Package 'pathviewr'

March 8, 2023

Title Wrangle, Analyze, and Visualize Animal Movement Data

Version 1.1.7

Description Tools to import, clean, and visualize movement data, particularly from motion capture systems such as Optitrack's 'Motive', the Straw Lab's 'Flydra', or from other sources. We provide functions to remove artifacts, standardize tunnel position and tunnel axes, select a region of interest, isolate specific trajectories, fill gaps in trajectory data, and calculate 3D and per-axis velocity. For experiments of visual guidance, we also provide functions that use subject position to estimate perception of visual stimuli.

Maintainer Vikram B. Baliga <vbaliga87@gmail.com>

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Imports R.matlab, data.table (>= 1.12.2), magrittr (>= 1.5), dplyr (>= 1.0.0), stringr (>= 1.4.0), tibble (>= 3.0.1), tidyr (>= 1.1.0), fANCOVA, purrr (>= 0.3.3), ggplot2 (>= 3.4.0), tidyselect (>= 1.1.0), cowplot, lubridate

Suggests knitr, rmarkdown, testthat, anomalize, covr

VignetteBuilder knitr

URL https://github.com/ropensci/pathviewr/, https://docs.ropensci.org/pathviewr/

BugReports https://github.com/ropensci/pathviewr/issues/

NeedsCompilation no

Author Vikram B. Baliga [aut, cre] (<https://orcid.org/0000-0002-9367-8974>), Melissa S. Armstrong [aut] (<https://orcid.org/0000-0002-3059-0094>), Eric R. Press [aut] (<https://orcid.org/0000-0002-1944-3755>), Anne-Sophie Bonnet-Lebrun [rev], Marco Sciaini [rev]

Repository CRAN

Date/Publication 2023-03-08 08:10:05 UTC

R topics documented:

as_viewr . . . 3
bind_viewr_objects . . . 5
calc_min_dist_box . . . 6
calc_min_dist_v . . . 8
clean_by_span . . . 9
clean_viewr . . . 10
clean_viewr_batch . . . 12
deg_2_rad . . . 14
exclude_by_velocity . . . 15
fill_traj_gaps . . . 16
find_curve_elbow . . . 18
gather_tunnel_data . . . 19
get_2d_angle . . . 20
get_3d_angle . . . 21
get_3d_cross_prod . . . 22
get_dist_point_line . . . 23
get_full_trajectories . . . 24
get_header_viewr . . . 25
get_sf . . . 26
get_traj_velocities . . . 28
get_velocity . . . 29
get_vis_angle . . . 31
import_and_clean_batch . . . 33
import_and_clean_viewr . . . 35
import_batch . . . 38
insert_treatments . . . 40
plot_by_subject . . . 43
plot_viewr_trajectories . . . 44
quick_separate_trajectories . . . 45
rad_2_deg . . . 47
read_flydra_mat . . . 48
read_motive_csv . . . 49
redefine_tunnel_center . . . 50
relabel_viewr_axes . . . 52
remove_duplicate_frames . . . 54
remove_vel_anomalies . . . 55
rename_viewr_characters . . . 56
rescale_tunnel_data . . . 57
rm_by_trajnum . . . 58
rotate_tunnel . . . 60
section_tunnel_by . . . 62
select_x_percent . . . 63
separate_trajectories . . . 65
set_traj_frametime . . . 67
standardize_tunnel . . . 68
trim_tunnel_outliers . . . 69

visualize_frame_gap_choice 71

Index **73**

`as_viewr` *Convert data from another format into a viewr object*

Description

Should you have data from a non-Motive, non-Flydra source, this function can be used to ensure your data are put into the right format to work with other pathviewr functions.

Usage

```
as_viewr(
  obj_name,
  frame_rate = 100,
  frame_col,
  time_col,
  subject_col,
  position_length_col,
  position_width_col,
  position_height_col,
  include_rotation = FALSE,
  rotation_real_col,
  rotation_length_col,
  rotation_width_col,
  rotation_height_col
)
```

Arguments

`obj_name`	A tibble or data frame containing movement trajectories
`frame_rate`	Must be a single numeric value indicating capture frame rate in frames per second.
`frame_col`	Column number of obj_name that contains frame numbers
`time_col`	Column number of obj_name that contains time (must be in seconds)
`subject_col`	Column number of obj_name that contains subject name(s)
`position_length_col`	Column number of obj_name that contains length-axis position values
`position_width_col`	Column number of obj_name that contains width-axis position values
`position_height_col`	Column number of obj_name that contains height-axis position values
`include_rotation`	Are rotation data included? Defaults to FALSE

`rotation_real_col`
: Column number of obj_name that contains the "real" axis of quaternion rotation data

`rotation_length_col`
: Column number of obj_name that contains the length axis of quaternion rotation data

`rotation_width_col`
: Column number of obj_name that contains the width axis of quaternion rotation data

`rotation_height_col`
: Column number of obj_name that contains the height axis of quaternion rotation data

Value

A tibble that is organized to be compliant with other `pathviewr` functions and that contains the attributes `pathviewr_steps` with entries set to `c("viewr", "renamed_tunnel", "gathered_tunnel")`

Author(s)

Vikram B. Baliga

See Also

Other data import functions: `import_and_clean_batch()`, `import_batch()`, `read_flydra_mat()`, `read_motive_csv()`

Examples

```
## Create a dummy data frame with simulated (nonsense) data
df <- data.frame(frame = seq(1, 100, by = 1),
                 time_sec = seq(0, by = 0.01, length.out = 100),
                 subject = "birdie_sanders",
                 z = rnorm(100),
                 x = rnorm(100),
                 y = rnorm(100))

## Use as_viewr() to convert it into a viewr object
test <-
  as_viewr(
    df,
    frame_rate = 100,
    frame_col = 1,
    time_col = 2,
    subject_col = 3,
    position_length_col = 5,
    position_width_col = 6,
    position_height_col = 4
  )
```

`bind_viewr_objects` *Bind viewr objects*

Description

Combine a list of multiple viewr objects into a single viewr object

Usage

```
bind_viewr_objects(obj_list)
```

Arguments

`obj_list`	A list of viewr objects

Value

A single viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) that combines all the rows of the source viewr objects in `obj_list`. Metadata may not necessarily be retained and therefore `attributes` should be used with caution.

Author(s)

Vikram B. Baliga

See Also

Other batch functions: `clean_viewr_batch()`, `import_and_clean_batch()`, `import_batch()`

Examples

```
## Since we only have one example file of each type provided
## in pathviewr, we will simply import the same example multiple
## times to simulate batch importing. Replace the contents of
## the following list with your own list of files to be imported.

## Make a list of the same example file 3x
import_list <-
  c(rep(
    system.file("extdata", "pathviewr_motive_example_data.csv",
                package = 'pathviewr'),
    3
  ))

## Batch import
motive_batch_imports <-
  import_batch(import_list,
               import_method = "motive",
               import_messaging = TRUE)
```

```
## Batch cleaning of these imported files
## via clean_viewr_batch()
motive_batch_cleaned <-
  clean_viewr_batch(
    file_announce = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Alternatively, use import_and_clean_batch() to
## combine import with cleaning on a batch of files
motive_batch_import_and_clean <-
  import_and_clean_batch(
    import_list,
    import_method = "motive",
    import_messaging = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Each of these lists of objects can be bound into
## one viewr object (i.e. one tibble) via
## bind_viewr_objects()
motive_bound_one <-
  bind_viewr_objects(motive_batch_cleaned)

motive_bound_two <-
  bind_viewr_objects(motive_batch_import_and_clean)

## Either route results in the same object ultimately:
identical(motive_bound_one, motive_bound_two)
```

`calc_min_dist_box` *Calculate minimum distance to lateral and end walls in a box-shaped experimental tunnel*

Description

Calculate minimum distance to lateral and end walls in a box-shaped experimental tunnel

Usage

```
calc_min_dist_box(obj_name)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that include "viewr" and treatments_added.

Details

calc_min_dist_box() assumes the subject locomotes facing forward, therefore min_dist_end represents the minimum distance between the subject and the end wall to which it is moving towards. All outputs are in meters.

Value

A tibble or data.frame with added variables for min_dist_pos, min_dist_neg, and min_dist_end,.

Author(s)

Eric R. Press

See Also

Other visual perception functions: get_sf(), get_vis_angle()

Examples

```
## Import sample data from package
 flydra_data <-
 read_flydra_mat(system.file("extdata", "pathviewr_flydra_example_data.mat",
                             package = 'pathviewr'),
                             subject_name = "birdie_sanders")

  ## Process data up to and including insert_treatments()
 flydra_data_full <-
  flydra_data %>%
  redefine_tunnel_center(length_method = "middle",
                         height_method = "user-defined",
                         height_zero = 1.44) %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95) %>%
  insert_treatments(tunnel_config = "box",
                    tunnel_length = 3,
                    tunnel_width = 1,
                    stim_param_lat_pos = 0.1,
                    stim_param_lat_neg = 0.1,
                    stim_param_end_pos = 0.3,
                    stim_param_end_neg = 0.3,
                    treatment = "lat10_end_30") %>%

  ## Now calculate the minimum distances to each wall
  calc_min_dist_box()
```

```
## See 3 new variables for calculations to lateral and end walls
names(flydra_data_full)
```

`calc_min_dist_v` *Calculate minimum distance to lateral and end walls in a V-shaped experimental tunnel*

Description

Calculate minimum distance to lateral and end walls in a V-shaped experimental tunnel

Usage

```
calc_min_dist_v(obj_name, simplify_output = TRUE)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"` and `treatments_added`.
`simplify_output`	If TRUE, the returned object includes only the minimum distance between the subject and the lateral/end walls. If FALSE, the returned object includes all variables internal to the calculation.

Details

For tunnels in which `vertex_angle` is >90 degree, bound_pos and bound_neg represent a planes orthogonal to the lateral walls and are used to modify `min_dist_pos` and `min_dist_neg` calculations to prevent erroneous outputs. `calc_min_dist_v()` assumes the subject locomotes facing forward, therefore `min_dist_end` represents the minimum distance between the subject and the end wall to which it is moving towards All outputs are in meters.

Value

A tibble or data.frame with added variables for `height_2_vertex`, `height_2_screen`, `width_2_screen_pos`, `width_2_screen_neg`, `min_dist_pos`, `min_dist_neg`, `min_dist_end`, bound_pos, and bound_neg.

Author(s)

Eric R. Press

See Also

Other mathematical functions: `deg_2_rad()`, `find_curve_elbow()`, `get_2d_angle()`, `get_3d_angle()`, `get_3d_cross_prod()`, `get_dist_point_line()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
 ## Import sample data from package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

 ## Process data up to and including insert_treatments()
motive_data_full <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel() %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95) %>%
  insert_treatments(tunnel_config = "v",
                    perch_2_vertex = 0.4,
                    vertex_angle = 90,
                    tunnel_length = 2,
                    stim_param_lat_pos = 0.1,
                    stim_param_lat_neg = 0.1,
                    stim_param_end_pos = 0.3,
                    stim_param_end_neg = 0.3,
                    treatment = "lat10_end_30") %>%

 ## Now calculate the minimum distances to each wall
  calc_min_dist_v(simplify_output = TRUE)

  ## See 3 new variables for calculations to lateral and end walls
  names(motive_data_full)
```

`clean_by_span` *Remove file_sub_traj entries that do not span the full region of interest*

Description

Remove file_sub_traj entries that do not span the full region of interest

Usage

```
clean_by_span(
  obj_name,
  axis = "position_length",
  min_value = NULL,
  max_value = NULL,
  tolerance = 0.1
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
axis	Along which axis should restrictions be enforced?
min_value	Minimum coordinate value; setting this to NULL will auto-compute the best value
max_value	Maximum coordinate; setting this to NULL will auto-compute the best value
tolerance	As a proporiton of axis value

Value

A viewr object (tibble or data.frame with attribute pathviewr_steps. Trajectories that do not span the full region of interest have been removed; trajectory identities (file_sub_traj) have not been changed.

Author(s)

Vikram B. Baliga

See Also

Other utility functions: insert_treatments(), remove_duplicate_frames(), remove_vel_anomalies(), set_traj_frametime()

---

clean_viewr *All-in-one function to clean imported objects*

Description

For an imported viewr object, run through the cleaning pipeline as desired

Usage

```
clean_viewr(
  obj_name,
  relabel_viewr_axes = TRUE,
  gather_tunnel_data = TRUE,
  trim_tunnel_outliers = TRUE,
  standardization_option = "rotate_tunnel",
  get_velocity = TRUE,
  select_x_percent = TRUE,
  rename_viewr_characters = FALSE,
  separate_trajectories = TRUE,
  get_full_trajectories = TRUE,
  fill_traj_gaps = FALSE,
  ...
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`relabel_viewr_axes`	default TRUE, should axes be relabeled?
`gather_tunnel_data`	default TRUE, should tunnel data be gathered?
`trim_tunnel_outliers`	default TRUE, outliers be trimmed?
`standardization_option`	default "rotate_tunnel"; which standardization option should be used? See Details for more.
`get_velocity`	default TRUE, should velocity be computed?
`select_x_percent`	default TRUE, should a region of interest be selected?
`rename_viewr_characters`	default FALSE, should subjects be renamed?
`separate_trajectories`	default TRUE, should trajectories be defined?
`get_full_trajectories`	default TRUE, should only full trajectories be retained?
`fill_traj_gaps`	default FALSE, should gaps in trajectories be filled?
`...`	Additional arguments passed to any of the corresponding functions

Details

Each argument corresponds to a standalone function in `pathviewr`. E.g. the parameter `relabel_viewr_axes` allows a user to choose whether `pathviewr::relabel_viewr_axes()` is run internally. Should the user desire to use any non-default parameter values for any functions included here, they should be supplied to this function as additional arguments formatted exactly as they would appear in their corresponding function(s). E.g. if the "autodetect" feature in `pathviewr::separate_trajectories()` is desired, add an argument `max_frame_gap = "autodetect"` to the arguments supplied to this function.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) that has passed through several `pathviewr` functions as desired by the user, resulting in data that have been cleaned and ready for analyses.

Author(s)

Vikram B. Baliga

See Also

Other all in one functions: `import_and_clean_viewr()`

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

motive_full <-
  motive_data %>%
  clean_viewr(desired_percent = 50,
              max_frame_gap = "autodetect",
              span = 0.95)

## Alternatively, used the import_and_clean_viewr()
## function to combine these steps
motive_import_and_clean <-
  import_and_clean_viewr(
    file_name = system.file("extdata", "pathviewr_motive_example_data.csv",
                            package = 'pathviewr'),
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )
```

---

clean_viewr_batch *Batch clean viewr files*

Description

For a list of viewr objects, run through the pipeline (from relabel axes up through get full trajectories, as desired) via clean_viewr()

Usage

```
clean_viewr_batch(obj_list, file_announce = FALSE, ...)
```

Arguments

obj_list	A list of viewr objects (i.e. a list of tibbles that each have attribute pathviewr_steps that includes "viewr")
file_announce	Should the function report each time a file is processed? Default FALSE; if TRUE, a message will appear in the console each time a file has been cleaned successfully.
...	Arguments to be passed in that specify how this function should clean files.

Details

viewr objects should be in a list, e.g. the object generated by `import_batch()`.

See `clean_viewr()` for details of how cleaning steps are handled and/or refer to the corresponding cleaning functions themselves.

Value

A list of viewr objects (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) that have been passed through the corresponding cleaning functions.

Author(s)

Vikram B. Baliga

See Also

Other batch functions: `bind_viewr_objects()`, `import_and_clean_batch()`, `import_batch()`

Examples

```
## Since we only have one example file of each type provided
## in pathviewr, we will simply import the same example multiple
## times to simulate batch importing. Replace the contents of
## the following list with your own list of files to be imported.

## Make a list of the same example file 3x
import_list <-
  c(rep(
    system.file("extdata", "pathviewr_motive_example_data.csv",
                package = 'pathviewr'),
    3
  ))

## Batch import
motive_batch_imports <-
  import_batch(import_list,
               import_method = "motive",
               import_messaging = TRUE)

## Batch cleaning of these imported files
## via clean_viewr_batch()
motive_batch_cleaned <-
  clean_viewr_batch(
    file_announce = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Alternatively, use import_and_clean_batch() to
```

```
## combine import with cleaning on a batch of files
motive_batch_import_and_clean <-
  import_and_clean_batch(
    import_list,
    import_method = "motive",
    import_messaging = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Each of these lists of objects can be bound into
## one viewr object (i.e. one tibble) via
## bind_viewr_objects()
motive_bound_one <-
  bind_viewr_objects(motive_batch_cleaned)

motive_bound_two <-
  bind_viewr_objects(motive_batch_import_and_clean)

## Either route results in the same object ultimately:
identical(motive_bound_one, motive_bound_two)
```

`deg_2_rad` *Convert degrees to radians*

Description

Convert degrees to radians

Usage

```
deg_2_rad(deg)
```

Arguments

`deg`	Degrees (a numeric of any length >= 1)

Value

The angle(s) in radians (as a numeric vector of the same length)

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: `calc_min_dist_v()`, `find_curve_elbow()`, `get_2d_angle()`, `get_3d_angle()`, `get_3d_cross_prod()`, `get_dist_point_line()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
## One input
deg_2_rad(90)

## Multiple inputs
deg_2_rad(c(5, 10, 15, 20))
```

---

exclude_by_velocity *Remove trajectories entirely, based on velocity thresholds*

Description

Remove trajectories from a viewr object that contain instances of velocity known to be spurious.

Usage

```
exclude_by_velocity(obj_name, vel_min = NULL, vel_max = NULL)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`vel_min`	Default `NULL`. If a numeric is entered, trajectories that have at least one observation with velocity less than `vel_min` are removed.
`vel_max`	Default `NULL`. If a numeric is entered, trajectories that have at least one observation with velocity greater than `vel_max` are removed.

Value

A new viewr object that is identical to the input object but now excludes any trajectories that contain observations with velocity less than `vel_min` (if specified) and/or velocity greater than `vel_max` (if specified)

Author(s)

Vikram B. Baliga

Examples

```
## Import and clean the example Motive data
motive_import_and_clean <-
  import_and_clean_viewr(
    file_name = system.file("extdata", "pathviewr_motive_example_data.csv",
                            package = 'pathviewr'),
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## See the distribution of velocities
hist(motive_import_and_clean$velocity)

## Let's remove any trajectories that contain
## velocity < 2
motive_vel_filtered <-
  motive_import_and_clean %>%
  exclude_by_velocity(vel_min = 2)

## See how the distribution of velocities has changed
hist(motive_vel_filtered$velocity)
```

`fill_traj_gaps` *Interpolate gaps within trajectories*

Description

Use LOESS smoothing to fill in gaps of missing data within trajectories in a viewr object

Usage

```
fill_traj_gaps(
  obj_name,
  loess_degree = 1,
  loess_criterion = c("aicc", "gcv"),
  loess_family = c("gaussian", "symmetric"),
  loess_user_span = NULL
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`. Trajectories must be predefined (i.e. via `separate_trajectories()`).
`loess_degree`	See "degree" argument of fANCOVA::loess.as()
`loess_criterion`	See "criterion" argument of fANCOVA::loess.as()
`loess_family`	See "family" argument of fANCOVA::loess.as()

loess_user_span
: See "user.span" argument of fANCOVA::loess.as()

Details

It is strongly recommended that the input viewr object be "cleaned" via `select_x_percent()` -> `separate_trajectories()` -> `get_full_trajectories()` prior to using this function. Doing so will ensure that only trajectories with minor gaps will be used in your analyses. This function will then enable you to interpolate missing data in those minor gaps.

Interpolation is handled by first fitting a series of LOESS regressions (via `fANCOVA::loess.as()`). In each regression, a position axis (e.g. `position_length`) is regressed against `frame` (`frame` is x-axis). From that relationship, values of missing position data are determined and then inserted into the original data set.

See loess.as for further details on parameters.

Value

A viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"` that now includes new observations (rows) as a result of interpolation to fill in missing data. A new column `gaps_filled` is added to the data to indicate original data ("No") vs data that have been inserted to fill gaps ("Yes").

Author(s)

Vikram B. Baliga

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean, isolate, and label trajectories
motive_full <-
  motive_data %>%
  clean_viewr(desired_percent = 50,
              max_frame_gap = "autodetect",
              span = 0.95)

## Interpolate missing data via this function
motive_filling <-
 motive_full %>%
 fill_traj_gaps()

## plot all trajectories (before)
plot_viewr_trajectories(motive_full, multi_plot = TRUE)
## plot all trajectories(after)
plot_viewr_trajectories(motive_filling, multi_plot = TRUE)
```

`find_curve_elbow` *Find the "elbow" of a curve.*

Description

For bivariate data that show monotonic decreases (e.g. plots of trajectory count vs. frame gap allowed, or scree plots from PCAs), this function will find the "elbow" point. This is done by drawing an (imaginary) line between the first observation and the final observation. Then, the distance between that line and each observation is calculated. The "elbow" of the curve is the observation that maximizes this distance.

Usage

```
find_curve_elbow(data_frame, export_type = "row_num", plot_curve = FALSE)
```

Arguments

`data_frame`	A two-column data frame (numeric entries only), ordered x-axis first, y-axis second.
`export_type`	If "row_num" (the default), the row number of the elbow point is returned. If anything else, the entire row of the original data frame is returned.
`plot_curve`	Default FALSE; should the curve be plotted?

Value

If `export_type` is `row_num` the row number of the elbow point is returned. If anything else is used for that argument, the entire row of the original data frame on which the "elbow" is located is returned. If `plot_curve` is `TRUE`, the curve is plotted along with a vertical line drawn at the computed elbow point.

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: `calc_min_dist_v()`, `deg_2_rad()`, `get_2d_angle()`, `get_3d_angle()`, `get_3d_cross_prod()`, `get_dist_point_line()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
df <- data.frame(x = seq(1:10),
                 y = 1/seq(1:10))
plot(df)
find_curve_elbow(df, plot_curve = TRUE)
```

gather_tunnel_data *Gather data columns into key-value pairs*

Description

Reformat `viewr` data into a "tidy" format so that every row corresponds to the position (and potentially rotation) of a single subject during an observed frame and time.

Usage

```
gather_tunnel_data(obj_name, NA_drop = TRUE, ...)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`NA_drop`	Should rows with NAs be dropped? Defaults to `TRUE`
`...`	Additional arguments that can be passed to other `pathviewr` functions such as `relabel_viewr_axes()` or `read_motive_csv()`

Details

The tibble or data.frame that is fed in must have variables that have subject names and axis names separated by underscores. Axis names must be one of the following: `position_length`, `position_width`, or `position_height`. Each of these three dimensions must be present in the data. Collectively, this means that names like `bird01_position_length` or `larry_position_height` are acceptable, but `bird01_x` or `bird01_length` are not.

Value

A tibble in "tidy" format which is formatted to have every row correspond to the position (and potentially rotation) of a single subject during an observed frame and time. Subjects' names are automatically parsed from original variable names (e.g. subject1_rotation_width extracts "subject1" as the subject name) and stored in a `Subjects` column in the returned tibble.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: `get_full_trajectories()`, `quick_separate_trajectories()`, `redefine_tunnel_center()`, `relabel_viewr_axes()`, `rename_viewr_characters()`, `rotate_tunnel()`, `select_x_percent()`, `separate_trajectories()`, `standardize_tunnel()`, `trim_tunnel_outliers()`, `visualize_frame_gap_choice()`

Examples

```
library(pathviewr)

## Import the Motive example data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## First use relabel_viewr_axes() to rename these variables using _length,
## _width, and _height instead
motive_data_relabeled <- relabel_viewr_axes(motive_data)

## Now use gather_tunnel_data() to gather colums into tidy format
motive_data_gathered <- gather_tunnel_data(motive_data_relabeled)

## Column names reflect the way in which data were reformatted:
names(motive_data_gathered)
```

`get_2d_angle` *Compute an angle in 2D space*

Description

Compute an angle in 2D space

Usage

```
get_2d_angle(x1, y1, x2, y2, x3, y3)
```

Arguments

x1	x-coordinate of first point
y1	y-coordinate of first point
x2	x-coordinate of second point (vertex)
y2	y-coordinate of second point (vertex)
x3	x-coordinate of third point
y3	y-coordinate of third point

Details

Everything supplied to arguments must be numeric values or vectors of numeric values. The second point (x2, y2) is treated as the vertex, and the angle between the three points in 2D space is computed.

Value

A numeric vector that provides the angular measurement in degrees.

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: `calc_min_dist_v()`, `deg_2_rad()`, `find_curve_elbow()`, `get_3d_angle()`, `get_3d_cross_prod()`, `get_dist_point_line()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
get_2d_angle(
  0, 1,
  0, 0,
  1, 0)
```

---

get_3d_angle *Compute an angle in 3D space*

Description

Compute an angle in 3D space

Usage

```
get_3d_angle(x1, y1, z1, x2, y2, z2, x3, y3, z3)
```

Arguments

x1	x-coordinate of first point
y1	y-coordinate of first point
z1	z-coordinate of first point
x2	x-coordinate of second point (vertex)
y2	y-coordinate of second point (vertex)
z2	y-coordinate of second point (vertex)
x3	x-coordinate of third point
y3	y-coordinate of third point
z3	z-coordinate of third point

Details

Everything supplied to arguments must be numeric values or vectors of numeric values. The second point (x2, y2, z2) is treated as the vertex, and the angle between the three points in 3D space is computed.

Value

A numeric vector that provides the angular measurement in degrees.

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: `calc_min_dist_v()`, `deg_2_rad()`, `find_curve_elbow()`, `get_2d_angle()`, `get_3d_cross_prod()`, `get_dist_point_line()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
get_3d_angle(
  0, 1, 0,
  0, 0, 0,
  1, 0, 0)
```

---

get_3d_cross_prod *Compute the cross product of two 3D vectors*

Description

Compute the cross product of two 3D vectors

Usage

```
get_3d_cross_prod(v1, v2)
```

Arguments

v1	First vector, as c(x,y,z)
v2	Second vector, as c(x,y,z)

Value

A vector of length 3 that describes the cross-product

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: `calc_min_dist_v()`, `deg_2_rad()`, `find_curve_elbow()`, `get_2d_angle()`, `get_3d_angle()`, `get_dist_point_line()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
v1 <- c(1, 1, 3)
v2 <- c(3, 1, 3)
get_3d_cross_prod(v1, v2)
```

`get_dist_point_line` *Compute distance between a point and a line*

Description

Compute distance between a point and a line

Usage

```
get_dist_point_line(point, line_coord1, line_coord2)
```

Arguments

`point`	2D or 3D coordinates of the point as c(x,y) or c(x,y,z)
`line_coord1`	2D or 3D coordinates of one point on the line as c(x,y) or c(x,y,z)
`line_coord2`	2D or 3D coordinates of a second point on the line as c(x,y) or c(x,y,z)

Details

The function accepts 2D coordinates or 3D coordinates, but note that the dimensions of all supplied arguments must match; all coordinates must be 2D or they all must be 3D.

Value

A numeric vector of length 1 that provides the euclidean distance between the point and the line.

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: `calc_min_dist_v()`, `deg_2_rad()`, `find_curve_elbow()`, `get_2d_angle()`, `get_3d_angle()`, `get_3d_cross_prod()`, `get_traj_velocities()`, `get_velocity()`, `rad_2_deg()`

Examples

```
## 2D case
get_dist_point_line(
  point = c(0, 0),
  line_coord1 = c(1, 0),
  line_coord2 = c(1, 5)
)

## 3D case
get_dist_point_line(
  point = c(0, 0, 0),
  line_coord1 = c(1, 0, 0),
  line_coord2 = c(1, 5, 0)
)
```

---

get_full_trajectories *Retain trajectories that span a selected region of interest*

Description

Specify a minimum span of the selected region of interest and then keep trajectories that are wider than that span and go from one end to the other of the region.

Usage

```
get_full_trajectories(obj_name, span = 0.8, ...)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
span	Span to use; must be numeric and between 0 and 1
...	Additional arguments passed to/from other pathviewr functions

Details

Because trajectories may not have observations exactly at the beginning or the end of the region of interest, it may be necessary to allow trajectories to be slightly shorter than the range of the selected region of interest. The span parameter of this function handles this. By supplying a numeric proportion from 0 to 1, a user may allow trajectories to span that proportion of the selected region. For example, setting span = 0.95 will keep all trajectories that span 95% of the length of the selected region of interest. Setting span = 1 (not recommended) will strictly keep trajectories that start and end at the exact cut-offs of the selected region of interest. For these reasons, spans of 0.99 to 0.95 are generally recommended.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) in which only trajectories that span the region of interest are retained. Data are labeled by direction (either "leftwards" or "rightwards") with respect to their starting and ending `position_length` values in the `direction` column.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: `gather_tunnel_data()`, `quick_separate_trajectories()`, `redefine_tunnel_center()` `relabel_viewr_axes()`, `rename_viewr_characters()`, `rotate_tunnel()`, `select_x_percent()`, `separate_trajectories()`, `standardize_tunnel()`, `trim_tunnel_outliers()`, `visualize_frame_gap_choice()`

Other functions that define or clean trajectories: `quick_separate_trajectories()`, `separate_trajectories()`, `visualize_frame_gap_choice()`

Examples

```
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "separate" step before running select_x_percent().
motive_separated <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel() %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect",
                        frame_rate_proportion = 0.1)

## Now retain only the "full" trajectories that span
## across 0.95 of the range of position_length
motive_full <-
  motive_separated %>%
  get_full_trajectories(span = 0.95)
```

---

get_header_viewr *Extract header info from imported viewr object*

Description

A function to quickly return the information stored in the header of the original data file imported via `pathviewr`'s `read_` functions.

Usage

```
get_header_viewr(obj_name, ...)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr" pathviewr_steps
...	Additional arguments that may be passed to other pathviewr functions

Value

The value of the header attribute, or NULL if no exact match is found and no or more than one partial match is found.

Author(s)

Vikram B. Baliga

Examples

```
library(pathviewr)

## Import the Motive example data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Now display the Header information
get_header_viewr(motive_data)
```

---

get_sf *Estimate the spatial frequency of visual stimuli from the subject's perspective in an experimental tunnel.*

Description

Estimate the spatial frequency of visual stimuli from the subject's perspective in an experimental tunnel.

Usage

```
get_sf(obj_name)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr" and vis_angles_calculated.

Details

`get_sf()` assumes the following:

- The subject's gaze is fixed at the point on the either side of the tunnel that minimizes the distance to visual stimuli and therefore maximizes visual angles.
- The subject's head is facing parallel to the length axis of the tunnel. Visual perception functions in future versions of pathviewr will integrate head orientation coordinates. Spatial frequency is reported in cycles/degree and is the inverse of visual angle (degrees/cycle).

Value

A tibble or data.frame with added variables for `sf_pos`, `sf_neg`, and `sf_end`. angle.

Author(s)

Eric R. Press

See Also

Other visual perception functions: `calc_min_dist_box()`, `get_vis_angle()`

Examples

```
 ## Import sample data from package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))
flydra_data <-
  read_flydra_mat(system.file("extdata", "pathviewr_flydra_example_data.mat",
                              package = 'pathviewr'),
                              subject_name = "birdie_sanders")

 ## Process data up to and including get_vis_angle()
motive_data_full <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel() %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95) %>%
  insert_treatments(tunnel_config = "v",
                    perch_2_vertex = 0.4,
                    vertex_angle = 90,
                    tunnel_length = 2,
                    stim_param_lat_pos = 0.1,
                    stim_param_lat_neg = 0.1,
                    stim_param_end_pos = 0.3,
                    stim_param_end_neg = 0.3,
                    treatment = "lat10_end_30") %>%
```

```
  calc_min_dist_v(simplify_output = TRUE) %>%
  get_vis_angle() %>%

  ## Now calculate the spatial frequencies
  get_sf()

  flydra_data_full <-
  flydra_data %>%
  redefine_tunnel_center(length_method = "middle",
                         height_method = "user-defined",
                         height_zero = 1.44) %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95) %>%
  insert_treatments(tunnel_config = "box",
                    tunnel_length = 3,
                    tunnel_width = 1,
                    stim_param_lat_pos = 0.1,
                    stim_param_lat_neg = 0.1,
                    stim_param_end_pos = 0.3,
                    stim_param_end_neg = 0.3,
                    treatment = "lat10_end_30") %>%
  calc_min_dist_box() %>%
  get_vis_angle() %>%

  ## Now calculate the spatial frequencies
  get_sf()
```

`get_traj_velocities` *Recompute trajectory-specific velocities*

Description

Recompute trajectory-specific velocities

Usage

```
get_traj_velocities(
  obj_name,
  time_col = "time_sec",
  length_col = "position_length",
  width_col = "position_width",
  height_col = "position_height",
  set_init_vel_zero = FALSE,
  velocity_min = NA,
  velocity_max = NA
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
time_col	Name of the column containing time
length_col	Name of the column containing length dimension
width_col	Name of the column containing width dimension
height_col	Name of the column containing height dimension
set_init_vel_zero	Should the first value be zero or can it be a duplicate of the second velocity value? Defaults to FALSE.
velocity_min	Should data below a certain velocity be filtered out of the object? If so, enter a numeric. If not, keep NA.
velocity_max	Should data above a certain velocity be filtered out of the object? If so, enter a numeric. If not, keep NA.

Details

Instantaneous velocity is not truly "instantaneous" but rather is approximated as the change in distance divided by change in time from one observation (row) to the previous observation (row). Each component of velocity is computed (i.e. per axis) along with the overall velocity of the subject.

Value

If add_to_viewr is TRUE, additional columns are appended to the input viewr object. If FALSE, a standalone tibble is created. Either way, an "instantaneous" velocity is computed as the difference in position divided by the difference in time as each successive row is encountered. Additionally, velocities along each of the three position axes are computed and provided as additional columns.

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: calc_min_dist_v(), deg_2_rad(), find_curve_elbow(), get_2d_angle(), get_3d_angle(), get_3d_cross_prod(), get_dist_point_line(), get_velocity(), rad_2_deg()

---

get_velocity *Get instantaneous velocity for subjects*

Description

Velocity (both overall and per-axis) is computed for each row in the data (see Details)

Usage

```
get_velocity(
  obj_name,
  time_col = "time_sec",
  length_col = "position_length",
  width_col = "position_width",
  height_col = "position_height",
  add_to_viewr = TRUE,
  velocity_min = NA,
  velocity_max = NA,
  ...
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
time_col	Name of the column containing time
length_col	Name of the column containing length dimension
width_col	Name of the column containing width dimension
height_col	Name of the column containing height dimension
add_to_viewr	Default TRUE; should velocity data be added as new columns or should this function create a new simpler object?
velocity_min	Should data below a certain velocity be filtered out of the object? If so, enter a numeric. If not, keep NA.
velocity_max	Should data above a certain velocity be filtered out of the object? If so, enter a numeric. If not, keep NA.
...	Additional arguments passed to or from other pathviewr functions.

Details

Instantaneous velocity is not truly "instantaneous" but rather is approximated as the change in distance divided by change in time from one observation (row) to the previous observation (row). Each component of velocity is computed (i.e. per axis) along with the overall velocity of the subject.

Value

If add_to_viewr is TRUE, additional columns are appended to the input viewr object. If FALSE, a standalone tibble is created. Either way, an "instantaneous" velocity is computed as the difference in position divided by the difference in time as each successive row is encountered. Additionally, velocities along each of the three position axes are computed and provided as additional columns.

Author(s)

Vikram B. Baliga and Melissa S. Armstrong

See Also

Other mathematical functions: calc_min_dist_v(), deg_2_rad(), find_curve_elbow(), get_2d_angle(), get_3d_angle(), get_3d_cross_prod(), get_dist_point_line(), get_traj_velocities(), rad_2_deg()

Examples

```
## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "standarization" step before running get_velocity().
 motive_cleaned <-
   motive_data %>%
   relabel_viewr_axes() %>%
   gather_tunnel_data() %>%
   trim_tunnel_outliers() %>%
   rotate_tunnel()

## Now compute velocity and add as columns
 motive_velocity_added <-
   motive_cleaned %>%
   get_velocity(add_to_viewr = TRUE)

## Or set add_to_viewr to FALSE for a standalone object
 motive_velocity_standalone <-
   motive_cleaned %>%
   get_velocity(add_to_viewr = TRUE)
```

---

get_vis_angle *Estimate visual angles from a subject's perspective in an experimental tunnel*

Description

Estimate visual angles from a subject's perspective in an experimental tunnel

Usage

```
get_vis_angle(obj_name)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attributes pathviewr_steps that include "viewr" and min_dist_calculated.

Details

`get_vis_angle()` assumes the following:

- The subject's gaze is fixed at the point on the either side of the tunnel that minimizes the distance to visual stimuli and therefore maximizes visual angles.
- The subject's head is facing parallel to the length axis of the tunnel. Visual perception functions in future versions of pathviewr will integrate head orientation coordinates. Angles are reported in radians/cycle (`vis_angle_pos_rad`) and degrees/cycle (`vis_angle_pos_deg`).

Value

A tibble or data.frame with added variables for `vis_angle_pos_rad`, `vis_angle_pos_deg`, `vis_angle_neg_rad`, `vos_angle_neg_deg`, `vis_angle_end_rad`, and `vis_angle_end_deg`.

Author(s)

Eric R. Press

See Also

Other visual perception functions: `calc_min_dist_box()`, `get_sf()`

Examples

```
 ## Import sample data from package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))
flydra_data <-
  read_flydra_mat(system.file("extdata", "pathviewr_flydra_example_data.mat",
                              package = 'pathviewr'),
                              subject_name = "birdie_sanders")

 ## Process data up to and including get_min_dist()
motive_data_full <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel() %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95) %>%
  insert_treatments(tunnel_config = "v",
                    perch_2_vertex = 0.4,
                    vertex_angle = 90,
                    tunnel_length = 2,
                    stim_param_lat_pos = 0.1,
                    stim_param_lat_neg = 0.1,
                    stim_param_end_pos = 0.3,
                    stim_param_end_neg = 0.3,
```

```
                  treatment = "lat10_end_30") %>%
  calc_min_dist_v(simplify_output = TRUE) %>%

  ## Now calculate the visual angles
  get_vis_angle()

  flydra_data_full <-
  flydra_data %>%
  redefine_tunnel_center(length_method = "middle",
                         height_method = "user-defined",
                         height_zero = 1.44) %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95) %>%
  insert_treatments(tunnel_config = "box",
                  tunnel_length = 3,
                  tunnel_width = 1,
                  stim_param_lat_pos = 0.1,
                  stim_param_lat_neg = 0.1,
                  stim_param_end_pos = 0.3,
                  stim_param_end_neg = 0.3,
                  treatment = "lat10_end_30") %>%
  calc_min_dist_box() %>%

   ## Now calculate the visual angles
  get_vis_angle()
```

`import_and_clean_batch` *Batch import and clean files*

Description

Like `clean_viewr_batch()`, but with import as the first step too

Usage

```
import_and_clean_batch(
  file_path_list,
  import_method = c("flydra", "motive"),
  file_id = NA,
  subject_name = NULL,
  frame_rate = NULL,
  simplify_marker_naming = TRUE,
  import_messaging = FALSE,
  ...
)
```

Arguments

file_path_list	A list of file paths leading to files to be imported.
import_method	Either "flydra" or "motive"
file_id	(Optional) identifier for this file. If not supplied, this defaults to basename(file_name).
subject_name	For Flydra, the subject name applied to all files
frame_rate	For Flydra, the frame rate applied to all files
simplify_marker_naming	For Motive, if Markers are encountered, should they be renamed from "Subject:marker" to "marker"? Defaults to TRUE
import_messaging	Should this function report each time a file has been processed?
...	Additional arguments to specify how data should be cleaned.

Details

viewr objects should be in a list, e.g. the object generated by import_batch().

See clean_viewr() for details of how cleaning steps are handled and/or refer to the corresponding cleaning functions themselves.

Value

A list of viewr objects (tibble or data.frame with attribute pathviewr_steps that includes "viewr") that have been passed through the corresponding cleaning functions.

Author(s)

Vikram B. Baliga

See Also

Other data import functions: as_viewr(), import_batch(), read_flydra_mat(), read_motive_csv()

Other batch functions: bind_viewr_objects(), clean_viewr_batch(), import_batch()

Examples

```
## Since we only have one example file of each type provided
## in pathviewr, we will simply import the same example multiple
## times to simulate batch importing. Replace the contents of
## the following list with your own list of files to be imported.

## Make a list of the same example file 3x
import_list <-
  c(rep(
    system.file("extdata", "pathviewr_motive_example_data.csv",
                package = 'pathviewr'),
    3
  ))
```

```
## Batch import
motive_batch_imports <-
  import_batch(import_list,
               import_method = "motive",
               import_messaging = TRUE)

## Batch cleaning of these imported files
## via clean_viewr_batch()
motive_batch_cleaned <-
  clean_viewr_batch(
    file_announce = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Alternatively, use import_and_clean_batch() to
## combine import with cleaning on a batch of files
motive_batch_import_and_clean <-
  import_and_clean_batch(
    import_list,
    import_method = "motive",
    import_messaging = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Each of these lists of objects can be bound into
## one viewr object (i.e. one tibble) via
## bind_viewr_objects()
motive_bound_one <-
  bind_viewr_objects(motive_batch_cleaned)

motive_bound_two <-
  bind_viewr_objects(motive_batch_import_and_clean)

## Either route results in the same object ultimately:
identical(motive_bound_one, motive_bound_two)
```

`import_and_clean_viewr` *Import + clean_viewr()*

Description

Import a file and then, akin to `clean_viewr`, run through as many cleaning steps as desired.

Usage

```
import_and_clean_viewr(
  file_name,
  file_id = NA,
  relabel_viewr_axes = TRUE,
  gather_tunnel_data = TRUE,
  trim_tunnel_outliers = TRUE,
  standardization_option = "rotate_tunnel",
  get_velocity = TRUE,
  select_x_percent = TRUE,
  rename_viewr_characters = FALSE,
  separate_trajectories = TRUE,
  get_full_trajectories = TRUE,
  fill_traj_gaps = FALSE,
  ...
)
```

Arguments

`file_name`	Target file
`file_id`	Optional
`relabel_viewr_axes`	default TRUE, should axes be relabeled?
`gather_tunnel_data`	default TRUE, should tunnel data be gathered?
`trim_tunnel_outliers`	default TRUE, outliers be trimmed?
`standardization_option`	default "rotate_tunnel"; which standardization option should be used? See Details for more.
`get_velocity`	default TRUE, should velocity be computed?
`select_x_percent`	default TRUE, should a region of interest be selected?
`rename_viewr_characters`	default FALSE, should subjects be renamed?
`separate_trajectories`	default TRUE, should trajectories be defined?
`get_full_trajectories`	default TRUE, should only full trajectories be retained?
`fill_traj_gaps`	default FALSE, should gaps in trajectories be filled?
`...`	Additional arguments passed to the corresponding functions.

Details

Each argument corresponds to a standalone function in `pathviewr`. E.g. the parameter `relabel_viewr_axes` allows a user to choose whether `pathviewr::relabel_viewr_axes()` is run internally. Should the

user desire to use any non-default parameter values for any functions included here, they should be supplied to this function as additional arguments formatted exactly as they would appear in their corresponding function(s). E.g. if the "autodetect" feature in `pathviewr::separate_trajectories()` is desired, add an argument `max_frame_gap = "autodetect"` to the arguments supplied to this function.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) that has passed through several `pathviewr` functions as desired by the user, resulting in data that have been cleaned and ready for analyses.

Author(s)

Vikram B. Baliga

See Also

Other all in one functions: `clean_viewr()`

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

motive_full <-
  motive_data %>%
  clean_viewr(desired_percent = 50,
              max_frame_gap = "autodetect",
              span = 0.95)

## Alternatively, used the import_and_clean_viewr()
## function to combine these steps
motive_import_and_clean <-
  import_and_clean_viewr(
    file_name = system.file("extdata", "pathviewr_motive_example_data.csv",
                            package = 'pathviewr'),
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )
```

`import_batch` *Batch import of files for either Motive or Flydra (but not a mix of both)*

Description

Batch import of files for either Motive or Flydra (but not a mix of both)

Usage

```
import_batch(
  file_path_list,
  import_method = c("flydra", "motive"),
  file_id = NA,
  subject_name = NULL,
  frame_rate = NULL,
  simplify_marker_naming = TRUE,
  import_messaging = FALSE,
  ...
)
```

Arguments

`file_path_list`	A list of file paths
`import_method`	Either "flydra" or "motive"
`file_id`	Optional
`subject_name`	For Flydra, the assigned subject name
`frame_rate`	For Flydra, the assigned frame rate
`simplify_marker_naming`	default TRUE; for Motive, whether marker naming should be simplified
`import_messaging`	default FALSE; should this function report each time a file has been imported?
`...`	Additional arguments (may remove this if needed)

Details

Refer to `read_motive_csv()` and `read_flydra_mat()` for details of data import methods.

Value

A list of viewr objects (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`).

Author(s)

Vikram B. Baliga

See Also

Other data import functions: as_viewr(), import_and_clean_batch(), read_flydra_mat(), read_motive_csv()

Other batch functions: bind_viewr_objects(), clean_viewr_batch(), import_and_clean_batch()

Examples

```
## Since we only have one example file of each type provided
## in pathviewr, we will simply import the same example multiple
## times to simulate batch importing. Replace the contents of
## the following list with your own list of files to be imported.

## Make a list of the same example file 3x
import_list <-
  c(rep(
    system.file("extdata", "pathviewr_motive_example_data.csv",
                package = 'pathviewr'),
    3
  ))

## Batch import
motive_batch_imports <-
  import_batch(import_list,
               import_method = "motive",
               import_messaging = TRUE)

## Batch cleaning of these imported files
## via clean_viewr_batch()
motive_batch_cleaned <-
  clean_viewr_batch(
    file_announce = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Alternatively, use import_and_clean_batch() to
## combine import with cleaning on a batch of files
motive_batch_import_and_clean <-
  import_and_clean_batch(
    import_list,
    import_method = "motive",
    import_messaging = TRUE,
    motive_batch_imports,
    desired_percent = 50,
    max_frame_gap = "autodetect",
    span = 0.95
  )

## Each of these lists of objects can be bound into
## one viewr object (i.e. one tibble) via
```

```
## bind_viewr_objects()
motive_bound_one <-
  bind_viewr_objects(motive_batch_cleaned)

motive_bound_two <-
  bind_viewr_objects(motive_batch_import_and_clean)

## Either route results in the same object ultimately:
identical(motive_bound_one, motive_bound_two)
```

`insert_treatments` *Inserts treatment and experiment information*

Description

Adds information about treatment and experimental set up to viewr objects for analysis in other pathviewr functions

Usage

```
insert_treatments(
  obj_name,
  tunnel_config = "box",
  perch_2_vertex = NULL,
  vertex_angle = NULL,
  tunnel_width = NULL,
  tunnel_length = NULL,
  stim_param_lat_pos = NULL,
  stim_param_lat_neg = NULL,
  stim_param_end_pos = NULL,
  stim_param_end_neg = NULL,
  treatment = NULL
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`tunnel_config`	The configuration of the experimental tunnel. Currently, pathviewr supports rectangular "box" and V-shaped tunnel configurations.
`perch_2_vertex`	If using a V-shaped tunnel, this is the vertical distance between the vertex and the height of the perches. If the tunnel does not have perches, insert the vertical distance between the vertex and the height of the origin (0,0,0).
`vertex_angle`	If using a V-shaped tunnel, the angle of the vertex (in degrees) `vertex_angle` defaults to 90.
`tunnel_width`	If using a box-shaped tunnel, the width of the tunnel.

`tunnel_length` The length of the tunnel.

`stim_param_lat_pos`
: The size of the stimulus on the lateral positive wall of the tunnel. Eg. for 10cm wide gratings, `stim_param_lat_pos` = 0.1.

`stim_param_lat_neg`
: The size of the stimulus on the lateral negative wall of the tunnel..

`stim_param_end_pos`
: The size of the stimulus on the end positive wall of the tunnel.

`stim_param_end_neg`
: The size of the stimulus on the end negative wall of the tunnel.

`treatment` The name of the treatment assigned to all rows of the viewr object. Currently only able to accept a single treatment per viewr data object.

Details

All length measurements reported in meters.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"treatments added"`). Depending on the argument `tunnel_config`, the viewr object also includes columns storing the values of the supplied arguments. This experimental information is also stored in the viewr object's metadata

Author(s)

Eric R. Press

See Also

Other utility functions: `clean_by_span()`, `remove_duplicate_frames()`, `remove_vel_anomalies()`, `set_traj_frametime()`

Examples

```
## Import sample data from package
motive_data <-
 read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                             package = 'pathviewr'))
flydra_data <-
read_flydra_mat(system.file("extdata", "pathviewr_flydra_example_data.mat",
                             package = 'pathviewr'),
                             subject_name = "birdie_sanders")

 ## Clean data up to and including get_full_trajectories()
motive_data_full <-
 motive_data %>%
 relabel_viewr_axes() %>%
 gather_tunnel_data() %>%
 trim_tunnel_outliers() %>%
```

```
 rotate_tunnel() %>%
 select_x_percent(desired_percent = 50) %>%
 separate_trajectories(max_frame_gap = "autodetect") %>%
 get_full_trajectories(span = 0.95)

 flydra_data_full <-
  flydra_data %>%
  redefine_tunnel_center(length_method = "middle",
                         height_method = "user-defined",
                         height_zero = 1.44) %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = "autodetect") %>%
  get_full_trajectories(span = 0.95)


## Now add information about the experimental configuration. In this example,
## a V-shaped tunnel in which the vertex is 90deg and lies 0.40m below the
## origin. The visual stimuli on the lateral and end walls have a cycle
## length of 0.1m and 0.3m respectively, and the treatment is labeled
## "lat10_end30"

motive_v <-
motive_data_full %>%
 insert_treatments(tunnel_config = "v",
                   perch_2_vertex = 0.4,
                   vertex_angle = 90,
                   tunnel_length = 2,
                   stim_param_lat_pos = 0.1,
                   stim_param_lat_neg = 0.1,
                   stim_param_end_pos = 0.3,
                   stim_param_end_neg = 0.3,
                   treatment = "lat10_end_30")

# For an experiment using the box-shaped configuration where the tunnel is 1m
# wide and 3m long and the visual stimuli on the lateral and end walls have a
# cycle length of 0.2 and 0.3m, respectively, and the treatment is labeled
# "lat20_end30".

flydra_box <-
 flydra_data_full %>%
 insert_treatments(tunnel_config = "box",
                   tunnel_width = 1,
                   tunnel_length = 3,
                   stim_param_lat_pos = 0.2,
                   stim_param_lat_neg = 0.2,
                   stim_param_end_pos = 0.3,
                   stim_param_end_neg = 0.3,
                   treatment = "lat20_end30")

## Check out the new columns in the resulting objects
names(motive_v)
names(flydra_box)
```

plot_by_subject *Plot trajectories and density plots of position by subject*

Description

Plots all trajectories and generates density plots of position by subject from elevation and bird's eye views.

Usage

```
plot_by_subject(obj_name, col_by_treat = FALSE, ...)
```

Arguments

obj_name	A viewr object (a tibble or data.frame with attribute pathviewr_steps that includes "viewr") that has been passed through separate_trajectories() or get_full_trajectories().
col_by_treat	If multiple treatments or sessions, color data per treatment or session. Treatments must be levels in a column named treatment.
...	Additional arguments passed to/from other pathviewr functions.

Details

The input viewr object should have passed through separate_trajectories() or get_full_trajectories(). Optionally, treatments should have been added as levels in a column named treatment. Two plots will be produced, one from a "bird's eye view" of width against length and one from an "elevation view" of height against length. All trajectories will be plotted on a per subject basis, along with density plots of width or height depending on the view. col_by_treat = TRUE, data will be plotted by color according to treatment in both the trajectory plots and the density plots.

Value

A "bird's eye view" plot and an "elevation view" plot, made via ggplot2.

Author(s)

Melissa S. Armstrong

See Also

Other plotting functions: plot_viewr_trajectories(), visualize_frame_gap_choice()

Examples

```
library(pathviewr)
library(ggplot2)
library(magrittr)

if (interactive()) {
  ## Import the example Motive data included in the package
  motive_data <-
    read_motive_csv(system.file("extdata",
                                "pathviewr_motive_example_data.csv",
                                package = 'pathviewr'))

  ## Clean, isolate, and label trajectories
  motive_full <-
    motive_data %>%
    clean_viewr(desired_percent = 50,
                max_frame_gap = "autodetect",
                span = 0.95)

  ## Plot all trajectories by subject
  motive_full %>%
    plot_by_subject()

  ## Add treatment information
  motive_full$treatment <- c(rep("latA", 100), rep("latB", 100),
                             rep("latA", 100), rep("latB", 149))

  ## Plot all trajectories by subject, color by treatment
  motive_full %>%
    plot_by_subject(col_by_treat = TRUE)
}
```

---

plot_viewr_trajectories

Plot each trajectory within a viewr object

Description

Plot each trajectory within a viewr object

Usage

```
plot_viewr_trajectories(
  obj_name,
  plot_axes = c("length", "width"),
  multi_plot = FALSE
)
```

Arguments

obj_name	A viewr object (a tibble or data.frame with attribute pathviewr_steps that includes "viewr") that has been passed through separate_trajectories() or get_full_trajectories().
plot_axes	Which position axes should be plotted? A character vector including exactly two of the following choices must be supplied: length, width, height. Default is c("length", "width").
multi_plot	Should separate plots (one per trajectory) be created or should one multi-plot grid be generated. Defaults to FALSE, which produces separate plots.

Value

A (base-R) series of plots or single plot (if multi_plot = TRUE) that depict each trajectory along the chosen axes.

Author(s)

Vikram B. Baliga

See Also

Other plotting functions: plot_by_subject(), visualize_frame_gap_choice()

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

motive_full <-
  motive_data %>%
  clean_viewr(desired_percent = 50,
              max_frame_gap = "autodetect",
              span = 0.95)

plot_viewr_trajectories(motive_full, multi_plot = FALSE)
plot_viewr_trajectories(motive_full, multi_plot = TRUE)
```

---

quick_separate_trajectories

Quick version of separate_trajectories()

Description

Mostly meant for internal use but available nevertheless.

Usage

```
quick_separate_trajectories(obj_name, max_frame_gap = 1, ...)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
max_frame_gap	Unlike the corresponding parameter in separate_trajectories, must be a single numeric here.
...	Additional arguments passed to/from other pathviewr functions

Details

This function is designed to separate rows of data into separately labeled trajectories.

The max_frame_gap parameter determines how trajectories will be separated. max_frame_gap defines the largest permissible gap in data before a new trajectory is forced to be defined. In this function, only a single numeric can be supplied to this parameter (unlike the case in separate_trajectories).

Value

A viewr object (tibble or data.frame with attribute pathviewr_steps that includes "viewr") in which a new column file_sub_traj is added, which labels trajectories within the data by concatenating file name, subject name, and a trajectory number (all separated by underscores).

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), redefine_tunnel_center(), relabel_viewr_axes(), rename_viewr_characters(), rotate_tunnel(), select_x_percent(), separate_trajectories(), standardize_tunnel(), trim_tunnel_outliers(), visualize_frame_gap_choice()

Other functions that define or clean trajectories: get_full_trajectories(), separate_trajectories(), visualize_frame_gap_choice()

Examples

```
## This function is not recommended for general use.
## See separate_trajectories() instead
```

rad_2_deg *Convert radians to degrees*

Description

Convert radians to degrees

Usage

```
rad_2_deg(rad)
```

Arguments

rad	Radians (a numeric of any length >= 1)

Value

The angle(s) in degrees (as a numeric vector of the same length)

Author(s)

Vikram B. Baliga

See Also

Other mathematical functions: calc_min_dist_v(), deg_2_rad(), find_curve_elbow(), get_2d_angle(), get_3d_angle(), get_3d_cross_prod(), get_dist_point_line(), get_traj_velocities(), get_velocity()

Examples

```
## One input
rad_2_deg(pi/2)

## Multiple inputs
rad_2_deg(c(pi / 2, pi, 2 * pi))
```

`read_flydra_mat` *Import data from a MAT file exported from Flydra software*

Description

`read_flydra_mat()` is designed to import data from a `.mat` file that has been exported from Flydra software. The resultant object is a tibble that additionally has important metadata stored as attributes (see Details).

Usage

```
read_flydra_mat(mat_file, file_id = NA, subject_name, frame_rate = 100, ...)
```

Arguments

`mat_file`	A file (or path to file) in .mat format, exported from Flydra
`file_id`	(Optional) identifier for this file. If not supplied, this defaults to `basename(file_name)`.
`subject_name`	Name that will be assigned to the subject
`frame_rate`	The capture frame rate of the session
`...`	Additional arguments that may be passed from other pathviewr functions

Value

A tibble with numerical data in columns. The first two columns will have frame numbers and time (assumed to be in secs), respectively. Columns 3 through 5 will contain position data. Note that unlike the behavior of `read_motive_csv()` this function produces "tidy" data that have already been gathered into key-value pairs based on subject.

Author(s)

Vikram B. Baliga

See Also

`read_motive_csv` for importing Motive data

Other data import functions: `as_viewr()`, `import_and_clean_batch()`, `import_batch()`, `read_motive_csv()`

Examples

```
library(pathviewr)

## Import the example Flydra data included in the package
flydra_data <-
  read_flydra_mat(system.file("extdata", "pathviewr_flydra_example_data.mat",
                              package = 'pathviewr'),
                  subject_name = "birdie_wooster")
```

```
## Names of variables in the resulting tibble
names(flydra_data)

## A variety of metadata are stored as attributes. Of particular interest:
attr(flydra_data, "pathviewr_steps")
```

`read_motive_csv` *Import data from a CSV exported from Optitrack's Motive software*

Description

`read_motive_csv()` is designed to import data from a CSV that has been exported from Optitrack's Motive software. The resultant object is a tibble that additionally has important metadata stored as attributes (see Details).

Usage

```
read_motive_csv(file_name, file_id = NA, simplify_marker_naming = TRUE, ...)
```

Arguments

`file_name`	A file (or path to file) in CSV format
`file_id`	(Optional) identifier for this file. If not supplied, this defaults to `basename(file_name)`.
`simplify_marker_naming`	If Markers are encountered, should they be renamed from "Subject:marker" to "marker"? Defaults to TRUE
`...`	Additional arguments passed from other `pathviewr` functions

Details

Uses `data.table::fread()` to import data from a CSV file and ultimately store it in a tibble. This object is also labeled with the attribute `pathviewr_steps` with value `viewr` to indicate that it has been imported by `pathviewr` and should be friendly towards use with other functions in our package. Additionally, the following metadata are stored in the tibble's attributes: header information from the Motive CSV file (`header`), original IDs for each object (`Motive_IDs`), the name of each subject in each data column (`subject_names_full`) and unique values of subject names (`subject_names_simple`), the type of data (rigid body or marker) that appears in each column (`data_types_full`) and overall (`data_types_simple`), and original data column names in the CSV (`d1, d2`). See Example below for example code to inspect attributes.

Value

A tibble with numerical data in columns. The first two columns will have frame numbers and time (assumed to be in secs), respectively. Columns 3 and beyond will contain the numerical data on the position or rotation of rigid bodies and/or markers that appear in the Motive CSV file. Each row corresponds to the position or rotation of all objects at a given time (frame).

Warning

This function was written to read CSVs exported using Motive's Format Version 1.23 and is not guaranteed to work with those from other versions. Please file an Issue on our Github page if you encounter any problems.

Author(s)

Vikram B. Baliga

See Also

`read_flydra_mat` for importing Flydra data

Other data import functions: `as_viewr()`, `import_and_clean_batch()`, `import_batch()`, `read_flydra_mat()`

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Names of variables in the resulting tibble
names(motive_data)

## A variety of metadata are stored as attributes. Of particular interest:
attr(motive_data, "pathviewr_steps")
attr(motive_data, "file_id")
attr(motive_data, "header")
attr(motive_data, "Motive_IDs")
attr(motive_data, "subject_names_full")
attr(motive_data, "subject_names_simple")
attr(motive_data, "motive_data_names")
attr(motive_data, "motive_data_types_full")
attr(motive_data, "motive_data_types_simple")

## Of course, all attributes can be viewed as a (long) list via:
attributes(motive_data)
```

---

redefine_tunnel_center

"Center" the tunnel data, i.e. translation but no rotation

Description

Redefine the center `(0, 0, 0,)` of the tunnel data via translating positions along axes.

Usage

```
redefine_tunnel_center(
  obj_name,
  axes = c("position_length", "position_width", "position_height"),
  length_method = c("original", "middle", "median", "user-defined"),
  width_method = c("original", "middle", "median", "user-defined"),
  height_method = c("original", "middle", "median", "user-defined"),
  length_zero = NA,
  width_zero = NA,
  height_zero = NA,
  ...
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
axes	Names of axes to be centered
length_method	Method for length
width_method	Method for width
height_method	Method for height
length_zero	User-defined value
width_zero	User-defined value
height_zero	User-defined value
...	Additional arguments passed to/from other pathviewr functions

Details

For each _method argument, there are four choices of how centering is handled: 1) "original" keeps axis as is – this is how width and (possibly) height should be handled for flydra data; 2) "middle" is the middle of the range of data: (min + max) / 2; 3) "median" is the median value of data on that axis. Probably not recommended; and 4) "user-defined" lets the user customize where the (0, 0, 0) point in the tunnel will end up. Each _zero argument is subtracted from its corresponding axis' data.

Value

A viewr object (tibble or data.frame with attribute pathviewr_steps that includes "viewr") in which data have been translated according to the user's inputs, generally with (0, 0, 0,) being relocated to the center of the tunnel.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), quick_separate_trajectories(), relabel_viewr_axes(), rename_viewr_characters(), rotate_tunnel(), select_x_percent(), separate_trajectories(), standardize_tunnel(), trim_tunnel_outliers(), visualize_frame_gap_choice()

Other tunnel standardization functions: rotate_tunnel(), standardize_tunnel()

Examples

```
## Import the Flydra example data included in
## the package
flydra_data <-
  read_flydra_mat(
    system.file("extdata",
                "pathviewr_flydra_example_data.mat",
                package = 'pathviewr'),
    subject_name = "birdie_wooster"
  )

## Re-center the Flydra data set.
## Width will be untouched
## Length will use the "middle" definition
## And height will be user-defined to be
## zeroed at 1.44 on the original axis
flydra_centered <-
  flydra_data %>%
  redefine_tunnel_center(length_method = "middle",
                         height_method = "user-defined",
                         height_zero = 1.44)
```

---

relabel_viewr_axes *Relabel the dimensions as length, width, and height*

Description

Axes are commonly labeled as "x", "y", and "z" in recording software yet pathviewr functions require these to be labeled as "length", "width", and "height". relabel_viewr_axes() is a function that takes a viewr object and allows the user to rename its variables.

Usage

```
relabel_viewr_axes(
  obj_name,
  tunnel_length = "_z",
  tunnel_width = "_x",
  tunnel_height = "_y",
  real = "_w",
  ...
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
tunnel_length	The dimension that corresponds to tunnel length. Set to tunnel_length = "_z" by default. Argument should contain a character vector with a leading underscore (see Details)
tunnel_width	The dimension that corresponds to tunnel width. Follows the same conventions as tunnel_length and defaults to tunnel_length = "_x"
tunnel_height	The dimension that corresponds to tunnel height. Follows the same conventions as tunnel_length and defaults to tunnel_length = "_y"
real	The dimension that corresponds to the "real" parameter in quaternion notation (for data with "rotation" values). Follows the same conventions as tunnel_length and defaults to real = "_w"
...	Additional arguments to be passed to read_motive_csv().

Details

Each argument must have a leading underscore ("_") and each argument must have an entry. E.g. tunnel_length = "_Y" will replace all instances of _Y with _length in the names of variables.

Value

A tibble or data.frame with variables that have been renamed.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), quick_separate_trajectories(), redefine_tunnel_center(), rename_viewr_characters(), rotate_tunnel(), select_x_percent(), separate_trajectories(), standardize_tunnel(), trim_tunnel_outliers(), visualize_frame_gap_choice()

Examples

```
library(pathviewr)

## Import the Motive example data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Names of variables are labeled with _x, _y, _z, which we'd like to rename
names(motive_data)

## Now use relabel_viewr_axes() to rename these variables using _length,
## _width, and _height instead
```

```
motive_data_relabeled <-
  relabel_viewr_axes(motive_data,
                     tunnel_length = "_z",
                     tunnel_width = "_x",
                     tunnel_height = "_y",
                     real = "_w")

## See the result
names(motive_data_relabeled)
```

`remove_duplicate_frames` *Remove any duplicates or aliased frames within trajectories*

Description

Remove any duplicates or aliased frames within trajectories

Usage

```
remove_duplicate_frames(obj_name)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`

Details

The separate_trajectories() and get_full_trajectories() must be run prior to use.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps`.

Author(s)

Vikram B. Baliga

See Also

Other utility functions: `clean_by_span()`, `insert_treatments()`, `remove_vel_anomalies()`, `set_traj_frametime()`

`remove_vel_anomalies` *Remove any rows which show sharp shifts in velocity that are likely due to tracking errors*

Description

Remove any rows which show sharp shifts in velocity that are likely due to tracking errors

Usage

```
remove_vel_anomalies(
  obj_name,
  target = "velocity",
  method = "gesd",
  alpha = 0.05,
  max_anoms = 0.2
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`target`	The column to target; defaults to "velocity"
`method`	The anomaly detection method; see anomalize::anomalize()
`alpha`	The width of the "normal" range; see anomalize::anomalize()
`max_anoms`	The max proportion of anomalies; see anomalize::anomalize()

Details

This function runs anomalize::anomalize() on a per-trajectory basis. The separate_trajectories() and get_full_trajectories() must be run prior to use.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps`. Rows in which large anomalies were detected have been removed. No additional columns are created.

Author(s)

Vikram B. Baliga

See Also

Other utility functions: `clean_by_span()`, `insert_treatments()`, `remove_duplicate_frames()`, `set_traj_frametime()`

`rename_viewr_characters` *Rename subjects in the data via pattern detection*

Description

Quick utility function to use str_replace with mutate(across()) to batch- rename subjects via pattern detection.

Usage

```
rename_viewr_characters(
  obj_name,
  target_column = "subject",
  pattern,
  replacement = ""
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`target_column`	The target column; defaults to "subject"
`pattern`	The (regex) pattern to be replaced
`replacement`	The replacement text. Must be a character

Value

A tibble or data frame in which subjects have been renamed according to the `pattern` and `replacement` supplied by the user.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: `gather_tunnel_data()`, `get_full_trajectories()`, `quick_separate_trajectories()`, `redefine_tunnel_center()`, `relabel_viewr_axes()`, `rotate_tunnel()`, `select_x_percent()`, `separate_trajectories()`, `standardize_tunnel()`, `trim_tunnel_outliers()`, `visualize_frame_gap_choice()`

Examples

```
## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "gather" step before running rescale_tunnel_data().
 motive_gathered <-
   motive_data %>%
   relabel_viewr_axes() %>%
   gather_tunnel_data()

## See the subject names
 unique(motive_gathered$subject)

## Now rename the subjects. We'll get rid of "device" and replace it
## with "subject"
motive_renamed <-
  motive_gathered %>%
  rename_viewr_characters(target_column = "subject",
                          pattern = "device",
                          replacement = "subject")

## See the new subject names
unique(motive_renamed$subject)
```

`rescale_tunnel_data` *Rescale position data within a* `viewr` *object*

Description

Should data have been exported at an incorrect scale, apply an isometric transformation to the position data and associated mean marker errors (if found)

Usage

```
rescale_tunnel_data(obj_name, original_scale = 0.5, desired_scale = 1, ...)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"` that has been passed through `relabel_viewr_axes()` and `gather_tunnel_data()` (or is structured as though it has been passed through those functions).
`original_scale`	The original scale at which data were exported. See Details if unknown.
`desired_scale`	The desired scale
`...`	Additional arguments passed to/from other pathviewr functions

Details

The desired_scale is divided by the original_scale to determine a scale_ratio internally. If the original_scale is not explicitly known, set it to 1 and then set desired_scale to be whatever scaling ratio you have in mind. E.g. setting original_scale to 1 and then desired_scale to 0.7 will multiply all position axis values by 0.7.

The default arguments of original_scale = 0.5 and desired_scale = 1 apply a doubling of tunnel size isometrically.

Value

A viewr object that has position data (and mean_marker_error data, if found) adjusted by the ratio of desired_scale/original_scale.

Author(s)

Vikram B. Baliga

Examples

```
## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "gather" step before running rescale_tunnel_data().
 motive_gathered <-
   motive_data %>%
   relabel_viewr_axes() %>%
   gather_tunnel_data()

## Now rescale the tunnel data
motive_rescaled <-
  motive_gathered %>%
  rescale_tunnel_data(original_scale = 0.5,
                      desired_scale = 1)

## See the difference in data range e.g. for length
range(motive_rescaled$position_length)
range(motive_gathered$position_length)
```

---

rm_by_trajnum *Remove subjects by trajectory number*

Description

Specify a minimum number of trajectories that each subject must complete during a treatment, trial, or session.

Usage

```
rm_by_trajnum(
  obj_name,
  trajnum = 5,
  mirrored = FALSE,
  treatment1,
  treatment2,
  ...
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr". Trajectories must be predefined (i.e. via separate_trajectories()).
trajnum	Minimum number of trajectories; must be numeric.
mirrored	Does the data have mirrored treatments? If so, arguments treatment1 and treatment2 must also be provided, indicating the names of two mirrored treatments, both of which must meet the trajectory threshold specified in trajnum. Default is FALSE.
treatment1	The first treatment or session during which the threshold must be met.
treatment2	A second treatment or session during which the threshold must be met.
...	Additional arguments passed to/from other pathviewr functions.

Details

Depending on analysis needs, users may want to remove subjects that have not completed a certain number of trajectories during a treatment, trial, or session. If mirrored = FALSE, no treatment information is necessary and subjects will be removed based on total number of trajectories as specified in trajnum. If mirrored = TRUE, the treatment1 and treatment2 parameters will allow users to define during which treatments or sessions subjects must reach threshold as specified in the trajnum argument. For example, if mirrored = TRUE, setting treatment1 = "latA", treatment2 = "latB" and trajnum = 5 will remove subjects that have fewer than 5 trajectories during the "latA" treatment AND the "latB" treatment. treatment1 and treatment2 should be levels within a column named "treatment".

Value

A viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr" that now has fewer observations (rows) as a result of removal of subjects with too few trajectories according to the trajnum parameter.

Author(s)

Melissa S. Armstrong

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean, isolate, and label trajectories
motive_full <-
  motive_data %>%
  clean_viewr(desired_percent = 50,
              max_frame_gap = "autodetect",
              span = 0.95)

##Remove subjects that have not completed at least 150 trajectories:
motive_rm_unmirrored <-
  motive_full %>%
  rm_by_trajnum(trajnum = 150)

## Add treatment information
motive_full$treatment <- c(rep("latA", 100),
                           rep("latB", 100),
                           rep("latA", 100),
                           rep("latB", 149))

## Remove subjects by that have not completed at least 10 trajectories in
## both treatments
motive_rm_mirrored <-
  motive_full %>%
  rm_by_trajnum(
    trajnum = 10,
    mirrored = TRUE,
    treatment1 = "latA",
    treatment2 = "latB"
  )
```

`rotate_tunnel` *Rotate a tunnel so that perches are approximately aligned*

Description

The rotation is applied about the height axis and affects tunnel length and width only, i.e. no rotation of height.

Usage

```
rotate_tunnel(
  obj_name,
```

```
  all_heights_min = 0.11,
  all_heights_max = 0.3,
  perch1_len_min = -0.06,
  perch1_len_max = 0.06,
  perch2_len_min = 2.48,
  perch2_len_max = 2.6,
  perch1_wid_min = 0.09,
  perch1_wid_max = 0.31,
  perch2_wid_min = 0.13,
  perch2_wid_max = 0.35,
  ...
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr" that has been passed through relabel_viewr_axes() and gather_tunnel_data() (or is structured as though it has been passed through those functions).
all_heights_min	Minimum perch height
all_heights_max	Maximum perch height
perch1_len_min	Minimum length value of perch 1
perch1_len_max	Maximum length value of perch 1
perch2_len_min	Minimum length value of perch 2
perch2_len_max	Maximum length value of perch 2
perch1_wid_min	Minimum width value of perch 1
perch1_wid_max	Maximum width value of perch 1
perch2_wid_min	Minimum width value of perch 2
perch2_wid_max	Maximum width value of perch 2
...	Additional arguments passed to/from other pathviewr functions

Details

The user first estimates the locations of the perches by specifying bounds for where each perch is located. The function then computes the center of each bounding box and estimates that to be the midpoint of each perch. Then the center point of the tunnel (center between the perch midpoints) is estimated. The angle between perch1_center, tunnel_center_point, and arbitrary point along the length axis (tunnel_center_point - 1 on length) is estimated. That angle is then used to rotate the data, again only in the length and width dimensions. Height is standardized by (approximate) perch height; values greater than 0 are above the perch and values less than 0 are below the perch level.

Value

A viewr object (tibble or data.frame with attribute pathviewr_steps that includes "viewr") in which data have been rotated according to user specifications.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), quick_separate_trajectories(), redefine_tunnel_center(), relabel_viewr_axes(), rename_viewr_characters(), select_x_percent(), separate_trajectories(), standardize_tunnel(), trim_tunnel_outliers(), visualize_frame_gap_choice()

Other tunnel standardization functions: redefine_tunnel_center(), standardize_tunnel()

Examples

```
## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "trimmed" step before running rotate_tunnel().
motive_trimmed <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers()

## Now rotate the tunnel using default values
motive_rotated <-
  motive_trimmed %>%
  rotate_tunnel()

## The following attributes store information about
## how rotation & translation was applied
attr(motive_rotated, "rotation_degrees")
attr(motive_rotated, "rotation_radians")
attr(motive_rotated, "perch1_midpoint_original")
attr(motive_rotated, "perch1_midpoint_current")
attr(motive_rotated, "tunnel_centerpoint_original")
attr(motive_rotated, "perch2_midpoint_original")
attr(motive_rotated, "perch2_midpoint_current")
```

---

section_tunnel_by *Bin data along a specified axis*

Description

Chop data into X sections (of equal size) along a specified axis

Usage

```
section_tunnel_by(obj_name, axis = "position_length", number_of_sections = 20)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
axis	Chosen axis, must match name of column exactly
number_of_sections	Total number of sections

Details

The idea is to bin the data along a specified axis, generally position_length.

Value

A new column added to the input data object called section_id, which is an ordered factor that indicates grouping.

Author(s)

Vikram B. Baliga

Examples

```
## Load data and run section_tunnel_by()
test_mat <-
  read_flydra_mat(system.file("extdata", "pathviewr_flydra_example_data.mat",
                              package = 'pathviewr'),
                  subject_name = "birdie_wooster") %>%
  redefine_tunnel_center(length_method = "middle",
                         height_method = "user-defined",
                         height_zero = 1.44) %>%
  select_x_percent(desired_percent = 50) %>%
  separate_trajectories(max_frame_gap = 1) %>%
  get_full_trajectories(span = 0.95) %>%
  section_tunnel_by(number_of_sections = 10)

## Plot; color by section ID
plot(test_mat$position_length,
     test_mat$position_width,
     asp = 1, col = as.factor(test_mat$section_id))
```

---

select_x_percent *Select a region of interest within the tunnel*

Description

Select data in the middle X percent of the length of the tunnel

Usage

```
select_x_percent(obj_name, desired_percent = 33, ...)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr"
desired_percent	Numeric, the percent of the total length of the tunnel that will define the region of interest. Measured from the center outwards.
...	Additional arguments passed to/from other pathviewr functions

Value

A viewr object (tibble or data.frame with attribute pathviewr_steps that includes "viewr") in which data outside the region of interest have been removed.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), quick_separate_trajectories(), redefine_tunnel_center(), relabel_viewr_axes(), rename_viewr_characters(), rotate_tunnel(), separate_trajectories(), standardize_tunnel(), trim_tunnel_outliers(), visualize_frame_gap_choice()

Examples

```
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "trimmed" step before running rotate_tunnel().
motive_rotated <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel()

## Now select the middle 50% of the tunnel
motive_selected <-
  motive_rotated %>%
  select_x_percent(desired_percent = 50)

## Compare the ranges of lengths to see the effect
range(motive_rotated$position_length)
range(motive_selected$position_length)
```

`separate_trajectories` *Separate rows of data into separately labeled trajectories.*

Description

Separate rows of data into separately labeled trajectories.

Usage

```
separate_trajectories(
  obj_name,
  max_frame_gap = 1,
  frame_rate_proportion = 0.1,
  frame_gap_messaging = FALSE,
  frame_gap_plotting = FALSE,
  ...
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`max_frame_gap`	Default 1; defines the largest permissible gap in data before a new trajectory is forced to be defined. Can be either a numeric value or "autodetect". See Details for more.
`frame_rate_proportion`	Default 0.10; if `max_frame_gap = "autodetect"`, proportion of frame rate to be used as a reference (see Details).
`frame_gap_messaging`	Default FALSE; should frame gaps be reported in the console?
`frame_gap_plotting`	Default FALSE; should frame gap diagnostic plots be shown?
`...`	Additional arguments passed to/from other pathviewr functions

Details

This function is designed to separate rows of data into separately labeled trajectories.

The `max_frame_gap` parameter determines how trajectories will be separated. If numeric, the function uses the supplied value as the largest permissible gap in frames before a new trajectory is defined.

If `max_frame_gap = "autodetect"` is used, the function attempts to guesstimate the best value(s) of `max_frame_gap`. This is performed separately for each subject in the data set, i.e. as many `max_frame_gap` values will be estimated as there are unique subjects. The highest possible value of any `max_frame_gap` is first set as a proportion of the capture frame rate, as defined by the `frame_rate_proportion` parameter (default 0.10). For each subject, a plot of total trajectory

counts vs. max frame gap values is created internally (but can be plotted via setting `frame_gap_plotting = TRUE`). As larger max frame gaps are allowed, trajectory count will necessarily decrease but may reach a value that likely represents the "best" option. The "elbow" of that plot is then used to find an estimate of the best max frame gap value to use.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) in which a new column `file_sub_traj` is added, which labels trajectories within the data by concatenating file name, subject name, and a trajectory number (all separated by underscores).

Author(s)

Vikram B. Baliga and Melissa S. Armstrong

See Also

Other data cleaning functions: `gather_tunnel_data()`, `get_full_trajectories()`, `quick_separate_trajectories()`, `redefine_tunnel_center()`, `relabel_viewr_axes()`, `rename_viewr_characters()`, `rotate_tunnel()`, `select_x_percent()`, `standardize_tunnel()`, `trim_tunnel_outliers()`, `visualize_frame_gap_choice()`

Other functions that define or clean trajectories: `get_full_trajectories()`, `quick_separate_trajectories()`, `visualize_frame_gap_choice()`

Examples

```
## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "select" step before running select_x_percent().
motive_selected <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel() %>%
  select_x_percent(desired_percent = 50)

## Now separate trajectories using autodetect
motive_separated <-
  motive_selected %>%
  separate_trajectories(max_frame_gap = "autodetect",
                        frame_rate_proportion = 0.1)

## See new column file_sub_traj for trajectory labeling
names(motive_separated)
```

`set_traj_frametime` *Redefine frames and time stamps on a per-trajectory basis*

Description

After a data set has been separated into trajectories, find the earliest frame in each trajectory and set the corresponding time to 0. All subsequent time_sec stamps are computed according to successive frame numbering.

Usage

```
set_traj_frametime(obj_name)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`

Details

The separate_trajectories() and get_full_trajectories() must be run prior to use. The initial traj_time and traj_frame values are set to 0 within each trajectory.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps`. New columns include traj_time (the trajectory-specific time values) and traj_frame (the trajectory-specific frame numbering).

Author(s)

Vikram B. Baliga

See Also

Other utility functions: `clean_by_span()`, `insert_treatments()`, `remove_duplicate_frames()`, `remove_vel_anomalies()`

`standardize_tunnel` *Rotate and center a tunnel based on landmarks*

Description

Similar to `rotate_tunnel()`; rotate and center tunnel data based on landmarks (specific subjects in the data).

Usage

```
standardize_tunnel(
  obj_name,
  landmark_one = "perch1",
  landmark_two = "perch2",
  ...
)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"` that has been passed through `relabel_viewr_axes()` and `gather_tunnel_data()` (or is structured as though it has been passed through those functions).
`landmark_one`	Subject name of the first landmark
`landmark_two`	Subject name of the second landmark
`...`	Additional arguments passed to/from other pathviewr functions

Details

The center point of the tunnel is estimated as the point between the two landmarks. It is therefore recommended that `landmark_one` and `landmark_two` be objects that are placed on opposite ends of the tunnel (e.g. in an avian flight tunnel, these landmarks may be perches that are placed at the extreme ends). The angle between landmark_one, tunnel_center_point, and arbitrary point along the length axis (tunnel_center_point - 1 on length) is estimated. That angle is then used to rotate the data, again only in the length and width dimensions. Height is standardized by average landmark height; values greater than 0 are above the landmarks and values less than 0 are below the landmark level.

Value

A viewr object (tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`) in which data have been rotated according to the positions of the landmarks in the data.

Warning

The `position_length` values of landmark_one MUST be less than the `position_length` values of landmark_two; otherwise, the rotation will apply to a mirror-image of the tunnel

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), quick_separate_trajectories(), redefine_tunnel_center(), relabel_viewr_axes(), rename_viewr_characters(), rotate_tunnel(), select_x_percent(), separate_trajectories(), trim_tunnel_outliers(), visualize_frame_gap_choice()

Other tunnel standardization functions: redefine_tunnel_center(), rotate_tunnel()

Examples

```
## Example data that would work with this function are
## not included in this version of pathviewr. Please
## contact the package authors for further guidance
## should you need it.
```

---

trim_tunnel_outliers *Trim out artifacts and other outliers from the extremes of the tunnel*

Description

The user provides estimates of min and max values of data. This function then trims out anything beyond these estimates.

Usage

```
trim_tunnel_outliers(
  obj_name,
  lengths_min = 0,
  lengths_max = 3,
  widths_min = -0.4,
  widths_max = 0.8,
  heights_min = -0.2,
  heights_max = 0.5,
  ...
)
```

Arguments

obj_name	The input viewr object; a tibble or data.frame with attribute pathviewr_steps that includes "viewr" that has been passed through relabel_viewr_axes() and gather_tunnel_data() (or is structured as though it has been passed through those functions).
lengths_min	Minimum length
lengths_max	Maximum length

widths_min	Minimum width
widths_max	Maximum width
heights_min	Minimum height
heights_max	Maximum height
...	Additional arguments passed to/from other pathviewr functions

Details

Anything supplied to _min or _max arguments should be single numeric values.

Value

A viewr object (tibble or data.frame with attribute pathviewr_steps that includes "viewr") in which data outside the specified ranges has been excluded.

Author(s)

Vikram B. Baliga

See Also

Other data cleaning functions: gather_tunnel_data(), get_full_trajectories(), quick_separate_trajectories(), redefine_tunnel_center(), relabel_viewr_axes(), rename_viewr_characters(), rotate_tunnel(), select_x_percent(), separate_trajectories(), standardize_tunnel(), visualize_frame_gap_choice()

Examples

```
## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

## Clean the file. It is generally recommended to clean up to the
## "gather" step before running trim_tunnel_outliers().
motive_gathered <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data()

## Now trim outliers using default values
motive_trimmed <-
  motive_gathered %>%
  trim_tunnel_outliers(lengths_min = 0,
                       lengths_max = 3,
                       widths_min = -0.4,
                       widths_max = 0.8,
                       heights_min = -0.2,
                       heights_max = 0.5)
```

`visualize_frame_gap_choice` *Visualize the consequence of using various max_frame_gap values*

Description

Run separate_trajectories() with many different frame gaps to help determine what value to use

Usage

```
visualize_frame_gap_choice(obj_name, loops = 20, ...)
```

Arguments

`obj_name`	The input viewr object; a tibble or data.frame with attribute `pathviewr_steps` that includes `"viewr"`
`loops`	How many total frame gap entries to consider. Each loop will increase the `max_fram_gap` argument in `separate_trajectories` by 1.
`...`	Additional arguments

Details

The input viewr object (`obj_name`) should likely be an object that has passed through the `select_x_percent()` step.

Value

A plot and a tibble, each of which shows the total number of trajectories that result from using the specified range of `max_frame_gap` values.

Author(s)

Melissa S. Armstrong and Vikram B. Baliga

See Also

Other data cleaning functions: `gather_tunnel_data()`, `get_full_trajectories()`, `quick_separate_trajectories()`, `redefine_tunnel_center()`, `relabel_viewr_axes()`, `rename_viewr_characters()`, `rotate_tunnel()`, `select_x_percent()`, `separate_trajectories()`, `standardize_tunnel()`, `trim_tunnel_outliers()`

Other plotting functions: `plot_by_subject()`, `plot_viewr_trajectories()`

Other functions that define or clean trajectories: `get_full_trajectories()`, `quick_separate_trajectories()`, `separate_trajectories()`

Examples

```
library(pathviewr)

## Import the example Motive data included in the package
motive_data <-
  read_motive_csv(system.file("extdata", "pathviewr_motive_example_data.csv",
                              package = 'pathviewr'))

motive_selected <-
  motive_data %>%
  relabel_viewr_axes() %>%
  gather_tunnel_data() %>%
  trim_tunnel_outliers() %>%
  rotate_tunnel() %>%
  get_velocity() %>%
  select_x_percent(desired_percent = 50)

visualize_frame_gap_choice(motive_selected, loops = 10)
```

Index

∗ **all in one functions**
clean_viewr, 10
import_and_clean_viewr, 35
∗ **batch functions**
bind_viewr_objects, 5
clean_viewr_batch, 12
import_and_clean_batch, 33
import_batch, 38
∗ **data cleaning functions**
gather_tunnel_data, 19
get_full_trajectories, 24
quick_separate_trajectories, 45
redefine_tunnel_center, 50
relabel_viewr_axes, 52
rename_viewr_characters, 56
rotate_tunnel, 60
select_x_percent, 63
separate_trajectories, 65
standardize_tunnel, 68
trim_tunnel_outliers, 69
visualize_frame_gap_choice, 71
∗ **data import functions**
as_viewr, 3
import_and_clean_batch, 33
import_batch, 38
read_flydra_mat, 48
read_motive_csv, 49
∗ **functions that define or clean trajectories**
get_full_trajectories, 24
quick_separate_trajectories, 45
separate_trajectories, 65
visualize_frame_gap_choice, 71
∗ **mathematical functions**
calc_min_dist_v, 8
deg_2_rad, 14
find_curve_elbow, 18
get_2d_angle, 20
get_3d_angle, 21
get_3d_cross_prod, 22
get_dist_point_line, 23
get_traj_velocities, 28
get_velocity, 29
rad_2_deg, 47
∗ **metadata handling functions**
get_header_viewr, 25
∗ **plotting functions**
plot_by_subject, 43
plot_viewr_trajectories, 44
visualize_frame_gap_choice, 71
∗ **tunnel standardization functions**
redefine_tunnel_center, 50
rotate_tunnel, 60
standardize_tunnel, 68
∗ **utility functions**
clean_by_span, 9
insert_treatments, 40
remove_duplicate_frames, 54
remove_vel_anomalies, 55
set_traj_frametime, 67
∗ **visual perception functions**
calc_min_dist_box, 6
get_sf, 26
get_vis_angle, 31

as_viewr, 3, *34*, *39*, *48*, *50*

bind_viewr_objects, 5, *13*, *34*, *39*

calc_min_dist_box, 6, *27*, *32*
calc_min_dist_v, 8, *15*, *18*, *21–23*, *29*, *31*, *47*
clean_by_span, 9, *41*, *54*, *55*, *67*
clean_viewr, 10, *37*
clean_viewr_batch, *5*, 12, *34*, *39*

deg_2_rad, *8*, 14, *18*, *21–23*, *29*, *31*, *47*

exclude_by_velocity, 15

fill_traj_gaps, 16

find_curve_elbow, *8*, *15*, 18, *21–23*, *29*, *31*, *47*

gather_tunnel_data, 19, *25*, *46*, *52, 53*, *56*, *62*, *64*, *66*, *69–71*
get_2d_angle, *8*, *15*, *18*, 20, *22, 23*, *29*, *31*, *47*
get_3d_angle, *8*, *15*, *18*, *21*, 21, *22, 23*, *29*, *31*, *47*
get_3d_cross_prod, *8*, *15*, *18*, *21, 22*, 22, *23*, *29*, *31*, *47*
get_dist_point_line, *8*, *15*, *18*, *21, 22*, 23, *29*, *31*, *47*
get_full_trajectories, *19*, 24, *46*, *52, 53*, *56*, *62*, *64*, *66*, *69–71*
get_header_viewr, 25
get_sf, *7*, 26, *32*
get_traj_velocities, *8*, *15*, *18*, *21–23*, 28, *31*, *47*
get_velocity, *8*, *15*, *18*, *21–23*, *29*, 29, *47*
get_vis_angle, *7*, *27*, 31

import_and_clean_batch, *4, 5*, *13*, 33, *39*, *48*, *50*
import_and_clean_viewr, *11*, 35
import_batch, *4, 5*, *13*, *34*, 38, *48*, *50*
insert_treatments, *10*, 40, *54, 55*, *67*

loess.as, *17*

plot_by_subject, 43, *45*, *71*
plot_viewr_trajectories, *43*, 44, *71*

quick_separate_trajectories, *19*, *25*, 45, *52, 53*, *56*, *62*, *64*, *66*, *69–71*

rad_2_deg, *8*, *15*, *18*, *21–23*, *29*, *31*, 47
read_flydra_mat, *4*, *34*, *39*, 48, *50*
read_motive_csv, *4*, *34*, *39*, *48*, 49
redefine_tunnel_center, *19*, *25*, *46*, 50, *53*, *56*, *62*, *64*, *66*, *69–71*
relabel_viewr_axes, *19*, *25*, *46*, *52*, 52, *56*, *62*, *64*, *66*, *69–71*
remove_duplicate_frames, *10*, *41*, 54, *55*, *67*
remove_vel_anomalies, *10*, *41*, *54*, 55, *67*
rename_viewr_characters, *19*, *25*, *46*, *52, 53*, 56, *62*, *64*, *66*, *69–71*
rescale_tunnel_data, 57
rm_by_trajnum, 58
rotate_tunnel, *19*, *25*, *46*, *52, 53*, *56*, 60, *64*, *66*, *69–71*

section_tunnel_by, 62
select_x_percent, *19*, *25*, *46*, *52, 53*, *56*, *62*, 63, *66*, *69–71*
separate_trajectories, *19*, *25*, *46*, *52, 53*, *56*, *62*, *64*, 65, *69–71*
set_traj_frametime, *10*, *41*, *54, 55*, 67
standardize_tunnel, *19*, *25*, *46*, *52, 53*, *56*, *62*, *64*, *66*, 68, *70, 71*

trim_tunnel_outliers, *19*, *25*, *46*, *52, 53*, *56*, *62*, *64*, *66*, *69*, 69, *71*

visualize_frame_gap_choice, *19*, *25*, *43*, *45, 46*, *52, 53*, *56*, *62*, *64*, *66*, *69, 70*, 71