] = 1
for i in range(len(yROBWorsened)):
    if yROBWorsened[i] != 0: yROBWorsened[i] = 1

#DBSCAN-Velocity
fnameVEL = fname+"DBSCANNED.csvVelocityPostProcessed.csv"
fnameVELWorsened = fname+worsenedtype+"DBSCANNED.
   csvVelocityPostProcessed.csv"
eye_trackVEL = pd.read_csv(fnameVEL, header = 0, index_col=0)
eye_trackVELWorsened = pd.read_csv(fnameVELWorsened, header = 0,
   index_col=0)
eye_trackVELmod = []

for timestamp in eye_trackVELWorsened['timestamp']:
    indexeyetrack = eye_trackVEL['timestamp'][eye_trackVEL['
       timestamp']==timestamp].index[0]
    eye_trackVELmod.append(eye_trackVEL.iloc[indexeyetrack])

eye_trackVELmod = pd.DataFrame(eye_trackVELmod, columns = ["
   timestamp", "horizontalPupilDiameter", "verticalPupilDiameter", "
   x-coordinate␣(gaze)", "y-coordinate␣(gaze)", "label", "Noise", "
   Cluster", "ErrorInFix", "corePoint"])
eye_trackVELmod = eye_trackVELmod.reset_index(drop=True)
eye_trackVEL = eye_trackVELmod
yVEL = eye_trackVEL['Cluster']
```

```
yVELWorsened = eye_trackVELWorsened['Cluster']
for i in range(len(yVEL)):
    if yVEL[i] != 0: yVEL[i] = 1
for i in range(len(yVELWorsened)):
    if yVELWorsened[i] != 0: yVELWorsened[i] = 1

#BIT
mapBIT = mapper[fname]-1
eye_trackBIT = eye_trackBITFull[(switch[mapBIT]+1):(switch[mapBIT
   +1]+1)]
eye_trackBIT = eye_trackBIT.reset_index(drop=True)
eye_trackBITWorsened = eye_trackBITFullWorsened[(switchWorsened[
   mapBIT]+1):(switchWorsened[mapBIT+1]+1)]
eye_trackBITWorsened = eye_trackBITWorsened.reset_index(drop=True)
eye_trackBITmod = []
for timestamp in eye_trackBITWorsened['timestamp␣(ms)']:
    indexeyetrack = eye_trackBIT['timestamp␣(ms)'][eye_trackBIT['
       timestamp␣(ms)']==timestamp].index[0]
    eye_trackBITmod.append(eye_trackBIT.iloc[indexeyetrack])

eye_trackBITmod = pd.DataFrame(eye_trackBITmod, columns = ["
   identifier", "timestamp␣(ms)", "x-coord", "y-coord", "distance",
   "Cluster"])
eye_trackBITmod = eye_trackBITmod.reset_index(drop=True)
eye_trackBIT = eye_trackBITmod


yBIT = eye_trackBIT['Cluster']
yBITWorsened = eye_trackBITWorsened['Cluster']
for i in range(len(yBIT)):
    if yBIT[i] != 0: yBIT[i] = 1
for i in range(len(yBITWorsened)):
    if yBITWorsened[i] != 0: yBITWorsened[i] = 1

#Kappa per sample
kappas[0, counter] = sklearn.metrics.cohen_kappa_score(yCONS,
   yCONSWorsened)
kappas[1, counter] = sklearn.metrics.cohen_kappa_score(yROB,
   yROBWorsened)
kappas[2, counter] = sklearn.metrics.cohen_kappa_score(yVEL,
   yVELWorsened)
kappas[3, counter] = sklearn.metrics.cohen_kappa_score(yBIT,
   yBITWorsened)

#For the calculation of Kappa for all samples appended
lablesDBSCANROBbinary = np.append(lablesDBSCANROBbinary, yROB)
lablesDBSCANROBbinaryWorsened = np.append(
   lablesDBSCANROBbinaryWorsened, yROBWorsened)
```

```
    lablesDBSCANCONSbinary = np.append(lablesDBSCANCONSbinary, yCONS)
    lablesDBSCANCONSbinaryWorsened = np.append(
       lablesDBSCANCONSbinaryWorsened, yCONSWorsened)
    lablesDBSCANVELbinary = np.append(lablesDBSCANVELbinary, yVEL)
    lablesDBSCANVELbinaryWorsened = np.append(
       lablesDBSCANVELbinaryWorsened, yVELWorsened)
    lablesBITbinary = np.append(lablesBITbinary, yBIT)
    lablesBITbinaryWorsened = np.append(lablesBITbinaryWorsened,
       yBITWorsened)
    counter+=1

lablesDBSCANROBbinary = lablesDBSCANROBbinary[1:]
lablesDBSCANROBbinaryWorsened = lablesDBSCANROBbinaryWorsened[1:]
lablesDBSCANCONSbinary = lablesDBSCANCONSbinary[1:]
lablesDBSCANCONSbinaryWorsened = lablesDBSCANCONSbinaryWorsened[1:]
lablesDBSCANVELbinary = lablesDBSCANVELbinary[1:]
lablesDBSCANVELbinaryWorsened = lablesDBSCANVELbinaryWorsened[1:]
lablesBITbinary = lablesBITbinary[1:]
lablesBITbinaryWorsened = lablesBITbinaryWorsened[1:]

kappaCONS = kappas[0, :]
kappaROB = kappas[1, :]
kappaVEL = kappas[2, :]
kappaBIT =  kappas[3, :]

#Sign test. Divide p-value by two to get one-sided p-value
import statsmodels.stats.descriptivestats

print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaVEL))
print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaCONS)
   )
print(statsmodels.stats.descriptivestats.sign_test(kappaBIT - kappaROB))

print(statsmodels.stats.descriptivestats.sign_test(kappaVEL - kappaCONS)
   )
print(statsmodels.stats.descriptivestats.sign_test(kappaVEL - kappaROB))

print(statsmodels.stats.descriptivestats.sign_test(kappaROB - kappaCONS)
   )

#Kappa score for the entire sample
import sklearn
from sklearn.metrics import cohen_kappa_score
print(sklearn.metrics.cohen_kappa_score(lablesBITbinaryWorsened,
   lablesBITbinary))
print(sklearn.metrics.cohen_kappa_score(lablesDBSCANROBbinary,
   lablesDBSCANROBbinaryWorsened))
print(sklearn.metrics.cohen_kappa_score(lablesDBSCANCONSbinary,
    lablesDBSCANCONSbinaryWorsened))
print(sklearn.metrics.cohen_kappa_score(lablesDBSCANVELbinary,
    lablesDBSCANVELbinaryWorsened))
```