Package 'moveVis'

March 20, 2019

Type Package

Title Movement Data Visualization

Version 0.10.0

Depends R (>= 3.5.0)

Date 2019-03-18

Description Tools to visualize movement data (e.g. from GPS tracking) and temporal changes of environmental data (e.g. from remote sensing) by creating video animations.

License GPL-3

Encoding UTF-8

RoxygenNote 6.1.1

Imports raster, sp, sf, move, RStoolbox, geosphere, slippymath, lubridate, rlang, ggplot2, cowplot, gifski, magick, av, pbapply, curl, plyr, dplyr, zoo, methods, grDevices

BugReports http://www.github.com/16eagle/moveVis/issues

SystemRequirements ImageMagick, FFmpeg, libav

URL http://movevis.org

Suggests mapview, leaflet, testthat

NeedsCompilation no

Author Jakob Schwalb-Willmann [aut, cre]

Maintainer Jakob Schwalb-Willmann <movevis@schwalb-willmann.de>

Repository CRAN

Date/Publication 2019-03-20 14:00:06 UTC

R topics documented:

`moveVis-package` *Tools to visualize movement data in R*

Description

`moveVis` provides tools to visualize movement data (e.g. from GPS tracking) and temporal changes of environmental data (e.g. from remote sensing) by creating video animations. The `moveVis` package is closely connected to the `move` package and builds up on `ggplot2` grammar of graphics.

Details

The package includes the following functions, sorted by the order they would be applied to create an animation from movement data:

- `df2move` converts a `data.frame` into a `move` or `moveStack` object. This is useful if you do not usually work with the `move` classes and your tracks are present as `data.frames`.
- `align_move` aligns single and multi-individual movement data to a uniform time scale with a uniform temporal resolution needed for creating an animation from it. Use this function to prepare your movement data for animation depending on the temporal resolution that suits your data.
- `subset_move` subsets a `move` or `moveStack` by a given time span. This is useful if you want to create a movement animation of only a temporal subset of your data, e.g. a particular day.
- `get_maptypes` returns a character vector of available map types that can be used with `frames_spatial`. `moveVis` supports OpenStreetMaps and Mapbox basemap imergay. Alternatively, you can provide custom imagery to `frames_spatial`.

- frames_spatial creates a list of ggplot2 maps displaying movement. Each object represents a single frame. Each frame can be viewed or modified individually. The returned list of frames can be animated using animate_frames.
- frames_graph creates a list of ggplot2 graphs displaying movement-environment interaction. Each object represents a single frame. Each frame can be viewed or modified individually. The returned list of frames can be animated using animate_frames.
- add_gg adds ggplot2 functions (e.g. to add layers such as points, polygons, lines, or to change scales etc.) to the animation frames created with frames_spatial or frames_graph. Instead of creating your own ggplot2 functions, you can use one of the other moveVis add_ functions:
- add_labels adds character labels such as title or axis labels to animation frames created with frames_spatial or frames_graph.
- add_scalebar adds a scalebar to the animation frames created with frames_spatial or frames_graph.
- add_northarrow adds a north arrow to the animation frames created with frames_spatial or frames_graph.
- add_progress adds a progress bar to animation frames created with frames_spatial or frames_graph.
- add_timestamps adds timestamps to animation frames created with frames_spatial or frames_graph.
- add_text adds static or dynamically changing text to the animation frames created with frames_spatial or frames_graph.
- add_colourscale adjusts the colour scales of the animation frames created with frames_spatial and custom map imagery.
- join_frames side-by-side joins the ggplot2 objects of two or more frames lists of equal lengths into a single list of ggplot2 objects per frame using plot_grid. This is useful if you want to side-by-side combine spatial frames returned by frames_spatial with graph frames returned by frames_graph.
- suggest_formats returns a selection of suggested file formats that can be used with out_file of animate_frames on your system.
- animate_frames creates an animation from a list of frames computed with frames_spatial or frames_graph.
- view_spatial displays movement tracks on an interactive mapview or leaflet map.

Author(s)

Jakob Schwalb-Willmann. Maintainer: Jakob Schwalb-Willmann, moveVis@schwalb-willmann.de

See Also

Useful links:

- http://movevis.org
- Report bugs at http://www.github.com/16eagle/moveVis/issues

add_colourscale *Add scale to frames*

Description

This function adjusts the colour scales of the animation frames created with frames_spatial and custom map imagery.

Usage

```
add_colourscale(frames, type, colours, labels = waiver(),
  legend_title = NULL, verbose = TRUE)
```

Arguments

frames	list of ggplot2 objects, crated with frames_spatial.
type	character, either "gradient" or "discrete". Must be equal to the defintion of argument r_type with which frames have been created (see frames_spatial).
colours	character, a vector of colours. If type = "discrete", number of colours must be equal to the number of classes contained in the raster imagery with which frames have been created. Provide a named vector to associate map values with colours, e.g. c("1" = "red", "2" = "green", "3" = "black")
labels	character, a vector of labels with the same length as colours. Ignored, if type = "gradient".
legend_title	character, a legend title.
verbose	logical, if TRUE, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
# align movement
m <- align_move(move_data, res = 4, unit = "mins")
```

```
# create spatial frames with frames_spatial:
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)
frames[[100]] # take a look at one of the frames

# default blue is boring, let's change the colour scale of all frames
frames <- add_colourscale(frames, type = "gradient", colours = c("orange", "white", "darkgreen"),
                          legend_title = "NDVI")
frames[[100]]


# let's make up some classification data with 10 classes
r_list <- lapply(r_list, function(x){
  y <- raster::setValues(x, round(raster::getValues(x)*10))
  return(y)
})
# turn fade_raster to FALSE, since it makes no sense to temporally interpolate discrete classes
frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "discrete",
                         fade_raster = FALSE)
frames[[100]]

# now, let's assign a colour per class value to frames
colFUN <- colorRampPalette(c("orange", "lightgreen", "darkgreen"))
cols <- colFUN(10)
frames <- add_colourscale(frames, type = "discrete", colours = cols, legend_title = "Classes")
frames[[100]]
```

---

add_gg *Add* ggplot2 *function to frames*

Description

This function adds ggplot2 functions (e.g. to add layers, change scales etc.) to the animation frames created with frames_spatial.

Usage

```
add_gg(frames, gg, data = NULL, ..., verbose = T)
```

Arguments

frames	list of ggplot2 objects, crated with frames_spatial.
gg	ggplot2 expressions (see details), either as

- an expression of one or a list of ggplot2 functions to be added to every frame,
- a list of such of the same length as frames to add different ggplot2 expressions per frame

data — optional data used by gg (see details), either

- an object of any class, e.g. a data.frame, used by gg that will be added to all frames,
- a list, e.g. of multiple data.frames, with length of frames to add different data to each frame.

... — additional (non-iterated) objects that should be visible to gg.

verbose — logical, if TRUE, messages on the function's progress are displayed (default).

Details

Agrument gg expects ggplot2 functions handed over as expressions (see expr) to avoid their evaluation before thex are called for the correct frame. Simply wrap your ggplot2 function into expr() and supply it to gg. To add multiple ggplot2 functions to be applied on every frame, supply an expression containing a list of ggplot2 functions (e.g. expr(list(geom_label(...), geom_text(...)))). This expression would be added to all frames. To add specific ggplot2 functions per frame, supply a list of expressions of the same length as frames. Each expression may contain a list of ggplot2 functions, if you want to add multiple functions per frame.

If data is used, the ggplot2 expressions supplied with gg can use the object by the name data for plotting. If data is a list, it must be of the same length as frames. The list will be iterated, so that functions in gg will have access to the individual objects within the list by the name data per each frame. If the data you want to display is does not change with frames and may only be a character vector or similiar, you may not need data, as you can supply the needed values within the expression supplied through gg.

If you supply gg as a list of expressions for each frame and data as a list of objects (e.g. data.frames) for each frame, each frame will be manipulated with the corresponding ggplot2 function and the corresponding data.

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)
library(ggplot2)

data("move_data", "basemap_data")
# align movement
m <- align_move(move_data, res = 4, unit = "mins")


frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor")
frames[[100]] # take a look at one of the frames

# let's draw a polygon on frames:
data <- data.frame(x = c(8.917, 8.924, 8.924, 8.916, 8.917),
                   y = c(47.7678, 47.7675, 47.764, 47.7646, 47.7678))

frames = add_gg(frames, gg = expr(geom_path(aes(x = x, y = y), data = data,
                                        colour = "red", linetype = "dashed")), data = data)

# add some text
frames <- add_text(frames, "Static feature", x = 8.9205, y = 47.7633,
                    colour = "black", size = 3)
frames[[100]]

# add_gg can also be used iteratively to manipulate each frame differently.
# Let's create unique polygons per frame:

# create data.frame containing corner coordinates
data <- data.frame(x = c(8.96, 8.955, 8.959, 8.963, 8.968, 8.963, 8.96),
                   y = c(47.725, 47.728, 47.729, 47.728, 47.725, 47.723, 47.725))
# make a list from it by replicating it by the length of frames
data <- rep(list(data), length.out = length(frames))

# now alter the coordinates to make them shift
data <- lapply(data, function(x){
  y <- rnorm(nrow(x)-1, mean = 0.00001, sd = 0.0001)
  x + c(y, y[1])
})

# draw each individual polygon to each frame
frames = add_gg(frames, gg = expr(geom_path(aes(x = x, y = y), data = data,
                                            colour = "black")), data = data)

# add a text label
frames <- add_text(frames, "Dynamic feature", x = 8.959, y = 47.7305,
                   colour = "black", size = 3)
frames[[100]]

# animate frames to see how the polygons "flip"
animate_frames(frames, out_file = tempfile(fileext = ".mov"))
```

```
# you can use add_gg on any list of ggplot2 objects,
# also on frames made using frames_gr
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]

frames.gr <- frames_graph(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                          fade_raster = TRUE, graph_type = "hist", val_by = 0.01)
frames.gr[[100]]
# manipulate the labels, since they are very dense:
# just replace the current scale
frames.gr <- add_gg(frames.gr, expr(scale_x_continuous(breaks=seq(0,1,0.1),
                                    labels=seq(0,1,0.1), expand = c(0,0))))
frames.gr[[100]]
```

`add_labels` *Add labels to frames*

Description

This function adds character labels such as title or axis labels to animation frames created with `frames_spatial`.

Usage

```
add_labels(frames, title = waiver(), subtitle = waiver(),
  caption = waiver(), tag = waiver(), x = waiver(), y = waiver(),
  verbose = TRUE)
```

Arguments

`frames`	list of ggplot2 objects, crated with `frames_spatial`.
`title`	character, frame title. If `NULL`, an existing title of `frames` is removed. If `waiver()` (default, see `ggplot2::waiver()`), an existing title of `frames` is kept.
`subtitle`	character, frame subtitle. If `NULL`, an existing title of `frames` is removed. If `waiver()` (default, see `ggplot2::waiver()`), an existing title of `frames` is kept.
`caption`	character, frame caption. If `NULL`, an existing title of `frames` is removed. If `waiver()` (default, see `ggplot2::waiver()`), an existing title of `frames` is kept.
`tag`	character, frame tag. If `NULL`, an existing title of `frames` is removed. If `waiver()` (default, see `ggplot2::waiver()`), an existing title of `frames` is kept.
`x`	character, label of the x axis. If `NULL`, an existing title of `frames` is removed. If `waiver()` (default, see `ggplot2::waiver()`), an existing title of `frames` is kept.
`y`	character, label of the y axis. If `NULL`, an existing title of `frames` is removed. If `waiver()` (default, see `ggplot2::waiver()`), an existing title of `frames` is kept.
`verbose`	logical, if `TRUE`, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames using a custom NDVI base layer
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# add labels to frames:
frames <- add_labels(frames, title = "Example animation using moveVis::add_labels()",
                 subtitle = "Adding a subtitle to frames created using frames_spatial()",
                 caption = "Projection: Geographical, WGS84. Sources: moveVis examples.",
                     x = "Longitude", y = "Latitude")
# have a look at one frame
frames[[100]]
```

---

add_northarrow *Add north arrow to frames*

Description

This function adds a north arrow to the animation frames created with frames_spatial.

Usage

```
add_northarrow(frames, height = 0.05, position = "bottomright",
  x = NULL, y = NULL, colour = "black", size = 1,
  label_text = "N", label_margin = 0.4, label_size = 5,
  verbose = TRUE)
```

Arguments

frames	list of ggplot2 objects, crated with frames_spatial.
height	numeric, height of the north arrow in a range from 0 to 1 as the proportion of the overall height of the frame map. Default is 0.5.
position	character, position of the north arrow on the map. Either "bottomleft", "upperleft", "upperright", Ignored, if x and y are set.
x	numeric, position of the bottom left corner of the north arrow on the x axis. If not set, position is used to calculate the position of the north arrow.
y	numeric, position of the bottom left corner of the north arrow on the y axis. If not set, position is used to calculate the position of the north arrow.
colour	character, colour. Default is "black".
size	numeric, arrow size. Default is 1.
label_text	character, text below the north arrow. Default is "North".
label_margin	numeric, margin between label and north arrow as a proportion of the size of the north arrow. Default is 0.3.
label_size	numeric, label font size. Default is 4.
verbose	logical, if TRUE, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames using a custom NDVI base layer
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# add a north arrow to frames:
```

```
frames.a <- add_northarrow(frames)
frames.a[[100]]

# or in white at another position
frames.b <- add_northarrow(frames, colour = "white", position = "bottomleft")
frames.b[[100]]
```

---

add_progress *Add progress bar to frames*

Description

This function adds a progress bar to animation frames created with frames_spatial.

Usage

```
add_progress(frames, colour = "grey", size = 1.8, verbose = TRUE)
```

Arguments

frames	list of ggplot2 objects, crated with frames_spatial.
colour	character, progress bar colour.
size	numeric, progress bar line size..
verbose	logical, if TRUE, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames using a custom NDVI base layer
```

```
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# add a progress bar:
frames.a <- add_progress(frames)
frames.a[[100]]

# or in red and larger
frames.b <- add_progress(frames, colour = "red", size = 2.5)
frames.b[[100]]
```

`add_scalebar` *Add scalebar to frames*

Description

This function adds a scalebar to the animation frames created with `frames_spatial`.

Usage

```
add_scalebar(frames, distance = NULL, height = 0.015,
  position = "bottomleft", x = NULL, y = NULL, colour = "black",
  label_margin = 1.2, verbose = TRUE)
```

Arguments

`frames`	list of `ggplot2` objects, crated with `frames_spatial`.
`distance`	numeric, optional. Distance displayed by the scalebar in km. By default, the displayed distance is calculated automatically.
`height`	numeric, height of the scalebar in a range from 0 to 1 as the proportion of the overall height of the frame map. Default is 0.015.
`position`	character, position of the scalebar on the map. Either `"bottomleft"`, `"upperleft"`, `"upperright"`, `"b` Ignored, if `x` and `y` are set.
`x`	numeric, position of the bottom left corner of the scalebar on the x axis. If not set, `position` is used to calculate the position of the scalebar.
`y`	numeric, position of the bottom left corner of the scalebar on the y axis. If not set, `position` is used to calculate the position of the scalebar.
`colour`	character, colour of the distance labels. Default is `"black"`.
`label_margin`	numeric, distance of the labels to the scalebar as a proportion of the height of the scalebar (e.g. if set to 2, the labels will be positioned with a distance to the scalebar of twice the scalebar height).
`verbose`	logical, if `TRUE`, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames using a custom NDVI base layer
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# add a scale bar to frames:
frames.a <- add_scalebar(frames)
frames.a[[100]]

# or in white at another position
frames.b <- add_scalebar(frames, colour = "white", position = "bottomright")
frames.b[[100]]

# or with another height
frames.c <- add_scalebar(frames, colour = "white", position = "bottomright", height = 0.025)
frames.c[[100]]
```

---

add_text *Add static or dynamic text to frames*

Description

This function adds static or dynamically changing text to the animation frames created with frames_spatial.

Usage

```
add_text(frames, labels, x, y, colour = "black", size = 3,
  type = "text", verbose = TRUE)
```

Arguments

frames	list of ggplot2 objects, crated with frames_spatial.
labels	character, text to be added to frames. Either a single character value or a character vector of same length as frames.
x	numeric, position of text on the x scale. Either a single numeric value or a numeric vector of same length as frames.
y	numeric, position of text on the y scale. Either a single numeric value or a numeric vector of same length as frames.
colour	character, the text colour(s). Either a single character value or a character vector of same length as frames.
size	numeric, the text size(s). Either a single numeric value or a numeric vector of same length as frames.
type	character, either "text" to draw text or "label" to draw text inside a box.
verbose	logical, if TRUE, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames using a custom NDVI base layer
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)
```

```
# add text somewhere to all frames:
frames.a <- add_text(frames, "Water area", x = 8.959, y = 47.7305,
                     colour = "white", size = 3)
frames.a[[100]]

# or use the ggplot2 "label" type:
frames.b <- add_text(frames, "Water area", x = 8.959, y = 47.7305,
                     colour = "black", size = 3, type = "label")
frames.b[[100]]
```

`add_timestamps` *Add timestamps to frames*

Description

This function adds timestamps to animation frames created with `frames_spatial`.

Usage

```
add_timestamps(frames, m, x = NULL, y = NULL, ..., verbose = TRUE)
```

Arguments

`frames`	list of `ggplot2` objects, crated with `frames_spatial`.
`m`	`move` or `moveStack` used to create `frames` with `frames_spatial` of uniform time scale and time lag, e.g. prepared with `align_move`.
`x`	numeric, optioanl, position of timestamps on the x scale. By default, timestamps will be displayed in the top center.
`y`	numeric, optioanl, position of timestamps on the y scale.
`...`	optional, arguments passed to `add_text`, such as `colour`, `size`, `type`.
`verbose`	logical, if `TRUE`, messages on the function's progress are displayed (default).

Value

List of frames.

Author(s)

Jakob Schwalb-Willmann

See Also

`frames_spatial` `frames_graph` `animate_frames`

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames using a custom NDVI base layer
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# add timestamps as text
frames.a <- add_timestamps(frames, m, type = "text")
frames.a[[100]]

# or use the ggplot2 "label" type:
frames.b <- add_timestamps(frames, m, type = "label")
frames.b[[100]]
```

`align_move` *Align movement data*

Description

This function aligns movement data to a uniform time scale with a uniform temporal resolution throughout the complete movement sequence. This prepares the provided movement data to be interpretable by `frames_spatial`, which necessitates a uniform time scale and a consistent, unique temporal resolution for all moving individuals to turn recording times into frame times.

Usage

```
align_move(m, res = "min", digit = "min", unit = "secs",
  spaceMethod = "greatcircle")
```

Arguments

`m`	`move` or `moveStack`, which is allowed to contain irregular timestamps and diverging temporal resolutions to be aligned (see `df2move` to convert a `data.frame` to a `move` object).
`res`	either numeric, representing the temporal resolution, to which `m` should be aligned to (see argument `unit`), or character: • `"min"` to use the smalles temporal resolution of `m` (default)

- "max" to use the largest temporal resolution of m
- "mean" to use the rounded average temporal resolution of m

digit — either numeric, indicating to which digits of a specifc unit (see argument unit) the time scale of m should be aligned (e.g. 0 to align the time scale to second ":00", if unit is set to secs), or character:

- "min" to use the smallest digit of the defined unit (default)
- "max" to use the largest digit of the defined unit
- "mean" to use the rounded average digit of the defined unit

unit — character, either "secs", "mins", "hours", "days", indicating the temporal unit, to which res and digit are referring.

spaceMethod — character, either "euclidean", "greatcircle" or "rhumbline", indicating the interpolation function to be used to interpolate locations of m to the aligned time scale. Interpolation is performed using move::interpolateTime.

Value

Aligned move or moveStack, ready to be used with frames_spatial-

Author(s)

Jakob Schwalb-Willmann

See Also

df2move frames_spatial frames_graph

Examples

```
library(moveVis)
library(move)
data("move_data")

# the tracks in move_data have irregular timestamps and sampling rates.
# print unique timestamps and timeLag
unique(timestamps(move_data))
unique(unlist(timeLag(move_data, units = "secs")))

# use align_move to correct move_data to a uniform time scale and lag using interpolation.
# resolution of 4 minutes (240 seconds) at digit 0 (:00 seconds) per timestamp:
m <- align_move(move_data, res = 240, digit = 0, unit = "secs")
unique(unlist(timeLag(m, units = "secs")))

# resolution of 1 hour (3600 seconds) at digit 0 (:00 seconds) per timestamp:
m <- align_move(move_data, res = 3600, digit = 0, unit = "secs")
unique(unlist(timeLag(m, units = "secs")))

# resolution of 1 hour (15 seconds) at digit 0 (:00 seconds) per timestamp:
m <- align_move(move_data, res = 15, digit = 0, unit = "secs")
unique(unlist(timeLag(m, units = "secs")))
```

```
# resolution of 1 hour:
m <- align_move(move_data, res = 60, unit = "mins")
unique(unlist(timeLag(m, units = "secs")))
```

`animate_frames` *Animate frames*

Description

`animate_frames` creates an animation from a list of frames computed with `frames_spatial`.

Usage

```
animate_frames(frames, out_file, fps = 25, width = 700, height = 700,
  res = 100, display = TRUE, overwrite = FALSE, verbose = TRUE,
  ...)
```

Arguments

`frames`	list of `ggplot2` objects, crated with `frames_spatial`.
`out_file`	character, the output file path, e.g. "/dir/to/file.mov". The file extension must correspond to a file format known by the available renderers of the running system. Use `suggest_formats` to get a vector of suggested known file formats.
`fps`	numeric, the number of frames to be displayed per second. Default is 2.
`width`	numeric, width of the output animation in pixels.
`height`	numeric, height of the output animation in pixels.
`res`	numeric, resolution of the output animation in ppi.
`display`	logical, whether the animation should be displayed after rendering or not.
`overwrite`	logical, wether to overwrite an existing file, if `out_file` is already present.
`verbose`	logical, if `TRUE`, messages on the function's progress are displayed (default).
`...`	additional arguments to be passed to the render function.

Details

An appropriate render function is selected depending on the file extension in `out_file`: For `.gif` files, `gifski::save_gif` is used, for any other (video) format, `av::av_capture_graphics` is used.

Value

None or an R graphics window displaying the animation

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph join_frames

Examples

```
library(moveVis)
library(move)

data("move_data", "basemap_data")
# align movement
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames with frames_spatial:
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# customize
frames <- add_colourscale(frames, type = "gradient",
                      colours = c("orange", "white", "darkgreen"), legend_title = "NDVI")
frames <- add_northarrow(frames, position = "bottomleft")
frames <- add_scalebar(frames, colour = "white", position = "bottomright")

frames <- add_progress(frames)
frames <- add_timestamps(frames, m, type = "label")

# check available formats
suggest_formats()

# animate frames as GIF
animate_frames(frames, out_file = tempfile(fileext = ".gif"))

# animate frames as mov
animate_frames(frames, out_file = tempfile(fileext = ".gif"))
```

---

basemap_data *Exemplary manipulated NDVI data*

Description

This datasetd contains two lists of equal lenghts:

- a list of ten single-layer `raster` objects, representing NDVI images covering the Lake of Constance area.
- a list of made-up times that simulate acquisition times with a temporal resolution, remote sensing scientiest would dream of...

Usage

```
basemap_data
```

Format

List containing two lists of equal lengths: a list of `raster` objects and a list of `POSIXct` times.

Details

This object is used by some `moveVis` examples and unit tests.

Note

All data contained should only be used for testing `moveVis` and are not suitable to be used for analysis or interpretation.

Source

MODIS (MOD13Q1 NDVI)

`deprecated` *Deprecated functions*

Description

Several functions are deprecated due to a rewrite of moveVis with version 0.10.

Usage

```
animate_move(...)

animate_raster(...)

animate_stats(...)

get_formats(...)

get_libraries(...)
```

Arguments

`...`	deprecated arguments.

Details

The new version of moveVis makes it much easier to animate movement data and multi-temporal imagery (see `?moveVis`). You gain more control about the preprocessing of your movement data as well as the visual customization of each animation frame through a more consequent link of `moveVis` to `gpplot2`.

Note

To install the old version of moveVis (0.9.9), see https://github.com/16EAGLE/moveVis/releases/tag/v0.9.9.

See Also

frames_spatial frames_graph join_frames animate_frames

---

df2move *Convert a data.frame into a move or moveStack object*

Description

This function is a simple wrapper that converts a data.frame into a move or moveStack object. Both can be used as inputs to frames_spatial or frames_graph.

Usage

```
df2move(df, proj, x, y, time, track_id = NULL, data = NULL, ...)
```

Arguments

df	data.frame, a data.frame with rows representing observations and columns representing x and y coordinates, time and optionally track IDs, if multiple tracks are contained.
proj	projection, character (proj4string) or CRS object, indicating the projection that the coordinates of df represent.
x	character, name of the column in df that represents x coordinates.
y	character, name of the column in df that represents y coordinates.
time	character, name of the column in df that represents timestamps. Timestamps need to be of class POSIXct.
track_id	character, optional, name of the column in df that represents track names or IDs. If set, a moveStack is returned, otherwise, a move object is returned.
data	data.frame, optional, to add additional data such as path colours (see move). Number of rows must equal number of rows of df.
...	additional arguments passed to move.

Value

A move or moveStack object.

See Also

frames_spatial frames_graph subset_move

Examples

```
library(moveVis)
library(move)

# load the example data and convert them into a data.frame
data("move_data")
move_df <- methods::as(move_data, "data.frame")

# use df2move to convert the data.frame into a moveStack
df2move(move_df,
      proj = "+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0",
        x = "coords.x1", y = "coords.x2", time = "timestamps", track_id = "trackId")
```

---

frames_graph *Create frames of movement-environment interaction graphs for animation*

Description

frames_graph creates a list of ggplot2 graphs displaying movement-environment interaction. Each object represents a single frame. Each frame can be viewed or modified individually. The returned list of frames can be animated using animate_frames.

Usage

```
frames_graph(m, r_list, r_times, r_type = "gradient",
  fade_raster = FALSE, return_data = FALSE, graph_type = "flow",
  path_size = 1, path_legend = TRUE, path_legend_title = "Names",
  val_min = NULL, val_max = NULL, val_by = 0.1, verbose = T)
```

Arguments

m	move or moveStack of uniform time scale and time lag, e.g. prepared with align_move (recommended). May contain a column named colour to control path colours (see details).
r_list	list of raster or rasterStack. Each list element referrs to the times given in r_times. Use single-layer raster objects for gradient or discrete data (see r_type). Use a rasterStack containing three bands for RGB imagery (in the order red, green, blue).
r_times	list of POSIXct times. Each list element represents the time of the corresponding element in r_list. Must be of same length as r_list.
r_type	character, either "gradient" or "discrete". Ignored, if r_list contains rasterStacks of three bands, which are treated as RGB.
fade_raster	logical, if TRUE, r_list is interpolated over time based on r_times. If FALSE, r_list elements are assigned to those frames closest to the equivalent times in r_times.

return_data — logical, if TRUE, instead of a list of frames, a data.frame containing the values extracted from r_list per individual, location and time is returned. This data.frame can be used to create your own multi- or monotemporal ggplot2 movement-environemnt interaction graphs.

graph_type — character, defines the type of multi-temporal graph that should be drawn as frames. Currently supported graphs are:

- "flow", a time flow graph with frame time on the x axis and values of the visited cell at x on the y axis per individual track
- "hist", a cumulative histogram with cell values on the x axis and time-cumulative counts of visits on the y axis per individual track.

path_size — numeric, size of each path.

path_legend — logical, wether to add a path legend from m or not. Legend tracks and colours will be ordered by the tracks' temporal apperances, not by their order in m.

path_legend_title — character, path legend title. Default is "Names".

val_min — numeric, minimum value of the value axis. If undefined, the minimum is collected automatically.

val_max — numeric, maximum value of the value axis. If undefined, the maximum is collected automatically.

val_by — numeric, increment of the value axis sequence. Default is 0.1. If graph_type = "discrete", this value should be an integer of 1 or greater.

verbose — logical, if TRUE, messages on the function's progress are displayed (default).

Details

To later on side-by-side join spatial frames created using frames_spatial with frames created with frames_graph for animation, equal inputs must have been used for both function calls for each of the arguments m, r_list, r_times and fade_raster.

Value

List of ggplot2 objects, each representing a single frame. If return_data is TRUE, a data.frame is returned (see return_data).

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial join_frames animate_frames

Examples

```
library(moveVis)
library(move)
library(ggplot2)

data("move_data", "basemap_data")
# align movement
m <- align_move(move_data, res = 4, unit = "mins")

r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]


# use the same inputs to create a non-spatial graph, e.g. a flow graph:
frames.gr <- frames_graph(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                          fade_raster = TRUE, graph_type = "flow")
# take a look
frames.gr[[100]]

# make a histogram graph:
frames.gr <- frames_graph(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                          fade_raster = TRUE, graph_type = "hist")
# change the value interval:
frames.gr <- frames_graph(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                          fade_raster = TRUE, graph_type = "hist", val_by = 0.01)

frames.gr[[100]]
# manipulate the labels, since now they are very dense:
# just replace the current scale
frames.gr <- add_gg(frames.gr, expr(scale_x_continuous(breaks=seq(0,1,0.1),
                                   labels=seq(0,1,0.1), expand = c(0,0))))
frames.gr[[100]]

# the same can be done for discrete data, histogram will then be shown as bin plots

# to make your own graphs, use frames_graph to return data instead of frames
data.gr <- frames_graph(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                        fade_raster = TRUE, return_data = TRUE)

# animate the frames created with frames_graph;
animate_frames(frames, out_file = tempfile(fileext = ".gif"))


# see all add_ functions on how to customize your frames created with frames_spatial
# or frames_graph

# see ?animate_frames on how to animate your list of frames
```

`frames_spatial` *Create frames of spatial movement maps for animation*

Description

frames_spatial creates a list of ggplot2 maps displaying movement. Each object represents a single frame. Each frame can be viewed or modified individually. The returned list of frames can be animated using animate_frames.

Usage

```
frames_spatial(m, r_list = NULL, r_times = NULL, r_type = "gradient",
  fade_raster = FALSE, map_service = "osm", map_type = "streets",
  map_res = 1, map_token = NULL, map_dir = NULL,
  margin_factor = 1.1, equidistant = NULL, ext = NULL,
  tail_length = 19, tail_size = 1, path_size = 3,
  path_end = "round", path_join = "round", path_mitre = 10,
  path_arrow = NULL, path_colours = NA, path_legend = TRUE,
  path_legend_title = "Names", ..., verbose = TRUE)
```

Arguments

m	move or moveStack of uniform time scale and time lag, e.g. prepared with align_move (recommended). May contain a column named colour to control path colours (see details).
r_list	list of raster or rasterStack. Each list element referrs to the times given in r_times. Use single-layer raster objects for gradient or discrete data (see r_type). Use a rasterStack containing three bands for RGB imagery (in the order red, green, blue).
r_times	list of POSIXct times. Each list element represents the time of the corresponding element in r_list. Must be of same length as r_list.
r_type	character, either "gradient" or "discrete". Ignored, if r_list contains rasterStacks of three bands, which are treated as RGB.
fade_raster	logical, if TRUE, r_list is interpolated over time based on r_times. If FALSE, r_list elements are assigned to those frames closest to the equivalent times in r_times.
map_service	character, either "osm", "carto" or "mapbox". Default is "osm".
map_type	character, a map type, e.g. "streets". For a full list of available map types, see get_maptypes.
map_res	numeric, resolution of base map in range from 0 to 1.
map_token	character, mapbox authentification token for mapbox basemaps. Register at https://www.mapbox.com/ to get a mapbox token. Mapbox is free of charge after registration for up to 50.000 map requests per month. Ignored, if map_service = "osm".
map_dir	character, directory where downloaded basemap tiles can be stored. By default, a temporary directory is used. If you use moveVis often for the same area it is recommended to set this argument to a directory persistent throughout sessions (e.g. in your user folder), so that baesmap tiles that had been already downloaded by moveVis do not have to be requested again.
margin_factor	numeric, factor relative to the extent of m by which the frame extent should be increased around the movement area. Ignored, if ext is set.

equidistant	logical, whether to make the map extent equidistant (squared) with y and x axis measuring equal distances or not. Especially in polar regions of the globe it might be necessaray to set equidistant to FALSE to avoid strong stretches. By default (equidistant = NULL), equidistant is set automatically to FALSE, if ext is set, otherwise TRUE. Read more in the details.
ext	sf bbox or sp extent in same CRS as m, optional. If set, frames are cropped to this extent. If not set, a squared extent around m, optional with a margin set by margin_factor, is used (default).
tail_length	numeric, length of tail per movement path.
tail_size	numeric, size of the last tail element. Default is 1.
path_size	numeric, size of each path.
path_end	character, either "round", "butt" or "square", indicating the path end style.
path_join	character, either "round", "mitre" or "bevel", indicating the path join style.
path_mitre	numeric, path mitre limit (number greater than 1).
path_arrow	arrow, path arrow specification, as created by grid::arrow().
path_colours	character, a vector of colours. Must be of same length as number of individual tracks in m and refers to the order of tracks in m. If undefined (NA) and m contains a column named colour, colours provided within m are used (see details). Othwersie, colours are selected randomly per individual track.
path_legend	logical, wether to add a path legend from m or not. Legend tracks and colours will be ordered by the tracks' temporal apperances, not by their order in m.
path_legend_title	character, path legend title. Default is "Names".
...	Additional arguments customizing the frame background, passed to RStoolbox::ggR or RStoolbox::ggRGB: • alpha, numeric, background transparency (0-1). • hue, numeric, hue value for color calculation (0-1). Change if you need anything else than greyscale. Only effective if sat > 0. • sat, numeric, saturation value for color calculation (0,1). Change if you need anything else than greyscale. • stretch, character, either 'none', 'lin', 'hist', 'sqrt' or 'log' for no stretch, linear, histogram, square-root or logarithmic stretch. • quantiles, numeric vector with two elements, min and max quantiles to stretch to. Defaults to 2
verbose	logical, if TRUE, messages on the function's progress are displayed (default).

Details

If argument path_colours is not defined (set to NA), path colours can be defined by adding a character column named colour to m, containing a colour code or name per row (e.g. "red". This way, for example, column colour for all rows belonging to individual A can be set to "green", while column colour for all rows belonging to individual B can be set to "red". Colours could also be arranged to change through time or by behavioral segments, geographic locations, age,

environmental or health parameters etc. If a column name `colour` in `m` is missing, colours will be selected automatically. Call `colours()` to see all available colours in R.

Basemap colour scales can be changed/added using `add_colourscale` or by using `ggplot2` commands (see `examples`). For continous scales, use `r_type = "gradient"`. For discrete scales, use `r_type = "discrete"`.

The projection of `m` is treated as target projection. Default base maps accessed through a map service will be reprojected into the projection of `m`. Thus, depending on the projection of `m`, it may happen that map labels are distorted. To get undistorted map labels, reproject `m` to the web mercator projection (the default projection of the base maps): `spTransform(m, crs("+init=epsg:3857"))`. The `ggplot2` coordinate system will be computed based on the projection of `m` using `coord_sf`. If argument `equidistant` is set, the map extent is calculated (thus enlarged into one axis direction) to represent equal surface distances on the x and y axis.

Value

List of ggplot2 objects, each representing a single frame.

Author(s)

Jakob Schwalb-Willmann

See Also

`frames_graph` `join_frames` `animate_frames`

Examples

```
library(moveVis)
library(move)
library(ggplot2)

data("move_data")
# align movement
m <- align_move(move_data, res = 4, unit = "mins")


# with osm watercolor base map
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor")
# take a look at one of the frames, e.g. the 100th
frames[[100]]

# make base map a bit transparent
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor", alpha = 0.5)
frames[[100]] # take a look

# use a larger margin around extent
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor", alpha = 0.5,
                         margin_factor = 1.8)

# use a extent object as your AOI
ext <- extent(m)
```

```
ext@xmin <- ext@xmin - (ext@xmin*0.003)
ext@xmax <- ext@xmax + (ext@xmax*0.003)
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor", alpha = 0.5,
                         ext = ext)

# alter path appearance (make it longer and bigger)
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor", alpha = 0.5,
                         path_size = 4, tail_length = 29)

# adjust path colours manually
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor", alpha = 0.5,
                         path_colours = c("black", "blue", "purple"))

# or do it directly within your moveStack, e.g. like:
m.list <- split(m) # split m into list by individual
m.list <- mapply(x = m.list, y = c("orange", "purple", "darkgreen"), function(x, y){
  x$colour <- y
  return(x)
}) # add colour per individual
m <- moveStack(m.list) # putting it back together into a moveStack
frames <- frames_spatial(m, map_service = "osm", map_type = "watercolor", alpha = 0.5)
# this way, you do not have to assign colours per individual track
# instead, you could assign colours by segment, age, speed or other variables

# get available map types
get_maptypes()

# use mapbox to get a satellite or other map types (register to on mapbox.com to get a token)
frames <- frames_spatial(m, map_service = "mapbox",
                         map_token = "your_token_from_your_mapbox_account",
                         map_type = "satellite")

# if you make a lot of calls to frames_spatial during mutliple sessions, use a map directory
# to save all base maps offline so that you do not have to query the servers each time
frames <- frames_spatial(m, map_service = "mapbox",
                         map_token = "your_token_from_your_mapbox_account",
                         map_type = "satellite",
                         map_dir = "your/map_directory/")

# use your own custom base maps
data("basemap_data")
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]

# using gradient data (e.g. NDVI)
frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                         fade_raster = TRUE)

# using discrete data (e.g. classifications)
# let's make up some classification data with 10 classes
r_list <- lapply(r_list, function(x){
  y <- raster::setValues(x, round(raster::getValues(x)*10))
  return(y)
```

```
})
# turn fade_raster to FALSE, since it makes no sense to temporally interpolate discrete classes
frames <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "discrete",
                         fade_raster = FALSE)

# animate the frames created with frames_spatial;
animate_frames(frames, out_file = tempfile(fileext = ".gif"))

# see ?add_colourscale to learn how to change colours of custom base maps
# see all add_ functions on how to customize your frames created with frames_spatial
# or frames_graph
# see ?animate_frames on how to animate your list of frames
```

`get_maptypes` *Get all supported map types*

Description

This function returns every supported map type that can be used as input to the `map_type` argument of `frames_spatial`.

Usage

```
get_maptypes(map_service = NULL)
```

Arguments

`map_service`	character, optional, either `"osm"`, `"carto"` or `"mapbox"`. Otherwise, a list of map types for both services is returned.

Value

A character vector of supported map types

See Also

`frames_spatial`

Examples

```
# for all services
get_maptypes()

# for osm only
get_maptypes("osm")
# or
get_maptypes()$osm

# for mapbox only
```

```
get_maptypes("mapbox")
# or
get_maptypes()$mapbox

# same for all other map services
```

`join_frames` *Join multiple frames lists into a single frames list*

Description

This function side-by-side joins the ggplot2 objects of two or more frames lists of equal lengths into a single plot per frame using `plot_grid`. This is useful if you want to side-by-side combine spatial frames returned by `frames_spatial` with graph frames returned by `frames_graph`.

Usage

```
join_frames(frames_lists, ..., verbose = T)
```

Arguments

`frames_lists`	list, a list of two or more frames lists that you want to combine. All frames lists contained in `frames_lists` must be of equal lengths. The contained ggplot2 objects are passed frame-wise to the `plotlist` argument of `plot_grid`.
`...`	Further arguments, specifying the appearance of the joined ggplot2 objects, passed to `plot_grid`. See `plot_grid` for further options.
`verbose`	logical, if `TRUE`, messages on the function's progress are displayed (default).

Value

List of ggplot2 objects, each representing a single frame.

See Also

`frames_spatial` `frames_graph` `animate_frames`

Examples

```
## Not run:
library(moveVis)
library(move)

data("move_data", "basemap_data")
# align movement
m <- align_move(move_data, res = 4, unit = "mins")

# create spatial frames and graph frames:
```

```
r_list <- basemap_data[[1]]
r_times <- basemap_data[[2]]

frames.sp <- frames_spatial(m, r_list = r_list, r_times = r_times, r_type = "gradient",
                            fade_raster = TRUE)
frames.sp <- add_colourscale(frames.sp, type = "gradient",
                        colours = c("orange", "white", "darkgreen"), legend_title = "NDVI")
frames.flow <- frames_graph(m, r_list, r_times, path_legend = FALSE, graph_type = "flow")
frames.hist <- frames_graph(m, r_list, r_times, path_legend = FALSE, graph_type = "hist")

# check lengths (must be equal)
sapply(list(frames.sp, frames.flow, frames.hist), length)

# Let's join the graph frames vertically
frames.join.gr <- join_frames(list(frames.flow, frames.hist), ncol = 1, nrow = 2)
frames.join.gr[[100]]

# Now, let's join the joined graph frames with the spatial frames horizontally
# in 2:1 ration and align all axis
frames.join <- join_frames(list(frames.sp, frames.join.gr),
                           ncol = 2, nrow = 1, rel_widths = c(2, 1), axis = "tb")
frames.join[[100]]
# in a standard graphics device, this looks a bit unproportional
# however when setting the correct width, height and resolution of a graphic device,
# it will come out well aligend.

# Do so for example with animate_move() with width = 900, dheight = 500 and res = 90
animate_frames(frames.join, out_file = tempfile(fileext = ".gif"), fps = 25,
               width = 900, height = 500, res = 90, display = TRUE, overwrite = TRUE)

## End(Not run)
```

`move_data` *Exemplary simualted movement tracks*

Description

This datasetd contains a `Move` object, representing cooridnates and acquisition times of three simulated movement tracks, covering a location nearby Lake of Constance, Germany. Individual names are made up for demonstration purposes.

Usage

```
move_data
```

Format

`Move` object, as used by the `move` package.

Details

This object is used by some moveVis examples and unit tests.

Note

All data contained should only be used for testing moveVis and are not suitable to be used for analysis or interpretation.

---

subset_move *Subset a* move *or* moveStack *object by a given time span*

Description

This function is a simple wrapper that subsets a move or moveStack by a given time span. A move or moveStack containing data only for the subset time span is returned.

Usage

```
subset_move(m, from, to, tz = "UTC")
```

Arguments

m	a move or moveStack object (see df2move to convert a data.frame to a move object).
from	character or POSIXct, representing the start time. If character, the format "%m-%d-%y %H:%M:%S" must be used (see strptime).
to	character or POSIXct, representing the stop time. If character, the format "%m-%d-%y %H:%M:%S" must be used (see strptime).
tz	character, time zone that should be used if from and/or to are of type character.

Value

A move or moveStack object.

See Also

df2move

Examples

```
library(moveVis)
library(move)

# load the example data
data("move_data")

# check min and max of move_data timestamps
```

```
min(timestamps(move_data))
max(timestamps(move_data))

# subset by character times
m <- subset_move(move_data, from = "2018-05-15 07:00:00", to = "2018-05-15 18:00:00")

# check min and max of result
min(timestamps(m))
max(timestamps(m))
```

`suggest_formats` *Suggest known file formats*

Description

This function returns a selection of suggested file formats that can be used with `out_file` of `animate_frames` on your system.

Usage

```
suggest_formats(suggested = c("gif", "mov", "mp4", "flv", "avi", "mpeg",
  "3gp", "ogg"))
```

Arguments

`suggested`	character, a vector of suggested file formats which are checked to be known by the available renderers on the running system. By default, these are `c("gif", "mov", "mp4", "flv", "a`

Value

A subset of `suggested`, containing only those file formats which are known by the renderers on the running system.

See Also

`animate_frames`

Examples

```
# find out which formats are available
suggest_formats()

# check for a particular format not listed in "suggested" that you want to use, e.g. m4v
suggest_formats("m4v")
# if "m4v" is returned, you can use this format with animate_frames
```

`view_spatial` *View movements on an interactive map*

Description

`view_spatial` is a simple wrapper that displays movement tracks on an interactive `mapview` or `leaflet` map.

Usage

```
view_spatial(m, render_as = "mapview", time_labels = TRUE,
  stroke = TRUE, path_colours = NA, path_legend = TRUE,
  path_legend_title = "Names", verbose = TRUE)
```

Arguments

`m`	`move` or `moveStack`. May contain a column named `colour` to control path colours (see `details`).
`render_as`	character, either `'mapview'` to return a `mapview` map or `'leaflet'` to return a `leaflet` map.
`time_labels`	logical, wether to display timestamps for each track fix when hovering it with the mouse cursor.
`stroke`	logical, whether to draw stroke around circles.
`path_colours`	character, a vector of colours. Must be of same length as number of individual tracks in `m` and refers to the order of tracks in `m`. If undefined (`NA`) and `m` contains a column named `colour`, colours provided within `m` are used (see details). Othwersie, colours are selected randomly per individual track.
`path_legend`	logical, wether to add a path legend from `m` or not. Legend tracks and colours will be ordered by the tracks' temporal apperances, not by their order in `m`.
`path_legend_title`	character, path legend title. Default is `"Names"`.
`verbose`	logical, if `TRUE`, messages on the function's progress are displayed (default).

Details

If argument `path_colours` is not defined (set to `NA`), path colours can be defined by adding a character column named `colour` to `m`, containing a colour code or name per row (e.g. `"red"`. This way, for example, column `colour` for all rows belonging to individual A can be set to `"green"`, while column `colour` for all rows belonging to individual B can be set to `"red"`. Colours could also be arranged to change through time or by behavioral segments, geographic locations, age, environmental or health parameters etc. If a column name `colour` in `m` is missing, colours will be selected automatically. Call `colours()` to see all available colours in R.

Value

An interatcive `mapview` or `leaflet` map.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial

Examples

```
## Not run:
library(moveVis)
library(move)

data("move_data")

# return a mapview map (mapview must be installed)
view_spatial(move_data)

# return a leaflet map (leaflet must be installed)
view_spatial(move_data, render_as = "leaflet")

# turn off time labels and legend
view_spatial(move_data, time_labels = FALSE, path_legend = FALSE)


## End(Not run)
```

Index