Package ‘rubitrail’

September 19, 2017

Version 2.0.1

Date 2016-09-20

Maintainer Joe Gallagher <joedgallagher@gmail.com>

Title rubitrail, a package to analyse the output of UbiTrail, an open-source video tracking program for analysis of animal locomotion.

Description This package provides tools to read, analyse and represent output data from UbiTrail. Used in combination, UbiTrail and rubitrail provide a complete toolkit to quantify single animal motion in arenas. Please visit the website for complete documentation and tutorials.

Depends R (>= 3.2.5)

Imports MASS, plotrix, plyr

Suggests rtiff

LazyLoad TRUE

License GPL (>=3.0)

URL http://ubitrail.sourceforge.net/

Collate 'addFactor.R' 'linearInterpolate.R' 'utils.R' 'medianFilter.R' 'removeOutliers.R' 'calcActivity.R' 'calcTurning.R' 'calcPosition.R' 'calcSpeed.R' 'metrics.R' 'calcFPS.R' 'loadFile.R' 'basic.R' 'getMinCircle2.R' 'getNullAreas.R' 'lensCorrection.R' 'plotHeatmap.R' 'plotPosition.R' 'stats.R' 'tenebrio_basic-data.R' 'toDF.R' 'weevils_basic-data.R'

RoxygenNote 6.0.1

NeedsCompilation no

Author Quentin Geissmann [aut, cre],
Joe Gallagher [aut]

R topics documented:

`calcFPS` *Calculates the framerate of a time series, in frames per second.*

Description

Calculates the modal number of frames per second from a matrix, dataframe or list of time series.

Usage

```
calcFPS(dat)
```

Arguments

`dat`	a matrix or dataframe, or list of matrices or dataframes, that contain a variable called `'time'`. Data should not not yet be resampled, e.g. by `rubitLinearInterpolate`

Value

a numeric: the calculated framerate, in frames per second. For a single time series, the median framerate is returned; for a list of time series, the mode of individual framerates is returned.

Note

This function is intended for outputs from the 'rubitrail' package.

See Also

`rubitBasic` and `rubitMetrics` for information on re-encoding a new framerate into tracking data.

Examples

```
data(weevils_raw)

### Framerate of area '08'
calcFPS(weevils_raw[['08']])

### Framerate of each individual area
sapply(weevils_raw, calcFPS)

### Modal framerate of all areas combined
calcFPS(weevils_raw)
```

`lensCorrection` *Corrects lens distortion in raw X,Y-coordinates.*

Description

Corrects lens distortion using the formula $R = r(ar^3 + br^2 + cr + d)$, where r is the distance to the image center and R is the equivalent radius for the corrected image.

Usage

```
lensCorrection(m, imWidth, imHeight, a = 0, b = 0, c = 0)
```

Arguments

`m`	a matrix with X,Y-coordinate data labelled 'X' and 'Y'.
`imWidth, imHeight`	the width and height of the original image.
`a, b, c`	parameters used to correct lens distortion.

Value

The inputted matrix with corrected X,Y-coordinates.

Note

Ready-made undistortion parameters for particular cameras can be found online (e.g. `http://sourceforge.net/projects/hugin/files/PTLens%20Database`). Parameters can be estimated manually using a checkerboard calibration image and the undistortion feature in image manipulation programs such as ImageMagick (`http://imagemagick.org`), or automatically using Hugin's lens calibration tool (`http://wiki.panotools.org/Calibrate_lens_gui`). See `http://www.imagemagick.org/Usage/lens/` for more information on lens correction.

Examples

```
data(tenebrio_basic)

tenebrio_basic <- lensCorrection(tenebrio_basic, 640, 480, b = -0.022)
```

`rubitBasic` *Reads a raw UbiTrail result file and returns filtered X,Y-trajectories.*

Description

This function smooths, interpolates and resamples X,Y-trajectories from raw tracking before calculating the distance between successive positions. These data can then be used with `rubitMetrics` to extract a range of behavioural metrics.

Usage

```
rubitBasic(FILE, scale = 1, hz = 30, start_at = NA, end_at = NA,
  adj_fps = NA, k = 15, p = 0.001, nmin = k * 10, a = 0, b = 0,
  c = 0, filterFUN = rubitMedianFilter,
  interpFUN = rubitLinearInterpolate, verbose = FALSE)
```

Arguments

`FILE`	a .csv result file outputted by UbiTrail.
`scale`	a numeric to calibrate the true spatial scale, in pixels per mm. At the default value, measurements are returned in pixels.
`hz`	the frequency of resampling, in Hz. This argument is passed to the interpolation function.
`start_at, end_at`	the desired start and end times to interpolate and/or cut data to, in minutes.
`adj_fps`	encodes a new framerate, in Hz.
`k`	the level of smoothing for raw trajectories. This argument is passed to the filter function.
`p`	the threshold used to remove outliers, as a proportion of the overall likelihood distribution; e.g. for `p = 0.01`, the largest 1% of outliers will be removed. See `rubitRemoveOutliers` for more information.
`nmin`	the minimal number of reads. If not enough reads are presents in an area, an empty matrix is returned.
`a, b, c`	parameters to correct lens distortion. See `lensCorrection` for more information.
`filterFUN`	the filter function to be used. It must have the same arguments as `rubitMedianFilter`.
`interpFUN`	the interpolation function to be used. It must have the same arguments as `rubitLinearInterpolate`.
`verbose`	logical; if TRUE, the function will print messages at every step.

Value

A list of numerical matrices, with each matrix corresponding to an area. The attributes of list contain metadata about the original video file and attributes of each matrix contain information on the dimensions of the area.

Note

Re-encoding a new framerate with `adj_fps` can correct potential errors made during video recording and/or tracking analysis. Check that the value returned by `calcFPS` matches the calculated framerate of the original video (e.g. using the 'ffprobe' function in FFmpeg [`https://ffmpeg.org/`].

See Also

`rubitMetrics` for extracting behavioural metrics from processed trajectories outputted by this function. `rubitToDF` converts the list (or list of lists) returned by this function into a dataframe for ease of further analysis. See `rubitLinearInterpolate`, `rubitMedianFilter`, and `rubitRemoveOutliers` to understand the different steps of processing used in this function. See codecalcFPS for calculating the framerate of data and see `lensCorrection` for more information on lens distortion.

Examples

```
### Read a single UbiTrail results file:
###---------------------------------------
## Locate raw data example including with package
FILE <- system.file("extdata", "tenebrio_ubitrail.csv.gz", package = "rubitrail")

tenebrio_basic <- rubitBasic(FILE, scale= 2.08, hz = 30, start_at = 0, end_at = 60, adj_fps = 19.05, k = 21, a = -

## See general metadata:
attributes(tenebrio_basic)

## See information on area '01':
attributes(tenebrio_basic[['01']])


### Read a list of UbiTrail results files:
###---------------------------------------
## Create a filelist of all UbiTrail results files in a directory:
# filelist <- list.files()

## Apply function over filelist:
lapply(filelist, rubitBasic, scale= 2.08, hz = 30, start_at = 0, end_at = 60, verbose = TRUE)
```

`rubitCalcActivity` *Calculates the frequency and duration of mobile / stationary phases*

Description

Defines stationary and mobile phases by determining the number of consecutive timepoints in which movement speeds are below (stationary; FALSE) or above (mobile; TRUE) a speed threshold, `min_speed`. The window size, `window`, defines the number of consecutive timepoints required to change the activity phase, in seconds.

Usage

```
rubitCalcActivity(m, window = 1, min_speed = 0.1, simple = FALSE)
```

Arguments

`m`	a matrix containing processed tracking data outputted by `rubitBasic`
`window`	the window size used to define changes in activity, in seconds.
`min_speed`	the minimum speed threshold used to define changes in activity below which no movement is inferred, in mm/second.
`simple`	logical; if FALSE, more complex information on average distance, velocity and position is returned for individual runs.

Value

A new matrix containing information on runs for the inputted area.

See Also

`rubitMetrics` to understand the different steps of processing. This function uses the run length encoding function, `rle()`, from base R.

Examples

```
data(tenebrio_basic)

### Apply the function over all areas in list
sapply(tenebrio_basic, rubitCalcActivity, window = 3, simple = TRUE)
```

`rubitCalcPosition` *Calculate positional information (e.g. thigmotaxis, exploration) from a circular area.*

Description

Divides a circular area into inner and outer zones in order to quantify thigmotaxis (distance from area perimeter), and divides a circular area into any number of equally-sized grid cells in order to quantify exploration (number of unique area grid cells visited).

Usage

```
rubitCalcPosition(m, scale = 1, area_rad = NA, thigmo_dist = NA,
  n_radials = 1, n_slices = 1, n_bootstraps = 20)
```

Arguments

`m`	a matrix containing processed tracking data outputted by `rubitBasic`.
`scale`	a numeric to calibrate the true spatial scale, in pixels per mm. If `scale` == 1, measurements are returned in pixels. This value should match that used in `rubitBasic`.
`area_rad`	the minimum radius of the area. If an area shows insufficient movement to define a minimum enclosing circle of at least this radius, then a new minimum enclosing circle is calculated using `area_rad` and area metainformation stored in `attributes(m)`. This unit is defined in pixels unless `scale` != 1.
`thigmo_dist`	the distance from the boundary perimeter defined as being central (i.e. not thigmotaxis). If thigmo_dist = NA, thigmotaxis is defined as movement in the outer 50% of the area (i.e. > $R/sqrt(2)$ from the area centre, where R is the radius of the area). This unit is defined in pixels unless `scale` != 1.
`n_radials`	the number of concentric circles to divide a circular area into.
`n_slices`	the number of slices to divide a circular area into.
`n_bootstraps`	the number of random data samples used to calculate the minimum enclosing circle defining each circular area.

Value

The input matrix with additional positional information added for each timepoint.

See Also

`rubitPlotPosition` to visualise positional information, and `rubitMetrics` to understand the different steps of processing.

Examples

```
data(tenebrio_basic)

### Divide circular area into 96 cells, and define thigmotaxis
### as movement within 20mm of the area perimeter.
sapply(tenebrio_basic, rubitCalcPosition, n_radials = 8, n_slices = 12, thigmo_dist = 20)
```

`rubitCalcSpeed` *Calculates speed and acceleration from processed tracking data.*

Description

Calculates raw and and smoothed (using a rolling median) speed and acceleration from smoothed tracking data.

Usage

```
rubitCalcSpeed(m, window = 21)
```

Arguments

`m`	a matrix containing processed tracking data outputted by `rubitBasic`.
`window`	the size of the rolling median window used to smooth speed and acceleration, in frames.

Value

The inputted matrix with additional information added for speed (in mm/s) and acceleration (in mm/s^2).

See Also

`rubitMetrics` to understand the different steps of processing.

Examples

```
data(tenebrio_basic)

### Apply the function over all areas in list
sapply(tenebrio_basic, rubitCalcSpeed, window = 1)
```

`rubitCalcTurning` *Resamples tracking data and calculates absolute and relative turning angles.*

Description

Calculates turning angle between successive vectors of movement by resampling data to minimise jitter, returning a matrix containing both absolute and relative angles in degrees.

Usage

```
rubitCalcTurning(m, resample = 1)
```

Arguments

m	a matrix containing processed tracking data outputted by rubitBasic.
resample	the number of seconds over which to resample tracking data and calculate a new trajectory and turning angle from.

Value

A new matrix containing information on turning angles for the inputted area.

See Also

rubitMetrics to understand the different steps of processing.

Examples

```
data(tenebrio_basic)
### Apply the function over all areas in list
sapply(tenebrio_basic, rubitCalcTurning, resample = 1)
```

rubitLinearInterpolate

Resamples and interpolates data to achieve regular time intervals, and trims and/or extends tracking data to fit within desired start and end timepoints.

Description

X,Y-positions are returned at a regular time interval using linear interpolation. Data can be trimmed to fit within desired start and end timepoints, or new positions inferred for additional timepoints (before or after object tracking).

Usage

```
rubitLinearInterpolate(m, hz = 30, start_at = NA, end_at = NA,
  adj_fps = NA, minRow = 11)
```

Arguments

m	a numerical matrix corresponding to an area.
hz	the desired resampling frequency, in Hz.
start_at, end_at	the starting / ending times to interpolate and/or cut data to, in minutes.
adj_fps	encodes a new framerate, in Hz.
minRow	an integer defining the minimal number of reads. If less than minRow reads are present in m, the function returns an empty matrix.

Value

A trimmed and resampled numerical matrix based upon the input data. The attributes of the input matrix are copied to the new matrix.

Note

X,Y-positions are defined as 'NA' between the `start_at` time and the point of first object tracking. For terminally interpolated timepoints (i.e. between last object detection and the `end_at` time), the last known X,Y-position is repeated. Re-encoding a new framerate with `adj_fps` can correct potential errors made during video recording and/or tracking analysis. Check that the value returned by `calcFPS` matches the calculated framerate of the original video (e.g. using the 'ffprobe' function in FFmpeg [`https://ffmpeg.org/`].

See Also

`rubitMedianFilter` to smooth data before interpolation.

Examples

```
data(weevils_raw)

### Interpolation before filtering
w1 <- lapply(weevils_raw, rubitLinearInterpolate, hz = 50)
plot(w1[['08']][,'X'] ~ w1[['08']][,'Y'], asp=1, type='l')

### Interpolation after filtering
w19 <- lapply(weevils_raw, rubitRemoveOutliers, p = 0.001)
w19 <- lapply(w19, rubitMedianFilter, k = 19)
w19 <- lapply(w19, rubitLinearInterpolate, hz = 50)
plot(w19[['08']][,'X'] ~ w15[['08']][,'Y'], asp=1, type='l')
```

`rubitLoadFile` *Reads raw data from a UbiTrail result file.*

Description

A general function to read raw tracking data and meta information from a .csv result file outputted by UbiTrail.

Usage

```
rubitLoadFile(FILE, verbose = FALSE)
```

Arguments

`FILE`	a CSV result file outputted by UbiTrail.
`verbose`	logical; if TRUE, the function will print messages at every step.

Value

A list of numerical matrices with each matrix corresponding to an area.

Note

The returned list contains a numerical matrix for each area. The attributes of list contain metadata and additional information is present in each of the attributes of each matrix.

See Also

`rubitBasic` and `rubitMetrics` for more further processing and analysis of tracking data.

Examples

```
## Locate raw data example including with package
FILE <- system.file("extdata", "tenebrio_ubitrail.csv.gz", package = "rubitrail")

tenebrio_raw <- rubitLoadFile(FILE)

###See general metadata:
attributes(tenebrio_raw)

###See how many reads in each area:
summary(tenebrio_raw)

###See informations about the area named '08':
attributes(tenebrio_raw[['08']])
```

`rubitMedianFilter` *Apply a running median filter on an an area matrix.*

Description

This function is used to eliminate outliers from an area matrix.

Usage

```
rubitMedianFilter(m, k = 15)
```

Arguments

`m`	a numerical matrix corresponding to an area.
`k`	an integer specifying the size of the smoothing window. *It must be odd.*

Value

A numerical matrix of the same dimension as m.

See Also

`rubitLinearInterpolate` to get regular sampling after filtering.

Examples

```
data(weevils_raw)

### Apply different 'k' values to a list of area matrices:
w15 <- lapply(weevils_raw, rubitMedianFilter, k = 15)
w101 <- lapply(weevils_raw, rubitMedianFilter, k = 101)

### See impacts of smoothing parameter 'k' on a trajectory:
## raw trajectory
plot(weevils_raw[['08']][1:100,'X'] ~ weevils_raw[['08']][1:100,'Y'], asp=1, type='l')
## acceptable level of smoothing
lines(w15[['08']][1:100,'X'] ~ w15[['08']][1:100,'Y'],col='green')
## oversmoothed
lines(w101[['08']][1:100,'X'] ~ w101[['08']][1:100,'Y'],col='red')
```

`rubitMetrics` *A wrapper combining several functions which calculate a range of behavioural metrics from processed X,Y-trajectories.*

Description

This function loads processed X,Y-trajectories returned from `rubitBasic`, and returns data on metrics on velocity, position (e.g. thigmotaxis, exploration), turning angle, and activity (i.e. pause / walk events).

Usage

```
rubitMetrics(LIST, scale = 1, speed_smooth = 21, turn_resample_rate = 1,
  activity_window = 1, activity_min_speed = 0.1, n_radials = 1,
  n_slices = 1, area_rad = NA, thigmo_dist = NA, n_bootstraps = 20,
  verbose = FALSE)
```

Arguments

`LIST`	a list of matrices containing processed X,Y-trajectories returned from `rubitBasic`.
`scale`	a numeric to calibrate the true spatial scale, in pixels per mm. At the default value, measurements are returned in pixels.
`speed_smooth`	the size of the rolling median window used to smooth speed and acceleration, in frames.
`turn_resample_rate`	the number of seconds over which to resample X,Y-coordinate data and calculate turning angle from.

`activity_window`	the window size used to define changes in activity, in seconds.
`activity_min_speed`	the minimum speed threshold used to define changes in activity below which no movement is inferred, in mm/second.
`n_radials`	the number of concentric circles to divide a circular area into.
`n_slices`	the number of slices to divide a circular area into.
`area_rad`	the minimum radius of the area. If an area shows insufficient movement to define a minimum enclosing circle of at least this radius, then a new minimum enclosing circle is calculated using `area_rad` and area metainformation stored in `attributes(m)`. This unit is defined in pixels unless `scale` != 1.
`thigmo_dist`	the distance from the boundary perimeter defined as being central (i.e. not thigmotaxis), in mm. If thigmo_dist = NA, thigmotaxis is defined as movement in the outer 50% of the area (i.e. > R/sqrt(2) from the area centre, where R is the radius of the whole area).
`n_bootstraps`	the number of random data samples used to calculate the minimum enclosing circle defining each circular area.
`verbose`	logical; if TRUE, the function will print messages at every step.

Value

A list of numerical matrices. Each matrix corresponds to an area.

Note

The returned list contains a numerical matrix for each area. The attributes of list contain metadata and additional information is present in each of the attributes of each matrix. Re-encoding a new framerate with `adj_fps` can correct potential errors made during video recording and/or tracking analysis. Check that the value returned by `calcFPS` matches the calculated framerate of the original video (e.g. using the 'ffprobe' function in FFmpeg [`https://ffmpeg.org/`].

See Also

`rubitToDF` for converting the returned lists (or list of lists) to a dataframe for ease of further analysis. See `rubitBasic`, `rubitCalcSpeed`, `rubitCalcPosition`, `rubitCalcTurning`, and `rubitCalcActivity` to understand the different steps of processing used in this function.

Examples

```
### Extract metrics from a single UbiTrail results file:
###-----------------------------------------------------
## Locate raw data example included with package
FILE <- system.file("extdata", "tenebrio_ubitrail.csv.gz", package = "rubitrail")

## Basic processing
tenebrio_basic <- rubitBasic(FILE, scale= 2.08, hz = 30, start_at = 0, end_at = 60, adj_fps = 19.05, k = 21, a = -

## Extract metrics from processed data
```

```
rubitMetrics(tenebrio_basic, scale = 2.08, n_radials = 8, n_slices = 12, area_rad = 90, thigmo_dist = 20, verbose


### Extract metrics from multiple UbiTrail results files:
###------------------------------------------------------
## Create a filelist of all UbiTrail results files in a directory:
#filelist <- list.files()

## Apply basic processing function over filelist
#basic_data <- lapply(filelist, rubitBasic, scale = 2.08, hz = 30, start_at = 0, end_at = 60, verbose = TRUE)

## Apply metrics function over processed list of data
#lapply(basic_data, rubitMetrics, scale = 2.08, n_radials = 8, n_slices = 12, area_rad = 90, thigmo_dist = 20, ver
```

rubitPlotHeatmap *Plot a 2D density estimate of all areas.*

Description

This function represents the relative position density of each agent during the entire experiment.

Usage

```
rubitPlotHeatmap(l, refImg = NA, resol = 50, h = 10,
  palet = rubitTransCol(50, 0.5))
```

Arguments

`l`	a list of area matrices returned by `rubitLoadFile` or `rubitBasic`.
`refImg`	the path to a reference TIFF image (see details).
`resol`	the resolution of the kernel density estimate.
`h`	the bandwidth ratio (see details).
`palet`	a vector of colours.

Note

The list `l` can be obtained by `rubitBasic`. `refImg` is an image of the same size as the video frames. If specified (*i.e.* if `refImg != NA`), the function will use the image as background for the plot. An easy way to obtain such an image is to ask UbiTrail to save the first frame (by ticking the corresponding box in the GUI). For each area, the kernel estimate density function will use a bandwidth `bw = W/h`, where `W` is the width of the area.

See Also

`rubitLinearInterpolate` for interpolating data. `MASS::kde2d` is used by this function.

Examples

```
data(tenebrio_basic)

rubitPlotHeatmap(tenebrio_basic)
rubitPlotHeatmap(tenebrio_basic, resol=150)
```

`rubitPlotPosition` *Visualise an individual trajectory and positional information in a circular area.*

Description

Plots a trajectory over a circular area, dividing the area into a number of grid cells of equal size to visualise exploration, and defining an outer perimeter to visualise thigmotaxis.

Usage

```
rubitPlotPosition(m, scale = 1, area_rad = NA, thigmo_dist = NA,
  n_radials = 1, n_slices = 1, n_bootstraps = 20)
```

Arguments

`m`	a matrix containing processed tracking data outputted by `rubitBasic`.
`scale`	a numeric to calibrate the true spatial scale, in pixels per mm. If `scale` == 1, measurements are returned in pixels. This value should match that used in `rubitBasic`.
`area_rad`	the minimum radius of the area. If an area shows insufficient movement to define a minimum enclosing circle of at least this radius, then a new minimum enclosing circle is calculated using `area_rad` and area metainformation stored in `attributes(m)`. This unit is defined in pixels unless `scale` != 1.
`thigmo_dist`	the distance from the boundary perimeter defined as being central (i.e. not thigmotaxis). If thigmo_dist = NA, thigmotaxis is defined as movement in the outer 50% of the area (i.e. > $R/sqrt(2)$ from the area centre, where R is the radius of the area). This unit is defined in pixels unless `scale` != 1.
`n_radials`	the number of concentric circles to divide a circular arena into
`n_slices`	the number of slices to divide a circular arena into
`n_bootstraps`	the number of random data samples used to calculate the minimum enclosing circle defining each circular area.

Value

a plot showing the divided circular area with full trajectory overlaid.

See Also

`rubitCalcPosition` for more on calculating positional information.

Examples

```
data(tenebrio_basic)

### Single plot, with area divided into 96 cells
### and thigmotaxis defined in outer 20mm of area:
my_scale <- 2.08
rubitPlotPosition(tenebrio_basic[['05']], scale = my_scale, thigmo_dist = 20, n_radials = 8, n_slices = 12)

### Print plots for all areas in list to PDF:
#pdf("plots.pdf")
lapply(tenebrio_basic, rubitPlotPosition, scale = my_scale, thigmo_dist = 20, n_radials = 8, n_slices = 12)
#dev.off()
```

`rubitRemoveOutliers` *Removes outliers from tracking data based upon log-likelihood.*

Description

Removes outliers from an area matrix based upon the log-likelihood of tracked X,Y-coordinates.

Usage

```
rubitRemoveOutliers(m, p = 0.001)
```

Arguments

`m`	a numerical matrix corresponding to an area.
`p`	the proportion of least likely X,Y-coordinates to remove based on log-likelihood. For example, for `p = 0.01`, the least likely 1% of points will be removed.

Value

A numerical matrix of the same dimensions as m.

See Also

`rubitLinearInterpolate` to interpolate X,Y-coordinates after removing outliers.

Examples

```
data(weevils_raw)

Remove least likely 0.01\% of points
w_filt <- lapply(weevils_raw, rubitRemoveOutliers, p = 0.001)
```

rubitToDF — *Flattens rubitrail lists into a single dataframe.*

Description

A helper function which flattens a list, or a list of lists, returned by `rubitBasic` and `rubitMetrics` and converts into a dataframe for ease of further analysis.

Usage

```
rubitToDF(LIST, basic = FALSE)
```

Arguments

`LIST`	a list (single result file) or list of lists (multiple results files) returned by `rubitBasic` or `rubitMetrics`.
`basic`	logical defining the type of data contained in `LIST`. Use FALSE for metrics data returned by `rubitMetrics` and TRUE for basic data returned by `rubitBasic`.

Value

A single dataframe for 'basic' input data. A list of three dataframes for 'metrics' input data, with elements named 'speed', 'turning' and 'activity'.

See Also

`rubitBasic` and `rubitMetrics` for information on processing raw trajectories and extracting metrics.

Examples

```
data(tenebrio_basic)

tenebrio_DF <- rubitToDF(tenebrio_basic, basic=TRUE)
```

Index