Package ‘mousetrap’

December 5, 2019

Type Package

Version 3.1.4

Date 2019-12-05

Title Process and Analyze Mouse-Tracking Data

Description Mouse-tracking, the analysis of mouse movements in computerized experiments, is a method that is becoming increasingly popular in the cognitive sciences. The mousetrap package offers functions for importing, preprocessing, analyzing, aggregating, and visualizing mouse-tracking data.

Maintainer Pascal J. Kieslich <kieslich@psychologie.uni-mannheim.de>

URL https://github.com/pascalkieslich/mousetrap

BugReports https://github.com/pascalkieslich/mousetrap/issues

License GPL-3

Depends R (>= 3.1.0),

Imports utils, stats, pracma, dplyr (>= 0.5.0), tidyr, magrittr, graphics, grDevices, ggplot2, scales, psych (>= 1.2.4), Rcpp (>= 0.11.4), diptest, RColorBrewer, cstab, fastcluster, parallel, fields

Suggests readbulk, testthat

LinkingTo Rcpp

Encoding UTF-8

LazyData TRUE

RoxygenNote 6.1.1

NeedsCompilation yes

Author Pascal J. Kieslich [aut, cre] (<https://orcid.org/0000-0002-0853-9364>),
Dirk U. Wulff [aut] (<https://orcid.org/0000-0002-4008-8022>),
Felix Henninger [aut] (<https://orcid.org/0000-0002-7730-9511>),
Jonas M. B. Haslbeck [aut],
Sarah Brockhaus [ctb]

Repository CRAN

Date/Publication 2019-12-05 14:40:02 UTC

R topics documented:

`bezier` *Create Bezier-curves using the Bernstein approximation.*

Description

`bezier` creates 3-point Bezier-curves using the Bernstein approximation to simulate continuous competition in mouse- and hand-trajectories.

Usage

```
bezier(x = c(0, 1, -1), y = c(0, 1.5, 1.5), w = 1, resol = 100)
```

Arguments

`x`	a numeric vector giving the x-coordinates of exactly three Bezier-points. Defaults to c(0,1,-1) matching the 'mt' format in mt_align.
`y`	a numeric vector giving the x-coordinates of exactly three Bezier-points. Defaults to c(0,1.5,1.5) matching the 'mt' format in mt_align.
`w`	a numeric value or vector specifying one or several Bezier curves, with `w` governing the pull towards the middle point. Each entry in `w` creates one Bezier-curve.
`resol`	a numeric value specifying the spatial resolution of the bezier curves. For example, `resol = 100` creates bezier curves comprised of 100 points each.

Value

A trajectory array containing the bezier curves.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Examples

```
# Generate range of Bezier-curves
bezier_curves <- bezier(w=seq(0,10,.1))

# Plot curves
mt_plot(bezier_curves)
```

`bimodality_coefficient`

Calculate bimodality coefficient.

Description

Calculate the bimodality coefficient for a numeric vector as specified in Pfister et al. (2013).

Usage

```
bimodality_coefficient(x, na.rm = FALSE)
```

Arguments

`x`	a numeric vector.
`na.rm`	logical specifying whether missing values should be removed.

Details

The calculation of the bimodality coefficient involves calculating the skewness and kurtosis of the distribution first. For this, the skew and kurtosi functions of the `psych` package are used. Note that type is set to "2" for these functions in accordance with Pfister et al. (2013).

Value

A numeric value.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

Pfister, R., Schwarz, K. A., Janczyk, M., Dale, R., & Freeman, J. B. (2013). Good things peak in pairs: A note on the bimodality coefficient. *Frontiers in Psychology, 4*, 700. http://dx.doi.org/10.3389/fpsyg.2013.00700

See Also

skew and kurtosi for calculating skewness and kurtosis.

mt_check_bimodality for assessing bimodality using several methods in a mousetrap data object.

Examples

```
pfister_data_a <- rep(1:11, times=c(3,5,5,10,17,20,17,10,5,5,3))
bimodality_coefficient(pfister_data_a) #.34
pfister_data_b <- rep(1:11, times=c(2,26,14,6,2,0,2,6,14,26,2))
bimodality_coefficient(pfister_data_b) #.79
```

KH2017 *Mouse-tracking dataset from Kieslich & Henninger (2017)*

Description

A data object of class "mousetrap" with the imported and preprocessed mouse-tracking data from Kieslich & Henninger (2017). More information about the study and raw data can be found in KH2017_raw.

Usage

```
KH2017
```

Format

A mousetrap data object is a list containing at least the following objects:

- `data`: a data.frame containing the trial data (from which the mouse-tracking data columns have been removed). More information about the content of the trial data in `KH2017` can be found in KH2017_raw. The rownames of `data` correspond to the trial identifier. For convenience, the trial identifier is also stored in an additional column called "mt_id".
- `trajectories`: an array containing the raw mouse-tracking trajectories. The first dimension represents the different trials and the dimension names (which can be accessed using rownames) correspond to the trial identifier (the same identifier that is used as the `rownames` in `data`). The second dimension corresponds to the samples taken over time which are included in chronological order. The third dimension corresponds to the different mouse-tracking variables (timestamps, x-positions, y-positions) which are usually called `timestamps`, `xpos`, and `ypos`.

Some functions in this package (e.g., mt_time_normalize and mt_average) add additional trajectory arrays (e.g., `tn_trajectories` and `av_trajectories`) to the mousetrap data object. Other functions modify the existing arrays (e.g., mt_derivatives adds distance, velocity, and acceleration to an existing dataset). Finally mt_measures adds an additional data.frame with mouse-tracking measures to it.

Details

The raw dataset (KH2017_raw) was filtered keeping only correctly answered trials. The filtered dataset was imported using mt_import_mousetrap. Trajectories were then remapped using mt_remap_symmetric so that all trajectories end in the top-left corner and their starting point was aligned to a common value (0,0) using mt_align_start.

References

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

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15-28. https://doi.org/10.3758/BF03195938

`KH2017_raw` *Raw mouse-tracking dataset from Kieslich & Henninger (2017)*

Description

Raw mouse-tracking dataset from Kieslich & Henninger (2017), an experiment using the material and procedure of experiment 1 by Dale et al. (2007). A preprocessed (as opposed to raw) version of the same data can be found in KH2017.

Usage

```
KH2017_raw
```

Format

A data.frame with 1140 rows and 19 variables. The data.frame is based on the combined raw data that were created using read_opensesame from the readbulk library. For ease of use, unnecessary columns were excluded.

The variables included relate to the item that was presented (`Exemplar`), the answer categories (`Category1` and `Category2`), the subject identifier (`subject_nr`), the subjects' response (`response`), the correctness of the response (`response`) as well as the mouse-tracking variables (`timestamps_get_response`, `xpos_get_response` and `ypos_get_response`)

Each mouse-tracking variable contains a list of values (separated by ', ') - one entry for each recorded position of the mouse. The position coordinates are given in pixels, such that values of zero for both `xpos_get_response` and `ypos_get_response` indicate that the cursor is located in the center of the screen. Both variables increase in value as the mouse moves toward the bottom right. Timestamps are given in milliseconds.

Details

The data stem from a study by Kieslich & Henninger (2017) which used the material and procedure of experiment 1 by Dale et al. (2007). In this experiment, participants have to assign exemplars (e.g., "whale") to one of two categories (e.g., "fish" or "mammal") by clicking on the button corresponding to the correct category. All exemplars and categories from the original study were translated to and presented in German.

The data was collected in OpenSesame using the mousetrap plugin (Kieslich & Henninger, 2017).

Across the 19 trials of the experiment, 60 participants categorized 13 exemplars that were typical of their category and 6 atypical exemplars for which this was not the case. For the atypical exemplars (e.g., "whale"), the competing category ("fish") was selected to compete with the correct category ("mammal"). The hypothesis under investigation is whether participants' mouse trajectories deviate more towards the competing category for the atypical exemplars, indicating increased conflict between the response options.

References

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

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15-28. https://doi.org/10.3758/BF03195938

`mousetrap` *Process and analyze mouse-tracking data*

Description

The mousetrap package provides functions for importing, preprocessing, analyzing, aggregating, and visualizing mouse-tracking data. In the following, a brief overview of the functions in this package is given.

Read functions

Depending on the file format, one of the standard R functions for reading files into R can be used (e.g., read.table or read.csv).

If raw data were collected using MouseTracker, the mousetrap package provides the read_mt function to read files in the ".mt" format.

If several raw data files should be read and merged, the read_bulk function from the readbulk package can be used (or the read_opensesame function, if data were collected using OpenSesame).

Import functions

The initial step to prepare data for analysis in the mousetrap package is to create a mousetrap data object. Depending on the input format, one of the following functions can be used. A detailed description (and example) of the resulting mousetrap data object can be found in mt_example.

mt_import_mousetrap imports mouse-tracking data that were recorded using the mousetrap plugin for OpenSesame.

mt_import_wide imports mouse-tracking data saved in a wide format (e.g., data collected using MouseTracker).

mt_import_long imports mouse-tracking data saved in a long format. (e.g., trajectories exported using mt_export_long).

Geometric preprocessing functions

A number of functions are available that perform geometric preprocessing operations.

mt_remap_symmetric remaps mouse trajectories to one side (or one quadrant) of the coordinate system.

mt_align is a general purpose function for aligning and rescaling trajectories. For specific operations, you can rely on one of the following functions.

mt_align_start aligns the start position of trajectories.

mt_align_start_end aligns all trajectories so that they share a common initial and final coordinate (this is also sometimes referred to as "space-normalization").

Resampling and interpolation functions

A number of functions are available that perform resampling and interpolation operations.

mt_exclude_initiation excludes the initial phase of a trial without mouse movement.

mt_time_normalize performs time-normalization using equidistant time intervals, resulting in an identical number of samples for all trajectories.

mt_resample resamples trajectories so that samples occur at constant intervals of a specified length.

mt_average averages trajectory coordinates (and related variables) for time bins of constant duration.

mt_spatialize re-represents each trajectory spatially so that adjacent points on the trajectory become equidistant to each other.

Data handling functions

A number of functions are available for data handling operations, such as filtering or adding of new variables or trajectories.

mt_subset filters mouse-tracking data by trials, so that only those meeting the defined criteria are included.

mt_add_variables adds new, self created variables to a trajectory array.

mt_add_trajectory adds a new trajectory to a trajectory array.

mt_bind joins two trajectory arrays.

Analysis functions

A number of different analysis procedures and summary statistics for mouse trajectories have been established in the existing literature. The following functions implement many of these approaches.

mt_derivatives calculates distance, velocity, and acceleration for trajectories.

mt_angles calculates movement angles for trajectories.

mt_deviations calculates the deviations from an idealized trajectory (straight line).

mt_measures calculates a set of mouse-tracking measures.

mt_sample_entropy calculates sample entropy.

mt_standardize standardizes mouse-tracking measures onto a common scale (separately for subsets of the data, e.g., per participant).

mt_scale_trajectories provides different options for standardizing variables in a mouse trajectory array.

mt_check_bimodality assesses the bimodality of mouse-tracking measure distributions.

mt_check_resolution checks the (temporal) logging resolution of raw trajectories.

mt_count counts the number of observations for each trajectory.

Cluster functions

A number of different functions for clustering trajectories is provided.

mt_distmat computes the dissimilarity/distance between each pair of trajectories.

mt_cluster performs trajectory clustering with a specified number of clusters.

mt_cluster_k estimates the optimal number of clusters using various methods.

mt_map maps trajectories onto a predefined set of prototype trajectories (a core set is provided in mt_prototypes).

Reshaping, aggregation, and export functions

A number of helper functions are provided for aggregating, plotting, and exporting the multi-dimensional mouse trajectory arrays.

mt_reshape is a general purpose reshaping and aggregation function for mousetrap data.

mt_aggregate aggregates mouse-tracking data per condition.

mt_aggregate_per_subject aggregates mouse-tracking data per (within subjects-) condition separately for each subject.

mt_export_long exports mouse-tracking data in long format.

mt_export_wide exports mouse-tracking data in wide format.

Visualization functions

The following functions can be used for plotting trajectory data, e.g., individual and aggregated trajectories or velocity profiles.

mt_plot plots individual trajectory data.

mt_plot_aggregate plots aggregated trajectory data.

mt_plot_add_rect adds rectangles to a trajectory plot.

mt_plot_riverbed plots the relative frequency of a selected variable across time.

mt_plot_per_trajectory creates a pdf with separate plots per trajectory.

mt_heatmap and mt_heatmap_ggplot plot trajectory heatmaps.

mt_diffmap for creating a difference-heatmap of two trajectory heatmap images.

mt_animate creates a gif trajectory animation.

Helper functions

bimodality_coefficient calculates the bimodality coefficient.

scale_within scales and centers variables within the levels of another variable.

bezier creates Bezier-curves using the Bernstein approximation.

Datasets

mt_example and mt_example_raw contain a mouse-tracking example dataset for demonstrations using the `mousetrap` package.

KH2017 and KH2017_raw contain a mouse-tracking dataset from Kieslich & Henninger (2017).

Examples

```
## Not run:
KH2017 <- mt_import_mousetrap(subset(KH2017_raw,correct==1))
KH2017 <- mt_remap_symmetric(KH2017)
KH2017 <- mt_align_start(KH2017)

## End(Not run)

KH2017 <- mt_time_normalize(KH2017)
KH2017 <- mt_measures(KH2017)

mt_aggregate(
  KH2017, use="measures",
  use_variables=c("MAD", "AD"),
  use2_variables="Condition",
  subject_id="subject_nr"
)

mt_plot_aggregate(KH2017,
  use="tn_trajectories",
  x="xpos", y="ypos", color="Condition",
  subject_id="subject_nr"
)

## Not run:
mt_plot(KH2017,
  use="tn_trajectories",
  x="xpos", y="ypos", color="Condition"
)
```

```
## End(Not run)
```

`mt_add_trajectory` *Add new trajectory to trajectory array.*

Description

Add a single new trajectory to trajectory array.

Usage

```
mt_add_trajectory(data, use = "trajectories", save_as = use,
  xpos = NULL, ypos = NULL, xypos = NULL, id = "new")
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`xpos`	a vector of x positions. Ignored, if `xypos` is provided.
`ypos`	a vector of y positions. Ignored, if `xypos` is provided.
`xypos`	a matrix, the first column corresponding to the x positions, the second to the y positions.
`id`	a character string specifying the identifier of the to be added trajectory.

Value

A mousetrap data object (see mt_example) where the new trajectory has been added. If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

Examples

```
# Add additional prototype to mt_prototypes
mt_prototypes_ext <- mt_add_trajectory(mt_prototypes,
   xpos = c(0,1,-1,1,-1), ypos = c(0,1.5,1.5,1.5,1.5), id = "dCoM3"
)
```

mt_add_variables *Add new variables to trajectory array.*

Description

Add new variables to the trajectory array (and remove potentially existing variables of the same name). This is mostly a helper function used by other functions in this package (e.g., mt_deviations). However, it can also be helpful if the user has calculated new variables for each logged coordinate and wants to add them to an existing trajectory array.

Usage

```
mt_add_variables(data, use = "trajectories", save_as = use, variables)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`variables`	a list of matrices that each contain the data of one of the to be added variables. In this case, the new variables with their values are added as a new entry in the trajectory arrays third dimension. Alternatively, a character vector specifying the name of the new variables that should be added to the trajectory array. In this case, the new variables are filled with `NA`s.

Value

A mousetrap data object (see mt_example) where the new variables have been added to the trajectory array. Depending on the input to `variables`, the values for the added variables are either `NA`s or their actual values. If columns of the same name already existed, they have been removed. If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (`<kieslich@psychologie.uni-mannheim.de>`)

Felix Henninger

Examples

```
# Calculate new (arbitrary) variables for this example
# ... the sum of the x- and y-positions
xy_sum <- mt_example$trajectories[,,"xpos"] + mt_example$trajectories[,,"ypos"]
# ... the product of the x- and y-positions
xy_prod <- mt_example$trajectories[,,"xpos"] * mt_example$trajectories[,,"ypos"]
```

```
# Add the new variables to the trajectory array
mt_example <- mt_add_variables(mt_example,
  variables=list(xy_sum=xy_sum, xy_prod=xy_prod))
```

---

mt_aggregate *Aggregate mouse-tracking data per condition.*

Description

mt_aggregate is used for aggregating mouse-tracking measures (or trajectories) per condition. One or several condition variables can be specified using use2_variables. Aggregation will be performed separately for each level of the condition variables. mt_aggregate is a wrapper function for mt_reshape.

Usage

```
mt_aggregate(data, use = "measures", use_variables = NULL,
  use2 = "data", use2_variables = NULL, subject_id = NULL,
  trajectories_long = TRUE, ...)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which dataset should be aggregated. The corresponding data are selected from data using data[[use]]. Usually, this value corresponds to either "tn_trajectories" or "measures", depending on whether the time-normalized trajectories or derived measures should be aggregated.
use_variables	a character vector specifying the mouse-tracking variables to aggregate. If a data.frame with mouse-tracking measures is provided as data, this corresponds to the column names. If a trajectory array is provided, this argument should specify the labels of respective array dimensions. If unspecified, all variables will be aggregated.
use2	a character string specifying where the data containing the condition information can be found. Defaults to "data" as data[["data"]] usually contains all non mouse-tracking trial data. Alternatively, a data.frame can be provided directly.
use2_variables	a character string (or vector) specifying the variables (in data[[use2]]) across which the trajectories / measures will be aggregated. For each combination of levels of the grouping variable(s), aggregation will be performed separately using summarize_at.
subject_id	an optional character string specifying the column that contains the subject identifier. If specified, aggregation will be performed within subjects first (i.e., within subjects for all available values of the grouping variables specified in use2_variables).

`trajectories_long`
: logical indicating if the reshaped trajectories should be returned in long or wide format. If TRUE, every recorded position in a trajectory is placed in another row (whereby the order of the positions is logged in the variable mt_seq). If FALSE, every trajectory is saved in wide format and the respective positions are indexed by adding an integer to the corresponding label (e.g., xpos_1, xpos_2, ...). Only relevant if data[[use]] contains trajectories.

`...`
: additional arguments passed on to mt_reshape (such as subset).

Value

A data.frame containing the aggregated data.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_aggregate_per_subject for aggregating mouse-tracking measures and trajectories per subject.

summarize_at for aggregating data using the dplyr package.

Examples

```
# Time-normalize trajectories
mt_example <- mt_time_normalize(mt_example)

# Aggregate time-normalized trajectories per condition
average_trajectories <-  mt_aggregate(mt_example,
  use="tn_trajectories",
  use2_variables="Condition"
)


# Calculate mouse-tracking measures
mt_example <- mt_measures(mt_example)

# Aggregate measures per condition
average_measures <- mt_aggregate(mt_example,
  use="measures", use_variables=c("MAD", "AD"),
  use2_variables="Condition"
)

# Aggregate measures per condition
# first within subjects and then across subjects
average_measures <- mt_aggregate(mt_example,
  use="measures", use_variables=c("MAD", "AD"),
  use2_variables="Condition",
  subject_id="subject_nr"
)
```

mt_aggregate_per_subject

Aggregate mouse-tracking data per condition separately for each subject.

Description

mt_aggregate_per_subject can be used for aggregating mouse-tracking measures (or trajectories) per condition separately for each subject. One or more condition variables can be specified using use2_variables. Aggregation will be performed separately for each level of the condition variables. mt_aggregate_per_subject is a wrapper function for mt_reshape.

Usage

```
mt_aggregate_per_subject(data, use = "measures", use_variables = NULL,
  use2 = "data", use2_variables = NULL, subject_id,
  trajectories_long = TRUE, ...)
```

Arguments

data
: a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).

use
: a character string specifying which dataset should be aggregated. The corresponding data are selected from data using data[[use]]. Usually, this value corresponds to either "tn_trajectories" or "measures", depending on whether the time-normalized trajectories or derived measures should be aggregated.

use_variables
: a character vector specifying the mouse-tracking variables to aggregate. If a data.frame with mouse-tracking measures is provided as data, this corresponds to the column names. If a trajectory array is provided, this argument should specify the labels of respective array dimensions. If unspecified, all variables will be aggregated.

use2
: a character string specifying where the data containing the condition information can be found. Defaults to "data" as data[["data"]] usually contains all non mouse-tracking trial data. Alternatively, a data.frame can be provided directly.

use2_variables
: a character string (or vector) specifying the variables (in data[[use2]]) across which the trajectories / measures will be aggregated. For each combination of levels of the grouping variable(s), aggregation will be performed separately using summarize_at.

subject_id
: a character string specifying which column contains the subject identifier.

trajectories_long
: logical indicating if the reshaped trajectories should be returned in long or wide format. If TRUE, every recorded position in a trajectory is placed in another row

(whereby the order of the positions is logged in the variable mt_seq). If FALSE, every trajectory is saved in wide format and the respective positions are indexed by adding an integer to the corresponding label (e.g., xpos_1, xpos_2, ...). Only relevant if data[[use]] contains trajectories.

... additional arguments passed on to mt_reshape (such as subset).

Value

A data.frame containing the aggregated data.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_aggregate for aggregating mouse-tracking measures and trajectories per condition.

summarize_at for aggregating data using the dplyr package.

Examples

```
# Time-normalize trajectories
mt_example <- mt_time_normalize(mt_example)

# Aggregate time-normalized trajectories per condition
# separately per subject
average_trajectories <- mt_aggregate_per_subject(
  mt_example,
  use="tn_trajectories",
  use2_variables="Condition",
  subject_id="subject_nr"
)


# Calculate mouse-tracking measures
mt_example <- mt_measures(mt_example)

# Aggregate measures per condition
# separately per subject
average_measures <- mt_aggregate_per_subject(
  mt_example,
  use="measures",
  use_variables=c("MAD", "AD"),
  use2_variables="Condition",
  subject_id="subject_nr"
)
```

mt_align — *Align trajectories.*

Description

mt_align aligns trajectories to a common start point, end point, and / or coordinate system.

Usage

```
mt_align(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), coordinates = "isotropic",
  align_start = FALSE, align_end = FALSE, align_side = "no",
  verbose = FALSE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
save_as	a character string specifying where the resulting trajectory data should be stored.
dimensions	a character string specifying which trajectory variables should be used. Can be of length 2 or 3 for two-dimensional or three-dimensional alignment respectively.
coordinates	either a numeric vector of length 4 specifying the xstart, ystart, xend, yend coordinates of the trajectory start and end points. Can also be isotropic (the default) to preserve the coordinates of dim1 and dim2, isotropic-norm to set the coordinates to c(0,0,-1,x) where x is chosen to preserve the aspect ratio of dim1 and dim2, mt to set coordinates to c(0,0,-1,1.5), norm to set coordinates to c(0,0,-1,1), and wide to set coordinates to c(0,0,-1,1.2). In the three-dimensional case, coordinates is a vector of length 6.
align_start	boolean specifying whether the start points of all trajectories should be aligned to the position specified in coordinates. See Details.
align_end	boolean specifying whether the end points of all trajectories should be aligned to the position specified in coordinates. See Details.
align_side	character string specifying whether all trajectories should be flipped to the left side (left), the right side (right), or not at all (no). Assumes that first entry in dimensions are the x positions.
verbose	logical indicating whether function should report its progress.

Details

If align_start / align_end is FALSE, coordinates define the position of the average start / end point across all trajectories.

Note that if the end points of trajectories are not aligned, coordinates refer to the hypothetical case where all trajectories are mapped to one side.

If `align_start` / `align_end` is `TRUE`, the start / end point of each trajectory is set to the exact position specified in `coordinates`. `align_start` and `align_end` can be set completely independently of one another, i.e., one can align only end points, only start points, none, or both.

If `align_start` is set to `"left"` or `"right"` trajectories will be flipped to the lower or upper spectrum of the first dimensions, respectively. If the first dimension is the x-coordinate this is equivalent to flipping the trajectories to the left and right side, respectively.

Value

A mousetrap data object (see mt_example) with aligned trajectories. Per default, the dimensions in the original trajectory array will be replaced. If a different trajectory array is specified using `save_as`, a new trajectory array will be created (including only the aligned dimensions). If a trajectory array was provided directly as `data`, only the aligned trajectories will be returned.

Author(s)

Dirk U. Wulff (`<dirk.wulff@gmail.com>`)

See Also

mt_align_start for aligning all trajectories to a common start position.

mt_align_start_end for aligning all trajectories so that they share a common initial and final coordinate.

mt_remap_symmetric for remapping trajectories to one side (or one quadrant) of the coordinate system.

Examples

```
mt_example <- mt_align(mt_example,
  align_start = TRUE, align_end = TRUE,
  coordinates = 'mt')
```

`mt_align_start` *Align start position of trajectories.*

Description

Adjust trajectories so that all trajectories have the same start position.

Usage

```
mt_align_start(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), start = c(0, 0), verbose = FALSE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
save_as	a character string specifying where the resulting trajectory data should be stored.
dimensions	a character vector specifying the dimensions in the trajectory array that should be aligned.
start	a numeric vector specifying the start values for each dimension, i.e., the values the first recorded position should have in every trial. If NULL, trajectories are aligned to the mean first position across all trials.
verbose	logical indicating whether function should report its progress.

Value

A mousetrap data object (see mt_example) with aligned trajectories. All other trajectory dimensions not specified in dimensions (e.g., timestamps) will be kept as is in the resulting trajectory array. If the trajectory array was provided directly as data, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_align_start_end for aligning the start and end position of trajectories.

mt_align as a general purpose function for aligning and rescaling trajectories.

mt_remap_symmetric for remapping trajectories.

Examples

```
# Import raw trajectories for demonstration
mt_example <- mt_import_mousetrap(mt_example_raw)

# Align trajectories to start coordinates (0,0)
mt_example <- mt_align_start(mt_example,
  start=c(0,0))


# Import raw trajectories for demonstration
mt_example <- mt_import_mousetrap(mt_example_raw)

# Align trajectories to mean first coordinates
mt_example <- mt_align_start(mt_example,
  start=NULL)
```

mt_align_start_end *Align start and end position of trajectories.*

Description

Adjust trajectories so that all trajectories have an identical start and end point. In some articles, this is also referred to as space-normalization (e.g. Dale et al., 2007).

Usage

```
mt_align_start_end(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), start = c(0, 0), end = c(-1, 1),
  verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`dimensions`	a character vector specifying the dimensions in the trajectory array that should be aligned.
`start`	a numeric vector specifying the start values for each dimension, i.e., the values the first recorded position should have in every trial. If `NULL`, trajectories are aligned to the mean first position across all trials.
`end`	a numeric vector specifying the end values for each dimension, i.e., the values the last recorded position should have in every trial. If `NULL`, trajectories are aligned to the mean last position across all trials. Note that in this case trajectories should be remapped to one side before alignment (e.g., using mt_remap_symmetric) so that the mean last position is meaningful.
`verbose`	logical indicating whether function should report its progress.

Value

A mousetrap data object (see mt_example) with aligned trajectories. All other trajectory dimensions not specified in `dimensions` (e.g., timestamps) will be kept as is in the resulting trajectory array. If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (`<kieslich@psychologie.uni-mannheim.de>`)

Felix Henninger

References

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15-28.

See Also

mt_align_start for aligning the start position of trajectories.

mt_align as a general purpose function for aligning and rescaling trajectories.

mt_remap_symmetric for remapping trajectories.

Examples

```
# Align start and end positions to specific coordinates
mt_example <- mt_align_start_end(mt_example,
  start=c(0,0), end=c(-1,1))


# Import raw trajectories for demonstration
mt_example <- mt_import_mousetrap(mt_example_raw)

# Remap trajectories
mt_example <- mt_remap_symmetric(mt_example)

# Align trajectories to mean first and last coordinates
mt_example <- mt_align_start_end(mt_example,
  start=NULL, end=NULL)
```

`mt_angles` *Calculate movement angles.*

Description

Calculate point-based and vertical-based angles for the points in the movement trajectory. Point-based angles are the angle defined by three subsequent points on the trajectory. Vertical-based angles are the angles between two subsequent points and the vertical axis.

Usage

```
mt_angles(data, use = "trajectories", dimensions = c("xpos", "ypos"),
  save_as = use, na_replace = FALSE, unit = "radian",
  verbose = FALSE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
dimensions	a character string specifying which trajectory variables should be used. Must be of length 2.
save_as	a character string specifying where the resulting trajectory data should be stored.
na_replace	logical specifying whether NAs in the angle values should be replaced using the next existing angle value (see Details). Defaults to FALSE.
unit	character specifying the unit for the angles. Default is "radian", alternative is "degree".
verbose	logical indicating whether function should report its progress.

Details

By default, angles are reported in radians, the alternative is degrees. For the first point in a trajectory, the angle values are always not defined (NA).

For vertical-based angles (angle_v), positive values indicate a movement to the left of the vertical, negative values to the right of the vertical. If there was no movement across two consecutive points, angle_v is not defined and, by default, NA is returned. If na_replace is TRUE, the next existing angle value is reported instead.

For point-based angles (angle_p), angles indicate changes of movement within three consecutive time steps. The reported angle is always the smaller one. A value of pi (= 3.14...) (for radians) or 180 (for degrees) indicates a constant movement direction, a value of 0 (both for radians and degrees) a complete reversal. If there was no movement across two consecutive points, angle_p is not defined and, by default, NA is returned. If na_replace is TRUE, the next existing angle value is reported instead. angle_p is also not defined for the last point of the trajectory.

Value

A mousetrap data object (see mt_example) with point-based and vertical-based angles added as additional variables to the trajectory array (called angle_p and angle_v). If a trajectory array was provided directly as data, only the trajectory array will be returned.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Examples

```
# Calculate movement angles
mt_example <- mt_angles(mt_example)

# Calculate movement angles (in degree)
# and replace NAs with next existing value
mt_example <- mt_angles(mt_example,
```

```
unit="degree", na_replace=TRUE)
```

---

mt_animate *Create gif trajectory animation.*

Description

mt_animate animates trajectories using the animation package. Note that this function has beta status.

Usage

```
mt_animate(data, use = "trajectories", dimensions = c("xpos", "ypos"),
  timestamps = "timestamps", filename = "trajectory_animation.gif",
  xres = 1000, seconds = 3, framerate = 24, speed = 0.5,
  density = 3, jitter = TRUE, remove = FALSE, bg = "black",
  col = "white", lwd = 1, loop = FALSE, bounds = NULL,
  norm = FALSE, upscale = 1, decay = 10, max_intensity = 5,
  discard_images = TRUE, im_path = NULL, parallel = TRUE,
  verbose = FALSE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
dimensions	a character vector specifying the two dimensions in the trajectory array that contain the mouse positions. Usually (and by default), the first value in the vector corresponds to the x-positions (xpos) and the second to the y-positions (ypos).
timestamps	a character string specifying the trajectory dimension containing the timestamps. If NULL linearly increasing timestamps are assumed, producing a perfectly constant timestamp interval.
filename	character string specifying the path and filename of the resulting *.gif*. If the extension of filename is not *.gif*, *.gif* is added at the end. Must not contain spaces.
xres	numeric specifying the resolution of the *.gif* file.
seconds	numeric specifying the duration of the *.gif* file.
framerate	numeric specifying the framerate of the *.gif* file. Defaults to 24 implying smooth non-discrete frame transitions for the human eye.
speed	numeric specifying the speed of the trajectories with regard to their original velocity profile. I.e., a value of .5 shows trajectories in half of the original velocities, whereas a value of 2 shows trajectories in double of the original velocities.

density	integer specifying the number of trajectories to be added each frame. I.e., if density = 10, seconds = 10, framerate = 24 and speed = .5 then the animation will show 10 x 10 x 24 x .5 = 1200 trajectories.
jitter	logical specifying whether the density should be jittered. If TRUE, density varies according to rgeom(1/density).
remove	logical specifying whether trajectories that reached their end points should be removed from the rest of the animation. Defaults to FALSE implying that all finished trajectories remain visible.
bg	character string specifying the background color.
col	character string specifying the foreground color, i.e., the color used to draw the trajectories.
lwd	numeric specifying the line width of the trajectories.
loop	logical specifying whether gif should be looped. If FALSE (the default), the last frame will remain visible after the animation is finished. If TRUE, the gif will infinitely repeat itself.
bounds	numeric vector specifying the xleft, xright, ybottom, and ytop limits of the animation canvas. Defaults to NULL in which case the animation canvas is set to include all existing trajectory points, irrespective of how extreme they may be.
norm	logical specifying whether the trajectories should be remapped to the *mt-space*. See mt_align. Note that aligning often requires that that all trajectories are flipped to one side first (see mt_remap_symmetric).
upscale	numeric specifying a scaling factor for the animation resolution. E.g, upscale = 2 implies that the x-resolution in *.gif* file is 2*xres.
decay	numeric defining a within-trajectory gradient of color intensity. Specifically, values larger than 1 will give more recent movements higher color intensities than movements that lie longer in the past, and vice versa.
max_intensity	numeric specifying the maximum color intensity. A value of, e.g., 5, implies that color intensity is limited to 5 overlapping trajectories. I.e., a point at which 4 trajectories overlap will in that case have a smaller color intensity than a point at which 5 trajectories overlap, but there will be no difference between the latter and a point at which 6 trajectories overlap. If decay is unequal 1, this metric refers to the most intense color point within the trajectory.
discard_images	logical specifying whether the temporary folder containing the temporary *.png* images should be deleted. Defaults to TRUE.
im_path	character string specifying the location of ImageMagick's *convert* function. If NULL, the *convert* function is expected in '/usr/local/bin/convert', the default location for Linux and OSX operating systems. The location has to be specified explicitly for Windows (see Details and Examples).
parallel	logical specifying whether the temporary *.png* images should be created using parallel processing (uses clusterApplyLB). Process will be run on the maximum number of available cores (as determined by detectCores).
verbose	logical indicating whether function should report its progress.

Details

`mt_animate` produces a .gif file showing a continuous stream of animated trajectories. The function first produces a series of *.png* images, which then are combined into a *.gif* animation using *ImageMagick* (see https://www.imagemagick.org/).

In order to run this function, ImageMagick must be installed (download from https://www.imagemagick.org/). Under Unix systems (Linux and Apple's OSX) the function will look for ImageMagick using its default installation path. Alternatively, the location of ImageMagick's *convert* function can be provided using the `im_path` argument. Under Windows, `im_path` must always be specified explicitly (e.g., it might look something like this `im_path = "C:/Program Files/ImageMagick-7.0.5-Q16/convert.exe"`).

During the animation trajectories are sampled from the data without replacement. The function stops when it reaches the last trajectory contained in `data`.

By default, `mt_animate` animates trajectories using the original timestamps. Timestamps are expected to be expressed in milliseconds. By setting `timestamps = NULL`, the function can also assume timestamps to be regular, i.e., of constant interval, in this case the longest duration is set to exactly one second.

In order to create high-resolution (large) animations in a relatively short time increase `upscale` in favor of `xres`. However, note that this will decrease the sharpness of the image.

In order to increase or decrease the overall color intensity decrease or increase the `max_intensity`, respectively.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Examples

```
## Not run:
# Preprocess trajectory data
mt_example <- mt_align_start(mt_example)
mt_example <- mt_remap_symmetric(mt_example)

# Create animated trajectory gif
# (under Linux / OSX)
mt_animate(mt_example,filename = "MyMovie.gif")

# Increase duration and density while decreasing speed
mt_animate(mt_example, filename = "MyMovie2.gif",
  seconds = 10, speed = .3, density = 10)

# Create animated trajectory gif
# (under Windows - ImageMagick version specific example)
mt_animate(mt_example,filename = "MyMovie.gif",
  im_path = "C:/Program Files/ImageMagick-7.0.5-Q16/convert.exe")


## End(Not run)
```

`mt_average` *Average trajectories across intervals.*

Description

Average trajectory data across specified intervals (e.g., constant time intervals). For every specified dimension in the trajectory array (by default, every dimension, i.e., x- and y-position, possibly also velocity and acceleration etc.), the mean value for the respective interval is calculated (see Details for information regarding the exact averaging procedure).

Usage

```
mt_average(data, use = "trajectories", save_as = "av_trajectories",
  dimensions = "all", av_dimension = "timestamps", intervals = NULL,
  interval_size = 100, max_interval = NULL, verbose = FALSE,
  dimension = NULL)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`dimensions`	a character vector specifying the dimensions in the trajectory array that should be averaged. By default (`"all"`), all trajectory dimensions will be averaged.
`av_dimension`	a character string specifying which values should be used for determining the intervals for averaging (`"timestamps"` by default).
`intervals`	an optional numeric vector. If specified, these values are taken as the borders of the intervals (`interval_size` and `max_interval` are ignored).
`interval_size`	an integer specifying the size of the constant dimension interval.
`max_interval`	an integer specifying the upper limit of the last dimension value that should be included (therefore, it should be a multiple of the `interval_size`). If specified, only values will be used for averaging where the dimension values are smaller than `max_interval`. If unspecified (the default), all values will be included.
`verbose`	logical indicating whether function should report its progress.
`dimension`	Deprecated. Please use `av_dimension` instead.

Details

For each interval, it is first determined which of the values lie within the respective interval of the dimension used for averaging (e.g., timestamps). Intervals are left-open, right-closed (e.g., if values are averaged across constant timestamps of 100 ms, a timestamp of 1200 would be included in the interval 1100-1200 while a timestamp of 1300 would be included in the interval 1200-1300).

Then, all values for which the corresponding average dimension values lie within the interval are averaged.

In case the last interval is not fully covered (e.g., if the last timestamp has the value 1250), values for the corresponding interval (1200-1300) will be computed based on the average of the values up to the last existing value.

Note that `mt_average` assumes that the trajectory variables are recorded with a constant sampling rate (i.e., with a constant difference in the timestamps). If the sampling rate varies considerably, mt_resample should be called before averaging to arrive at equally spaced timestamps. The sampling rate can be investigated using mt_check_resolution.

If average velocity and acceleration are of interest, mt_derivatives should be called before averaging.

Value

A mousetrap data object (see mt_example) with an additional array (by default called `av_trajectories`) that contains the average trajectory data per dimension interval. If a trajectory array was provided directly as `data`, only the average trajectories will be returned.

For the dimension values used for averaging (specified in `av_dimension`), the mid point of the respective interval is reported, which is helpful for plotting the trajectory data later on. However, this value does not necessarily correspond to the empirical mean of the dimension values in the interval.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_derivatives for calculating velocity and acceleration.

mt_resample for resampling trajectories using a constant time interval.

Examples

```
mt_example <- mt_derivatives(mt_example)

# average trajectories across 100 ms intervals
mt_example <- mt_average(mt_example, save_as="av_trajectories",
  interval_size=100)

# average time-normalized trajectories across specific intervals
# of the time steps
mt_example <- mt_time_normalize(mt_example)
mt_example <- mt_average(mt_example,
  use="tn_trajectories", save_as="av_tn_trajectories",
  av_dimension = "steps", intervals = c(0.5,33.5,67.5,101.5))
```

mt_bind — *Join two trajectory arrays*

Description

Join two trajectory arrays. This function is mainly used internally, but can be helpful in those (relatively rare) occasions where additional processed trajectory data should be added to another trajectory array.

Usage

```
mt_bind(trajectories1, trajectories2, verbose = FALSE)
```

Arguments

`trajectories1`	a trajectory array (see mt_example).
`trajectories2`	a trajectory array (see mt_example).
`verbose`	logical indicating whether function should report its progress.

Value

A trajectory array.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

Examples

```
trajectories_combined <- mt_bind(
  mt_example$trajectories,
  mt_prototypes
)
```

`mt_check_bimodality` *Assess bimodality of mouse-tracking measure distributions.*

Description

Assess bimodality of the distribution of mouse-tracking measures using the bimodality coefficient and Hartigan's dip statistic (see Details). If bimodality should be assessed separately for different conditions, the corresponding variables can be specified under `grouping_variables`.

Usage

```
mt_check_bimodality(data, use = "measures", use_variables = NULL,
  methods = c("BC", "HDS"), B = 2000, grouping_variables = NULL, ...)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details).
`use`	a character string specifying which data should be used. By default, points to the `measures` data.frame created using mt_measures.
`use_variables`	a vector specifying for which mouse-tracking measures bimodality should be assessed.
`methods`	a character string (or vector) specifying which methods should be used for assessing bimodality (see Details).
`B`	an integer specifying the number of replicates used in the Monte Carlo test (only relevant if "HDS_sim" is included in methods, see Details).
`grouping_variables`	a character string (or vector) specifying one or more variables in `data[["data"]]`. If specified, bimodality will be assessed separately for each level of the variable. If unspecified (the default), bimodality is checked across all trials.
`...`	additional arguments passed on to mt_reshape (such as `subset`).

Details

Different methods have been suggested for assessing the bimodality of mouse-tracking measure distributions, each of which has advantages and disadvantages (see Freeman & Dale, 2013).

Hehman et al. (2015) focus on two specific methods (bimodality coefficient and Hartigan's dip statistic) which are implemented here.

If `methods` include `BC`, the bimodality coefficient is calculated using the bimodality_coefficient function in this package. According to Freeman and Ambady (2010), a distribution is considered bimodal if $BC > 0.555$.

Note that MouseTracker (Freeman & Ambady, 2010) standardizes variables within each subject before computing the BC. This is also possible here using mt_standardize (see Examples).

If methods include HDS, Hartigan's dip statistic is calculated using the dip.test function of the diptest package. The corresponding p value (computed via linear interpolation) is returned.

If methods include HDS_sim, Hartigan's dip statistic is calculated using the dip.test function with the additional argument simulate.p.values=TRUE. In this case, the p value is computed from a Monte Carlo simulation of a uniform distribution with B (default: 2000) replicates.

Value

A list of several data.frames. Each data.frame contains the value returned by the respective method for assessing bimodality (see Details) - separately per condition (specified in the row dimension) and measure (specified in the column dimension).

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

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.

Freeman, J. B., & Dale, R. (2013). Assessing bimodality to detect the presence of a dual cognitive process. *Behavior Research Methods, 45*(1), 83-97.

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.

See Also

bimodality_coefficient for more information about the bimodality coefficient.

dip.test for more information about Hartigan's dip test.

Examples

```
# Calculate measures
mt_example <- mt_measures(mt_example)

# Assess bimodality for untransformed variables
mt_check_bimodality(mt_example,
  use_variables=c("MAD", "AD"))

# Standardize variables per participant
mt_example <- mt_standardize(mt_example,
  use_variables=c("MAD", "AD"), within="subject_nr")

# Assess bimodality for standardized variables
mt_check_bimodality(mt_example,
  use_variables=c("z_MAD", "z_AD"))

# Assess bimodality with simulated p values for HDS
```

```
mt_check_bimodality(mt_example,
  use_variables=c("z_MAD", "z_AD"),
  methods=c("BC", "HDS_sim"))

# Assess bimodality per condition
mt_check_bimodality(mt_example,
  use_variables=c("z_MAD", "z_AD"),
  grouping_variables="Condition")
```

`mt_check_resolution` *Check logging resolution by looking at timestamp differences.*

Description

`mt_check_resolution` computes the timestamp differences as a measure of the logging resolution. It provides various descriptive statistics to check the logging resolution.

Usage

```
mt_check_resolution(data, use = "trajectories",
  timestamps = "timestamps", desired = NULL)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
`use`	a character string specifying which trajectory data should be used.
`timestamps`	a character string specifying the trajectory dimension containing the timestamps.
`desired`	an optional integer. If specified, additional statistics are computed concerning the (relative) frequencies with which exactly the desired timestamp difference (with tolerance 1e-12) occurred.

Details

If mouse-tracking experiments are conducted using the mousetrap plug-ins for OpenSesame, the logging resolution can be specified explicitly in the experiment under "Logging resolution", which corresponds to the delay (in milliseconds) between recordings of the mouse position. By default, mouse positions are recorded every 10 ms (corresponding to a 100 Hz sampling rate). As the actual resolution achieved depends on the performance of the hardware, it makes sense to check the logging resolution using `mt_check_resolution`. Note that delays smaller than the specified delay typically result from mouse clicks in the experiment.

Value

A list with various descriptive statistics. For convenience, the relative frequencies are rounded to 4 decimal places.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

Examples

```
mt_check_resolution(mt_example)
```

`mt_cluster` *Cluster trajectories.*

Description

Performs trajectory clustering. It first computes distances between each pair of trajectories and then applies off-the-shelf clustering tools to explain the resulting dissimilarity matrix using a predefined number of clusters.

Usage

```
mt_cluster(data, use = "sp_trajectories", save_as = "clustering",
  dimensions = c("xpos", "ypos"), n_cluster = 5, method = "hclust",
  weights = rep(1, length(dimensions)), pointwise = TRUE,
  minkowski_p = 2, hclust_method = "ward.D", kmeans_nstart = 10,
  na_rm = FALSE, cluster_output = FALSE, verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting data should be stored.
`dimensions`	a character vector specifying which trajectory variables should be used. Can be of length 2 or 3, for two-dimensional or three-dimensional trajectories respectively.
`n_cluster`	an integer specifying the number of clusters to estimate.
`method`	character string specifying the clustering procedure. Either hclust (the default) or kmeans.
`weights`	numeric vector specifying the relative importance of the variables specified in `dimensions`. Defaults to a vector of 1s implying equal importance. Technically, each variable is rescaled so that the standard deviation matches the corresponding value in `weights`. To use the original variables, set `weights = NULL`.

pointwise
: boolean specifying the way in which dissimilarity between the trajectories is measured. If TRUE (the default), mt_distmat measures the average dissimilarity and then sums the results. If FALSE, mt_distmat measures dissimilarity once (by treating the various points as independent dimensions). This is only relevant if method is "hclust". See mt_distmat for further details.

minkowski_p
: an integer specifying the distance metric for the cluster solution. minkowski_p = 1 computes the city-block distance, minkowski_p = 2 (the default) computes the Euclidian distance, minkowski_p = 3 the cubic distance, etc. Only relevant if method is "hclust". See mt_distmat for further details.

hclust_method
: character string specifying the linkage criterion used. Passed on to the method argument of hclust. Default is set to ward.D. Only relevant if method is "hclust".

kmeans_nstart
: integer specifying the number of reruns of the kmeans procedure. Larger numbers minimize the risk of finding local minima. Passed on to the nstart argument of kmeans. Only relevant if method is "kmeans".

na_rm
: logical specifying whether trajectory points containing NAs should be removed. Removal is done column-wise. That is, if any trajectory has a missing value at, e.g., the 10th recorded position, the 10th position is removed for all trajectories. This is necessary to compute distance between trajectories.

cluster_output
: logical. If FALSE (the default), the mousetrap data object with the cluster assignments is returned (see Value). If TRUE, the output of the cluster method (kmeans or hclust) is returned directly.

verbose
: logical indicating whether function should report its progress.

Details

mt_cluster uses off-the-shelf clustering tools, i.e., hclust and kmeans, for cluster estimation. Cluster estimation using hclust relies on distances computed by mt_distmat.

Mouse trajectories often occur in distinct, qualitative types (see Wulff et al., in press; Wulff et al., 2018). Common trajectory types are linear trajectories, mildly and strongly curved trajctories, and single and multiple change-of-mind trials (see also mt_map). mt_cluster can tease these types apart.

mt_cluster uses hclust or kmeans to explain the distances between every pair of trajectories using a predefined number of clusters. If method is "hclust", mt_cluster computes the dissimiliarity matrix for all trajectory pairs using mt_distmat. If method is "kmeans", this is done internally by kmeans.

We recommend setting method to hclust using ward.D as the linkage criterion (via hclust_method). Relative to kmeans, the other implemented clustering method, and other linkage criteria, this setup handles the skewed distribution cluster sizes and trajectory outliers found in the majority of datasets best.

For clustering trajectories, it is often useful that the endpoints of all trajectories share the same direction, e.g., that all trajectories end in the top-left corner of the coordinate system (mt_remap_symmetric or mt_align can be used to achieve this). Furthermore, it is recommended to use spatialized trajectories (see mt_spatialize; Wulff et al., in press; Haslbeck et al., 2018).

Value

A mousetrap data object (see mt_example) with an additional data.frame added to it (by default called `clustering`) that contains the cluster assignments. If a trajectory array was provided directly as `data`, only the clustering `data.frame` will be returned.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Jonas M. B. Haslbeck (<jonas.haslbeck@gmail.com>)

References

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2018). *Measuring the (dis-)continuous mind: What movement trajectories reveal about cognition*. Manuscript in preparation.

Haslbeck, J. M. B., Wulff, D. U., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2018). *Advanced mouse- and hand-tracking analysis: Detecting and visualizing clusters in movement trajectories*. Manuscript in preparation.

See Also

mt_distmat for more information about how the distance matrix is computed when the hclust method is used.

mt_cluster_k for estimating the optimal number of clusters.

Examples

```
# Spatialize trajectories
KH2017 <- mt_spatialize(KH2017)

# Cluster trajectories
KH2017 <- mt_cluster(KH2017, use="sp_trajectories")

# Plot clustered trajectories
mt_plot(KH2017,use="sp_trajectories",
  use2="clustering",facet_col="cluster")
```

mt_cluster_k *Estimate optimal number of clusters.*

Description

Estimates the optimal number of clusters (k) using various methods.

Usage

```
mt_cluster_k(data, use = "sp_trajectories", dimensions = c("xpos",
  "ypos"), kseq = 2:15, compute = c("stability", "gap", "jump",
  "slope"), method = "hclust", weights = rep(1, length(dimensions)),
  pointwise = TRUE, minkowski_p = 2, hclust_method = "ward.D",
  kmeans_nstart = 10, n_bootstrap = 10, model_based = FALSE,
  n_gap = 10, na_rm = FALSE, verbose = FALSE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
dimensions	a character vector specifying which trajectory variables should be used. Can be of length 2 or 3, for two-dimensional or three-dimensional trajectories respectively.
kseq	a numeric vector specifying set of candidates for k. Defaults to 2:15, implying that all values of k within that range are compared using the metrics specified in compute.
compute	character vector specifying the to be computed measures. Can be any subset of c("stability","gap","jump","slope").
method	character string specifying the type of clustering procedure for the stability-based method. Either hclust or kmeans.
weights	numeric vector specifying the relative importance of the variables specified in dimensions. Defaults to a vector of 1s implying equal importance. Technically, each variable is rescaled so that the standard deviation matches the corresponding value in weights. To use the original variables, set weights = NULL.
pointwise	boolean specifying the way in which dissimilarity between the trajectories is measured. If TRUE (the default), mt_distmat measures the average dissimilarity and then sums the results. If FALSE, mt_distmat measures dissimilarity once (by treating the various points as independent dimensions). This is only relevant if method is "hclust". See mt_distmat for further details.
minkowski_p	an integer specifying the distance metric for the cluster solution. minkowski_p = 1 computes the city-block distance, minkowski_p = 2 (the default) computes the Euclidian distance, minkowski_p = 3 the cubic distance, etc. Only relevant if method is "hclust". See mt_distmat for further details.

hclust_method — character string specifying the linkage criterion used. Passed on to the method argument of hclust. Default is set to ward.D. Only relevant if method is "hclust".

kmeans_nstart — integer specifying the number of reruns of the kmeans procedure. Larger numbers minimize the risk of finding local minima. Passed on to the nstart argument of kmeans. Only relevant if method is "kmeans".

n_bootstrap — an integer specifying the number of bootstrap comparisons used by stability. See cStability.

model_based — boolean specifying whether the model-based or the model-free should be used by stability, when method is kmeans. See cStability and Haslbeck & Wulff (2016).

n_gap — integer specifying the number of simulated datasets used by gap. See Tibshirani et al. (2001).

na_rm — logical specifying whether trajectory points containing NAs should be removed. Removal is done column-wise. That is, if any trajectory has a missing value at, e.g., the 10th recorded position, the 10th position is removed for all trajectories. This is necessary to compute distance between trajectories.

verbose — logical indicating whether function should report its progress.

Details

mt_cluster_k estimates the number of clusters (k) using four commonly used k-selection methods (specified via compute): cluster stability (stability), the gap statistic (gap), the jump statistic (jump), and the slope statistic (slope).

Cluster stability methods select k as the number of clusters for which the assignment of objects to clusters is most stable across bootstrap samples. This function implements the model-based and model-free methods described by Haslbeck & Wulff (2016). See references.

The remaining three methods select k as the value that optimizes the gap statistic (Tibshirani, Walther, & Hastie, 2001), the jump statistic (Sugar & James, 2013), and the slope statistic (Fujita, Takahashi, & Patriota, 2014), respectively.

For clustering trajectories, it is often useful that the endpoints of all trajectories share the same direction, e.g., that all trajectories end in the top-left corner of the coordinate system (mt_remap_symmetric or mt_align can be used to achieve this). Furthermore, it is recommended to use spatialized trajectories (see mt_spatialize; Wulff et al., in press; Haslbeck et al., 2018).

Value

A list containing two lists that store the results of the different methods. kopt contains the estimated k for each of the methods specified in compute. paths contains the values for each k in kseq as computed by each of the methods specified in compute. The values in kopt are optima for each of the vectors in paths.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Jonas M. B. Haslbeck (<jonas.haslbeck@gmail.com>)

References

Haslbeck, J., & Wulff, D. U. (2016). Estimating the Number of Clusters via Normalized Cluster Instability. *arXiv preprint* arXiv:1608.07494.

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Haslbeck, J. M. B., Wulff, D. U., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2018). *Advanced mouse- and hand-tracking analysis: Detecting and visualizing clusters in movement trajectories*. Manuscript in preparation.

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63*(2), 411-423.

Sugar, C. A., & James, G. M. (2013). Finding the number of clusters in a dataset. *Journal of the American Statistical Association, 98*(463), 750-763.

Fujita, A., Takahashi, D. Y., & Patriota, A. G. (2014). A non-parametric method to estimate the number of clusters. *Computational Statistics & Data Analysis, 73*, 27-39.

See Also

mt_distmat for more information about how the distance matrix is computed when the hclust method is used.

mt_cluster for performing trajectory clustering with a specified number of clusters.

Examples

```
## Not run:
# Spatialize trajectories
KH2017 <- mt_spatialize(KH2017)

# Find k
results <- mt_cluster_k(KH2017, use="sp_trajectories")

# Retrieve results
results$kopt
results$paths

## End(Not run)
```

mt_count *Count number of observations.*

Description

Count number of observations per trial for a specified dimension (or several) in the trajectory array. This is mostly a helper function used by other functions in this package.

Usage

```
mt_count(data, use = "trajectories", save_as = "measures",
  dimensions = "xpos")
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
save_as	a character string specifying where the resulting trajectory data should be stored.
dimensions	a character vector specifying the name of the dimension(s) that should be used for counting the number of observations. If several dimensions are specified, the number of complete observations are reported.

Value

A mousetrap data object (see mt_example).

If a data.frame with label specified in save_as (by default "measures") already exists, the number of observations (called nobs) are added as additional column. If not, an additional data.frame will be added.

If a trajectory array was provided directly as data, only a named character vector will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Examples

```
# Retrieve vector that counts number of observations
mt_count(mt_example$trajectories)
```

mt_derivatives *Calculate distance, velocity, and acceleration.*

Description

Calculate distance traveled, velocity, and acceleration for each logged position. Distance is calculated as the Euclidean distance between successive coordinates, and velocity as distance covered per time interval. The acceleration denotes the difference in absolute velocity, again normalized per time.

Usage

```
mt_derivatives(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), timestamps = "timestamps",
  prefix = "", absolute = FALSE, return_delta_time = FALSE,
  verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`dimensions`	a character vector specifying across which dimension(s) distances, velocity, and acceleration are calculated. By default (`c("xpos","ypos")`), they are calculated across both x and y dimensions. Alternatively, only one dimension can be specified, e.g., `"xpos"` or `"ypos"`.
`timestamps`	a character string specifying the trajectory dimension containing the timestamps.
`prefix`	an optional character string that is added as a prefix to the to be created new trajectory dimensions.
`absolute`	logical indicating if absolute values for distances and velocities should be reported. Only relevant if a single dimension is specified in `dimensions` (see Details).
`return_delta_time`	logical indicating if the timestamp differences should be returned as well (as "delta_time").
`verbose`	logical indicating whether function should report its progress.

Details

Distances, velocities and acceleration are computed as follows:

The first entry in each respective vector is always zero. Each subsequent entry thus represents the Euclidean distance traveled since the previous recorded set of coordinates and the velocity with which the movement between both samples took place. Thus, both distance and velocity represent the intervening period between the previous sample and the one with which the numeric value is saved.

The acceleration, by contrast, denotes the change in absolute velocity between two adjacent periods. Because of this, it is shifted forward to best match the actual time point at which the acceleration was measured. Because there will always be one less value computed for acceleration than for velocity, the final value in the acceleration vector has been padded with an NA.

If the distance is calculated across both horizontal and vertical (x and y) dimensions, distance and velocity is always positive (or 0). If only one dimension is used, by default (`absolute=TRUE`), increases in x (or y) values result in positive distances and velocity values, decreases in negative distances and velocity values. If `absolute=FALSE`, absolute values for distance and velocity are reported.

Value

A mousetrap data object (see mt_example) with Euclidian distance, velocity, and acceleration added as additional variables to the trajectory array (called `dist`, `vel`, and `acc`, if no prefix was specified). If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_average for averaging trajectories across constant time intervals.

mt_measures for calculating per-trial mouse-tracking measures.

Examples

```
# Calculate derivatives looking at movement
# across both dimensions
mt_example <- mt_derivatives(mt_example)

# Calculate derivatives only looking at movement along x dimension
# reporting absolute values for distance and velocity
mt_example <- mt_derivatives(mt_example,
  dimensions="xpos", absolute=TRUE)
```

`mt_deviations` *Calculate deviations from idealized trajectory.*

Description

Calculate the idealized trajectory and the perpendicular deviations of the actual trajectory from it for each logged position.

Usage

```
mt_deviations(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), start_ideal = NULL,
  end_ideal = NULL, prefix = "", verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`dimensions`	a character vector specifying the two dimensions in the trajectory array that contain the mouse positions. By default (`c("xpos","ypos")`), the x- and y-positions are used.
`start_ideal`	an optional vector specifying the start position (see Example). If specified, this position will be used as the starting point of the idealized trajectory (instead of the actual starting point).
`end_ideal`	an optional vector specifying the end position (see Example). If specified, this position will be used as the end point of the idealized trajectory (instead of the actual end point).
`prefix`	an optional character string that is added as a prefix to the to be created new trajectory dimensions.
`verbose`	logical indicating whether function should report its progress.

Details

The idealized trajectory is defined as the straight line connecting the start and end point of the actual trajectory (e.g., Freeman & Ambady, 2010). The deviation for each position is calculated as the perpendicular deviation of the actual trajectory from the idealized trajectory.

If a deviation occurs above the direct path, this is denoted by a positive value. If it occurs below the direct path, this is denoted by a negative value. This assumes that the complete movement in the trial was from bottom to top (i.e., the end point has a higher y-position than the start points). In case the movement was from top to bottom, `mt_deviations` automatically flips the signs. Note that the second dimension specified in `dimensions` is used for determining all this.

Value

A mousetrap data object (see mt_example) where the positions of the idealized trajectory (by default called `xpos_ideal` and `ypos_ideal`) and the perpendicular deviations of the actual trajectory from the idealized trajectory (by default called `dev_ideal`) have been added as additional variables to the trajectory array. If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

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.

See Also

mt_measures for calculating per-trial mouse-tracking measures.

Examples

```
# Calculate deviations from idealized trajectory
# (straight line connecting the start and end point of each trial)
mt_example <- mt_deviations(mt_example)

# Calculate deviations from idealized trajectory with
# constant start and end points across trials
mt_example <- mt_deviations(mt_example,
  start_ideal=c(0,0), end_ideal=c(-665,974))
```

---

mt_diffmap — *Creates a difference-heatmap of two trajectory heatmap images.*

Description

`mt_diffmap` creates a difference-heatmap of the trajectory data using gaussian smoothing. Note that this function has beta status.

Usage

```
mt_diffmap(x, y = NULL, condition = NULL, use = "trajectories",
  dimensions = c("xpos", "ypos"), use2 = "data", filename = NULL,
  bounds = NULL, xres = 500, upscale = 4, smooth_radius = 10,
  colors = c("#00863F", "#000000", "#FF1900"), n_shades = 1000,
  plot = TRUE, ..., verbose = TRUE)
```

Arguments

- `x` — an object of class `mousetrap`), a trajectory object of class `array`, or an object of class `mt_heatmap_raw` (as created by mt_heatmap_raw).
- `y` — an object of class `mousetrap`), a trajectory object of class `array`, or an object of class `mt_heatmap_raw` (as created by mt_heatmap_raw). The class of `y` must match the class of `x`, unless `y` is `NULL`.
- `condition` — either a character value specifying which variable codes the two conditions (in `x[[use2]]`) that should be compared - or a vector matching the number of trajectories in `x[[use]]` that has exactly two levels. `mt_diffmap` will create a difference-heatmap comparing all trajectories between the two conditions. If `condition` is specified, `y` will be ignored (unless `x` and `y` are of class `heatmap_raw`).
- `use` — a character string specifying which trajectory data should be used.
- `dimensions` — a character vector specifying the trajectory variables used to create the heatmap. The first two entries are used as x and y-coordinates, the third, if provided, will be added as color information.
- `use2` — an optional character string specifying where the data that contain the condition variable can be found. Defaults to "data" as `x[["data"]]` usually contains all non mouse-tracking trial data.
- `filename` — a character string giving the name of the file. If `NULL` (the default), the R standard device is used for plotting. Otherwise, the plotting device is inferred from the file extension. Only supports devices tiff, png, pdf.
- `bounds` — numeric vector specifying the corners (xmin, ymin, xmax, ymax) of the plot region. By default (`bounds = NULL`), bounds are determined based on the data input.
- `xres` — an integer specifying the number of pixels along the x-dimension. An `xres` of 1000 implies an 1000*N px, where N is determined so that the trajectories aspect ratio is preserved (provided the `bounds` are unchanged).
- `upscale` — a numeric value by which the output resolution of the image is increased or decreased. Only applies if device is one of tiff, png, or pdf.
- `smooth_radius` — a numeric value specifying the standard deviation of the gaussian smoothing. If zero, smoothing is omitted.
- `colors` — a character vector specifying the colors used to color cases of `image1 > image2, image1 ~ image2, image1 < image2`, respectively. Note that the colors are used in that specific order. Defaults to c("#00863F", "#FFFFFF", "#FF1900") which specifies a green-black-red color gradient.
- `n_shades` — integer specifying the number of shades for the color gradient between the first and second, and the second and third color in `colors`.
- `plot` — logical specifying whether resulting image should be plotted (`plot = TRUE`, the default). If (`plot = FALSE`), an object of class `mt_object_raw` is returned.
- `...` — arguments passed to mt_heatmap_raw.
- `verbose` — logical indicating whether function should report its progress.

Details

`mt_diffmap` takes two objects that either contain trajectory heatmaps or from which trajectory heatmaps can be computed. Difference-heatmaps are constructed analogously to mt_heatmap_raw.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Pascal J. Kieslich

References

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Kieslich, P. J., Henninger, F., Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: A practical guide to implementation and analysis. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 111-130). New York, NY: Routledge.

See Also

mt_heatmap and mt_heatmap_ggplot for plotting trajectory heatmaps.

Examples

```
mt_diffmap(KH2017, condition="Condition",
  xres=400, smooth_radius=6, n_shades=5)
```

---

mt_distmat — *Compute distance matrix.*

Description

Computes the point- or vector-wise dissimilarity between each pair of trajectories.

Usage

```
mt_distmat(data, use = "sp_trajectories", save_as = "distmat",
  dimensions = c("xpos", "ypos"), weights = rep(1, length(dimensions)),
  pointwise = TRUE, minkowski_p = 2, na_rm = FALSE)
```

Arguments

`data` a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).

`use` a character string specifying which trajectory data should be used.

`save_as` a character string specifying where the resulting data should be stored.

`dimensions` a character vector specifying which trajectory variables should be used. Can be of length 2 or 3 for two-dimensional or three-dimensional trajectories respectively.

`weights` numeric vector specifying the relative importance of the variables specified in `dimensions`. Defaults to a vector of 1s implying equal importance. Technically, each variable is rescaled so that the standard deviation matches the corresponding value in `weights`. To use the original variables, set `weights = NULL`.

`pointwise` boolean specifying the way dissimilarity between the trajectories is measured (see Details). If `TRUE` (the default), `mt_distmat` measures the average dissimilarity and then sums the results. If `FALSE`, `mt_distmat` measures dissimilarity once (by treating the various points as independent dimensions).

`minkowski_p` an integer specifying the distance metric. `minkowski_p = 1` computes the city-block distance, `minkowski_p = 2` (the default) computes the Euclidian distance, `minkowski_p = 3` the cubic distance, etc.

`na_rm` logical specifying whether trajectory points containing NAs should be removed. Removal is done column-wise. That is, if any trajectory has a missing value at, e.g., the 10th recorded position, the 10th position is removed for all trajectories. This is necessary to compute distance between trajectories.

Details

`mt_distmat` computes point- or vector-wise dissimilarities between pairs of trajectories. Point-wise dissimilarity refers to computing the distance metric defined by `minkowski_p` for every point of the trajectory and then summing the results. That is, if `minkowski_p = 2` the point-wise dissimilarity between two trajectories, each defined by a set of x and y coordinates, is calculated as `sum(sqrt((x_i-x_j)^2 + (y_i-y_j)^2))`. Vector-wise dissimilarity, on the other hand refers to computing the distance metric once for the entire trajectory. That is, vector-wise dissimilarity is computed as `sqrt(sum((x_i-x_j)^2 + (y_i-y_j)^2))`.

Value

A mousetrap data object (see mt_example) with an additional object added (by default called `distmat`) containing the distance matrix. If a trajectory array was provided directly as `data`, only the distance matrix will be returned.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Jonas M. B. Haslbeck (<jonas.haslbeck@gmail.com>)

Examples

```
# Spatialize trajectories
mt_example <- mt_spatialize(mt_example)

# Compute distance matrix
mt_example <- mt_distmat(mt_example, use="sp_trajectories")
```

`mt_example` *A mousetrap data object.*

Description

A data object of class "mousetrap" with example data created by importing mt_example_raw and applying basic post-processing.

Usage

```
mt_example
```

Format

A mousetrap data object is a list containing at least the following objects:

- `data`: a data.frame containing the trial data (from which the mouse-tracking data columns have been removed). More information about the content of the trial data in `mt_example` can be found in mt_example_raw. The rownames of `data` correspond to the trial identifier. For convenience, the trial identifier is also stored in an additional column called "mt_id".
- `trajectories`: an array containing the raw mouse-tracking trajectories. The first dimension represents the different trials and the dimension names (which can be accessed using rownames) correspond to the trial identifier (the same identifier that is used as the `rownames` in `data`). The second dimension corresponds to the samples taken over time which are included in chronological order. The third dimension corresponds to the different mouse-tracking variables (timestamps, x-positions, y-positions) which are usually called `timestamps`, `xpos`, and `ypos`.

Some functions in this package (e.g., mt_time_normalize and mt_average) add additional trajectory arrays (e.g., `tn_trajectories` and `av_trajectories`) to the mousetrap data object. Other functions modify the existing arrays (e.g., mt_derivatives adds distance, velocity, and acceleration to an existing dataset). Finally mt_measures adds an additional data.frame with mouse-tracking measures to it.

Details

The raw data set was imported using mt_import_mousetrap. Trajectories were then remapped using mt_remap_symmetric so that all trajectories end in the top-left corner and their starting point was aligned using mt_align_start to a common value (0,0).

mt_example_raw *Raw mouse-tracking dataset for demonstrations of the mousetrap package*

Description

An exemplary mouse-tracking dataset collected OpenSesame using the mousetrap plugin (Kieslich & Henninger, 2017). A preprocessed (as opposed to raw) version of the same data can be found in mt_example.

Usage

```
mt_example_raw
```

Format

A data.frame with 38 rows and 19 variables. The data.frame is based on the combined raw data that were created using read_opensesame from the readbulk library. For ease of use, unnecessary columns were excluded.

The variables included relate to the item that was presented (`Exemplar`), the answer categories (`Category1` and `Category2`), the subject identifier (`subject_nr`) the subjects' response (`response_get_response`), as well as the mouse-tracking variables (`timestamps_get_response`, `xpos_get_response` and `ypos_get_response`). Besides, a number of additional variables are included, e.g., some variables relating to the general settings of the experiment (e.g., the `width` and `height` of the screen in pixels).

Each mouse-tracking variable contains a list of values (separated by ', ') - one entry for each recorded position of the mouse. The position coordinates are given in pixels, such that values of zero for both `xpos_get_response` and `ypos_get_response` indicate that the cursor is located in the center of the screen. Both variables increase in value as the mouse moves toward the bottom right. Timestamps are given in milliseconds.

Details

The data stem from a study based on experiment 1 by Dale et al. (2007). In this experiment, participants have to assign exemplars (e.g., "shark") to one of two categories (e.g., "fish" or "mammal") by clicking on the button corresponding to the correct category. All exemplars and categories were translated to and presented in German.

Across the 19 trials of the experiment, participants categorized 13 exemplars that were typical of their category and 6 atypical exemplars for which this was not the case. For the atypical exemplars (e.g., "whale"), the competing category ("fish") was selected to compete with the correct category ("mammal"). The hypothesis under investigation is whether participants' mouse trajectories deviate more towards the competing category for the atypical exemplars, indicating increased conflict between the response options.

Please note that `mt_example_raw` should only be used for exploring the features of the mousetrap package and not for any substantive analysis.

References

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

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15-28. https://doi.org/10.3758/BF03195938

`mt_exclude_initiation` *Exclude initial phase without mouse movement.*

Description

Exclude the initial phase in a trial where the mouse was not moved. The corresponding samples (x- and y-positions and timestamps) in the trajectory data will be removed.

Usage

```
mt_exclude_initiation(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), timestamps = "timestamps",
  reset_timestamps = TRUE, verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`dimensions`	a character vector specifying the dimensions in the trajectory array that contain the mouse positions.
`timestamps`	a character string specifying the trajectory dimension containing the timestamps.
`reset_timestamps`	logical indicating whether the timestamps should be reset after removing the initial phase without movement (see Details).
`verbose`	logical indicating whether function should report its progress.

Details

`mt_exclude_initiation` removes all samples (x- and y-positions as well as timestamps) at the beginning of the trial during which the mouse was not moved from its initial position. The last unchanged sample is retained in the data.

If `reset_timestamps == TRUE` (the default), it subtracts the last timestamp before a movement occurs from all timestamps , so that the series of timestamps once more begin with zero. If the argument is set to `FALSE`, the values of the timestamps are unchanged.

Please note that resetting the timestamps will result in changes in several mouse-tracking measures, notably those which report timestamps (e.g., `MAD_time`). Typically, however, these changes are desired when using this function.

Value

A mousetrap data object (see mt_example) from which the initial phase without mouse movement was removed. If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_measures for calculating the initiation time.

Examples

```
mt_example <- mt_exclude_initiation(mt_example,
  save_as="mod_trajectories")
```

`mt_export_long` *Export mouse-tracking data.*

Description

`mt_export_long` and `mt_export_wide` can be used for exporting mouse-tracking data from a mousetrap data object in long or wide format. If desired, additional data (stored in `data[[use2]]`) can be merged with the trajectory data before export. `mt_export_long` and `mt_export_wide` are wrapper functions for mt_reshape.

Usage

```
mt_export_long(data, use = "trajectories", use_variables = NULL,
  use2 = "data", use2_variables = NULL, ...)

mt_export_wide(data, use = "trajectories", use_variables = NULL,
  use2 = "data", use2_variables = NULL, ...)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which data should be exported. The corresponding data are selected from data using `data[[use]]`. Usually, this value corresponds to either "trajectories" or "tn_trajectories", depending on whether the raw or the time-normalized trajectories should be exported.

use_variables	a character vector specifying which mouse-tracking variables should be exported. This corresponds to the labels of the trajectory array dimensions. If unspecified, all variables will be exported.
use2	an optional character string specifying where the other trial data can be found. Defaults to "data" as `data[["data"]]` usually contains all non mouse-tracking trial data. Alternatively, a data.frame can be provided directly.
use2_variables	an optional character string (or vector) specifying the variables (in `data[[use2]]`) that should be merged with the data.
...	additional arguments passed on to mt_reshape (such as `subset`).

Value

A data.frame containing the exported data.

Functions

- `mt_export_long`: Export mouse-tracking data in long format
- `mt_export_wide`: Export mouse-tracking data in wide format

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_import_long for importing mouse-tracking data saved in a long format.

mt_import_wide for importing mouse-tracking data saved in a wide format.

Examples

```
# Export data in long format
# (and include information about condition and subject_nr)
mt_data_long <- mt_export_long(mt_example,
  use2_variables=c("subject_nr","Condition"))

# Export data in wide format
# (and include information about condition and subject_nr)
mt_data_wide <- mt_export_wide(mt_example,
  use2_variables=c("subject_nr","Condition"))
```

mt_heatmap *Plot trajectory heatmap.*

Description

mt_heatmap plots high resolution raw trajectory maps. Note that this function has beta status.

Usage

```
mt_heatmap(x, use = "trajectories", dimensions = c("xpos", "ypos"),
  filename = NULL, ..., upscale = 1, plot_dims = FALSE,
  verbose = TRUE)
```

Arguments

x	usually an object of class mousetrap. Alternatively, a trajectory array or an object of class mt_heatmap_raw.
use	a character string specifying which trajectory data should be used.
dimensions	a character vector specifying the trajectory variables used to create the heatmap. The first two entries are used as x and y-coordinates, the third, if provided, will be added as color information.
filename	a character string giving the name of the file. If NULL (the default), the R standard device is used for plotting. Otherwise, the plotting device is inferred from the file extension. Only supports devices tiff, png, pdf.
...	arguments passed to mt_heatmap_raw.
upscale	a numeric value by which the output resolution of the image is increased or decreased. Only applies if device is one of tiff, png, or pdf.
plot_dims	adds the coordinates of the four image corners to the plot. Helps setting bounds.
verbose	logical indicating whether function should report its progress.

Details

mt_heatmap wraps mt_heatmap_raw and provides direct plotting output in tiff, png, pdf, or R's default window output. For further details on how the trajectory heatmaps are constructed, see mt_heatmap_raw.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

References

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Kieslich, P. J., Henninger, F., Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: A practical guide to implementation and analysis. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 111-130). New York, NY: Routledge.

See Also

mt_heatmap_ggplot for plotting a trajectory heatmap using ggplot2.

mt_diffmap for plotting trajectory difference-heatmaps.

Examples

```
mt_heatmap(KH2017, xres=500, n_shades=5, mean_image=0.2)
```

`mt_heatmap_ggplot` *Plot trajectory heatmap using ggplot.*

Description

`mt_heatmap_ggplot` plots high resolution raw trajectory maps. Note that this function has beta status.

Usage

```
mt_heatmap_ggplot(data, use = "trajectories", dimensions = c("xpos",
  "ypos"), use2 = "data", facet_row = NULL, facet_col = NULL, ...)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`dimensions`	a character vector specifying the trajectory variables used to create the heatmap. The first two entries are used as x and y-coordinates, the third, if provided, will be added as color information.
`use2`	an optional character string specifying where the data that contain the variables used for faceting can be found (in case these arguments are specified). Defaults to "data" as `data[["data"]]` usually contains all non mouse-tracking trial data.

facet_row	an optional character string specifying a variable in data[[use2]] that should be used for (row-wise) faceting.
facet_col	an optional character string specifying a variable in data[[use2]] that should be used for (column-wise) faceting.
...	arguments passed to mt_heatmap_raw.

Details

mt_heatmap_ggplot wraps mt_heatmap_raw and returns a ggplot object containing the plot. In contrast to mt_heatmap_plot plots created by mt_heatmap_ggplot can be extended using ggplot's + operator. For further details on how the trajectory heatmaps are constructed, see mt_heatmap_raw.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

Dirk U. Wulff

References

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Kieslich, P. J., Henninger, F., Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: A practical guide to implementation and analysis. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 111-130). New York, NY: Routledge.

See Also

mt_heatmap for plotting a trajectory heatmap using base plots.

mt_diffmap for plotting trajectory difference-heatmaps.

Examples

```
mt_heatmap_ggplot(KH2017, xres=500, n_shades=5, mean_image=0.2)
```

mt_heatmap_raw — *Creates high-resolution heatmap of trajectory data.*

Description

mt_heatmap_raw creates a high-resolution heatmap image of the trajectory data using gaussian smoothing. Note that this function has beta status.

Usage

```
mt_heatmap_raw(data, use = "trajectories", dimensions = c("xpos",
  "ypos"), variable = NULL, bounds = NULL, xres = 1000,
  upsample = 1, norm = FALSE, colors = c("black", "blue", "white"),
  n_shades = c(1000, 1000), smooth_radius = 1.5, low_pass = 200,
  auto_enhance = TRUE, mean_image = 0.15, mean_color = 0.25,
  aggregate_lwd = 0, aggregate_col = "black", n_trajectories = NULL,
  seed = NULL, verbose = TRUE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
dimensions	a character vector specifying the trajectory variables used to create the heatmap. The first two entries are used as x and y-coordinates, the third, if provided, will be added as color information.
variable	boolean or numeric vector matching the number of trajectories that if provided will be used as color information. variable is only considered when length(dimensions) < 3.
bounds	numeric vector specifying the corners (xmin, ymin, xmax, ymax) of the plot region. By default (bounds = NULL), bounds are determined based on the data input.
xres	an integer specifying the number of pixels along the x-dimension. An xres of 1000 implies an 1000*N px, where N is determined so that the trajectories aspect ratio is preserved (provided the bounds are unchanged).
upsample	a numeric value by which the number of points used to represent individual trajectories are increased or decreased. Values of smaller than one will improve speed but also introduce a certain level of granularity.
norm	a logical specifying whether the data should be warped into standard space. If norm = TRUE, this overrules bounds.
colors	a character vector specifying two or three colors used to color the background, the foreground (trajectories), and the values of a third dimension (if specified).
n_shades	an integer specifying the number of shades for the color gradient between the first and second, and the second and third color in colors.

smooth_radius	a numeric value specifying the standard deviation of the gaussian smoothing. If zero, smoothing is omitted.
low_pass	an integer specifying the allowed number of counts per pixel. This arguments limits the maximum pixel color intensity.
auto_enhance	boolean. If TRUE (the default), the image is adjusted so that the mean color intensity matches mean_image and mean_color.
mean_image	a numeric value between 0 and 1 specifying the average foreground color intensity across the entire image. Defaults to 0.1.
mean_color	a numeric value between 0 and 1 specifying the average third dimension's color intensity across the entire image. Defaults to 0.1. Only relevant if a third dimension is specified in colors.
aggregate_lwd	an integer specifying the width of the aggregate trajectory. If aggregate_lwd is 0 (the default), the aggregate trajectory is omitted.
aggregate_col	a character value specifying the color of the aggregate trajectory.
n_trajectories	an optional integer specifying the number of trajectories used to create the image. By default, all trajectories are used. If n_trajectories is specified and smaller than the number of trajectories in the trajectory array, then n_trajectories are randomly sampled.
seed	an optional integer specifying the seed used for the trajectory sampling.
verbose	logical indicating whether function should report its progress.

Details

To create the image, mt_heatmap_raw takes the following steps. First, the function maps the trajectory points to a pixel space with x ranging from 1 to xres and y ranging from 1 to xres divided by the ratio of x and y's value range. Second, the function counts and normalizes the number of trajectory points occupying each of the x,y-pixels to yield image intensities between 0 and 1. Third, the function smooths the image using an approximative guassian approach governed by smooth_radius, which controls the dispersion of the gaussian smoothing. Fourth, the function automatically enhances the image (unless auto_enhance = FALSE) using a non-linear transformation in order to yield a desired mean_image intensity. Fifth, the function translates the image intensity into color using the colors specified in colors. Finally, the function returns the image data in a long format containing the x, y, and color information.

mt_heatmap_raw also offers the possibility to overlay the heatmap with an additional variable, such as for instance velocity, so that both the density of mouse trajectories and the information of the additional variable are visible. In order to do this, specify a third variable label in dimensions and control its appearance using the color and mean_color arguments.

Value

An object of class mt_object_raw containing in a matrix format the image's pixel information, the aggregate trajectory, and the colors.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

References

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Kieslich, P. J., Henninger, F., Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: A practical guide to implementation and analysis. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 111-130). New York, NY: Routledge.

See Also

mt_heatmap and mt_heatmap_ggplot for plotting trajectory heatmaps.

mt_diffmap for plotting trajectory difference-heatmaps.

`mt_import_long` *Import mouse-tracking data saved in long format.*

Description

`mt_import_long` receives a data.frame in which mouse-tracking data are stored in long format, i.e., where one row contains the logging data (timestamp, x- and y-position etc.) at one specific point in the trial. This is, for example, the case when exporting the trajectory data from the mousetrap package using mt_export_long. From this data.frame, `mt_import_long` creates a mousetrap data object containing the trajectories and additional data for further processing within the mousetrap package. Specifically, it returns a list that includes the trajectory data as an array (called `trajectories`), and all other data as a data.frame (called `data`). This data structure can then be passed on to other functions within this package (see mousetrap for an overview).

Usage

```
mt_import_long(raw_data, xpos_label = "xpos", ypos_label = "ypos",
  zpos_label = NULL, timestamps_label = "timestamps",
  add_labels = NULL, mt_id_label = "mt_id", mt_seq_label = "mt_seq",
  reset_timestamps = TRUE, verbose = TRUE)
```

Arguments

`raw_data`	a data.frame in long format, containing the raw data.
`xpos_label`	a character string specifying the column containing the x-positions.
`ypos_label`	a character string specifying the column containing the y-positions.
`zpos_label`	an optional character string specifying the column containing the z-positions.
`timestamps_label`	a character string specifying the column containing the timestamps. If no timestamps are found in the data, a timestamps variable with increasing integers will be created (assuming equidistant time steps).

add_labels
: a character vector specifying columns containing additional mouse-tracking variables.

mt_id_label
: a character string (or vector) specifying the name of the column that provides a unique ID for every trial (the trial identifier). If more than one variable name is provided, a new ID variable will be created by combining the values of each variable. The trial identifier will be set as the rownames of the resulting trajectories and trial data, and additionally be stored in the column "mt_id" in the trial data.

mt_seq_label
: a character string specifying the column that indicates the order of the logged coordinates within a trial. If no column of the specified name is found in the data.frame, the coordinates will be imported in the order in which they were stored in `raw_data`.

reset_timestamps
: logical indicating if the first timestamp should be subtracted from all timestamps within a trial. Default is `TRUE` as it is recommended for all following analyses in mousetrap.

verbose
: logical indicating whether function should report its progress.

Details

The default arguments are set so that no adjustments have to be made when importing a data.frame that was created using mt_export_long.

The coordinates are ordered according to the values in the column provided in the `mt_seq_label` parameter (`mt_seq` by default). If the corresponding column does not exist, the coordinates will be imported in the order in which they were stored in the raw_data.

If no timestamps are found in the data, `mt_import_long` automatically creates a timestamps variable with increasing integers (starting with 0) assuming equally spaced sampling intervals.

Value

A mousetrap data object (see mt_example).

Author(s)

Pascal J. Kieslich (`<kieslich@psychologie.uni-mannheim.de>`)

Felix Henninger

See Also

mt_import_mousetrap and mt_import_wide for importing mouse-tracking data in other formats.

Examples

```
# Create data in long format for test purposes
mt_data_long <- mt_export_long(mt_example,
  use2_variables=c("subject_nr","Condition"))

# Import the data using mt_import_long
```

```
mt_data <- mt_import_long(mt_data_long)


## Not run:
# Import a hypothetical dataset that contains the
# custom mouse-tracking variables angle and velocity
mt_data <- mt_import_long(exp_data,
  add_labels= c("angle", "velocity"))

## End(Not run)
```

---

mt_import_mousetrap   *Import mouse-tracking data recorded using the mousetrap plug-ins in OpenSesame.*

Description

mt_import_mousetrap accepts a data.frame of (merged) raw data from a mouse-tracking experiment implemented in OpenSesame using the mousetrap plugin (Kieslich & Henninger, 2017). From this data.frame, mt_import_mousetrap creates a mousetrap data object containing the trajectories and additional data for further processing within the mousetrap package. Specifically, it returns a list that includes the trajectory data as an array (called trajectories), and all other data as a data.frame (called data). This data structure can then be passed on to other functions within this package (see mousetrap for an overview).

Usage

```
mt_import_mousetrap(raw_data, xpos_label = "xpos", ypos_label = "ypos",
  timestamps_label = "timestamps", mt_id_label = NULL, split = ",",
  duplicates = "remove_first", reset_timestamps = TRUE,
  verbose = FALSE)
```

Arguments

raw_data
: a data.frame containing the raw data.

xpos_label
: a character string specifying the name of the column(s) in which the x-positions are stored (see Details).

ypos_label
: a character string specifying the name of the column(s) in which the y-positions are stored (see Details).

timestamps_label
: a character string specifying the name of the column(s) in which the timestamps are stored (see Details).

mt_id_label
: an optional character string (or vector) specifying the name of the column that provides a unique ID for every trial (the trial identifier). If unspecified (the default), an ID variable will be generated. If more than one variable name is provided, a new ID variable will be created by combining the values of each

variable. The trial identifier will be set as the rownames of the resulting trajectories and trial data, and additionally be stored in the column "mt_id" in the trial data.

`split` a character string indicating how the different timestamps and coordinates within a trial are separated.

`duplicates` a character string indicating how duplicate timestamps within a trial are handled (see Details).

`reset_timestamps`
logical indicating if the first timestamp should be subtracted from all timestamps within a trial. Default is `TRUE` as it is recommended for all following analyses in mousetrap.

`verbose` logical indicating whether function should report its progress.

Details

When working with mouse-tracking data that were recorded using the mousetrap plug-ins for OpenSesame, usually only the `raw_data` need to be provided. All other arguments have sensible defaults.

If the relevant timestamps, x-positions, and y-positions are each stored in one variable, a character string specifying (parts of) the respective column name needs to be provided. In this case, the column names are extracted using grep to find the column that starts with the respective character string (in OpenSesame these will typically contain the name of the item that was used to record them, such as `xpos_get_response`). This means that the exact column names do not have to be provided - as long as only one column starts with the respective character string (otherwise, the exact column names have to be provided).

If several variables contain the timestamps, x-positions, and y-positions within a trial (e.g., `xpos_part1` and `xpos_part2`), a vector of the exact column names has to be provided (e.g., `xpos_label=c("xpos_part1","xpos_part2` `mt_import_mousetrap` will then merge all raw data in the order with which the variable labels have been specified. If one variable contains NAs or an empty string in a trial, these cases will be ignored (this covers the special case that, e.g., `xpos_part2` is only relevant for some trials and contains NAs in the other trials).

`duplicates` allows for different options to handle duplicate timestamps within a trial:

- `remove_first`: First timestamp and corresponding x-/y-positions are removed (the default).
- `remove_last`: Last timestamp and corresponding x-/y-positions are removed.
- `ignore`: Duplicates are kept.

Value

A mousetrap data object (see mt_example).

If mouse-tracking data were recorded using the mousetrap plug-ins for OpenSesame, the unit of the timestamps is milliseconds.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

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

See Also

read_opensesame from the `readbulk` library for reading and combining raw data files that were collected with OpenSesame.

mt_import_wide and mt_import_long for importing mouse-tracking data from other sources.

Examples

```
mt_data <- mt_import_mousetrap(mt_example_raw)
```

---

mt_import_wide *Import mouse-tracking data saved in wide format.*

Description

`mt_import_wide` receives a data.frame where mouse-tracking data are stored in wide format, i.e., where one row contains the data of one trial and every recorded mouse position and variable is saved in a separate variable (e.g., X_1, X_2, ..., Y_1, Y_2, ...). This is, e.g., the case when collecting data using MouseTracker (Freeman & Ambady, 2010). From this data.frame, `mt_import_wide` creates a mousetrap data object containing the trajectories and additional data for further processing within the mousetrap package. Specifically, it returns a list that includes the trajectory data as an array (called `trajectories`), and all other data as a data.frame (called `data`). This data structure can then be passed on to other functions within this package (see mousetrap for an overview).

Usage

```
mt_import_wide(raw_data, xpos_label = "X", ypos_label = "Y",
  zpos_label = NULL, timestamps_label = "T", add_labels = NULL,
  mt_id_label = NULL, pos_sep = "_", pos_ids = NULL,
  reset_timestamps = TRUE, verbose = TRUE)
```

Arguments

`raw_data`	a data.frame containing the raw data.
`xpos_label`	a character string specifying the core of the column labels containing the x-positions (e.g., "X" for "X_1", "X_2", ...).
`ypos_label`	a character string specifying the core of the column labels containing the y-positions (e.g., "Y" for "Y_1", "Y_2", ...).
`zpos_label`	a character string specifying the core of the column labels containing the z-positions.

timestamps_label
: an optional character string specifying the core of the column labels containing the timestamps. If no timestamps are found in the data, a timestamps variable with increasing integers will be created (assuming equidistant time steps).

add_labels
: a character vector specifying the core of columns containing additional mouse-tracking variables.

mt_id_label
: an optional character string (or vector) specifying the name of the column that provides a unique ID for every trial (the trial identifier). If unspecified (the default), an ID variable will be generated. If more than one variable name is provided, a new ID variable will be created by combining the values of each variable. The trial identifier will be set as the rownames of the resulting trajectories and trial data, and additionally be stored in the column "mt_id" in the trial data.

pos_sep
: a character string indicating the character that connects the core label and the position, (e.g., "_" for "X_1", "Y_1", ...).

pos_ids
: the vector of IDs used for indexing the x-coordinates, y-coordinates etc. (e.g., 1:101 for time-normalized trajectories from MouseTracker). If unspecified (the default), column labels for the respective variable will be extracted using grep (see Details).

reset_timestamps
: logical indicating if the first timestamp should be subtracted from all timestamps within a trial. Default is `TRUE` as it is recommended for all following analyses in mousetrap.

verbose
: logical indicating whether function should report its progress.

Details

`mt_import_wide` is designed to import mouse-tracking data saved in a wide format. The defaults are set so that usually only the `raw_data` need to be provided when data have been collecting using MouseTracker (Freeman & Ambady, 2010) and have been read into R using read_mt.

If no `pos_ids` are provided, column labels for the respective variable (e.g., x-positions) are extracted using `grep` returning all variables that start with the respective character string (e.g., "X_" if `xpos_label="X"` and `pos_sep="_"`).

If no timestamps are found in the data, `mt_import_wide` automatically creates a timestamps variable with increasing integers (starting with 0) assuming equally spaced sampling intervals.

Value

A mousetrap data object (see mt_example).

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

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.

See Also

read_mt for reading raw data that was collected using MouseTracker (Freeman & Ambady, 2010) and stored as a file in the ".mt" format.

mt_import_mousetrap and mt_import_long for importing mouse-tracking data in other formats.

Examples

```
# Create data in wide format for test purposes
mt_data_wide <- mt_export_wide(mt_example,
  use2_variables=c("subject_nr", "Condition"))

# Import the data using mt_import_wide
mt_data <- mt_import_wide(mt_data_wide,
  xpos_label="xpos", ypos_label="ypos",
  timestamps_label="timestamps")
```

---

mt_map *Map trajectories to prototypes.*

Description

mt_map maps trajectories onto a predefined set of prototype trajectories. It first computes distances between the trajectories and each of the supplied trajectory types and then assigns each trajectory to the prototype that produced the smallest distance.

Usage

```
mt_map(data, use = "sp_trajectories", save_as = "prototyping",
  dimensions = c("xpos", "ypos"),
  prototypes = mousetrap::mt_prototypes, weights = rep(1,
  length(dimensions)), pointwise = TRUE, na_rm = FALSE,
  minkowski_p = 2, use2 = "data", grouping_variables = NULL)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
save_as	a character string specifying where the resulting data should be stored.

dimensions
: a character vector specifying which trajectory variables should be used. Can be of length 2 or 3 for two-dimensional or three-dimensional trajectories respectively.

prototypes
: a trajectory array containing the prototypes the trajectories are mapped to. As a starting point, the trajectories stored in mt_prototypes can be used. See Details and Examples for selecting prototypes and creating new ones.

weights
: numeric vector specifying the relative importance of the variables specified in `dimensions`. Defaults to a vector of 1s implying equal importance. Technically, each variable is rescaled so that the standard deviation matches the corresponding value in `weights`. To use the original variables, set `weights = NULL`.

pointwise
: boolean specifying the way dissimilarity between the trajectories is measured (see Details). If `TRUE` (the default), `mt_distmat` measures the average dissimilarity and then sums the results. If `FALSE`, `mt_distmat` measures dissimilarity once (by treating the various points as independent dimensions).

na_rm
: logical specifying whether trajectory points containing NAs should be removed. Removal is done column-wise. That is, if any trajectory has a missing value at, e.g., the 10th recorded position, the 10th position is removed for all trajectories. This is necessary to compute distance between trajectories.

minkowski_p
: an integer specifying the distance metric. `minkowski_p = 1` computes the city-block distance, `minkowski_p = 2` (the default) computes the Euclidian distance, `minkowski_p = 3` the cubic distance, etc.

use2
: an optional character string specifying where the data that contain the variables used for grouping can be found (in case `grouping_variables` are specified). Defaults to "data" as `data[["data"]]` usually contains all non mouse-tracking trial data.

grouping_variables
: a character string (or vector) specifying one or more variables in `use2`. If specified, prototypes will be rescaled separately to match the coordinate system of the trajectories for each level of the variable(s). If unspecified (the default), the prototypes are rescaled in the same way across all trajectories.

Details

Mouse trajectories often occur in distinct, qualitative types (see Wulff et al., in press; Wulff et al., 2018). Common trajectory types are linear trajectories, mildly and strongly curved trajctories, and single and multiple change-of-mind trials. `mt_map` allows to map trajectories to a predefined set of trajectory types.

First, `mt_map` adjusts prototypes to match the coordinate system of the trajectories specified by `use`. Next, `mt_map` computes the distances between each trajectory and each of the supplied prototypes (see mt_distmat) and then assigns each trajectory to the closest prototype (i.e., the prototype that produced the smallest distance).

Mapping trajectories to prototypes requires that the endpoints of all trajectories (and added prototypes) share the same direction, i.e., that all trajectories end in the top-left corner of the coordinate system (mt_remap_symmetric or mt_align can be used to achieve this). Furthermore, it is recommended to use spatialized trajectories (see mt_spatialize; Wulff et al., in press; Haslbeck et al., 2018).

Value

A mousetrap data object (see mt_example) with an additional data.frame (by default called `prototyping`) that contains the best fitting prototype for each trajectory (the number of the prototype is specified under `prototype`, the label of the prototype under `prototype_label`) and the distance of the trajectory to the best fitting prototype (`min_dist`). If a trajectory array was provided directly as `data`, only the data.frame containing the results will be returned.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Jonas M. B. Haslbeck (<jonas.haslbeck@gmail.com>)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

References

Wulff, D. U., Haslbeck, J. M. B., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: Detecting types in movement trajectories. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 131-145). New York, NY: Routledge.

Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2018). *Measuring the (dis-)continuous mind: What movement trajectories reveal about cognition*. Manuscript in preparation.

Haslbeck, J. M. B., Wulff, D. U., Kieslich, P. J., Henninger, F., & Schulte-Mecklenbeck, M. (2018). *Advanced mouse- and hand-tracking analysis: Detecting and visualizing clusters in movement trajectories*. Manuscript in preparation.

Examples

```
# Spatialize trajectories
KH2017 <- mt_spatialize(KH2017)

# Map trajectories onto standard prototype set
KH2017 <- mt_map(KH2017,
  use="sp_trajectories")


# Plot prototypes
mt_plot(mt_prototypes,facet_col="mt_id") +
  ggplot2::facet_grid(.~factor(mt_id,levels=unique(mt_id)))

# Plot trajectories per assigned prototype
mt_plot(KH2017,use="sp_trajectories",
  use2="prototyping",facet_col="prototype_label")


# Map trajectories onto reduced prototype set
KH2017 <- mt_map(KH2017,
  use="sp_trajectories",
  prototypes=mt_prototypes[c("straight","curved","cCoM"),,],
  save_as="prototyping_red")
```

```
# Map trajectories onto extended prototype set

# Add additional prototypes
mt_prototypes_ext <- mt_add_trajectory(mt_prototypes,
   xpos = c(0,1,-1,1,-1), ypos = c(0,1.5,1.5,1.5,1.5), id = "dCoM3"
)
mt_prototypes_ext <- mt_add_trajectory(mt_prototypes_ext,
   xpos = c(0,0,-1), ypos = c(0,1.5,1.5), id = "neutral"
)

# Map trajectories
KH2017 <- mt_map(KH2017,
  use="sp_trajectories", prototypes=mt_prototypes_ext,
  save_as="prototyping_ext")
```

---

mt_measures *Calculate mouse-tracking measures.*

Description

Calculate a number of mouse-tracking measures for each trajectory, such as minima, maxima, and flips for each dimension, and different measures for curvature (e.g., `MAD`, `AD`, and `AUC`). Note that some measures are only returned if distance, velocity and acceleration are calculated using mt_derivatives before running `mt_measures`. More information on the different measures can be found in the Details and Values sections.

Usage

```
mt_measures(data, use = "trajectories", save_as = "measures",
  dimensions = c("xpos", "ypos"), timestamps = "timestamps",
  flip_threshold = 0, hover_threshold = NULL,
  hover_incl_initial = TRUE, verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the calculated measures should be stored.
`dimensions`	a character vector specifying the two dimensions in the trajectory array that contain the mouse positions. Usually (and by default), the first value in the vector corresponds to the x-positions (`xpos`) and the second to the y-positions (`ypos`).

`timestamps` a character string specifying the trajectory dimension containing the timestamps.

`flip_threshold` a numeric value specifying the distance that needs to be exceeded in one direction so that a change in direction counts as a flip. If several thresholds are specified, flips will be returned in separate variables for each threshold value (the variable name will be suffixed with the threshold value).

`hover_threshold` an optional numeric value. If specified, `hovers` (and `hover_time`) will be calculated as the number (and total time) of periods without movement in a trial (whose duration exceeds the value specified in `hover_threshold`). If several thresholds are specified, hovers and hover_time will be returned in separate variables for each threshold value (the variable name will be suffixed with the threshold value).

`hover_incl_initial` logical indicating if the calculation of hovers should include a potential initial phase in the trial without mouse movements (this initial phase is included by default).

`verbose` logical indicating whether function should report its progress.

Details

Note that some measures are only returned if distance, velocity and acceleration are calculated using mt_derivatives before running `mt_measures`. Besides, the meaning of these measures depends on the values of the arguments in mt_derivatives.

If the deviations from the idealized response trajectory have been calculated using mt_deviations before running `mt_measures`, the corresponding data in the trajectory array will be used. If not, `mt_measures` will calculate these deviations automatically.

The calculation of most measures can be deduced directly from their definition (see Value). For several more complex measures, a few details are provided in the following.

The signed **maximum absolute deviation** (`MAD`) is the maximum perpendicular deviation from the straight path connecting start and end point of the trajectory (e.g., Freeman & Ambady, 2010). If the `MAD` occurs above the direct path, this is denoted by a positive value. If it occurs below the direct path, this is denoted by a negative value. This assumes that the complete movement in the trial was from bottom to top (i.e., the end point has a higher y-position than the start point). In case the movement was from top to bottom, `mt_measures` automatically flips the signs. Both `MD_above` and `MD_below` are also reported separately.

The **average deviation** (`AD`) is the average of all deviations across the trial. Note that `AD` ignores the timestamps when calculating this average. This implicitly assumes that the time passed between each recording of the mouse is the same within each individual trajectory. If the `AD` is calculated using raw data that were obtained with an approximately constant logging resolution (sampling rate), this assumption is usually justified (mt_check_resolution can be used to check this). Alternatively, the `AD` can be calculated based on time-normalized trajectories; these can be computed using mt_time_normalize which creates equidistant time steps within each trajectory.

The `AUC` represents the **area under curve**, i.e., the geometric area between the actual trajectory and the direct path. Areas above the direct path are added and areas below are subtracted. The `AUC` is calculated using the polyarea function from the pracma package.

Note that all **time** related measures (except idle_time and hover_time) are reported using the timestamp metric as present in the data. To interpret the timestamp values as time since tracking start, the assumption has to be made that for each trajectory the tracking started at timestamp 0 and that all timestamps indicate the time passed since tracking start. Therefore, all timestamps should be reset during data import by subtracting the value of the first timestamp from all timestamps within a trial (assuming that the first timestamp corresponds to the time when tracking started). Timestamps are reset by default when importing the data using one of the mt_import functions (e.g., mt_import_mousetrap).

Value

A mousetrap data object (see mt_example) where an additional data.frame has been added (by default called "measures") containing the per-trial mouse-tracking measures. Each row in the data.frame corresponds to one trajectory (the corresponding trajectory is identified via the rownames and, additionally, in the column "mt_id"). Each column in the data.frame corresponds to one of the measures. If a trajectory array was provided directly as data, only the measures data.frame will be returned.

The following measures are computed for each trajectory (the labels relating to x- and y-positions will be adapted depending on the values specified in dimensions). Please note that additional information is provided in the Details section.

mt_id	Trial ID (can be used for merging measures data.frame with other trial-level data)
xpos_max	Maximum x-position
xpos_min	Minimum x-position
ypos_max	Maximum y-position
ypos_min	Minimum y-position
MAD	Signed Maximum absolute deviation from the direct path connecting start and end point of the trajectory (straight line). If the MAD occurs above the direct path, this is denoted by a positive value; if it occurs below, by a negative value.
MAD_time	Time at which the maximum absolute deviation was reached first
MD_above	Maximum deviation above the direct path
MD_above_time	Time at which the maximum deviation above was reached first
MD_below	Maximum deviation below the direct path
MD_below_time	Time at which the maximum deviation below was reached first
AD	Average deviation from direct path
AUC	Area under curve, the geometric area between the actual trajectory and the direct path where areas below the direct path have been subtracted
xpos_flips	Number of directional changes along x-axis (exceeding the distance specified in flip_threshold)
ypos_flips	Number of directional changes along y-axis (exceeding the distance specified in flip_threshold)
xpos_reversals	Number of crossings of the y-axis
ypos_reversals	Number of crossings of the x-axis

RT	Response time, time at which tracking stopped
initiation_time	Time at which first mouse movement was initiated
idle_time	Total time without mouse movement across the entirety of the trial
hover_time	Total time of all periods without movement in a trial (whose duration exceeds the value specified in hover_threshold)
hovers	Number of periods without movement in a trial (whose duration exceeds the value specified in hover_threshold)
total_dist	Total distance covered by the trajectory
vel_max	Maximum velocity
vel_max_time	Time at which maximum velocity occurred first
vel_min	Minimum velocity
vel_min_time	Time at which minimum velocity occurred first
acc_max	Maximum acceleration
acc_max_time	Time at which maximum acceleration occurred first
acc_min	Minimum acceleration
acc_min_time	Time at which minimum acceleration occurred first

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

Kieslich, P. J., Henninger, F., Wulff, D. U., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2019). Mouse-tracking: A practical guide to implementation and analysis. In M. Schulte-Mecklenbeck, A. Kühberger, & J. G. Johnson (Eds.), *A Handbook of Process Tracing Methods* (pp. 111-130). New York, NY: Routledge.

Kieslich, P. J., Wulff, D. U., Henninger, F., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2018). *Mouse- and hand-tracking as a window to cognition: A tutorial on implementation, analysis, and visualization*. Manuscript in preparation.

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.

See Also

mt_sample_entropy for calculating sample entropy.

mt_standardize for standardizing the measures per subject.

mt_check_bimodality for checking bimodality of the measures using different methods.

mt_aggregate and mt_aggregate_per_subject for aggregating the measures.

inner_join for merging data using the dplyr package.

Examples

```
mt_example <- mt_derivatives(mt_example)
mt_example <- mt_deviations(mt_example)
mt_example <- mt_measures(mt_example)

# Merge measures with trial data
mt_example_results <- dplyr::inner_join(
  mt_example$data, mt_example$measures,
  by="mt_id")
```

---

mt_plot *Plot trajectory data.*

Description

mt_plot can be used for plotting a number of individual trajectories. mt_plot_aggregate can be used for plotting aggregated trajectories. The color and linetype can be varied depending on a set of condition variables using the color and linetype arguments. If the x and y arguments are varied, this function can also be used for plotting velocity and acceleration profiles.

Usage

```
mt_plot(data, use = "trajectories", use2 = "data", x = "xpos",
  y = "ypos", color = NULL, linetype = NULL, alpha = NA,
  size = 0.5, facet_row = NULL, facet_col = NULL, points = FALSE,
  only_ggplot = FALSE, mt_id = "mt_id", ...)

mt_plot_aggregate(data, use = "trajectories", use2 = "data",
  x = "xpos", y = "ypos", color = NULL, linetype = NULL,
  alpha = NA, size = 0.5, facet_row = NULL, facet_col = NULL,
  points = FALSE, only_ggplot = FALSE, subject_id = NULL, ...)
```

Arguments

data — a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).

use — a character string specifying which trajectories should be plotted. The corresponding trajectories are selected from data using data[[use]]. Usually, this value corresponds to either "trajectories", "tn_trajectories" or "av_trajectories", depending on whether the raw, time-normalized or averaged trajectories should be plotted.

use2 — a character string specifying where the data that contain the variables used for determining the color and linetype can be found (in case these arguments are specified). Defaults to "data" as data[["data"]] usually contains all non mouse-tracking trial data.

x	a character string specifying which dimension in the trajectory array should be displayed on the x-axis (defaults to xpos).
y	a character string specifying which dimension in the trajectory array should be displayed on the y-axis (defaults to ypos).
color	an optional character string specifying which variable in data[[use2]] should be used for coloring the trajectories.
linetype	an optional character string specifying which variable in data[[use2]] should be used for varying the linetype of the trajectories.
alpha	an optional numeric value between 0 and 1 that can be used to make the plotted lines (and points) semitransparent.
size	an optional numeric value that can be used to vary the width of the plotted trajectory lines.
facet_row	an optional character string specifying a variable in data[[use2]] that should be used for (row-wise) faceting.
facet_col	an optional character string specifying a variable in data[[use2]] that should be used for (column-wise) faceting.
points	logical. If TRUE, points will be added to the plot using geom_point.
only_ggplot	logical. If TRUE, only the ggplot object without geoms is returned. If FALSE (the default), the trajectories are plotted using geom_path.
mt_id	a character string specifying the internal label used for the trial identifier (passed on to the group aesthetic). Only relevant for mt_plot.
...	additional arguments passed on to mt_reshape (such as subset).
subject_id	a character string specifying which column contains the subject identifier. Only relevant for mt_plot_aggregate. If specified, aggregation will be performed within subjects first. Note that aggregation will be performed separately for each level, including all subjects for whom data are available.

Details

mt_plot internally uses mt_reshape for reshaping trajectories into a long format. Next, it creates a ggplot object using the ggplot function of the ggplot2 package. The aes mappings are taken from the function arguments for x, y etc.; in addition, the group mapping is set to the internal trial identifier (by default called "mt_id").

If only_ggplot==FALSE, the trajectories are plotted using the geom_path function of the ggplot2 package. If only_ggplot==TRUE, the ggplot object is returned without layers, which can be used to further customize the plot, e.g., to specify a custom size for the lines or to create semitransparent lines by specifying an alpha level < 1 (see Examples).

mt_plot_aggregate works similarly, but uses mt_aggregate for reshaping and aggregating trajectories prior to plotting.

Please note that this function is intended as a quick and easy solution for visualizing mouse trajectories. For additional flexibility, we recommend that mt_reshape or mt_aggregate be used in conjunction with ggplot to create custom visualizations.

Functions

- `mt_plot`: Plot individual trajectory data
- `mt_plot_aggregate`: Plot aggregated trajectory data

Author(s)

Pascal J. Kieslich (`<kieslich@psychologie.uni-mannheim.de>`)

Felix Henninger

See Also

mt_plot_add_rect for adding rectangles representing the response buttons to the plot.

mt_plot_riverbed for plotting the relative frequency of a selected variable across time.

mt_plot_per_trajectory for individually plotting all trajectories as individual pdf files.

Examples

```
# Load ggplot2
library(ggplot2)


## Plot individual example trajectories

# Time-normalize trajectories
mt_example <- mt_time_normalize(mt_example)

# Plot all time-normalized trajectories
# varying the color depending on the condition
mt_plot(mt_example, use="tn_trajectories",
  color="Condition")

# ... with custom colors
mt_plot(mt_example, use="tn_trajectories",
  color="Condition") +
  ggplot2::scale_color_brewer(type="qual")

# Create separate plots per Condition
mt_plot(mt_example, use="tn_trajectories",
  facet_col="Condition")

# Plot velocity profiles based on the averaged trajectories
# varying the color depending on the condition
mt_example <- mt_derivatives(mt_example)
mt_example <- mt_average(mt_example, interval_size = 100)
mt_plot(mt_example, use="av_trajectories",
  x="timestamps", y="vel", color="Condition")


## Plot aggregate trajectories for KH2017 data
```

```
# Time-normalize trajectories
KH2017 <- mt_time_normalize(KH2017)

# Plot aggregated time-normalized trajectories per condition
mt_plot_aggregate(KH2017, use="tn_trajectories",
  color="Condition")

# ... first aggregating trajectories within subjects
mt_plot_aggregate(KH2017, use="tn_trajectories",
  color="Condition", subject_id="subject_nr")

# ... adding points for each position to the plot
mt_plot_aggregate(KH2017, use="tn_trajectories",
  color="Condition", points=TRUE)

## Not run:
# Create customized aggregate trajectory plot
# by using only_ggplot option to return a ggplot object without geoms
# and by adding a geom to it with a custom line width
mt_plot_aggregate(KH2017, use="tn_trajectories",
  color="Condition", only_ggplot=TRUE) +
  geom_path(size=1.5)

# Create customized plot of individual trajectories
# by using only_ggplot option to return a ggplot object without geoms
# and by adding a geom to it with semitransparent lines
# (by specifying alpha < 1)
mt_plot(KH2017, use="tn_trajectories", only_ggplot=TRUE) +
  geom_path(alpha=0.2)

## End(Not run)
```

---

mt_plot_add_rect *Add rectangles to trajectory plot.*

Description

mt_plot_add_rect adds one or several rectangles to a mousetrap plot. These buttons usually correspond to the borders of the buttons in the mouse-tracking experiment. It is specifically designed so that the arguments from the mousetrap_response plugin in OpenSesame can be used.

Usage

```
mt_plot_add_rect(rect, color = "black", fill = NA, ...)
```

Arguments

rect a data.frame or matrix with one row per box. For each rectangle, the x-position (x), y-position (y), width (w), and height (h) needs to be provided. If columns are not labeled, the order x,y,w,h is assumed.

`color`	argument passed on to geom_rect. Specifies the color of the border of the rectangles.
`fill`	argument passed on to geom_rect. Specifies the color of the interior of the rectangles. If `NA` (the default), rectangles are unfilled.
`...`	additional arguments passed on to geom_rect.

Details

`mt_plot_add_rect` internally uses geom_rect of the `ggplot2` package for plotting.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_plot for plotting trajectory data.

Examples

```
# Load ggplot2
library(ggplot2)

# Import, flip, and time-normalize raw trajectories
mt_example <- mt_import_mousetrap(mt_example_raw)
mt_example <- mt_remap_symmetric(mt_example,remap_xpos="no")
mt_example <- mt_time_normalize(mt_example)

# Create rectangles matrix
rectangles <- matrix(
  # (The matrix is n x 4, and contains
  # all relevant data for every button,
  # (i.e. x, y, width and height values)
  # in separate rows)
  c(
   -840, 525,  350, -170,
    840, 525, -350, -170
  ),
  ncol=4, byrow=TRUE)

# Plot all time-normalized trajectories
# varying the color depending on the condition
# and add rectangles
mt_plot(mt_example,
  use="trajectories",
  x="xpos", y="ypos", color="Condition"
) + mt_plot_add_rect(rect=rectangles)
```

mt_plot_per_trajectory

Create PDF with separate plots per trajectory.

Description

mt_plot_per_trajectory creates a PDF file with separate plots per trajectory. This PDF can be used for inspecting individual trajectories. Note that plotting all trajectories can be time-consuming, especially for raw trajectories. If the appropriate x and y arguments are inserted, this function can also be used for plotting velocity and acceleration profiles.

Usage

```
mt_plot_per_trajectory(file, data, use = "trajectories", x = "xpos",
  y = "ypos", xlim = NULL, ylim = NULL, axes_exact = FALSE,
  points = FALSE, rect = NULL, color = "black", fill = NA,
  verbose = FALSE, ...)
```

Arguments

file	a character string specifying the name of the PDF file. Passed on to pdf.
data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectories should be plotted. The corresponding trajectories are selected from data using data[[use]]. Usually, this value corresponds to either "trajectories", "tn_trajectories" or "av_trajectories", depending on whether the raw, time-normalized or averaged trajectories should be plotted.
x	a character string specifying which dimension in the trajectory array should be displayed on the x-axis (defaults to xpos).
y	a character string specifying which dimension in the trajectory array should be displayed on the y-axis (defaults to ypos).
xlim	optional argument specifying the limits for the x axis (passed on to coord_cartesian). If not specified (the default), sensible axis limits will be computed.
ylim	optional argument specifying the limits for the y axis (passed on to coord_cartesian). If not specified (the default), sensible axis limits will be computed.
axes_exact	logical. If TRUE, axes will be set without offset exactly at the limits of the x and y axes (which can be specified using xlim and ylim]).
points	logical. If TRUE, points will be added to the plot using geom_point.
rect	optional argument passed on to mt_plot_add_rect. If specified, rectangles (usually representing the response buttons) will be plotted for each trajectory plot.
color	optional argument passed on to mt_plot_add_rect. Only relevant if rect is specified.

`fill`	optional argument passed on to mt_plot_add_rect. Only relevant if `rect` is specified.
`verbose`	logical indicating whether function should report its progress.
`...`	additional arguments passed on to pdf.

Details

`mt_plot_per_trajectory` creates a PDF using pdf. Next, it plots all trajectories individually using mt_plot. Every plot is labeled using the rownames of the trajectories.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_plot for plotting trajectory data.

Examples

```
## Not run:
mt_plot_per_trajectory(mt_example,
  file="trajectories.pdf",
  use="trajectories")

## End(Not run)
```

`mt_plot_riverbed` *Plot density of mouse positions across time steps.*

Description

`mt_plot_riverbed` creates a plot showing the distribution of one trajectory variable (e.g., the x-positions or velocity) per time step.

Usage

```
mt_plot_riverbed(data, use = "tn_trajectories", y = "xpos",
  y_range = NULL, y_bins = 250, facet_row = NULL, facet_col = NULL,
  facet_data = "data", grid_colors = c("gray30", "gray10"))
```

Arguments

data	mousetrap data object containing the data to be plotted.
use	character string specifying the set of trajectories to use in the plot. The steps of this set will constitute the x axis. Defaults to 'tn_trajectories', which results in time steps being plotted on the x axis.
y	variable in the mousetrap data object to be plotted on the output's y dimension. Defaults to 'xpos', the cursor's x coordinate.
y_range	numerical vector containing two values that represent the upper and lower ends of the y axis. By default, the range is calculated from the data provided.
y_bins	number of bins to distribute along the y axis (defaults to 250).
facet_row	an optional character string specifying a variable in `data[[facet_data]]` that should be used for (row-wise) faceting. If specified, separate riverbed plots for each level of the variable will be created.
facet_col	an optional character string specifying a variable in `data[[facet_data]]` that should be used for (column-wise) faceting. If specified, separate riverbed plots for each level of the variable will be created.
facet_data	a character string specifying where the (optional) data containing the faceting variables can be found.
grid_colors	a character string or vector of length 2 specifying the grid color(s). If a single value is provided, this will be used as the grid color. If a vector of length 2 is provided, the first value will be used as the color for the major grid lines, the second value for the minor grid lines. If set to `NA`, no grid lines are plotted.

Details

This function plots the relative frequency of the values of a trajectory variable separately for each of a series of time steps. This type of plot has been used in previous research to visualize the distribution of x-positions per time step (e.g., Scherbaum et al., 2010).

`mt_plot_riverbed` usually is applied to time-normalized trajectory data as all trajectories must contain the same number of values.

Author(s)

Felix Henninger (<mailbox@felixhenninger.com>)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

References

Scherbaum, S., Dshemuchadse, M., Fischer, R., & Goschke, T. (2010). How decisions evolve: The temporal dynamics of action selection. *Cognition, 115*(3), 407-416.

Scherbaum, S., & Kieslich, P. J. (in press). Stuck at the starting line: How the starting procedure influences mouse-tracking data. *Behavior Research Methods*.

See Also

mt_plot for plotting trajectory data.

mt_time_normalize for time-normalizing trajectories.

Examples

```
# Time-normalize trajectories
KH2017 <- mt_time_normalize(KH2017)

# Create riverbed plot for all trials
mt_plot_riverbed(KH2017)

## Not run:
# Create separate plots for typical and atypical trials
mt_plot_riverbed(mt_example, facet_col="Condition")


# Create riverbed plot for all trials with custom x and y axis labels
mt_plot_riverbed(mt_example) +
  ggplot2::xlab("Time step") + ggplot2::ylab("X coordinate")

# Note that it is also possible to replace the
# default scale for fill with a custom scale
mt_plot_riverbed(mt_example, facet_col="Condition") +
  ggplot2::scale_fill_gradientn(colours=grDevices::heat.colors(9),
    name="Frequency", trans="log", labels=scales::percent)

## End(Not run)
```

---

mt_prototypes *Mouse trajectory prototypes.*

Description

A core set of five mouse trajectory prototypes including the 'straight' trajectory, the mildly curved trajectory, the continuous change-of-mind trajectory, the discrete change-of-mind trajectory, and the double discrete change-of-mind trajectory.

Usage

```
mt_prototypes
```

Format

An object of class `array` of dimension 5 x 100 x 2.

Details

Mouse- and hand-trajectories often occur in types (Wulff, Haslbeck, & Schulte-Mecklenbeck, 2017). In such cases, movement trajectory data should be analyzed in terms of discrete type assignments. To this end mt_map can be used to map mouse- or hand-trajectory to the closest of several predefined prototypes. `mt_prototypes` provides a core set of prototypes that has been shown to represent well a large fraction of empirical movement trajectories.

To tailor the set of prototypes to a given study, `mt_prototypes` can be extended using mt_add_trajectory.

References

Wulff, D. U., Haslbeck, J. M. B., Schulte-Mecklenbeck, M. (2018). *Measuring the (dis-)continuous mind: What movement trajectories reveal about cognition*. Manuscript in preparation.

---

mt_qeffect — *Create quantile-effect plot*

Description

Function in beta and currently only for internal purposes.

Usage

```
mt_qeffect(data, compare, use = "measures", measure = "MAD",
  direction = "upward", n_steps = 100, return_data = FALSE, ...)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`compare`	either a vector, the label of a variable in , or a mousetrap object.
`use`	a character string specifying which trajectory data should be used.
`measure`	a character value specifying the variable used to calculate the effect between
`direction`	a character value.
`n_steps`	an integer.
`return_data`	boolean.
`...`	additional arguments passed on to points.

Value

Nothing, when image is plotted using an external device. Otherwise an object of class `mt_object_raw` containing in a matrix format the image's pixel information.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Examples

```
# Plot regular heatmap
#SpiveyEtAl2005 = mt_import_long(SpiveyEtAl2005_raw,'x','y',NULL,'t',
#mt_id_label = c('ptp','trial'))
#heatmap = mt_heatmap_raw(SpiveyEtAl2005,xres = 2000)
#mt_heatmap(heatmap,file = NULL)

# compute measures
#SpiveyEtAl2005 = mt_measures(SpiveyEtAl2005)

# Plot heatmap using velocity
#mt_heatmap(SpiveyEtAl2005)
```

---

mt_remap_symmetric   *Remap mouse trajectories.*

Description

Remap all trajectories to one side (or one quadrant) of the coordinate system. In doing so, mt_remap_symmetric assumes a centered coordinate system and a symmetric design of the response buttons (see Details).

Usage

```
mt_remap_symmetric(data, use = "trajectories", save_as = use,
  dimensions = c("xpos", "ypos"), remap_xpos = "left",
  remap_ypos = "up")
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
save_as	a character string specifying where the resulting trajectory data should be stored.
dimensions	a character vector specifying the two dimensions in the trajectory array that contain the mouse positions, the first value corresponding to the x-positions, the second to the y-positions.
remap_xpos	character string indicating the direction in which to remap values on the x axis. If set to "left" (as per default), trajectories with an endpoint on the right (i.e. with a positive x-value) will be remapped to the left. The alternatives are "right" which has the reverse effect, and "no", which disables remapping on the horizontal dimension.

remap_ypos — character string defining whether tracks directed downwards on the y axis should be remapped so that they end with a positive y value. This will be performed if this parameter is set to "up" (which is the default), and the reverse occurs if the parameter is set to "down". If it is set to "no", y-values remain untouched.

Details

When mouse trajectories are compared across different conditions, it is typically desirable that the endpoints of the trajectories share the same direction (e.g., diagonally up and left). This way, the trajectories can be compared regardless of the button they were directed at.

`mt_remap_symmetric` can be used to achieve this provided that two assumptions hold:

First, this function assumes a centered coordinate system, i.e. the coordinate system is centered on the screen center. This is the case when the data is produced by the mousetrap plug-ins in OpenSesame.

Second, it assumes that the response buttons in the mouse-tracking experiment are symmetric, in that they all are equally distant from the screen center.

Value

A mousetrap data object (see mt_example) with remapped trajectories. If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

Examples

```
# Remap trajectories so that all trajectories
# end in the top-left corner
mt_example <- mt_import_mousetrap(mt_example_raw)
mt_example <- mt_remap_symmetric(mt_example)

# Only flip trajectories vertically so that all
# trajectories end in the upper half of the screen
mt_example <- mt_import_mousetrap(mt_example_raw)
mt_example <- mt_remap_symmetric(mt_example,
  remap_xpos="no", remap_ypos="up")
```

`mt_resample` *Resample trajectories using a constant time interval.*

Description

Resample trajectory positions using a constant time interval. If no timestamp that represents an exact multiple of this time interval is found, linear interpolation is performed using the two adjacent timestamps.

Usage

```
mt_resample(data, use = "trajectories", save_as = "rs_trajectories",
  dimensions = c("xpos", "ypos"), timestamps = "timestamps",
  step_size = 10, exact_last_timestamp = TRUE,
  constant_interpolation = NULL, verbose = FALSE)
```

Arguments

`data` a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).

`use` a character string specifying which trajectory data should be used.

`save_as` a character string specifying where the resulting trajectory data should be stored.

`dimensions` a character vector specifying the dimensions in the trajectory array that should be resampled. If `"all"`, all trajectory dimensions except the timestamps will be resampled.

`timestamps` a character string specifying the trajectory dimension containing the timestamps.

`step_size` an integer specifying the size of the constant time interval. The unit corresponds to the unit of the timestamps.

`exact_last_timestamp` logical indicating if the last timestamp should always be appended (which is the case by default). If `FALSE`, the last timestamp is only appended if it is a multiple of the step_size.

`constant_interpolation` an optional integer. If specified, constant instead of linear interpolation will be performed for all adjacent timestamps whose difference exceeds the number specified for `constant_interpolation`. The unit corresponds to the unit of the timestamps.

`verbose` logical indicating whether function should report its progress.

Details

`mt_resample` can be used if the number of logged positions in a trial should be reduced. `mt_resample` achieves this by artificially decreasing the resolution with which the positions were recorded. For example, if mouse positions were recorded every 10 ms in an experiment, but one was only interested in the exact mouse position every 50 ms, `mt_resample` with `step_size=50` could be used. In this case, only every fifth sample would be kept.

In addition, `mt_resample` can be used to only retain values for specific timestamps across trials (e.g., if for each trial the position of the mouse exactly 250 ms and 500 ms after onset of the trial are of interest). In case that a trial does not contain samples at the specified timestamps, linear interpolation is performed using the two adjacent timestamps.

If a number is specified for `constant_interpolation`, constant instead of linear interpolation will be performed for all adjacent timestamps whose difference exceeds this number. Specifically, a period without mouse movement will be assumed starting at the respective timestamp until the next timestamp - `constant_interpolation/2`.

Note that `mt_resample` does not average across time intervals. For this, mt_average can be used.

Value

A mousetrap data object (see mt_example) with an additional array (by default called `rs_trajectories`) containing the resampled trajectories. If a trajectory array was provided directly as `data`, only the resampled trajectories will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

approx for information about the function used for linear interpolation.

mt_average for averaging trajectories across constant time intervals.

mt_time_normalize for time-normalizing trajectories.

Examples

```
mt_example <- mt_resample(mt_example,
  save_as="rs_trajectories",
  step_size=50)
```

`mt_reshape` *General-purpose reshape and aggregation function for mousetrap data.*

Description

`mt_reshape` is the general function used in the `mousetrap` package for filtering, merging, reshaping, and aggregating mouse-tracking measures or trajectories in combination with other trial data. Several additional (wrapper) functions for more specific purposes (cf. "See Also") are available.

Usage

```
mt_reshape(data, use = "trajectories", use_variables = NULL,
  use2 = "data", use2_variables = NULL, subset = NULL,
  subject_id = NULL, aggregate = FALSE,
  aggregate_subjects_only = FALSE, .funs = "mean",
  trajectories_long = TRUE, convert_df = TRUE, mt_id = "mt_id",
  mt_seq = "mt_seq", aggregation_function = NULL)
```

Arguments

`data` a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).

`use` a character string specifying which data should be reshaped. The corresponding data are selected from data using `data[[use]]`. Usually, this value corresponds to either "trajectories", "tn_trajectories", or "measures", depending on whether the analysis concerns raw trajectories, time-normalized trajectories, or derived measures.

`use_variables` a character vector specifying which mouse-tracking variables should be reshaped. Corresponds to the column names in case a data.frame with mouse-tracking measures is provided. Corresponds to the labels of the array dimensions in case a trajectory array is provided. If unspecified, all variables will be reshaped.

`use2` an optional character string specifying where the other trial data can be found. Defaults to "data" as `data[["data"]]` usually contains all non mouse-tracking trial data. Alternatively, a data.frame can be provided directly.

`use2_variables` an optional character string (or vector) specifying the variables (in `data[[use2]]`) that should be merged with the data. If `aggregate==TRUE`, the trajectories / measures will be aggregated separately for each of the levels of these variables using summarize_at.

`subset` a logical expression (passed on to subset) indicating elements or rows to keep. If specified, `data[[use2]]` will be subsetted using this expression, and, afterwards, `data[[use]]` will be filtered accordingly.

`subject_id` an optional character string specifying which column contains the subject identifier in `data[[use2]]`. If specified and `aggregate==TRUE`, aggregation will be performed within subjects first.

`aggregate` logical indicating whether data should be aggregated. If `use2_variables` are specified, aggregation will be performed separately for each of the levels of the `use2_variables`.

`aggregate_subjects_only`
logical indicating whether data should only be aggregated per subject (if `subject_id` is specified and `aggregate==TRUE`).

`.funs` the aggregation function(s) passed on to `summarize_at`. By default, the `mean` is calculated.

`trajectories_long`
logical indicating if the reshaped trajectories should be returned in long or wide format. If `TRUE`, every recorded position in a trajectory is placed in another row

(whereby the order of the positions is logged in the variable mt_seq). If FALSE, every trajectory is saved in wide format and the respective positions are indexed by adding an integer to the corresponding label (e.g., xpos_1, xpos_2, ...). Only relevant if data[[use]] contains trajectories.

convert_df
: logical indicating if the reshaped data should be converted to a data.frame using as.data.frame. This will drop potentially existing additional classes (such as tbl_df) that result from the internally used dplyr functions for data grouping and aggregation. As these additional classes might - on rare occasions - cause problems with functions from other packages, the reshaped data are converted to "pure" data.frames by default.

mt_id
: a character string specifying the name of the column that will contain the trial identifier in the reshaped data. The values for the trial identifier correspond to the rownames of data[[use]] and data[[use2]].

mt_seq
: a character string specifying the name of the column that will contain the integers indicating the order of the mouse positions per trajectory in the reshaped data. Only relevant if data[[use]] contains trajectories and trajectories_long==TRUE.

aggregation_function
: Deprecated. Please use .funs instead.

Details

mt_reshape uses the rownames of data[[use]] and data[[use2]] for merging the trajectories / measures and the trial data. For convenience (and for trajectories in long format also of necessity), an additional column (labelled as specified in the mt_id argument) is added to the reshaped data containing the rownames as trial identifier.

The main purpose of this function is to reshape the trajectory data into a two-dimensional data.frame, as this format is required for many further analyses and plots in R.

Besides, it should aid the user in combining data contained in different parts of the mousetrap data object, e.g., a condition variable stored in data[["data"]] with trajectory data stored in data[["trajectories"]] (or mouse-tracking measures stored in data[["measures"]]).

Finally, it offers the possibility to aggregate trajectories and measures for different conditions and/or subjects.

The package also includes several functions that wrap mt_reshape and serve specific purposes. They are often easier to use, and thus recommended over mt_reshape unless the utmost flexibility is required. These functions are described in the section "See Also".

Note also that many merging, reshaping, and aggregation procedures can be performed directly by using some of the basic R functions, e.g., merge and aggregate, or through the R packages dplyr or reshape2, if desired.

Value

A data.frame containing the reshaped data.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_aggregate for aggregating mouse-tracking measures and trajectories.

mt_aggregate_per_subject for aggregating mouse-tracking measures and trajectories per subject.

mt_export_long for exporting mouse-tracking data in long format.

mt_export_wide for exporting mouse-tracking data in wide format.

inner_join for merging data and `summarize_at` for aggregating data using the `dplyr` package.

Examples

```
# Time-normalize trajectories
mt_example <- mt_time_normalize(mt_example)

# Reshape time-normalized trajectories data into long format
# adding Condition variable
trajectories_long <- mt_reshape(mt_example,
 use="tn_trajectories",
 use2_variables="Condition"
 )

# Reshape time-normalized trajectories data into wide format
# only keeping xpos and ypos
# and adding Condition variable
trajectories_wide <- mt_reshape(mt_example,
  use="tn_trajectories", use_variables = c("xpos","ypos"),
  use2_variables = "Condition",
  trajectories_long = FALSE
  )
```

`mt_sample_entropy` *Calculate sample entropy.*

Description

Calculate sample entropy for each trajectory as a measure of the complexity of movements along one specific dimension.

Usage

```
mt_sample_entropy(data, use = "tn_trajectories", save_as = "measures",
  dimension = "xpos", m = 3, r = NULL, use_diff = TRUE,
  verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the calculated measures should be stored.
`dimension`	a character string specifying the dimension based on which sample entropy should be calculated. By default (xpos), the x-positions are used.
`m`	an integer passed on to the sample entropy function (see Details).
`r`	a numeric value passed on to the sample entropy function (see Details).
`use_diff`	logical indicating if the differences of the dimension values should be computed before calculating sample entropy (which is done by default, see Details).
`verbose`	logical indicating whether function should report its progress.

Details

`mt_sample_entropy` calculates the sample entropy for each trajectory as a measure of its complexity. Hehman et al (2015) provide details on how sample entropy can be calculated and applied in mouse-tracking (following Dale et al., 2007). They apply the sample entropy measure to the differences between adjacent x-positions (which is also the default here, as in a standard mouse-tracking task with buttons located in the top-left and right corners mostly the movements in the horizontal direction are of interest). Besides, they recommend using the time-normalized trajectories so all trajectories have the same length.

Sample entropy is computed by comparing windows of a fixed size (specified using `m`) across all recorded positions. Sample entropy is the negative natural logarithm of the conditional probability that this window remains similar across the trial (Hehman et al., 2015). A window is considered to be similar to another if their distance is smaller than a specified tolerance value (which can be specified using `r`). Hehman et al. (2015) use a tolerance value of 0.2 * standard deviation of all differences between adjacent x-positions in the dataset (which is the default implemented here).

Value

A mousetrap data object (see mt_example).

If a data.frame with label specified in `save_as` (by default "measures") already exists, the sample entropy values are added as additional column.

If not, an additional data.frame will be added.

If a trajectory array was provided directly as `data`, only the data.frame will be returned.

Author(s)

Pascal J. Kieslich (`<kieslich@psychologie.uni-mannheim.de>`)

Dirk Wulff

Felix Henninger

References

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15-28.

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.

See Also

mt_measures for calculating other mouse-tracking measures.

Examples

```
# Calculate sample entropy based on time-normalized
# trajectories and merge results with other meausres
# derived from raw trajectories
mt_example <- mt_measures(mt_example)
mt_example <- mt_time_normalize(mt_example,
  save_as="tn_trajectories", nsteps=101)
mt_example <- mt_sample_entropy(mt_example,
  use="tn_trajectories", save_as="measures",
  dimension="xpos", m=3)
```

---

mt_scale_trajectories *Standardize variables in mouse trajectory array.*

Description

`mt_scale_trajectories` centers and / or standardizes selected trajectory variables within or across trajectories.

Usage

```
mt_scale_trajectories(data, use = "trajectories", save_as = use,
  var_names, center = TRUE, scale = TRUE, within_trajectory = FALSE,
  prefix = "z_", transform = NULL)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`var_names`	character vector giving the labels of the to be standardized variables.

`center` logical specifying whether variables should be centered (i.e., `mean = 0`). Can be a logical vector, in which case the values of `scale` are mapped to the variables specified in `var_names`.

`scale` logical or numeric specifying the scaling of the variables. When logical, `scale = TRUE` normalizes the trajectory variable to sd = 1, whereas `scale = FALSE` leaves the variable on its original scale. When numeric, the trajectory variables are scaled by (i.e., divided by) the specific value in scale. Can also be a numeric vector, in which case the values of `scale` are mapped to the variables specified in `var_names`.

`within_trajectory` logical specifying whether trajectory variables should be scaled within or across trajectories. If `within_trajectory == TRUE`, scaling trajectories to mean = 0 and sd = 1 means that every to be standardized trajectory variable will have mean = 0 and sd = 1. If `within_trajectory == FALSE` (the default), mean = 0 and sd = 1 are only true in the aggregate (i.e., across all trajectories). Can be a logical vector, in which case the values of `scale` are mapped to the variables specified in `var_names`.

`prefix` character string added to the names of the new standardized variables. If `prefix = ""`, the original variables will be overwritten.

`transform` function that takes a numeric matrix as argument and returns a numeric matrix of same size with transformed values. If `NULL` the original values are passed on to standardization.

Value

A mousetrap data object (see mt_example) with an additional variable containing the standardized trajectory variable added to the trajectory array). If the trajectory array was provided directly as `data`, only the trajectory array will be returned.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

See Also

mt_standardize for standardizing mouse-tracking measures per level of other variables.

Examples

```
# Calculate derivatives
mt_example <- mt_derivatives(mt_example)

# Standardize velocity across trajectories
mt_example <- mt_scale_trajectories(mt_example,var_names = "vel")
```

mt_space_normalize *Space normalize trajectories.*

Description

Adjust trajectories so that all trajectories have an identical start and end point. If no end points are provided, trajectories are only adjusted so that they have the same start position. Please note that this function is **deprecated** and that mt_align_start_end should be used instead, which provides the same (and additional) functionality.

Usage

```
mt_space_normalize(data, use = "trajectories",
  save_as = "sn_trajectories", dimensions = c("xpos", "ypos"),
  start = c(0, 0), end = NULL, verbose = FALSE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
save_as	a character string specifying where the resulting trajectory data should be stored.
dimensions	a character vector specifying the dimensions in the trajectory array that should be space-normalized.
start	a numeric vector specifying the start values for each dimension, i.e., the values the first recorded position should have in every trial.
end	a numeric vector specifying the end values for each dimension, i.e., the values the last recorded position should have in every trial. If NULL, trajectories are only adjusted so that they have the same start position.
verbose	logical indicating whether function should report its progress.

Value

A mousetrap data object (see mt_example) with an additional array (by default called sn_trajectories) containing the space-normalized trajectories. All other trajectory dimensions not specified in dimensions (e.g., timestamps) will be kept as is in the resulting trajectory array. If a trajectory array was provided directly as data, only the space-normalized trajectories will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

Dale, R., Kehoe, C., & Spivey, M. J. (2007). Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition, 35*(1), 15-28.

See Also

mt_align_start for aligning the start position of trajectories.

mt_remap_symmetric for remapping trajectories.

Examples

```
## Not run:
mt_example <- mt_space_normalize(mt_example,
  save_as ="sn_trajectories",
  start=c(0,0), end=c(-1,1))

## End(Not run)
```

---

mt_spatialize *Spatialize trajectories.*

Description

Re-represent each trajectory spatially using a constant number of points so that adjacent points on the trajectory become equidistant to each other.

Usage

```
mt_spatialize(data, use = "trajectories", dimensions = c("xpos",
  "ypos"), save_as = "sp_trajectories", n_points = 20)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case use will be ignored).
use	a character string specifying which trajectory data should be used.
dimensions	a character string specifying which trajectory variables should be used. Can be of length 2 or 3 for two-dimensional or three-dimensional data.
save_as	a character string specifying where the resulting trajectory data should be stored.
n_points	an integer or vector of integers specifying the number of points used to represent the spatially rescaled trajectories. If a single integer is provided, the number of points will be constant across trajectories. Alternatively, a vector of integers can provided that specify the number of points for each trajectory individually.

Details

mt_spatialize is used to emphasize the trajectories' shape. Usually, the vast majority of points of a raw or a time-normalized trajectory lie close to the start and end point. mt_spatialize re-distributes these points so that the spatial distribution is uniform across the entire trajectory. mt_spatialize is mainly used to improve the results of clustering (in particular mt_cluster) and visualization.

Value

A mousetrap data object (see mt_example) with an additional array (by default called sp_trajectories) containing the spatialized trajectories. If a trajectory array was provided directly as data, only the spatialized trajectories will be returned.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

Jonas M. B. Haslbeck (<jonas.haslbeck@gmail.com>)

Examples

```
KH2017 <- mt_spatialize(data=KH2017,
  dimensions = c('xpos','ypos'),
  n_points = 20)
```

---

mt_standardize *Standardize mouse-tracking measures per level of other variables.*

Description

Standardize selected mouse-tracking measures across all trials or per level of one or more other variable, and store them in new variables. This function is a thin wrapper around scale_within, focussed on mouse-tracking data stored in a mousetrap data object.

Usage

```
mt_standardize(data, use = "measures", use_variables = NULL,
  within = NULL, prefix = "z_", center = TRUE, scale = TRUE)
```

Arguments

data	a mousetrap data object created using one of the mt_import functions (see mt_example for details).
use	a character string specifying which data should be used. By default points to the measures data.frame created using mt_measures.
use_variables	a vector specifying which variables should be standardized. If unspecified, all variables will be standardized.

within — an optional character string specifying one or more variables in `data[["data"]]`. If specified, all measures will be standardized separately for each level of the variable (or for each combination of levels, if more than one variable is specified).

prefix — a character string that is inserted before each standardized variable. If an empty string is specified, the original variables are replaced.

center — argument passed on to scale.

scale — argument passed on to scale.

Value

A mousetrap data object (see mt_example) including the standardized measures.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

mt_scale_trajectories for standardizing variables in mouse trajectory arrays.

scale_within which is called by `mt_standardize`.

scale for the R base scale function.

Examples

```
mt_example <- mt_measures(mt_example)

# Standardize MAD and AD per subject
mt_example <- mt_standardize(mt_example,
  use_variables=c("MAD", "AD"),
  within="subject_nr", prefix="z_")

# Standardize MAD and AD per subject and Condition
mt_example <- mt_standardize(mt_example,
  use_variables=c("MAD", "AD"),
  within=c("subject_nr", "Condition"),
  prefix="z_")
```

mt_subset *Filter mousetrap data.*

Description

Return a subset of the mousetrap data including only the trial data and corresponding trajectories that meet the conditions specified in the arguments.

Usage

```
mt_subset(data, subset, check = "data")
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details).
`subset`	a logical expression (passed on to subset) indicating the rows to keep. Missing values are taken as `FALSE`.
`check`	a character string specifying which data should be used for checking the subset condition.

Details

`mt_subset` is helpful when trials should be removed from all analyses. By default, `check` is set to "data" meaning that the subset condition is evaluated based on the trial data (stored in `data[["data"]]`). However, it might also be of interest to only include trials based on specific mouse-tracking measures (e.g., all trials with an `MAD` smaller than 200). In this case, `check` needs to be set to the respective name of the data.frame (e.g., "measures").

Note that if specific trials should be removed from all analyses based on a condition known a priori (e.g., practice trials), it is more efficient to use the subset function on the raw data before importing the trajectories using one of the mt_import functions (such as mt_import_mousetrap).

Besides, if trials should only be removed from some analyses or for specific plots, note that other mousetrap functions (e.g., mt_reshape, mt_aggregate, and mt_plot) also allow for subsetting.

Value

A mousetrap data object (see mt_example) with filtered data and trajectories.

Author(s)

Pascal J. Kieslich (`<kieslich@psychologie.uni-mannheim.de>`)

Felix Henninger

See Also

subset for the R base subset function for vectors, matrices, or data.frames.

mt_reshape for information about the subset argument in various other mousetrap functions.

Examples

```
# Subset based on trial data
mt_example_atypical <- mt_subset(mt_example, Condition=="Atypical")

# Subset based on mouse-tracking measure (MAD)
mt_example <- mt_measures(mt_example)
mt_example_mad_sub <- mt_subset(mt_example, MAD<400, check="measures")
```

---

mt_time_normalize *Time normalize trajectories.*

Description

Compute time-normalized trajectories using a constant number of equally sized time steps. Time normalization is performed separately for all specified trajectory dimensions (by default, the x- and y-positions) using linear interpolation based on the timestamps. By default, 101 time steps are used (following Spivey et al., 2005).

Usage

```
mt_time_normalize(data, use = "trajectories",
  save_as = "tn_trajectories", dimensions = c("xpos", "ypos"),
  timestamps = "timestamps", nsteps = 101, verbose = FALSE)
```

Arguments

`data`	a mousetrap data object created using one of the mt_import functions (see mt_example for details). Alternatively, a trajectory array can be provided directly (in this case `use` will be ignored).
`use`	a character string specifying which trajectory data should be used.
`save_as`	a character string specifying where the resulting trajectory data should be stored.
`dimensions`	a character vector specifying the dimensions in the trajectory array that should be time-normalized. If `"all"`, all trajectory dimensions except the timestamps will be time-normalized.
`timestamps`	a character string specifying the trajectory dimension containing the timestamps.
`nsteps`	an integer specifying the number of equally sized time steps.
`verbose`	logical indicating whether function should report its progress.

Details

Time-normalization is often performed if the number of recorded x- and y-positions varies across trajectories, which typically occurs when trajectories vary in their response time. After time-normalization, all trajectories have the same number of recorded positions (which is specified using `nsteps`) and the positions at different relative time points can be compared across trajectories.

For example, time normalized trajectories can be compared across conditions that differed in their overall response time, as the timestamps are now relative to the overall trial duration. This is also helpful for creating average trajectories, which are often used in plots.

Value

A mousetrap data object (see mt_example) with an additional array (by default called `tn_trajectories`) containing the time-normalized trajectories. In this array, another dimension (called `steps`) has been added with increasing integer values indexing the time-normalized position. If a trajectory array was provided directly as `data`, only the time-normalized trajectories will be returned.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

References

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.

See Also

approx for information about the function used for linear interpolation.

mt_resample for resampling trajectories using a constant time interval.

Examples

```
mt_example <- mt_time_normalize(mt_example,
  save_as="tn_trajectories", nsteps=101)
```

---

print.mt_heatmap_raw   *Generic print for class mt_heatmap_raw*

Description

`print.mt_heatmap_raw` shows str.

Usage

```
## S3 method for class 'mt_heatmap_raw'
print(x, ...)
```

Arguments

`x`	an object of class mt_heatmap_raw.
`...`	further arguments passed to or from other methods.

read_mt *Read MouseTracker raw data.*

Description

read_mt reads raw data that was collected using MouseTracker (Freeman & Ambady, 2010) and stored as a file in the ".mt" format. If multiple files should be read into R, read_mt can be used in combination with the read_bulk function from the readbulk package (see Examples). After reading the data into R, mt_import_wide can be used to prepare the trajectory data for analyses using the mousetrap library. The current version of read_mt has been tested with data from MouseTracker Version 2.84 - but please be sure to double-check.

Usage

```
read_mt(file, columns = "all", add_trialid = FALSE,
  add_filename = FALSE)
```

Arguments

file	a character string specifying the filename of the .mt file.
columns	either 'all' or a character vector specifying the to be extracted variables. Defaults to 'all' in which case all existing variables will be extracted.
add_trialid	boolean specifying whether an additional column containing the trial number should be added.
add_filename	boolean specifying whether an additional column containing the file name should be added.

Value

A data.frame with one row per trial. Variables are ordered according to columns, x-coordinates, y-coordinates, and timestamps.

Author(s)

Dirk U. Wulff (<dirk.wulff@gmail.com>)

References

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.

See Also

read_bulk from the readbulk package for reading and combining multiple raw data files.

mt_import_wide to prepare the imported data for analyses in mousetrap.

Examples

```
## Not run:
# Read a single raw data file from MouseTracker
# (stored in the current working directory)
mt_data_raw <- read_mt("example.mt")

# Use read_bulk to read all raw data files ending with ".mt" that are
# stored in the folder "raw_data" (in the current working directory)
library(readbulk)
mt_data_raw <- read_bulk("raw_data", fun=read_mt, extension=".mt")

# Import the data into mousetrap
mt_data <- mt_import_wide(mt_data_raw)

## End(Not run)
```

`scale_within` *Scale and center variables within the levels of another variable.*

Description

`scale_within` centers and/or scales variables in a data.frame (using scale) depending on the levels of one or more other variables. By default, variables are standardized (i.e., centered and scaled). A typical application is the within-subject standardization of variables in a repeated measures design.

Usage

```
scale_within(data, variables = NULL, within = NULL, prefix = "",
  center = TRUE, scale = TRUE)
```

Arguments

`data`	a data.frame.
`variables`	a character string (or vector) specifying one or more variables that scale is applied to. If unspecified, `scale_within` will be applied to all variables in data.
`within`	an optional character string specifying the name of one or more variables in `data`. If specified, scale is applied separately for each of the levels of the variable (or for each combination of levels, if more than one variable is specified). Alternatively, a vector directly containing the level values.
`prefix`	a character string that is inserted before each scaled variable. By default (empty string) the original variables are replaced.
`center`	argument passed on to scale.
`scale`	argument passed on to scale.

Value

The original data.frame including the centered and / or scaled variables.

Author(s)

Pascal J. Kieslich (<kieslich@psychologie.uni-mannheim.de>)

Felix Henninger

See Also

scale for the R base scale function.

mt_standardize for standardizing measures in a mousetrap data object.

Examples

```
ChickWeight_scaled <-  scale_within(
 ChickWeight, variables="weight",
 within="Chick", prefix="z_")
```

Index