2024 ■ Vol. 20 ■ no. 3

mousetRajectory: Mouse tracking analyses for behavioral scientists

Roland Pfister [a,b], Solveig Tonn [a], Moritz Schaaf [a] & Robert Wirth [c]

[a]Department of Psychology, Trier University
[b]Institute for Cognitive and Affective Neuroscience, Trier University
[c]Department of Psychology, University of Würzburg

Abstract ■ Mouse tracking and the recording of movement trajectories have become powerful tools to investigate cognitive processes. Dedicated analysis software for this type of data is now readily available to empirical researchers, promising a substantial simplification of the required data processing tasks. However, existing solutions are designed for specific recording software or analysis workflows, thus lacking the flexibility to adapt analyses to individual needs. The R package mousetRajectory addresses this gap. By placing strong emphasis on code clarity and modularity, it facilitates customization for researchers, especially those favoring a modern tidyverse programming style. Here, we provide example analyses that explain the essential preprocessing tools, such as time normalization or resampling, and key functions for computing trajectory markers. These markers include classic spatial metrics such as area under the curve and maximum absolute deviation along with more advanced measures such as sample entropy. In summary, mousetRajectory offers a toolkit for researchers seeking a lightweight and easily adaptable foundation for custom analyses of 2D movement trajectory data.

Keywords ■ mouse tracking, movement trajectories, open source. **Tools** ■ R.

✉ tonn@uni-trier.de or roland.pfister@uni-trier.de

10.20982/tqmp.20.3.p217

Acting Editor ■ Denis Cousineau (Université d'Ottawa)

Reviewers
■ One anonymous reviewer

Introduction

The analysis of movement trajectories is a powerful method for investigating mental processes. Mouse tracking, the most prevalent variant of this method, offers rich information while minimizing data collection costs: By continuously logging the position of a computer mouse while participants engage in cognitive tasks, it precisely captures the temporal and spatial evolvement of movements and thereby provides a comprehensive understanding of the cognitive foundations of decision-making, perception, and language processing (Erb et al., 2016; McKinstry et al., 2008; Song & Nakayama, 2009; Spivey et al., 2005; van der Wel et al., 2014). The means to distill insights from these complex movement data have rapidly evolved over the last two decades, and this continuous development has generated diverse and sophisticated techniques for data collection and analysis (Camerer et al., 1993; Kieslich et al., 2018; Schoemann et al., 2021; Wirth et al., 2020). Hence, researchers employing this method for the first time are confronted with manifold complex design choices and programming demands. To mitigate the implementation challenges and to streamline the process of recording and analyzing mouse tracking data, several ready-made solutions are available (e.g., Freeman & Ambady, 2010; Kieslich & Henninger, 2017; Mathur & Reichling, 2019; Wulff et al., 2021).

On the one hand, these software solutions offer substantial benefits. Utilizing existing software, as opposed to crafting custom scripts, saves time and improves the reproducibility and comparability across studies. It provides standardized defaults and documentation, enabling other researchers to quickly infer design decisions from the code of a given experiment. Moreover, suggestions from an active community can contribute to quality enhancements in the codebase over time. On the other hand, these software solutions come with several drawbacks. First, many packages are strictly designed for certain experimental setups, thereby limiting the choice over display geometry, tracking procedures, and data formats. As many of these design fac-

tors can affect the outcome of empirical studies (Grage et al., 2019; Kieslich et al., 2020; Schoemann et al., 2021; Wirth et al., 2020), relying on one particular solution may bias an entire field of research. Second, while research employing such integrated analysis solutions may benefit from standardized defaults, this uniformity might hinder the creative freedom inherent to custom-written scripts. Consequently, this limited variability in the employed techniques could slow the methodological progress of the field. Finally, code in mature software projects is often highly optimized for improved computational efficiency (which is generally a good choice). This optimization can render the source code less intuitive, however, especially for users seeking a deeper understanding of the inner workings of their analyses.

Within the R programming language (R Core Team, 2023), the `mousetrap` package (Wulff et al., 2021) is such an integrated analysis solution for mouse tracking data. When used in conjunction with the corresponding `mousetrap-os` plugin (Kieslich & Henninger, 2017) for the `OpenSesame` experiment builder (Mathôt et al., 2012), researchers can conduct and analyze mouse tracking experiments without directly engaging with the raw data produced by such experiments. Additionally, code is optimized for both time and space efficiency.[1] This efficiency is achieved by, for example, passing data to functions written in C++ and the use of custom data structures.

However, when researchers want to expand the functionality offered by mousetrap, this optimization introduces various challenges that can hinder straightforward modifications: First, trajectory data is stored in wide (instead of long) format and separately from trial-related data. While this is more efficient in terms of memory usage, users accustomed to `tidyverse` (Wickham et al., 2019) functionality – the de-facto standard in modern-day R usage in behavioral sciences – may perceive this data structure as rather counterintuitive. Second, C++ is a strongly typed, object-oriented, zero-indexed, compiled language, while R is a weakly typed, one-indexed, interpreted language. Consequently, C++ code can undergo thorough optimization by the compiler. However, it also requires a solid understanding of memory management and object-oriented principles. Since these concepts are not essential for R programming, R users may encounter difficulties when attempting to comprehend and modify C++ code.[2] Third, CRAN distributes C++ functions in compiled form, necessitating the retrieval of the underlying source code from the `.tar.gz` files available on CRAN or from the respective github repository. The source code of functions written in R, in contrast, is readily accessible by invoking `body()` on the function of interest.

The mousetrap package thus offers convenient access to common trajectory measures coupled with highly efficient code. Nevertheless, this focus on performance optimization can pose challenges for customization. Our package `mousetRajectory` addresses this need. Because all code is written in R, users can effortlessly retrieve the source code and tailor it to their specific needs. Furthermore, its code places strong emphasis on clarity, even if it occasionally means sacrificing some efficiency. Combined with its modular approach, researchers can easily interweave `mousetRajectory` functions with their own custom code, simplifying the modification of selected facets of their analysis. Finally, our recommended analysis approach smoothly combines with the `tidyverse` ecosystem.

Functionality provided by `mousetRajectory`

`mousetRajectory` provides both convenience functions for preprocessing and functions that compute trajectory metrics. At present, it provides the following set of features:

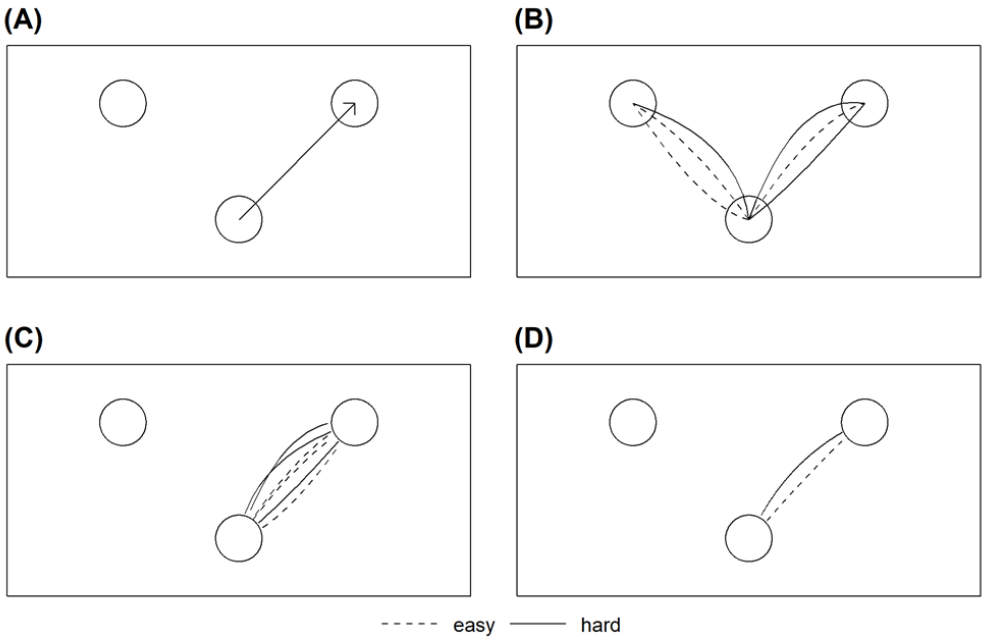
Preprocessing

- `is_monotonic()`: Verifies whether a number sequence is monotonically in-/decreasing. This is particularly useful when validating the temporal order of trajectory data.
- `is_monotonic_along_ideal()`: Verifies whether the empirical trajectory increases monotonically relative to the optimal path (straight line-segment from start to end). This is particularly useful when extracting trajectories that surpassed the target area and subsequently had to reverse their direction.
- `time_circle_left()`: Calculates the time at which the trajectory was first outside a circular area. This is particularly useful when determining the initiation time of a movement.
- `time_circle_entered()`: Calculates the time at which the trajectory was first inside a circular area. This is particularly useful when determining the completion time of a movement.
- `interp2()`: Convenience wrapper that performs resampling via linear interpolation. This is achieved by first normalizing the timestamps and subsequently invoking `interp1()` from the signal package.

[1] With the term "computational efficiency," we refer to practical efficiency rather than theoretical, asymptotic complexity classes.

[2] R also supports certain aspects of object-oriented programming via polymorphism of generic functions (`S3` and `S4` classes), reference classes, or dedicated packages (e.g., `R6`, Chang, 2021; and `proto`, Wickham et al., 2016).

Figure 1 ■ Typical processing steps involved in mouse tracking analyses. (A) Participants respond to a stimulus (not displayed) by moving the computer mouse from a starting area (bottom circle) to one of two target areas (top circles). (B) Unprocessed trajectory data. (C) Trajectories ending on the left are mirrored across the middle of the screen and truncated to relevant parts. (D) After resampling (sometimes referred to as time-normalization), average trajectories can be computed and visualized for the experimental conditions.



Trial metrics

- `starting_angle()`: Calculates angles. Output values are expressed in degrees (not radians).
- `point_crosses()`: Quantifies how often a specific value on the x- or y-axis is intersected or traversed.
- `direction_changes()`: Quantifies the frequency of directional changes along the x- or y-axis.
- `auc()`: Calculates the signed Area Under the Curve (AUC), a metric that quantifies the magnitude and directional attributes of the area enclosed between an empirical trajectory and the optimal path (straight line-segment from start to end).
- `max_ad()`: Calculates the signed Maximum Absolute Deviation (MAD), a metric that quantifies the magnitude and direction of the maximum discrepancy between an empirical trajectory and the optimal path (straight line-segment from start to end).
- `curvature()`: Calculates the curvature, a metric that quantifies the degree of bending. Curvature is defined as ratio between the length of the empirical trajectory and the length of the optimal path (straight line-segment from start to end).
- `index_max_velocity()`: Computes the time at which the velocity reaches its maximum.
- `index_max_acceleration()`: Computes the time at which the acceleration reaches its maximum.
- `sampen()`: Computes the sample entropy, a metric that quantifies the complexity of trajectories. Specifically, sample entropy assesses the degree of unpredictability or irregularity in a time series (e.g., the x or y-coordinates, or their derivatives) and quantifies how likely it is for a particular pattern to repeat itself (Hehman et al., 2015; Richman & Moorman, 2000).

A concise example

To provide a practical demonstration of how to use the `mousetRajectory` package, we will outline the essential components of analyzing raw data from a fictitious data set. The data and code used to generate this tutorial, along with corresponding analyses that do not rely on `tidyverse` functionality, are available at osf.io/75se3. In this fictive experimental scenario, participants encounter stimuli that are either easy or difficult to classify, which serves as our experimental manipulation. They respond by moving a mouse cursor from a starting area (bottom circle with co-

Listing 1 ■ Inspecting the data for example analysis

```
library(mousetRajectory)
library(tidyverse)

head(dat_track, n = 3)
#>   subject block trial time     X       Y
#> 1       1     1     1    0  0.00 0.00000
#> 2       1     1     1    1 -0.01 0.01396
#> 3       1     1     1    2 -0.02 0.02784

head(dat_trial, n = 3)
#>   subject block trial direction difficulty
#> 1       1     1     1      left       easy
#> 2       1     1     2      left       hard
#> 3       2     1     1     right       easy

dat_track <- dat_track %>%
   left_join(dat_trial, by = join_by(subject, block, trial))


head(dat_track, n = 3)
#>   subject block trial time     X       Y direction difficulty
#> 1       1     1     1    0  0.00 0.00000      left       easy
#> 2       1     1     1    1 -0.01 0.01396      left       easy
#> 3       1     1     1    2 -0.02 0.02784      left       easy
```

ordinates (0,0)) to one of two target areas (top circles with coordinates (-1,1) and (1,1); see Figure 1A).

The tracking data is stored in the data frame `dat_track`, which is in long (or tidy; Wickham, 2014) format. In other words, every row in `dat_track` corresponds to a single `time` point in one `trial` of one participant, and every column represents one variable. In addition to the `X` and `Y` coordinates and time-stamps of the trajectory, `dat_track` requires one or more variables that unambiguously specify, for each row, the precise trial the trajectory data belongs to. In this example, the `subject` × `block` × `trial` combination fulfills this role.

How trial-specific information is stored depends on the data logging method. It can be recorded in a separate data file (e.g., in the data frame `dat_trial` containing the variables stimulus `difficulty` and correct `direction`), or alternatively, in additional columns in the tracking data. The former approach saves memory because the information is recorded only once for each trial and not duplicated for each timestamp. The latter approach, however, streamlines data manipulation, particularly when utilizing `tidyverse` tools. Fortunately, it is possible to convert the former data representation into the latter seamlessly as seen in Listing 1.[3]

At this point, `dat_track` contains all the data that are essential for the analysis. A common first step is to adjust the representation of coordinates, for example by transforming coordinates that are recorded relative to the upper left corner into coordinates that are relative to the starting area (note that our example data are already in the desired representation). To allow for data aggregation across experimental conditions, irrespective of the target location, the initial preprocessing typically also involves mirroring certain movements (those initially ending on one particular side of the screen) across the central vertical axis (here: `X` = 0). This action effectively inverts the direction of these movements along the horizontal dimension, causing them to end on the opposite side. Additionally, it is advisable to verify certain assumptions about the data. For instance, are the data for each trial arranged in chronological order, based on their timestamps? Depending on preferences and the research objectives, timing-related variables can be extracted. Furthermore, focusing exclusively on segments of the data that directly target the specific research question at hand can increase the signal-to-noise ratio. The analysis could, for example, be limited to the part of the movement that occurs between the departure from the starting area and the entry into the target area (see Figure 1C). See top part of Listing 2.

Note that trials differ in length, and consequently, they differ in the number of data points (i.e., rows) that they encompass. To plot average trajectories, it therefore becomes necessary to align the data along a specific dimension (e.g., time). Adhering to the method advocated by Spivey et al. (2005), a commonly adopted approach is to resample

[3] When many trial-level variables are required for statistical analysis but not for trajectory-level data manipulation, it can be an efficient strategy to initially join only the essential information. The remaining variables can then be joined before conducting the statistical analysis.

Listing 2 ■ Computing `dat_track`, `dat_interp` and `dat_plot`

```
dat_track <- dat_track %>%
   group_by(subject, block, trial) %>%
   mutate(
      X = ifelse(direction == "left", -X, X),
      is_ok = is_monotonic(time),
      initiation_t = time_circle_left(X, Y, time, x_mid = 0, y_mid = 0, radius = 0.2),
      completion_t = time_circle_entered(X, Y, time, x_mid = 1, y_mid = 1, radius = 0.2),
      movement_t = completion_t - initiation_t
   ) %>%
   filter(time >= initiation_t & time < completion_t)

dat_interp <- dat_track %>%
   group_by(subject, block, trial, difficulty) %>%   # movement time etc.
   reframe(
      t_interp = 0:100,
      x_interp = interp2(time, X, 101),
      y_interp = interp2(time, Y, 101)
   )

dat_plot <- dat_interp %>%
   group_by(subject, difficulty, t_interp) %>%
   summarize(
      x_subject_mean = mean(x_interp),
      y_subject_mean = mean(y_interp)
   ) %>%
   group_by(difficulty, t_interp) %>%
   summarize(
      x_condition_mean = mean(x_subject_mean),
      y_condition_mean = mean(y_subject_mean)
   )

# Plot:
dat_plot %>%
   ggplot(aes(
      x = x_condition_mean,
      y = y_condition_mean,
      linetype = difficulty)) +
   geom_path()
```

each trajectory to 101 coordinates, thus representing the trajectory with 100 movement steps. This resampling is achieved by applying linear interpolation across normalized time-stamps and consequently, the resulting coordinates can be interpreted as representing the estimated position after a specified percentage of the movement duration has elapsed. Despite the apparent complexity of this process, it can easily be achieved by separately passing the `X` and `Y` coordinates to the function `interp2()`, which essentially functions as a user-friendly wrapper for the `interp1()` function from the `signal` package (Signal Developers, 2014). It is worth noting that functions like `summarize()` and `reframe()` may discard variables from the `dataframe` if they were not included in the most recent `group_by()` call. As the stimulus `difficulty` is a crucial component in the statistical analysis, this variable must be included in the `group_by()` function, even though the stimulus difficulty remains constant throughout a trial, as seen in the first part of Listing 2.

After computing the average `subject × difficulty` coordinates (and average `difficulty` coordinates based on these `subject × difficulty` coordinates), the results can be visualized (see Figure 1D); see second part of Listing 2.

A final step is to assess the reliability of possible between-condition differences in the trajectories with inferential statistics. Conventional statistical methods like *t*-tests and analyses of variance (ANOVAs) require a single data point per subject and condition. Thus, each trial must be broken down to characteristics that can be expressed in a single value. Here, numerous metrics are available, and it is feasible to calculate these metrics based on the original, raw trajectory data or the resampled, interpolated data. Frequently, trial characteristics like `AUC` or `MAD` are extracted from the resampled data; see Listing 3.

Listing 3 ■ Extracting MAD, AUC, and sample entropy for statistical testing.

```
dat_dvs_trialwise <- dat_interp %>%
   group_by(subject, block, trial, difficulty) %>%
   summarize(
      MAD = max_ad(x_interp, y_interp),
      AUC = auc(x_interp, y_interp),
      SAM = sampen(x_interp)    # other measures are computed similarly
   )

dat_dv_means <- dat_dvs_trialwise %>%
   group_by(subject, difficulty) %>%
   summarize(
      MAD_mean = mean(MAD),
      AUC_mean = mean(AUC),
      SAM_mean = mean(SAM)
   )

t.test(MAD_mean ~ difficulty, data = dat_dv_means, paired = TRUE)
```

A realistic application

Of course, researchers rarely analyze data from just six trials. However, the same approach as exemplified above also works for larger datasets, as we will illustrate by analyzing published data (Wirth et al., 2020, Experiment 1, custom mouse tracker data only). Again, the data and code for this tutorial, along with the scripts used to create the figures, can be accessed at osf.io/75se3.

The experiment that will be analyzed in this example employed a standard stimulus-response compatibility task. Participants classified a color (red vs. green) by moving a mouse cursor from a starting area at the bottom of the screen to one of two target areas at the top (left vs. right). Crucially, the to-be classified color was displayed in one of these two laterally displaced target areas. This task-irrelevant spatial stimulus dimension allows to differentiate between two trial types: spatially compatible trials, where the irrelevant stimulus location matches the required movement direction (e.g., left red stimulus prompting leftward movement), and spatially incompatible trials, where the irrelevant stimulus location does not match the required movement direction (e.g., right red stimulus prompting leftward movement).

The trial-specific data for all participants are stored in the file `trial_data.csv` and are loaded into `dat_trial` which consists of 7200 rows (36 participants with 200 trials each). However, different from the previous example, the trajectory data for each participant are stored in individual files within the `tracking_data/` directory. Thus, `dat_track`, the `data.frame` that contains the pooled tracking information, must be created by merging multiple files. This can be done with dedicated functions or alternatively, by first listing all the filenames, subsequently creating a list where each file is loaded into a `data.frame`, and finally binding the rows of these individual `data.frames` together. Again, we recommend joining `dat_track` and `dat_trial` to create one `data.frame` that contains all relevant information in a tidy format before applying `mousetRajectory` functions; see Listing 4.

It is again advisable to verify certain assumptions about the data, particularly that the spatial information (`X` and `Y` coordinates) is indeed represented in the expected coordinate system. In this experiment, coordinates are recorded relative to the center of the starting area, with positive `X` values indicating positions to the right of the starting area, and positive `Y` values indicating positions above the starting area. `X` and `Y` denote pixels and hence, are not normalized to a trajectory starting at (0, 0) and ending at (1, 1). Therefore, the functions need information about the position and size of the relevant areas: The centers of the target areas were positioned 600 px above the starting area and were laterally displaced by ± 300 px from the center of the starting area. Furthermore, each area had a diameter of 60 px, that is, a radius of 30 px. With this information, the same logic as above can be applied. See Listing 5.

Realistic scenarios differ from the artificial data of the first example in one key aspect: Most experiments exclude certain movements, like those not reaching the target area within a designated time limit or those selecting the wrong target area. In the data of Wirth et al. (2020), the experimental software already recoded in the `accuracy` column whether the correct target area was clicked. However, it can be insightful to explore additional criteria like which target area the mouse first hovered over (see Schoemann et al., 2021, for a systematic comparison of click and hover responses). The code above uses `final_x = nth(X, n = -1)` to achieve this: It creates a new column `final_x` by retrieving the `nth()` value from the `X` column. `n =`

Listing 4 ■ Loading and inspecting the data for the realistic example

```
library(mousetRajectory)
library(tidyverse)

dat_trial <- data.table::fread("trial_data.csv")

head(dat_trial, n = 3)
#>    subject trial  congruency correct_response_location accuracy
#> 1        1   201 incongruent   required_response_right    error
#> 2        1   202   congruent    required_response_left    error
#> 3        1   203 incongruent    required_response_left  correct

nrow(dat_trial)
#> 7200

tracking_files <- list.files("tracking_data")
tracking_files <- paste0("tracking_data/", tracking_files)
tracking_data_list <- lapply(tracking_files, data.table::fread)
dat_track <- bind_rows(tracking_data_list)

head(dat_track, n = 3)
#>    subject trial time X  Y
#> 1        1   201    0 8 -4
#> 2        1   201   18 8 -4
#> 3        1   201   36 8 -4

dat_track <- dat_track %>%
   left_join(dat_trial, by = join_by(subject, trial))

head(dat_track, n = 3)
#>    subject trial time X  Y  congruency correct_response_location accuracy
#> 1        1   201    0 8 -4 incongruent   required_response_right    error
#> 2        1   201   18 8 -4 incongruent   required_response_right    error
#> 3        1   201   36 8 -4 incongruent   required_response_right    error
```

`-1` specifies that `nth()` returns the 1st value of the current group, counting from the end. Since movements were mirrored when the correct target area was on the left, the correct target area is now on the right for all movements, rendering negative `final_x` values an (hover) error.

After merging and filtering the data to keep only the correct movements and relevant movement parts, much of the work is done, and the remaining processing steps are straightforward. To compute average trajectories (see Figure 2A), only the variables in the `group_by()` call have to be adjusted, relative to the code in the previous example. The variables `congruency`, `initiation_t`, and `movement_t` are included in the `group_by()` call because `reframe()` is an aggregation function that discards all non-grouping variables; see the second and third parts of Listing 5. From `dat_plot`, we can get a figure with:

```
dat_plot %>%
   ggplot(aes(
          x = x_condition_mean,
          y = y_condition_mean,
          linetype = congruency)) +
   geom_path()
```

The calculation of trial metrics and their statistical evaluation likewise only requires modifications to the arguments of the `group_by()` calls; see Listing 6.

As illustrated in this example, with the package `mousetRajectory`, a few lines of simple and comprehensible R code suffice to convert complex raw data into easily interpretable metrics that can be evaluated with basic inferential statistical methods.

Extending `mousetRajectory` functionality: Visualizing variability

Results sections in experiments relying on movement trajectory data often include two types of visualizations: First, a plot of average trajectories (as in Figure 2A), and second, line or bar charts summarizing key metrics like area under the curve or maximum absolute deviation. While trajectory plots are typically purely descriptive (i.e., they display solely averages), plots of key metrics are commonly enriched by information on variability to facilitate statistical interpretation (e.g., Pfister et al., 2016). In latter case, obtaining the necessary information to visualize variability is straightforward and thus, standard errors as well

Listing 5 ■ Computing `dat_track`, `dat_interp`, and `dat_plot`

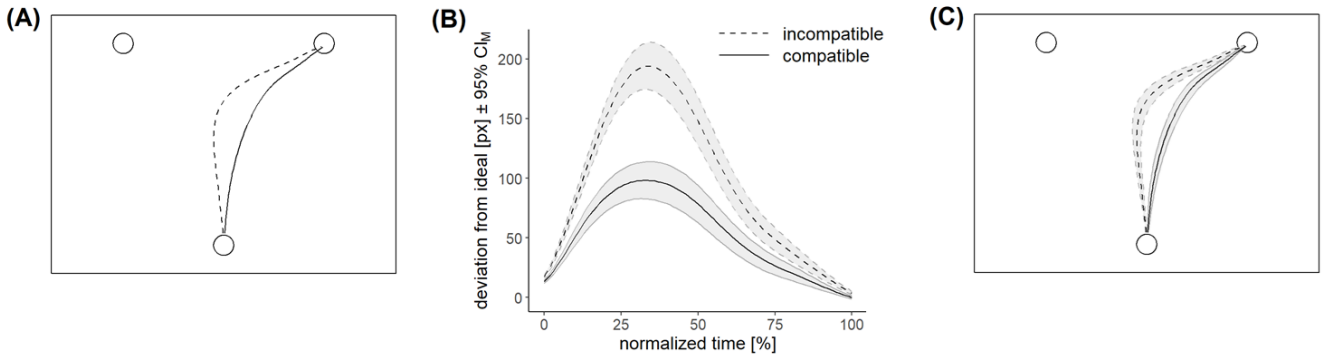
```
dat_track <- dat_track %>%
   group_by(subject, trial) %>%
   mutate(
      X = ifelse(correct_response_location == "required_response_left", -X, X),
      initiation_t = time_circle_left(X, Y, time, x_mid = 0, y_mid = 0, radius = 30),
      completion_t = time_circle_entered(abs(X), Y, time, x_mid = 300, y_mid = 600, radius = 30),
      accuracy = ifelse(is.na(completion_t), "omission_error", accuracy),
      movement_t = completion_t - initiation_t
   ) %>%
   filter(time >= initiation_t & time <= completion_t) %>%
   mutate(
      final_x = nth(X, n = -1)
      accuracy = ifelse(final_x < 0, "comission_error", accuracy)
   ) %>%
   filter(accuracy == "correct")

dat_interp <- dat_track %>%
   group_by(subject, trial, congruency, initiation_t, movement_t) %>%
   reframe(
      t_interp = 0:100,
      x_interp = interp2(time, X, 101),
      y_interp = interp2(time, Y, 101)
   )

dat_plot <- dat_interp %>%
   group_by(subject, congruency, t_interp) %>%
   summarize(
      x_subject_mean = mean(x_interp),
      y_subject_mean = mean(y_interp)
   ) %>%
   group_by(congruency, t_interp) %>%
   summarize(
      x_condition_mean = mean(x_subject_mean),
      y_condition_mean = mean(y_subject_mean)
   )
```

as confidence intervals can easily be calculated from typical aggregate measures (`IT_mean`, `MT_mean`, `MAD_mean`, and `AUC_mean` in the above example). Since mouse tracking studies often employ within-participant designs, participant × condition averages can be directly subjected to methods for computing design-appropriate confidence intervals (Cousineau, 2017; Estes, 1997; Loftus & Masson, 1994). With only two conditions, for example, it is possible to plot the standard error of paired differences or the confidence interval of paired differences, providing a clear index that allows for direct statistical inferences (for a tutorial see Pfister & Janczyk, 2013). More complex designs require different measures, but detailed guides (e.g., Cousineau, 2005; Morey, 2008) and R packages (e.g., the `superb` package; Cousineau et al., 2021) are readily available for those designs as well.

The situation becomes more complex if researchers aim to visualize uncertainty around average trajectories. In this case, each individual timepoint is associated with variability along both the x- and y-axes. For common analysis approaches, however, the variability in 'raw' positional values may not even be particularly meaningful. Rather, the variability in the deviation of a trajectory from the ideal, straight line (the line that is also used to compute the MAD and AUC aggregates) may provide a closer match to the aggregates used in statistical evaluation and thus, can provide a compact and insightful measure of variability for every data point along the trajectory. Because `mousetRajectory` is open-source and written directly in R, its functions can easily be modified to obtain such information. The function `max_ad()` is an ideal starting point for such an endeavor and its code can be inspected by invoking `print(max_ad)` from which we obtain: [4]

[4] Note that `max_ad` must be written without parentheses to `print()` the function itself. Using `print(max_ad())` would `print()` the return value of invoking `max_ad()` without arguments.

Figure 2 ■ Results of Wirth et al. (2020), Experiment 1, custom mouse tracker condition. (A) Average trajectories of movements where the direction was either compatible (solid line) or incompatible (dashed line) to the location of the imperative stimulus. (B) Deviation from the ideal line segment over time, separately for compatible and incompatible movements. Shaded areas represent the corresponding 95% confidence intervals of the mean (95% CI_M). (C) The average trajectories depicted in panel A can be augmented with the 95% CI_M depicted in panel B (see text for details).




```
#> function (x_vector, y_vector,
#>           x_start, y_start, x_end, y_end) {
#>    if (missing(x_start)) {
#>       x_start <- x_vector[1]
#>    }
#>    if (missing(y_start)) {
#>       y_start <- y_vector[1]
#>    }

#>    if (missing(x_end)) {
#>       x_end <- x_vector[length(x_vector)]
#>    }
#>    if (missing(y_end)) {
#>       y_end <- y_vector[length(x_vector)]
#>    }
#>    x_shift <- x_vector - x_start
#>    y_shift <- y_vector - y_start
#>    angle <- atan2((y_end - y_start),
#>                   (x_end - x_start))
#>    m_sin <- sin(-angle)
#>    m_cos <- cos(-angle)
#>    y_rot <- (x_shift * m_sin) +
#>             (y_shift * m_cos)
#>    index <- which.max(abs(y_rot))
#>    return(y_rot[index])
#> }
```

The first 12 lines are solely for user convenience: If certain arguments are missing, that is, if the user did not explicitly specify them, `max_ad()` defaults to using the first and last coordinates of the trajectory as start and end points of the ideal line segment. The subsequent 8 lines of code are more relevant: Initially, the trajectory is 'translated' (i.e., shifted) to move the starting point (`x_start`, `y_start`) to the origin (0, 0). Next, the `angle` between the x-axis and the half-line from the origin through the end point (`x_end`, `y_end`) is calculated. Note that the end point is shifted by the same amount as the trajectory. Using the sine and cosine of this angle, the trajectory is 'rotated' around the origin, thereby aligning the end point with the x-axis. Expressed more formally, the translated coordinates are rotated by –angle radians. After these two rigid body transformations (translation and rotation), the resulting x-coordinates represent[5] the position *on* the ideal line, and the resulting y-coordinates `y_rot` represent the perpendicular deviation from the ideal line. Finally, `max_ad()` returns the y-coordinate at the index where the absolute value of `y_rot` is maximal.

To modify the functionality of `max_ad()`, it is not necessary to thoroughly understand how the coordinates are rotated. Rather, it suffices to understand that the coordinates are rotated around the start point of the ideal line, and that the resulting vector `y_rot` contains the deviations from the ideal line. Thus, omitting the first 12 lines and replacing `return(y_rot[index]))` with `return(y_rot)` yields a custom function that outputs the complete set of deviations from the ideal line rather than a single maximum absolute deviation:

```
all_deviations <- function(x_vector, y_vector,
                           x_start, y_start, x_end, y_end){
   x_shift <- x_vector - x_start
   y_shift <- y_vector - y_start
   angle <- atan2((y_end - y_start),
                  (x_end - x_start))
   m_sin <- sin(-angle)
   m_cos <- cos(-angle)
```

[5] The resulting x-coordinates are irrelevant for questions focusing solely on deviations *from* the ideal line and therefore, it is not necessary to compute them in `max_ad()`. Code calculating rotated x-coordinates can be explored by invoking `print(auc)`.

Listing 6 ■ Extracting `MAD`, `AUC`, and sample entropy for statistical testing.

```
dat_dvs_trialwise <- dat_interp %>%
   group_by(subject, trial, congruency, initiation_t, movement_t) %>%
   summarize(
      MAD = max_ad(x_interp, y_interp, 0, 0, 300, 600),
      AUC = auc(x_interp, y_interp, 0, 0, 300, 600)
   )

dat_dv_means <- dat_dvs_trialwise %>%
   group_by(subject, congruency) %>%
   summarize(
      IT_mean  = mean(initiation_t),
      MT_mean  = mean(movement_t),
      MAD_mean = mean(MAD),
      AUC_mean = mean(AUC)
   )

t.test(IT_mean  ~ congruency, data = dat_dv_means, paired = TRUE)
t.test(MT_mean  ~ congruency, data = dat_dv_means, paired = TRUE)
t.test(MAD_mean ~ congruency, data = dat_dv_means, paired = TRUE)
t.test(AUC_mean ~ congruency, data = dat_dv_means, paired = TRUE)
```

```
   y_rot <- (x_shift * m_sin) +
            (y_shift * m_cos)
   return(y_rot)
}
```

This customized function can then be employed to easily calculate the central moment and dispersion of the deviations from the ideal line for each time point. Here, the 95% confidence intervals of the means (95% CI_M) are computed separately for each condition to provide an estimate of the between-participants variability of the subject × compatibility averages (see Figure 2B). In general, the choice of the measure of uncertainty depends not only on the experimental design, but also on the specific research question. However, one particularly compelling approach is to quantify the dispersion independently for each time point, as this makes the computation of the variability estimate of trajectories similar to the computation of variability estimates of conventional measures with only one data point per trial. In `tidyverse` code, the only required modification is to include the (normalized) time in the appropriate `group_by()` calls; see Listing 7, top part.

Understanding the implementation details of functions not only enables customization of these functions to individual needs but might also spark creative ideas that extend beyond the original functionality. One such idea might be to 'somehow' rotate the 'upright' confidence intervals of Figure 2B to make them perpendicular to the ideal line of Figure 2A, and to then map them onto the corresponding coordinates of the average trajectories (see Figure 2C). Three steps are required to implement this idea: First, the rotation can be adapted from `max_ad()` and `auc()`, but the rotation direction must be reversed by negating the sign of the `angle` (from negative to positive). Second, the upright confidence interval (size) is represented as two-dimensional line segment connecting the points (0, `-dev_condition_CI_half_size`) and (0, `+dev_condition_CI_half_size`), a representation of the confidence interval that the rotation can be applied to. Finally, the rotated confidence interval representation is joined with the corresponding average coordinates. Although this approach is functional, it involves redundant computations because the rotated upper bound of the confidence interval is point-symmetric to the rotated lower bound. Therefore, the code can be shortened by rotating only the upper bound and mirroring it at the end; see Listing 7, bottom part.

Conclusion

Mouse-tracking offers valuable data for behavioral scientists. Unleashing this method's full potential, however, requires access to manifold analytical tools and the possibility to combine them flexibly. The R package `mousetRajectory` provides those tools, covering both preprocessing and the extraction of relevant metrics from 2D movement trajectories. Its algorithms are transparent and easily accessible as they are written directly in the R programming language. Furthermore, it seamlessly integrates with the `tidyverse` framework, and its modular structure allows for customization and extension of each step of the analysis. We, therefore, hope to provide scientists with a low-threshold entry point to apply mouse tracking in their own research.

Authors' note

This package is available on CRAN (doi.org/10.32614/CRAN.package.mousetRajectory) and has a homepage hosted on

Listing 7 ■ Augmenting trajectory plots with confidence intervals.

```
dat_plot_deviations <- dat_interp %>%
   group_by(subject, trial) %>%
   mutate(
      current_deviation = all_deviations(x_interp, y_interp, 0, 0, 300, 600)
   ) %>%
   group_by(subject, congruency, t_interp) %>%
   summarize(
      dev_subject_mean = mean(current_deviation)
   ) %>%
   group_by(congruency, t_interp) %>%
   summarize(
      dev_condition_mean = mean(dev_subject_mean),
      dev_condition_CI_half_size = (sd(dev_subject_mean)/sqrt(n())) * qt(0.975, n()-1),
      dev_condition_CI_upper = dev_condition_mean + dev_condition_CI_half_size,
      dev_condition_CI_lower = dev_condition_mean - dev_condition_CI_half_size
   )

dat_add_CI <- dat_plot_deviations %>%
   mutate(
      angle = atan2(600, 300),
      m_sin = sin(angle),
      m_cos = cos(angle),
      CI_x_offset_rot = 0 * m_cos + dev_condition_CI_half_size * (-m_sin),
      CI_y_offset_rot = 0 * m_sin + dev_condition_CI_half_size * m_cos
   ) %>%
   right_join(dat_plot, by = join_by(t_interp, congruency)) %>%
   mutate(
      CI_x_upper = x_condition_mean + CI_x_offset_rot,
      CI_x_lower = x_condition_mean - CI_x_offset_rot,
      CI_y_upper = y_condition_mean + CI_y_offset_rot,
      CI_y_lower = y_condition_mean - CI_y_offset_rot
   )
```

GitHub (mc-schaaf.github.io/mousetRajectory/). This work was funded by the Heisenberg Programme of the German Research Foundation (PF 853/10-1 490925504, and PF 853/11-1 490927437). Correspondence concerning this article should be addressed to Roland Pfister or Solveig Tonn, Department of Psychology, Trier University, Johanniterufer 15, 54292 Trier, Germany.

References

Camerer, C. F., Johnson, E. J., Rymon, T., & Sen, S. (1993). Cognition and framing in sequential bargaining for gains and losses. In K. Binmore, A. Kirman, & P. Tani (Eds.), *Frontiers of game theory* (pp. 27–47). MIT Press.

Chang, W. (2021). *R6: Encapsulated classes with reference semantics [computer software]*. doi: 10.32614/CRAN.package.R6.

Cousineau, D. (2005). Confidence intervals in within-subject designs: A simpler solution to loftus and masson's method. *Tutorials in Quantitative Methods for Psychology*, *1*(1), 42–45. doi: 10.20982/tqmp.01.1.p042.

Cousineau, D. (2017). Varieties of confidence intervals. *Advances in Cognitive Psychology*, *13*(2), 140–155. doi: 10.5709/acp-0214-z.

Cousineau, D., Goulet, M.-A., & Harding, B. (2021). Summary plots with adjusted error bars: The superb framework with an implementation in r. *Advances in Methods and Practices in Psychological Science*, *4*(3), 1–18. doi: 10.1177/25152459211035109.

Erb, C. D., Moher, J., Sobel, D. M., & Song, J.-H. (2016). Reach tracking reveals dissociable processes underlying cognitive control. *Cognition*, *152*, 114–126. doi: 10.1016/j.cognition.2016.03.015.

Estes, W. K. (1997). On the communication of information by displays of standard errors and confidence intervals. *Psychonomic Bulletin & Review*, *4*(3), 330–341. doi: 10.3758/BF03210790.

Freeman, J. B., & Ambady, N. (2010). Mousetracker: Software for studying real-time mental processing using a computer mouse-tracking method. *Behavior Research Methods*, *42*(1), 226–241. doi: 10.3758/BRM.42.1.226.

Grage, T., Schoemann, M., Kieslich, P. J., & Scherbaum, S. (2019). Lost to translation: How design factors of the mouse-tracking procedure impact the inference from action to cognition. *Attention, Perception & Psychophysics*, *81*(7), 2538–2557. doi: 10.3758/s13414-019-01889-z.

Hehman, E., Stolier, R. M., & Freeman, J. B. (2015). Advanced mouse-tracking analytic techniques for enhancing psychological science. *Group Processes & Intergroup Relations*, *18*(3), 384–401. doi: 10.1177/1368430214538325.

Kieslich, P., & Henninger, F. (2017). Mousetrap: An integrated, open-source mouse-tracking package. *Behavior Research Methods*, *49*(5), 1652–1667. doi: 10.3758/s13428-017-0900-z.

Kieslich, P., Henninger, F., Wulff, D., Haslbeck, J., & Schulte-Mecklenbeck, M. (2018). Mouse-tracking: A practical guide to implementation and analysis. In A. K. Schulte-Mecklenbeck & J. G. Johnson (Eds.), *M* (pp. 111–129). A Handbook of Process Tracing Methods (2nd ed. doi: 10.31234/osf.io/zuvqa.

Kieslich, P., Schoemann, M., Grage, T., Hepp, J., & Scherbaum, S. (2020). Design factors in mouse-tracking: What makes a difference? *Behavior Research Methods*, *52*(1), 317–341. doi: 10.3758/s13428-019-01228-y.

Loftus, G. R., & Masson, M. E. (1994). Using confidence intervals in within-subject designs. *Psychonomic Bulletin & Review*, *1*(4), 476–490. doi: 10.3758/bf03210951.

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). Opensesame: An open-source, graphical experiment builder for the social sciences. *Behavior Research Methods*, *44*(2), 314–324. doi: 10.3758/s13428-011-0168-7.

Mathur, M. B., & Reichling, D. B. (2019). Open-source software for mouse-tracking in qualtrics to measure category competition. *Behavior Research Methods*, *51*(5), 1987–1997. doi: 10.3758/s13428-019-01258-6.

McKinstry, C., Dale, R., & Spivey, M. J. (2008). Action dynamics reveal parallel competition in decision making. *Psychological Science*, *19*(1), 22–24. doi: 10.1111/j.1467-9280.2008.02041.x.

Morey, R. D. (2008). Confidence intervals from normalized data: A correction to cousineau (2005). *Tutorials in Quantitative Methods for Psychology*, *4*(2), 61–64. doi: 10.20982/tqmp.04.2.p061.

Pfister, R., & Janczyk, M. (2013). Confidence intervals for two sample means: Calculation, interpretation, and a few simple rules. *Advances in Cognitive Psychology*, *9*(2), 74–80. doi: 10.2478/v10053-008-0133-x.

Pfister, R., Wirth, R., Schwarz, K. A., Steinhauser, M., & Kunde, W. (2016). Burdens of non-conformity: Motor execution reveals cognitive conflict during deliberate rule violations. *Cognition*, *147*, 93–99. doi: 10.1016/j.cognition.2015.11.009.

R Core Team. (2023). *R: A language and environment for statistical computing*. https://www.R-project.org/

Richman, J. S., & Moorman, J. R. (2000). Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology: Heart and Circulatory Physiology*, *278*(6), H2039–49. doi: 10.1152/ajpheart.2000.278.6.H2039.

Schoemann, M., O'Hora, D., Dale, R., & Scherbaum, S. (2021). Using mouse cursor tracking to investigate online cognition: Preserving methodological ingenuity while moving toward reproducible science. *Psychonomic Bulletin & Review*, *28*(3), 766–787. doi: 10.3758/s13423-020-01851-3.

Signal Developers. (2014). *Signal: Signal processing [computer software]*. doi: 10.32614/CRAN.package.signal.

Song, J.-H., & Nakayama, K. (2009). Hidden cognitive states revealed in choice reaching tasks. *Trends in Cognitive Sciences*, *13*(8), 360–366. doi: 10.1016/j.tics.2009.04.009.

Spivey, M. J., Grosjean, M., & Knoblich, G. (2005). Continuous attraction toward phonological competitors. *Proceedings of the National Academy of Sciences of the United States of America*, *102*(29), 10393–10398. doi: 10.1073/pnas.0503903102.

van der Wel, R. P. R. D., Sebanz, N., & Knoblich, G. (2014). Do people automatically track others' beliefs? evidence from a continuous measure. *Cognition*, *130*(1), 128–133. doi: 10.1016/j.cognition.2013.10.004.

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, *59*(10), 1–23. doi: 10.18637/jss.v059.i10.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T., Miller, E., Bache, S., Müller, K., Ooms, J., Robinson, D., Seidel, D., Spinu, V., Yutani, H., et al. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. doi: 10.21105/joss.01686.

Wickham, H., Grothendieck, G., Kates, L., & Petzoldt, T. (2016). *Proto: Prototype object-based programming [computer software]*. doi: 10.32614/CRAN.package.proto.

Wirth, R., Foerster, A., Kunde, W., & Pfister, R. (2020). Design choices: Empirical recommendations for designing two-dimensional finger-tracking experiments. *Behavior Research Methods*, *52*(6), 2394–2416. doi: 10.3758/s13428-020-01409-0.

Wulff, D. U., Kieslich, P. J., Henninger, F., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2021). *Movement tracking of cognitive processes: A tutorial using mousetrap*. doi: 10.31234/osf.io/v685r.

Open practices

The *Open Material* badge was earned because supplementary material(s) are available on mc-schaaf.github.io/mousetRajectory/

Citation

Pfister, R., Tonn, S., Schaaf, M., & Wirth, R. (2024). Mousetrajectory: Mouse tracking analyses for behavioral scientists. *The Quantitative Methods for Psychology*, *20*(3), 217–229. doi: 10.20982/tqmp.20.3.p217.

Received: 26/03 /2024 ∼ Accepted: 02/09/2024