Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 192 (2021) 2971–2979

Procedia Computer Science

www.elsevier.com/locate/procedia

25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Slice: an algorithm to assign fixations in multi-line texts

Dominik Glandorf[a,*], Sascha Schroeder[a]

[a] *University of Goettingen, Institute of Psychology, Waldweg 26, 37075 Göttingen, Germany*

Abstract

When analyzing eye movement data from the reading of multi-line texts, it is important to ensure that fixations are assigned correctly to the lines of the text. This is a non-trivial problem as eye movement data are noisy and often show complex non-linear distortions. Here, we introduced Slice, a new algorithm to assign fixations in multi-line text. We describe how Slice operates and evaluate it using a data set of natural reading data. Results show that Slice performs better than the default method used by many software packages, is robust to many forms of distortions, and approximates manual coding decisions.

Keywords:
eye tracking; reading; line assignment; drift correction

1. Introduction

Reading behavior can give insights to proficiency in natural and programming languages [1, 6], the mode of text reception [7] or serve to autogenerate summaries of read text [8]. As the eye constrains what one is able to perceive from a written text, eye movements are a reliable source for measuring reading behavior during the processing of naturalistic, multi-line texts and can be used to infer cognitive processes. One of the major challenges here is that eye movements are inherently noisy. This noise emerges during the complex chain from variability in visual attention over head and body movements to calibration problems. This is particularly true for low-cost eye trackers that are suitable to assess eye movements in a broad range of naturalistic settings, but often trade off robustness for accuracy [9].

Correcting for this noise is particularly important when reading is investigated where fixations first have to be mapped to different units of analysis such as words or sentences. All high-level reading measures (such as first-pass reading time or rereading time) can only be computed in a next step. However, their calculation heavily depends on the order in which fixations have been assigned to the words of a text. For example, the go-past time of a word is defined

* Corresponding author. Tel.: +49 157 366 20 114
E-mail address: dominik.glandorf@stud.uni-goettingen.de

1877-0509
10.1016/j.procs.2021.09.069

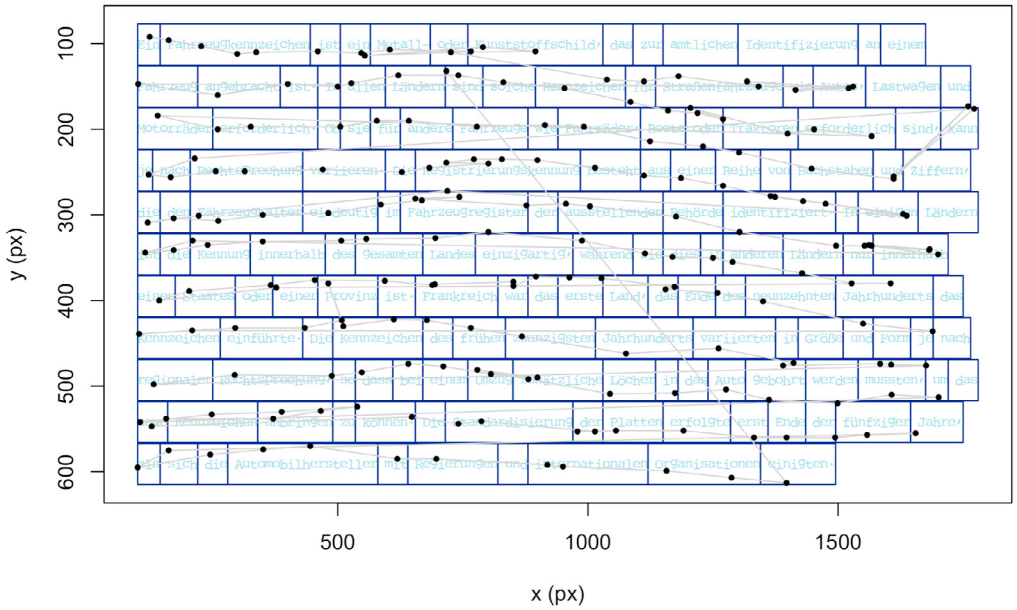

Fig. 1. A visualization of the fixations on a multi-line text. Fixations are connected with grey lines according to their temporal order. Words are surrounded by blue boxes. Texts have been displayed correctly, the small vertical offset in the letter positions was created during plotting.

as the time between the first fixation on a word and the time when it is exited to the right. If one of the intermediate fixations is mistakenly assigned to a subsequent line, the go-past time for this word will necessarily be incorrect too.

This example highlights that line assignment, i.e., mapping fixations to the individual lines of a multi-line text, is a fundamental and important process in the analysis of reading data. The accuracy of the line assignment process determines the upper limit of the accuracy of the fixation-to-word mapping. Unfortunately, mapping fixations to lines is a complicated and non-trivial problem in most natural reading settings. Fig. 1 visualizes a typical line assignment problem. The text was the last one a participant read in a longer experimental session when calibration is usually wearing off. As a consequence, the positions of some of the fixations (particularly in the right upper corner of the screen) are likely to be incorrect. Fixations still form distinguishable horizontal lines representing reading subsequences, but they do not completely overlap with the lines that have presumably been read. A naive approach, as implemented in standard software for eye movements analysis, which simply assigns fixations to the nearest line, will necessarily yield incorrect results here. Typically, such data have to be manually corrected by the experimenter. However, this process is time-intensive, error-prone, and can only be used offline, i.e., not while the text is actually being read [2]. Thus, it would be desirable to have an algorithm that assigns fixations to lines automatically with sufficient precision.

Carr et al. [2] describe a number of existing approaches to the line assignment problem. As elaborated above, the simplest method is to assign fixations to the closest line in the stimulus material. This method is computationally straightforward and can be regarded as some form of baseline method. Although it works in many cases, it requires fairly good data quality and a high amount of line spacing. If the vertical distance between the line is increased, then the line assignment process becomes increasingly more easy. However, increased line spacing comes at the cost of decreasing the ecological validity of reading situation. More sophisticated approaches often rely on segmenting the sequence of fixations into subsequences by some criterion and then assigning them to the lines of the stimulus. For example, the method proposed by [9] iteratively merges fixations that are locally close into lines which are then assigned to the stimulus lines from top to bottom.

The methods described by [2] rely on a number of assumptions summarized by [9]. For example, lines are assumed to be read in a specific direction or order, fixations are made from left to right except for return sweeps at line breaks, etc. While all of these assumptions are reasonable at first sight, most of them are overly restrictive and violated in many natural reading settings. For example, algorithms that aim at identifying coherent sequences of fixations or line

breaks, cannot detect when lines are skipped (e.g. [9]) or reread ([2]). Other approaches, which are regression-based, cannot handle non-linear patterns (e.g. [3]).

Here, we introduce Slice, a new algorithm to assign fixations to lines in multi-line texts. It rests on very few assumptions about the reading process and maps fixations to lines with high accuracy. In the following, we first describe the underlying rational of the algorithm as well as the steps and parameters involved in it. We then evaluate Slice against a baseline assignment method on a naturalistic data set for which manual line assignments were available. Finally, we discuss the results of the evaluation experiment and relate them to other existing line assignment approaches.

2. Description of the algorithm

Slice assigns all fixations of a single trial to one of the lines of the stimulus material. Slice comprises three main conceptual steps that are passed through on each trial.

1. Find sequences of fixations that are likely to belong to the same line.
2. Starting from the longest sequence, find other sequences that either belong to the same or to an adjacent line and repeat this process progressively until no further sequences can be found.
3. Ensure that the number of detected sequences corresponds to the number of lines in the text. The remaining sequences are mapped to lines by ordering them from top to bottom.

In the following, we will provide a more detailed explanation of these three main steps. We will first describe the structure of the algorithm's input and output. Next, we will elaborate the underlying assumptions of the algorithm and provide a step-by-step description of the algorithm and its parameters. An implementation of Slice in R can be found at `https://github.com/sascha2schroeder/popEye/blob/devel/R/Slice.R`. However, it would be relatively easy to also implement the algorithm in other programming languages such as Python or Java.

2.1. Input and output

The algorithm takes two matrices as input. The first file is a data frame with all fixations of a trial in consecutive order with their corresponding x and y coordinates. The second input file is a data frame describing the stimulus materials providing the x and y coordinates for all the words of the text. From this representation the number of lines and their position on the screen can be derived as well as other relevant variables such as line height (defined as the average vertical distance between lines) and average letter width. The output of the algorithm is a data frame that corresponds to the fixation input file but with an additional column providing the assigned line number for each fixation, using the same enumeration as the stimulus material.

2.2. Assumptions

As any algorithm, Slice makes some assumptions, that we want to make explicit:

1. Slice assumes that the first line was, at least partially, read at some point in time during the trial.
2. If the pattern of fixations contains an unexpected large horizontal gap, a line is considered as skipped.
3. Distortions are assumed to be dependent on space, not time. For example, if the first line's fixations are shifted downwards at the beginning of a trial, a similar shift is expected when the subject returns to the first line in the end of the trial.

Sequences of fixations that are located within a certain vertical and horizontal distance to each other are grouped together and will be assigned to the same line. We will refer to those sequences of fixations as "runs". Runs usually exhibit clear pattern of lines (see Fig. 2). The aim of the algorithm is to assign runs to "proto lines" and to map these proto lines to text lines from top to bottom. In order to create the proto lines, the algorithm tries to find horizontal gaps between runs and already existing proto lines. This can be thought as slicing from left to right with a tool through the whitespace between the runs (which is giving the method its name). Runs that are separated by these cuts are

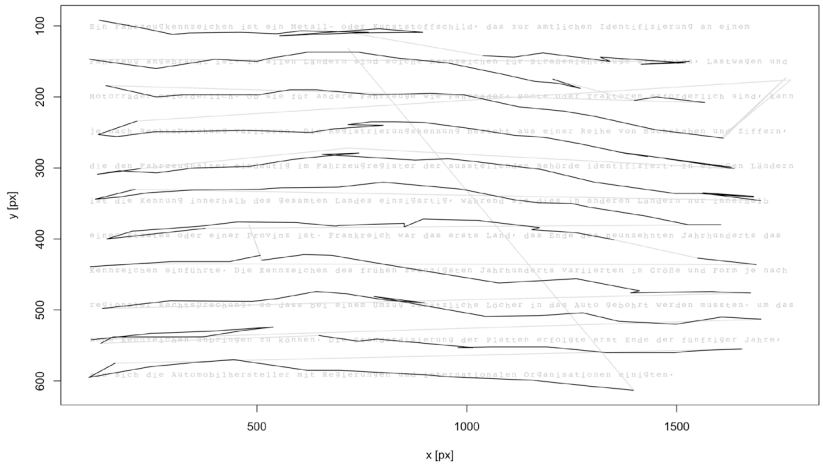

Fig. 2. The same trial as shown in Fig. 1 with black lines highlighting connected runs of fixations

assigned to adjacent lines. Basically, Slice only relies on the whitespace between runs and can thus tolerate a number of distortions.

2.3. Step by step

During the assignment process, Slice executes the following steps:

1. Segment runs: The vertical and horizontal distance between subsequent fixations is calculated. Every time one of the distances exceeds its respective threshold r_h or r_v (see section Parameters), a new run is initiated. This step is similar to the run segmentation as used by [9]. The result of this step is shown in Fig. 2 with runs plotted in black. The number of runs is typically clearly larger than the number of lines in the text as runs are solely defined by saccades exceeding the vertical or horizontal threshold, i.e., not only line breaks, but also regressions and long forward saccades.
2. Determine starting run: The run with the largest horizontal span is used as the starting point for the following grouping process. The starting run is highlighted in blue in Fig. 3 on the left.
3. Group runs into proto lines: In the next step, runs are grouped together into so called ”proto lines” which will later be assigned to actual lines of the text. This step is the core of the algorithm. For example, the next-to-last line in Fig. 2 comprises two runs that are likely to belong together and should be grouped into the same proto line. Again, the number of proto lines is not limited to the number of real lines in the text. The starting run is assigned the proto line number 0. Next, the distances between this first proto line and all other runs are calculated. The distance is defined as the average vertical distance of all fixations of a run to the horizontally closest fixation on the proto line. On the left side of Fig. 3, the distance calculation is visualized for two runs that are close to the starting run. The average distances of the green run to proto line 0 is -47 px. The average of the red run is +38 px. This calculation is performed for all runs. The decision to which proto line a run should be grouped into is based on the following criteria:

    - If the distance is smaller than the same-line threshold d_w (see section Parameters), then the run is merged into the proto line.
    - If the distance is larger than d_w and smaller than the next-line threshold d_n (see section Parameters), then the run will be assigned to a new adjacent proto line. For example, both the green and the brown run in the left plot of Fig. 3 will be grouped into new proto lines which are labeled -1 (green) and +1 (brown). Thus, upper adjacent lines receive a decreased number whereas lower adjacent lines receive an increased number.

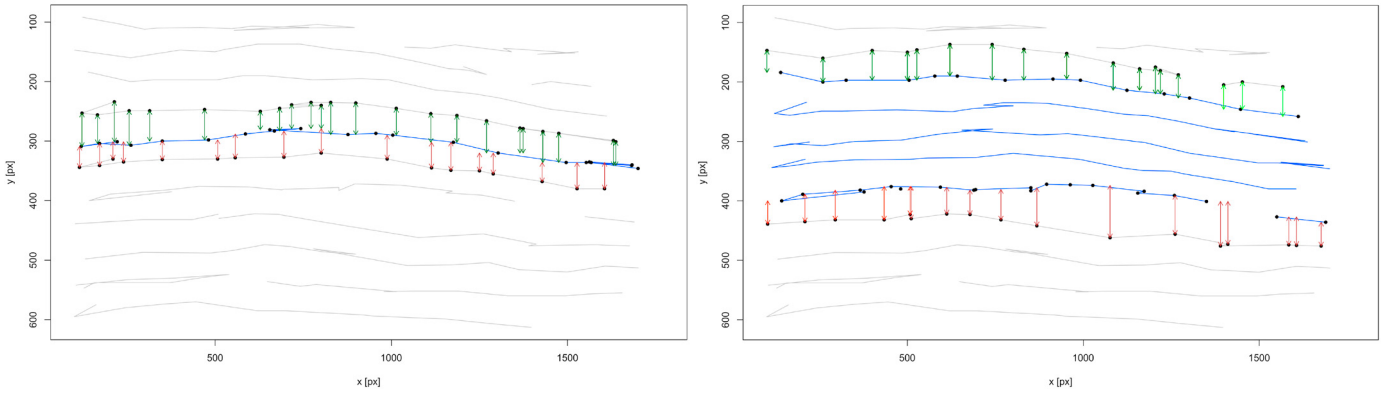

Fig. 3. The same trial as shown in Fig. 1 at two different states of the grouping process. Runs are highlighted in gray. The base run is marked in blue. The left plot shows the first iteration of grouping, the right shows the third iteration.

- If the distance of a run is larger than d_w as well as d_n, the run will not be assigned to any proto line in the current iteration.

If no run was grouped to one of the two adjacent proto lines, that line is considered to be skipped. In order to be able to continue the grouping process, an artificial empty proto line is created at the expected position of the skipped line with no run assigned to it. The position of this proto line is the average horizontal and vertical position of the last proto line vertically shifted by the line height. In the next iteration, both the new upper and lower proto lines are used as new reference lines for the grouping process. The distances are calculated in the same manner as before for all runs with regard to the two reference lines and the same grouping criteria are applied. For example, the distance calculations for the third iteration of the grouping process are shown on the right side of Fig. 3. Both the darkgreen and the lightgreen runs at the top will be grouped into a new proto line -3. The brown and the red run at the bottom will be grouped into proto line +3. Note that the distance calculation for the brown run is made based on a proto line that comprises of two different runs because they both were grouped into the proto line number +2 before. The process will be repeated until the outer fixations are reached.

4. Check assignment of runs: In some boundary cases, some runs might not have been assigned to a proto line in the previous step. For this reason, unassigned runs are merged into the closest proto line.
5. Prune proto lines: If there are more proto lines than lines in the text, the topmost or bottommost proto line with the fewest fixations is merged with its adjacent proto line until the number of proto lines corresponds to the number of text lines.
6. Map proto lines to text lines: Proto lines already are ordered correctly. They are numbered consecutively and mapped to the corresponding line of the text.
7. Delete empty proto lines: Proto lines that were created to represent skipped lines are deleted.

2.4. Parameters

The algorithm has four parameters, two that are relevant for run segmentation and two that are relevant for the grouping step.

- The two parameters related to the segmentation into runs are the horizontal threshold r_h (expressed in number of letters to the last fixation) and the vertical threshold r_v (expressed in terms of the average line height). They are defined as the spatial distance in their respective axis that is tolerated during run segmentation until two subsequent fixations are split into two distinct runs.
- The two parameters relevant for the proto line grouping are the within-line threshold d_w and next-line threshold d_n, that are expressed in terms of the average line distance. The thresholds are the decision base for the assignment of runs to proto lines. If the average distance of a run is below d_w, a run is merged into the current proto

line that serves as a reference. If the average distance of a run is larger than d_w but smaller than d_n, it will be assigned to a new proto line.

For all parameters, sensible default values are provided that are reasonable for text displays similar to the one described above, namely $r_h = 0.5$, $r_w = 25$, $d_s = 0.5$, $d_s = 1.4$. However, we advise users to to initially test whether the default values are appropriate for their data. In particular, it is crucial to work with suitable thresholds for the run segmentation as the quality of following step depends on it. If runs are too long, they may actually span more than one text line. If they are too short, runs are more noisy and errors are more likely to occur during the grouping process. Ideally, the thresholds are adjusted in a way that produces the longest possible runs that never span two actual lines at the same time. For different populations the optimal parameters might thus vary. The default values for the two parameters of the grouping step, by contrast, are rather stable and will likely generalize to new data sets.

3. Evaluation

We evaluated the performance of Slice against the naive baseline method using data from the MECO corpus [5]. MECO is a cross-linguistic, multi-site study investigating eye movements while participants read texts in their native language. At present, the corpus comprises data from 13 different languages (including Dutch, English, Finnish, German, Greek, Hebrew, Italian, Korean, Norwegian, Russian, Spanish, and Turkish). Here, data from the German sub-sample with 50 participants was used. The reading materials consisted of 12 short expository texts (100-200 words, 8-12 sentences, 10-15 lines, all on 1 page) which provided Wikipedia-style information. A representative trial is displayed in Fig. 1.

Texts were presented on a 25 in LCD monitor with a resolution of 1080×1920 px at a distance of ca. 60 cm. The font was 20 pt Consolas (in which each letter covers an area of 15×24 px) using double line spacing (i.e., 48 px). Eye movement data were collected using an Eyelink 1000 which has a high temporal (1 ms) and spatial resolution (ca. 0.5 deg visual angle, which corresponds to 2-3 letters given the used font size and screen resolution).

In order to create a gold standard for the evaluation, the fixations of all $50 \times 12 = 600$ trials were manually assigned to one of the lines in each text. The quality of the manual assignment was checked by computing the interrater agreement between two independent raters for 20 % of the data. Although there were some discrepancies, the percentage of agreement was extremely high (above 98 % for each participant). This value is thus the upper limit that could potentially be achieved by any line assignment algorithm.

Table 1. Average accuracy of the two line assignment methods on the MECO corpus

Method	Mean	Median	> 95%	> 99%
Baseline	81.1%	88.1%	24.4%	5.2%
Slice	95.3%	99.2%	89.6%	54.7%

In order to also assess the performance of the baseline algorithm and Slice we used the proportions of fixations that were assigned to the correct line (defined as the manual assignments of Coder 1) for each individual trial. The baseline method was a naive line-matching approach which simply assigns fixations to the line with the smallest vertical distance (see section 1). This approach is the only method that is (implicitly) implemented in standard software for the analysis of eye movements (e.g., SR Research's Data Viewer, SMI's BeGaze) and is thus a useful baseline condition. Slice was run with default parameter values (see section 2.4) which were checked for appropriateness for the present data. Table 1 lists the arithmetic mean, the median accuracy for both algorithms. In addition, the percentage of trials with mean accuracy above 95 % and 99 % are also reported.

As can be seen, Slice clearly outperformed the baseline algorithm in terms of both mean and median accuracy. However, from a practical perspective, average accuracy across trials is not the most useful statistic. Instead, it is typically more important to know whether a trial has to be discarded or not. In order to address this issue, Table 1 also reports the proportion of trials which has an average accuracy above 95 and 99 % which are the trials that are most likely to be usable. The performance of both algorithms is more modest here. However, Slice again clearly

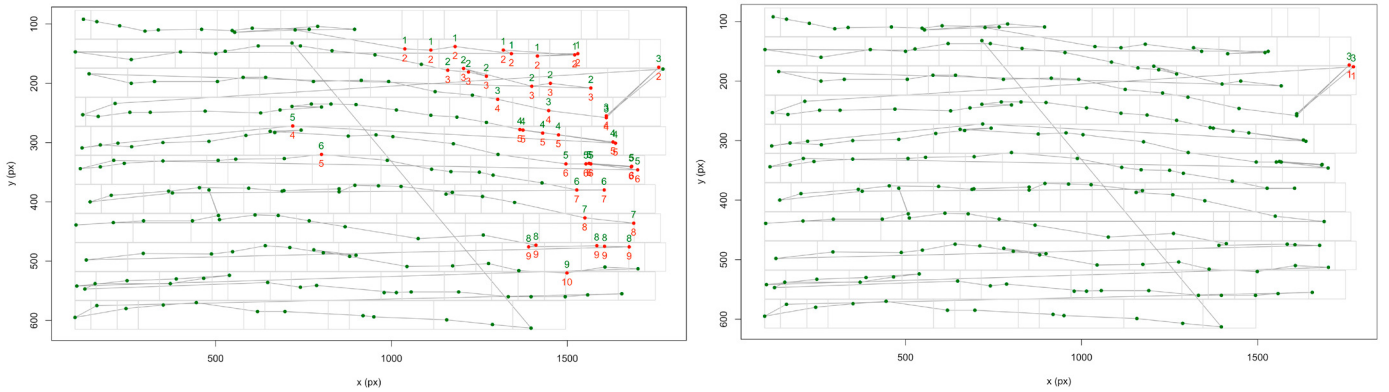

Fig. 4. The same example trial as in Fig. 1 with assignments by the baseline algorithm on the left and Slice on the right. Correctly assigned fixations are printed in green and incorrectly assigned fixations printed in red. For each incorrect assignment, the red number indicates the wrongly assigned line while the green number indicates the correct line.

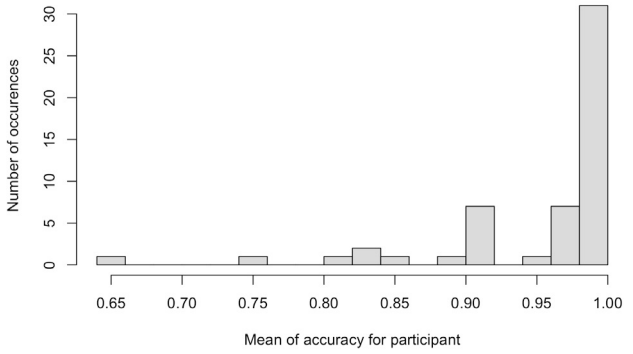

Fig. 5. For every of the 50 participants the accuracy of Slice across their 12 trials was averaged. The accuracy of a trial is the relative amount of correctly assigned fixations. The diagram shows the frequencies of those mean accuracies for the evaluation data set.

outperformed the baseline method, especially for the 99 % measure. Using this criterion, less than 10 % of the data would be usable using the baseline method while the proportion increases above 50 % when Slice is used.

To follow-up on this, we manually inspected all 228 trials from the first 19 participants for the errors produced by the two algorithms (see Fig. 4). In addition, we explicitly coded whether a trial is acceptable or not. For example, if a complete line segment was assigned to the wrong line (Fig. 4 on the left) the trial was coded as unacceptable. By contrast, trials in which only single fixations were assigned incorrectly (Fig. 4 on the right) were coded as acceptable. The overall interrater agreement was very high (91%). The mean acceptability rate was 43.4% for the baseline method and 82.9% for Slice. Thus, the explicit acceptability rating replicated the pattern from the overall analysis and indicates that the 95% cut-off approximates human acceptability judgments best.

In another follow-up analysis, we identified the data patterns that were particularly difficult to analyze for Slice. In order to do this, we first computed the average accuracy for each participant and each trial. The distribution of the mean accuracy values is shown in Fig. 5. As can be seen, the distribution is left-skewed with accuracy at ceiling for most participants. The two participants with the worst average performance were Participant 46 (65.9%) and Participant 43 (76.0%). Each of them had two trials in which the average accuracy was below 10%, i.e., near chance level. Closer inspection revealed that in these cases all lines were systematically misaligned by one line. Fig. 6 shows two trials by Participant 28 (on whom the algorithm had the third-worst performance) demonstrating this behavior. Both trials showed strong distortions in the top left corner due to calibration problems. However, while the trial on the left was completely misaligned, the trial on the right was aligned with near-perfect accuracy. How can this discrepancy be explained? The problem on the left side is created during the pruning step of the algorithm when the proto lines are

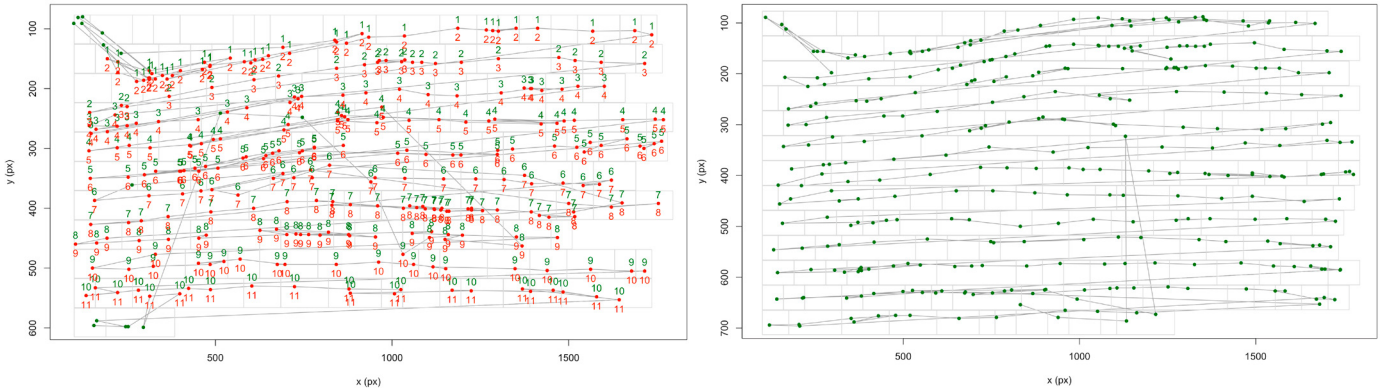

Fig. 6. Two different trials of Participant 28 both assigned by Slice. Correctly assigned fixations are printed in green and incorrectly assigned fixations printed in red. For each incorrect assignment, the red number indicates the wrongly assigned line while the green number indicates the correct line. Please note that no errors have been made on the trial presented on the right.

merged with each other. By default, lines at the begin or end of the text which have few fixations will be pruned first. The last line of the trial on the left is rather short and naturally comprises only few fixations. It was thus erroneously merged with the second last line, creating an overall shift of all other lines. In the trial on the right, by contrast, the fixation in the upper left corner has been merged with the second line during pruning, which was the correct decision. Other systematic errors of Slice were not identified by visual inspection on the used data set. In the majority of trials with a lot of mistakenly assigned fixations, a lot of very small runs or rereading occurred, so that lines were also visually indistinguishable and no clear pattern of runs could be recognized.

4. Discussion

In the present paper we have introduced Slice, a new algorithm to assign fixations in multi-line texts. Slice builds on existing concepts such as run segmentation and merging (as summarized by [2]). However, it introduces a novel slicing technique in the merge step that makes the algorithm very flexible and able to fit even complex, naturalistic reading patterns. Slice massively outperformed a commonly used baseline algorithm on the MECO data set comprising texts with 10-15 lines. The analysis on subject level revealed that most participants and trials were assigned with perfect or near-perfect accuracy and error rates were mainly determined by few outlier trials in which the assignment process failed completely.

One of Slice's main strengths is the self-correcting grouping mechanism that seems to resemble the human ability to identify and discriminate lines based on their global features: Slice first identifies the largest and most distinct line as a starting point and then tries to detect new adjacent lines, tolerating small distortions. Another strength is the final pruning step which enables the algorithm to ignore single, outlying fixations which would distort the line detection process. Finally, one of Slice's unique features is that it can deal with situations in which lines have been skipped. As a consequence, Slice can map detected proto lines to the actual text in an ordinal manner without the necessity to refer to the absolute position of the lines which causes problems if lines are bent. In contrast, regression-based algorithms (e.g., [3]) do not perform well with non-linear noise patterns.

Of course, Slice also has some important limitations. First, Slice does only make partial use of temporal information implicitly comprised in fixation sequence. The order of the fixations is only relevant during run segmentation, but is entirely ignored afterwards. Thus, Slice will have problems in situations when noise varies at a function of time, but not only the spatial position of the fixation, e.g., when an individual trial is presented for a long time (i.e., several minutes). Typically, however, the noise pattern is more strongly determined by spatial than by the temporal dimension (e.g., [4]). In addition, long texts are usually presented in several, consecutive trials (e.g., different pages or screens) which attenuates this problem. Another limitation is the choice of a reference line used for the line mapping process. Slice detects lines in an ordinal manner and the absolute positions of lines are discarded. As a consequence, Slice has

to assume that the first proto line corresponds to the first line of a text. Finally, our error analysis has demonstrated that short lines with very few fixations can create problems during line pruning step and should thus be avoided.

In the present study, we have used the default parameters that are provided for Slice. Although we checked that they were reasonable for the MECO data set, we did not did systematically evaluated whether other parameter settings would increase the effectiveness of the algorithm. However, it would certainly be possible to first tune these parameters for different data sets or even individual participants provided that a sufficient amount of labeled training data is available. In order to avoid overfitting, it would be advisable to use a separate training set for this. Exploring the potential of optimal parameter choices would be an interesting question for future research.

Future studies should systematically compare Slice against other existing line assignment algorithms (e.g. the ones listed in [2]) in order to evaluate its competitive performance. Ideally, the different algorithms would be evaluated on several data sets simultaneously that use different reading materials and participant populations. It is also necessary to go beyond assignment accuracy as the only performance criterion. As described in the introduction, researchers are typically interested in higher-level reading measures such as first-pass reading time or rereading time. Thus, a logical next step would be to use the correspondence with such derived measures as an additional outcome criterion.

References

[1] Augereau, O., Fujiyoshi, H., Kise, K., 2016. Towards an automated estimation of English skill via TOEIC score based on reading analysis, in: 23rd International Conference on Pattern Recognition (ICPR), IEEE, Cancun. pp. 1285–1290. doi:10.1109/ICPR.2016.7899814.

[2] Carr, J.W., Pescuma, V.N., Furlan, M., Ktori, M., Crepaldi, D., 2021. Algorithms for the Automated Correction of Vertical Drift in Eye-Tracking Data. Behavior Research Methods doi:10.3758/s13428-021-01554-0.

[3] Cohen, A.L., 2013. Software for the automatic correction of recorded eye fixation locations in reading experiments. Behavior Research Methods 45, 679–683. doi:10.3758/s13428-012-0280-3.

[4] Holmqvist, K., Andersson, R., 2017. Eye tracking: a comprehensive guide to methods, paradigms, and measures. 2nd ed., CreateSpace.

[5] Kuperman, V., Siegelman, N., Amenta, S., Aartürk, C., Alexeeva, S., Ahn, H., Bertram, R., Bonandrini, R., Brysbaert, M., Chernova, D., Maria Da Fonseca, S., Dirix, N., Duyck, W., Fella, A., Frost, R., Gattei, C.A., Kalaitzi, A., Kwon, N., Marelli, M., Papadopoulos, T.C., Protopapas, A., Savo, S., Shalom, D., Sloioussar, N., Stein, R., Sui, L., Taboh, A., Tønnesen, V., Usal, K.A., . Expanding horizons of cross-linguistic research on reading: The Multilingual Eye-movement Corpus (MECO). Behavior Research Methods .

[6] Madi, N.A., 2020. Modeling Eye Movement For The Assessment of programming Proficiency. Ph.D. Dissertation. Kent State University. URL: http://rgdoi.net/10.13140/RG.2.2.30567.27042.

[7] Sanches, C.L., Kise, K., Augereau, O., 2015. Eye gaze and text line matching for reading analysis, in: Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers, Association for Computing Machinery, Osaka, Japan. pp. 1227–1233. doi:10.1145/2800835.2807936.

[8] Xu, S., Jiang, H., Lau, F.C., 2009. User-oriented document summarization through vision-based eye-tracking, in: Proceedings of the 14th international conference on Intelligent user interfaces, ACM, Sanibel Island Florida USA. pp. 7–16. doi:10.1145/1502650.1502656.

[9] Špakov, O., Istance, H., Hyrskykari, A., Siirtola, H., Räihä, K.J., 2019. Improving the performance of eye trackers with limited spatial accuracy and low sampling rates for reading analysis by heuristic fixation-to-word mapping. Behavior Research Methods 51, 2661–2687. doi:10.3758/s13428-018-1120-x.