Package: anispace (via r-universe)

June 3, 2026

Type Package

Title An R package providing spatial transformation methods for movement data

Version 0.1.3

Description An R package providing spatial transformation methods for movement data.

License MIT + file LICENSE

URL http://animovement.dev/anispace/, https://github.com/animovement/anispace/

BugReports https://github.com/animovement/anispace/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports aniframe, cli, dplyr, rlang

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/Needs/website rmarkdown

Repository https://animovement.r-universe.dev

Date/Publication 2025-12-05 15:11:31 UTC

RemoteUrl https://github.com/animovement/anispace

RemoteRef HEAD

RemoteSha afd07b56111870131f01ae7a6a0527f479c07e44

Contents

`calculate_angular_difference`
Calculate angular difference

Description

Computes the shortest signed angular distance (in radians) from `from_angle` to `to_angle`.

Usage

```
calculate_angular_difference(from_angle, to_angle)
```

Arguments

`from_angle`	Numeric. Starting angle (radians).
`to_angle`	Numeric. Target angle (radians).

Value

Numeric scalar – the angular difference wrapped to [- ,].

`cartesian_to_phi` *Cartesian azimuth ( ) from coordinates*

Description

Returns the planar angle measured from the positive x-axis toward the positive y-axis. By default the result is mapped to $[0, 2\ )$; setting `centered = TRUE` leaves the native `atan2` range [- ,].

Usage

```
cartesian_to_phi(x, y, centered = FALSE)
```

Arguments

`x`	numeric vector of x-coordinates
`y`	numeric vector of y-coordinates
`centered`	logical; if `TRUE` keep the [- ,] range, otherwise map to $[0, 2\ \backslash)$

Value

numeric vector of azimuth angles ( ) in radians

`cartesian_to_rho` *Cartesian radius ( ) from coordinates*

Description

Computes the Euclidean distance from the origin to a point in either 2-D (`z` omitted) or 3-D space.

Usage

```
cartesian_to_rho(x, y, z = NULL)
```

Arguments

`x`	numeric vector of x-coordinates
`y`	numeric vector of y-coordinates
`z`	optional numeric vector of z-coordinates; if `NULL` a 2-D radius is returned

Value

numeric vector of radii ( )

cartesian_to_theta *Polar angle ( ) from Cartesian coordinates*

Description

Calculates the inclination angle measured from the positive z-axis (the "polar" angle) for each point.

Usage

```
cartesian_to_theta(x, y, z)
```

Arguments

x	numeric vector of x-coordinates
y	numeric vector of y-coordinates
z	numeric vector of z-coordinates

Value

numeric vector of polar angles ( ) in radians

diff_angle *Difference of angular values*

Description

Computes lagged differences between successive angles (in radians) and converts each raw subtraction into the shortest signed angular distance using `calculate_angular_difference()`. The output mimics `base::diff()`, returning `NA`s for the first `lag` positions so it works nicely inside `dplyr::mutate()`.

Usage

```
diff_angle(x, lag = 1L)
```

Arguments

x	Numeric vector of angles (radians).
lag	Positive integer indicating the lag (default = 1L). Must be an integer 1.

Value

Numeric vector of the same length as `x`. The first `lag` entries are `NA`; subsequent entries contain the angular differences.

Examples

```
# Simple example
angles <- c(0, pi/2, pi, 3*pi/2)
diff_angle(angles)

# Using a lag of 2
diff_angle(angles, lag = 2L)
```

---

map_to_cartesian *Map from polar to Cartesian coordinates*

Description

Map from polar to Cartesian coordinates

Usage

```
map_to_cartesian(data)
```

Arguments

data	an aniframe with polar coordinates

Value

an aniframe with Cartesian coordinates

---

map_to_cylindrical *Map from Cartesian to cylindrical coordinates*

Description

Map from Cartesian to cylindrical coordinates

Usage

```
map_to_cylindrical(data)
```

Arguments

data	movement data frame with Cartesian coordinates

Value

movement data frame with cylindrical coordinates

`map_to_polar` *Map from Cartesian to polar coordinates*

Description

Map from Cartesian to polar coordinates

Usage

```
map_to_polar(data)
```

Arguments

`data`	movement data frame with Cartesian coordinates

Value

movement data frame with polar coordinates

`map_to_spherical` *Map from Cartesian to spherical coordinates*

Description

Map from Cartesian to spherical coordinates

Usage

```
map_to_spherical(data)
```

Arguments

`data`	A data frame/tibble containing columns `x`, `y`, `z`

Value

Same data frame with columns `rho`, `theta`, `phi`

polar_to_x — *Convert polar radius to Cartesian x-coordinate*

Description

Convert polar radius to Cartesian x-coordinate

Usage

```
polar_to_x(rho, phi)
```

Arguments

rho	numeric vector of radial distances
phi	numeric vector of azimuth angles (radians)

Value

numeric vector of x-coordinates

polar_to_y — *Convert polar radius to Cartesian y-coordinate*

Description

Convert polar radius to Cartesian y-coordinate

Usage

```
polar_to_y(rho, phi)
```

Arguments

rho	numeric vector of radial distances
phi	numeric vector of azimuth angles (radians)

Value

numeric vector of y-coordinates

`rotate_coords` *Rotate coordinates in Cartesian space (2D or 3D)*

Description

Automatically detects whether data are 2D or 3D and applies the corresponding rotation method.

Usage

```
rotate_coords(data, alignment_points, align_perpendicular = FALSE)
```

Arguments

`data`	movement data frame with columns: time, individual, keypoint, x, y, z (optional)
`alignment_points`	character vector of length 2 specifying the keypoints used for alignment
`align_perpendicular`	logical; if TRUE, alignment_points are rotated to be perpendicular to the 0-degree axis (y-axis). If FALSE (default), they are aligned with the x-axis.

Value

movement data frame with rotated coordinates

`spherical_to_z` *Convert cylindrical radius and polar angle to Cartesian z-coordinate*

Description

Handles regular points as well as the two pole regions (0 and). Non-finite inputs remain `NA`.

Usage

```
spherical_to_z(rho, theta)
```

Arguments

`rho`	Numeric vector – cylindrical radius ($\sqrt{(x^2 + y^2)}$).
`theta`	Numeric vector – polar angle measured from the +z axis (radians).

Value

Numeric vector of z-coordinates (same length as input)

`transform_to_egocentric` *Transform coordinates to egocentric reference frame*

Description

Transforms Cartesian coordinates into an egocentric reference frame through a two-step process: translation followed by rotation. First translates all coordinates relative to a reference keypoint, then rotates the coordinate system based on specified alignment points.

Usage

```
transform_to_egocentric(
  data,
  to_keypoint,
  alignment_points,
  align_perpendicular = FALSE
)
```

Arguments

`data`	movement data frame with columns: time, individual, keypoint, x, y
`to_keypoint`	character; keypoint to use as the new origin
`alignment_points`	character vector of length 2 specifying the keypoint names to use for alignment
`align_perpendicular`	logical; if TRUE, alignment_points will be rotated to be perpendicular to the 0-degree axis. If FALSE (default), alignment_points will be rotated to align with the 0-degree axis

Details

This function combines translation and rotation to create an egocentric reference frame. It:

1. Translates all coordinates relative to the specified keypoint (to_keypoint)
2. Rotates the coordinate system based on the alignment points

The translation makes the reference keypoint the new origin (0,0), while the rotation standardizes the orientation. This is particularly useful for:

- Creating egocentric reference frames
- Standardizing pose data across frames or individuals
- Analyzing relative motion patterns

Value

movement data frame in egocentric reference frame

Examples

```
## Not run:
# Transform coordinates to make nose the origin and align body axis
transformed_data <- transform_to_egocentric(
  data,
  to_keypoint = "nose",
  alignment_points = c("nose", "tail"),
  align_perpendicular = FALSE
)

# Transform to make nose origin and ears perpendicular to forward axis
transformed_data <- transform_to_egocentric(
  data,
  to_keypoint = "nose",
  alignment_points = c("ear_left", "ear_right"),
  align_perpendicular = TRUE
)

## End(Not run)
```

---

translate_coords *Translate coordinates (Cartesian)*

Description

Translates coordinates in Cartesian space. Takes either a single point (`to_x` and `to_y`), a vector with the same length as the time dimension or a keypoint (`to_keypoint`), which can be used to transform the data into an egocentric reference frame.

Usage

```
translate_coords(data, to_x = 0, to_y = 0, to_z = NULL, to_keypoint = NULL)
```

Arguments

`data`	movement data frame with columns: time, individual, keypoint, x, y
`to_x`	x coordinates; either a single value or a time-length vector
`to_y`	y coordinates; either a single value or a time-length vector
`to_z`	z coordinates (only if 3D); either a single value or a time-length vector
`to_keypoint`	all other coordinates becomes relative to this keypoint

Value

movement data frame with translated coordinates

unwrap_angle *Remove constrain for angles to keep within [0, 2)*

Description

Unwraps any numeric vector from the interval [0, 2).

Usage

```
unwrap_angle(x)
```

Arguments

x	Numeric vector of angles (radians).

Value

Numeric vector of the same length.

wrap_angle *Constrain angles to a standard range*

Description

Wraps any numeric vector of angles (in radians) to a standard interval using modulo arithmetic.

Usage

```
wrap_angle(x, modulo = c("2pi", "pi", "asis"))
```

Arguments

x	Numeric vector of angles (radians).
modulo	Character string specifying the target range: "2pi" Wrap to [0, 2) (default) "pi" Wrap to (- ,] "asis" No wrapping, return unchanged

Value

Numeric vector of the same length as x, with angles wrapped to the specified range.

Examples

```
angles <- c(-pi, 0, pi, 2 * pi, 3 * pi)

# Wrap to [0, 2 )
wrap_angle(angles, "2pi")

# Wrap to (- , ]
wrap_angle(angles, "pi")

# No wrapping
wrap_angle(angles, "asis")
```

Index