

Package: anispace (via r-universe)

October 14, 2026

Type Package

Title An R package providing spatial transformation methods for movement data

Version 0.3.0.9006

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

License MIT + file LICENSE

URL <https://animovement.dev/anispace/>,
<https://github.com/animovement/anispace/>

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

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports anicore (>= 0.8.0.9004), cli, dplyr, lifecycle, rlang

Suggests testthat (>= 3.0.0)

Additional_repositories <https://animovement.r-universe.dev>

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/Needs/website animovement/animovementtemplate

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-10-14 12:03:24 UTC

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

RemoteRef HEAD

RemoteSha 35cccfb93e3f9b287cbdab0371d57f783cedd13c

Contents

cartesian_to_phi	2
cartesian_to_rho	3
cartesian_to_theta	4
map_to_cartesian	5
map_to_cylindrical	6
map_to_polar	6
map_to_spherical	7
polar_to_x	8
polar_to_y	8
quat_from_axis_angle	9
quat_from_euler	10
quat_from_vectors	11
quat_slerp	12
quaternions	13
rotate_coords	14
spherical_to_z	16
transform_euler_to_quaternion	17
transform_to_egocentric	18
translate_coords	20
Index	22

cartesian_to_phi	<i>Azimuth (phi) from Cartesian coordinates</i>
------------------	---

Description

Returns the planar angle measured from the positive x-axis towards the positive y-axis, in $(-\pi, \pi]$. That is the signed range the animovement suite uses for every direction (see [anicores::wrap_angle\(\)](#)), and the one [atan2\(\)](#) gives, except that a point on the negative x-axis is π , never $-\pi$. For $[\emptyset, 2 * \pi)$, wrap the result with [anicores::wrap_angle\(phi, "2pi"\)](#).

Usage

```
cartesian_to_phi(x, y)
```

Arguments

- x A numeric vector of x-coordinates.
- y A numeric vector of y-coordinates.

Value

A numeric vector of azimuth angles in radians, in $(-\pi, \pi]$.

See Also

Other coordinate conversion: [cartesian_to_rho\(\)](#), [cartesian_to_theta\(\)](#), [polar_to_x\(\)](#), [polar_to_y\(\)](#), [spherical_to_z\(\)](#)

Examples

```
cartesian_to_phi(1, 1)

# Below the x-axis, the azimuth is negative
cartesian_to_phi(-1, -1)

# On the negative x-axis, it is pi
cartesian_to_phi(-1, 0)

# For [0, 2 * pi), wrap the result
anicore::wrap_angle(cartesian_to_phi(-1, -1), "2pi")
```

cartesian_to_rho	<i>Radius (rho) from Cartesian coordinates</i>
------------------	--

Description

Computes the Euclidean distance from the origin to a point, in either two dimensions (z omitted) or three.

Usage

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

Arguments

x	A numeric vector of x-coordinates.
y	A numeric vector of y-coordinates.
z	An optional numeric vector of z-coordinates (default NULL). When NULL, a two-dimensional radius is returned.

Value

A numeric vector of radii, the same length as x.

See Also

Other coordinate conversion: [cartesian_to_phi\(\)](#), [cartesian_to_theta\(\)](#), [polar_to_x\(\)](#), [polar_to_y\(\)](#), [spherical_to_z\(\)](#)

Examples

```
cartesian_to_rho(3, 4)

# Supplying z gives the three-dimensional radius
cartesian_to_rho(3, 4, 12)
```

cartesian_to_theta	<i>Inclination (theta) from Cartesian coordinates</i>
--------------------	---

Description

Calculates the angle measured from the positive z-axis. Points at the origin return 0.

Usage

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

Arguments

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

Value

A numeric vector of inclination angles in radians, between 0 and pi.

See Also

Other coordinate conversion: [cartesian_to_phi\(\)](#), [cartesian_to_rho\(\)](#), [polar_to_x\(\)](#), [polar_to_y\(\)](#), [spherical_to_z\(\)](#)

Examples

```
# On the positive z-axis, the inclination is zero
cartesian_to_theta(0, 0, 1)

# In the xy-plane, it is a quarter turn
cartesian_to_theta(1, 0, 0)
```

map_to_cartesian	<i>Map to Cartesian coordinates</i>
------------------	-------------------------------------

Description

Converts an aniframe back to Cartesian coordinates, detecting whether it is currently polar, cylindrical or spherical.

Usage

```
map_to_cartesian(data)
```

Arguments

data An aniframe in a polar, cylindrical or spherical coordinate system.

Details

The polar columns are read from the frame's declared axes, so they can have any name, and the angles in the frame's unit_angle. phi can be in any range: $(-\pi, \pi]$, as [map_to_polar\(\)](#) writes it, $[0, 2 * \pi)$, or unwrapped.

Value

An aniframe with x and y (and z, where the input was three-dimensional) in place of the polar columns.

See Also

Other coordinate systems: [map_to_cylindrical\(\)](#), [map_to_polar\(\)](#), [map_to_spherical\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 5, n_individuals = 1, n_keypoints = 1)

# Round-trips back to the coordinates it started from
map_to_cartesian(map_to_polar(af))
```

map_to_cylindrical	<i>Map from Cartesian to cylindrical coordinates</i>
--------------------	--

Description

The Cartesian columns are read from the frame's declared axes, so they can have any name. phi is written in the frame's unit_angle, in $(-\pi, \pi]$ or $(-180, 180]$ (see [cartesian_to_phi\(\)](#)).

Usage

```
map_to_cylindrical(data)
```

Arguments

data	An aniframe in a Cartesian coordinate system.
------	---

Value

An aniframe with rho and phi in place of x and y, keeping z.

See Also

Other coordinate systems: [map_to_cartesian\(\)](#), [map_to_polar\(\)](#), [map_to_spherical\(\)](#)

Examples

```
af <- anicore::example_anipoint(
  n_obs = 5, n_individuals = 1, n_keypoints = 1, n_dims = 3
)
map_to_cylindrical(af)
```

map_to_polar	<i>Map from Cartesian to polar coordinates</i>
--------------	--

Description

The Cartesian columns are read from the frame's declared axes, so they can have any name. phi is written in the frame's unit_angle, in $(-\pi, \pi]$ or $(-180, 180]$ (see [cartesian_to_phi\(\)](#)).

Usage

```
map_to_polar(data)
```

Arguments

data	An aniframe in a Cartesian coordinate system.
------	---

Value

An aniframe with rho and phi in place of x and y.

See Also

Other coordinate systems: [map_to_cartesian\(\)](#), [map_to_cylindrical\(\)](#), [map_to_spherical\(\)](#)

Examples

```
af <- anicore::example_anipoint(
  n_obs = 5, n_individuals = 1, n_keypoints = 1
)
map_to_polar(af)
```

map_to_spherical

Map from Cartesian to spherical coordinates

Description

The Cartesian columns are read from the frame's declared axes, so they can have any name. phi and theta are written in the frame's unit_angle: phi in $(-\pi, \pi]$ or $(-180, 180]$ (see [cartesian_to_phi\(\)](#)), and theta in $[0, \pi]$ or $[0, 180]$.

Usage

```
map_to_spherical(data)
```

Arguments

data An aniframe in a Cartesian coordinate system.

Value

An aniframe with rho, phi and theta in place of x, y and z. rho is the radial distance from the origin, theta the inclination from the positive z-axis, and phi the azimuth — the convention of ISO 80000-2.

See Also

Other coordinate systems: [map_to_cartesian\(\)](#), [map_to_cylindrical\(\)](#), [map_to_polar\(\)](#)

Examples

```
af <- anicore::example_anipoint(
  n_obs = 5, n_individuals = 1, n_keypoints = 1, n_dims = 3
)
map_to_spherical(af)
```

polar_to_x	<i>Cartesian x-coordinate from polar coordinates</i>
------------	--

Description

Cartesian x-coordinate from polar coordinates

Usage

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

Arguments

rho	A numeric vector of radial distances.
phi	A numeric vector of azimuth angles, in radians, in any range.

Value

A numeric vector of x-coordinates.

See Also

Other coordinate conversion: [cartesian_to_phi\(\)](#), [cartesian_to_rho\(\)](#), [cartesian_to_theta\(\)](#), [polar_to_y\(\)](#), [spherical_to_z\(\)](#)

Examples

```
polar_to_x(1, pi / 3)
```

polar_to_y	<i>Cartesian y-coordinate from polar coordinates</i>
------------	--

Description

Cartesian y-coordinate from polar coordinates

Usage

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

Arguments

rho	A numeric vector of radial distances.
phi	A numeric vector of azimuth angles, in radians, in any range.

Value

A numeric vector of y-coordinates.

See Also

Other coordinate conversion: [cartesian_to_phi\(\)](#), [cartesian_to_rho\(\)](#), [cartesian_to_theta\(\)](#), [polar_to_x\(\)](#), [spherical_to_z\(\)](#)

Examples

```
polar_to_y(1, pi / 3)
```

quat_from_axis_angle	<i>Quaternions from and to other rotation representations</i>
----------------------	---

Description

- `quat_from_axis_angle()` / `quat_to_axis_angle()`: a rotation of angle radians about axis, counter-clockwise by the right-hand rule.
- `quat_from_matrix()` / `quat_to_matrix()`: 3x3 rotation matrices, or a 3x3xn array with one matrix per row.

Usage

```
quat_from_axis_angle(axis, angle)
```

```
quat_to_axis_angle(q)
```

```
quat_from_matrix(rotation)
```

```
quat_to_matrix(q)
```

Arguments

axis	Axis vectors (length 3, or 3 columns); need not be unit length.
angle	Angles in radians.
q	Quaternions; see quaternions .
rotation	A 3x3 rotation matrix, or a 3x3xn array.

Value

`quat_from_*`(): a quaternion matrix. `quat_to_axis_angle()`: a list with axis (3 columns) and angle in $[0, \pi]$; the axis of a zero rotation is NA. `quat_to_matrix()`: a 3x3xn array.

Examples

```
q <- quat_from_axis_angle(c(0, 0, 2), pi / 2)
quat_to_axis_angle(q)
quat_to_matrix(q)[, , 1]
quat_from_matrix(quat_to_matrix(q))
```

quat_from_euler

Quaternions from and to Euler angles

Description

Three rotations about coordinate axes, applied in sequence. There are twelve sequences: six Tait-Bryan ("ZYX", "XYZ", ..., all three axes) and six proper Euler ("XZX", "XYX", ..., first and last axis the same).

- **Intrinsic** rotations are about the body's own axes, which move with each rotation. Intrinsic "ZYX" is yaw, pitch, roll.
- **Extrinsic** rotations are about the fixed frame's axes. An intrinsic sequence equals the reversed extrinsic sequence with the angles reversed.

Neither sequence nor intrinsic has a default: an assumed convention is the classic source of Euler-angle bugs, so the call must say which is meant. Euler angles are a report format; store orientation as quaternions.

Usage

```
quat_from_euler(angles, sequence, intrinsic)
```

```
quat_to_euler(q, sequence, intrinsic)
```

Arguments

angles	Angles in radians: a vector of length 3, or three columns, in sequence order.
sequence	Three axes, e.g. "ZYX". Case is ignored.
intrinsic	TRUE for rotations about the moving body axes, FALSE for rotations about the fixed axes.
q	Quaternions; see quaternions .

Value

quat_from_euler(): a quaternion matrix. quat_to_euler(): a matrix with three columns in sequence order. The middle angle is in $[-\pi/2, \pi/2]$ for Tait-Bryan sequences and $[0, \pi]$ for proper ones; the others are in $(-\pi, \pi]$. At gimbal lock the third angle is 0.

Examples

```
ypr <- c(pi / 2, 0.1, -0.2)
q <- quat_from_euler(ypr, sequence = "ZYX", intrinsic = TRUE)
quat_to_euler(q, sequence = "ZYX", intrinsic = TRUE)

# The same rotation, described extrinsically
quat_to_euler(q, sequence = "XYZ", intrinsic = FALSE)
```

quat_from_vectors	<i>Quaternions from two body axes given as vectors</i>
-------------------	--

Description**[Experimental]**

The orientation whose body axis `axes[1]` points along primary, and whose body axis `axes[2]` points towards secondary. Only the part of secondary perpendicular to primary is used, so it need not be at right angles to it; it only has to fix the roll about primary. The third axis completes a right-handed set.

This is how three points define an orientation: primary from one point to a second, and secondary from the first towards any third point off that line. With the default axes, primary is the body's forward axis (x) and the third point lies on the side of its y axis. With `axes = c("y", "x")`, primary runs across the body and secondary fixes which way is forward.

Usage

```
quat_from_vectors(primary, secondary, axes = c("x", "y"))
```

Arguments

primary	Vectors (length 3, or 3 columns) along body axis <code>axes[1]</code> . Need not be unit length.
secondary	Vectors (length 3, or 3 columns) towards body axis <code>axes[2]</code> . Need not be unit length or perpendicular to primary.
axes	Two distinct body axes from "x", "y", "z": the one primary is mapped onto, then the one secondary points towards.

Value

A quaternion matrix; see [quaternions](#). A row is NA where either vector is missing or zero, or the two are parallel, since the roll is then undefined.

See Also

[quat_from_matrix\(\)](#), which this builds the rotation for.

Examples

```
# Facing +y, with the body's left (+y) towards -x: a quarter turn about z
quat_from_vectors(c(0, 1, 0), c(-1, 0, 0))

# The secondary vector need not be perpendicular
quat_from_vectors(c(0, 1, 0), c(-1, 5, 0))

# Across the body: right-to-left along +y, forward towards +x
quat_from_vectors(c(0, 1, 0), c(1, 0, 0), axes = c("y", "x"))
```

quat_slerp

Interpolate, average and differentiate rotations

Description

- `quat_slerp(p, q, t)`: spherical linear interpolation, $t = 0$ giving p and $t = 1$ giving q , along the shorter arc.
- `quat_mean(q, weights)`: the average rotation, as the eigenvector of the largest eigenvalue of $\text{sum}(w \cdot q \cdot q')$ (Markley et al. 2007). Unlike averaging components, it is unaffected by the sign ambiguity.
- `quat_continuous(q)`: flips signs so each row lies on the same side as the previous one. q and $-q$ are the same rotation, so this changes no rotation, but smoothing or differentiating a series without it goes wrong wherever the sign jumps.
- `quat_angular_velocity(q, dt)`: angular velocity between successive rows, in radians per unit of dt , about the fixed frame's axes (`frame = "fixed"`) or the body's own (`frame = "body"`). The first row is NA.

[Experimental] `quat_angular_velocity()` is experimental, and may change without a deprecation cycle: how it differences successive rows – here, each row against the one before – is to be settled with animetric's angular velocity from orientation (animovement/animetric#85). The other three are stable.

Rows with NA are skipped by `quat_mean()` and carried through by the others.

Usage

```
quat_slerp(p, q, t)

quat_mean(q, weights = NULL)

quat_continuous(q)

quat_angular_velocity(q, dt, frame = c("fixed", "body"))
```

Arguments

p, q	Quaternions; see quaternions .
t	Interpolation fraction(s), recycled against the rows.
weights	Optional non-negative weights, one per row.
dt	Time between rows: one value, or one per row.
frame	"fixed" or "body".

Value

A quaternion matrix; `quat_mean()` returns one row; `quat_angular_velocity()` returns a matrix with columns x, y, z.

References

Markley, F. L., Cheng, Y., Crassidis, J. L., & Oshman, Y. (2007). Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4), 1193-1197. doi:10.2514/1.28949.

Examples

```
a <- quat_from_axis_angle(c(0, 0, 1), 0)
b <- quat_from_axis_angle(c(0, 0, 1), pi / 2)
quat_slerp(a, b, 0.5)
quat_mean(rbind(a, b))

spin <- quat_from_axis_angle(c(0, 0, 1), seq(0, 1, by = 0.1))
quat_angular_velocity(spin, dt = 1 / 30)
```

quaternions

Quaternion algebra

Description

Row-wise operations on unit quaternions. A quaternion argument is a numeric vector of length 4, or a matrix or data frame with four columns, in the order w, x, y, z (Hamilton convention, scalar first). Columns named w/x/y/z or qw/qx/qy/qz are matched by name. A single row is recycled against the other argument.

- `quat_multiply(p, q)`: the Hamilton product $p \cdot q$, the rotation q followed by p .
- `quat_conjugate(q)`: the inverse of a unit quaternion.
- `quat_normalise(q)`: scaled to unit norm; a zero quaternion gives NA.
- `quat_rotate(q, v)`: rotates 3D vectors v (length 3, or 3 columns).
- `quat_distance(p, q)`: the angle of the rotation between p and q , in $[0, \pi]$. q and $-q$ are the same rotation, so their distance is 0.

An orientation q maps a vector in the body's own axes into the frame's coordinates, `quat_rotate(q, v)`, so which side a rotation r multiplies on decides which axes it turns about. A rotation in the frame's coordinates – the one `rotate_coords()` applies to positions – pre-multiplies: `quat_multiply(r, q)`, which carries `quat_rotate(q, v)` to `quat_rotate(r, quat_rotate(q, v))`. A rotation about the body's own axes post-multiplies: `quat_multiply(q, r)`.

Usage

`quat_multiply(p, q)`

`quat_conjugate(q)`

`quat_normalise(q)`

`quat_rotate(q, v)`

`quat_distance(p, q)`

Arguments

<code>p, q</code>	Quaternions.
<code>v</code>	3D vectors.

Value

A matrix with columns w, x, y, z (or x, y, z for `quat_rotate()`), or a numeric vector for `quat_distance()`.

See Also

`quat_from_euler()`, `quat_slerp()`

Examples

```
quarter_z <- quat_from_axis_angle(c(0, 0, 1), pi / 2)
quat_rotate(quarter_z, c(1, 0, 0))
quat_multiply(quarter_z, quarter_z)
quat_distance(quarter_z, quat_conjugate(quarter_z))
```

Description

Rotates each subject's coordinates so that chosen members of its identity define the axes. Two members give a direction; in three dimensions a third fixes the roll about it, which two cannot.

A declared orientation (where `$orientation`) turns with the coordinates, by the same rotation, so the two keep describing the same body: in 2D the rotation's angle is added to yaw, in the frame's `unit_angle` and wrapped to the range the input used (signed if any value is negative); in 3D the quaternion is pre-multiplied by the rotation's, `quat_multiply(r, q)` (see [quaternions](#)). The centre of rotation moves positions but not orientation. A moment whose rotation is undefined, because an alignment point is missing, is left as it was, orientation included.

Usage

```
rotate_coords(
  data,
  align,
  level = NULL,
  about = NULL,
  align_perpendicular = FALSE
)
```

Arguments

<code>data</code>	An aniframe in a Cartesian coordinate system.
<code>align</code>	Two or three values of <code>level</code> . The first two define the primary axis. A third, in 3D, defines the plane and so the orientation outright.
<code>level</code>	The identity variable <code>align</code> names members of. Defaults to the frame's only one; a frame declaring several has to be told.
<code>about</code>	Centre of rotation: a value of <code>level</code> to rotate around, or a named numeric such as <code>c(x = 500, y = 500)</code> . Defaults to the coordinate origin, which is what to rotate about once the frame has been translated onto its subject.
<code>align_perpendicular</code>	Put the primary axis across the target rather than along it.

Value

An aniframe with rotated coordinates, and orientation if declared.

See Also

Other coordinate transforms: [transform_to_egocentric\(\)](#), [translate_coords\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 3)

# Align the head-neck axis with x, rotating about the origin
rotate_coords(af, align = c("head", "neck"), level = "keypoint")
```

```
# Rotate each animal about its own head instead
rotate_coords(af, align = c("head", "neck"), level = "keypoint", about = "head")
```

spherical_to_z	<i>Cartesian z-coordinate from spherical coordinates</i>
----------------	--

Description

Non-finite inputs return NA.

Usage

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

Arguments

rho	A numeric vector of cylindrical radii, that is $\sqrt{x^2 + y^2}$.
theta	A numeric vector of inclination angles measured from the positive z-axis, in radians.

Value

A numeric vector of z-coordinates, the same length as rho.

See Also

Other coordinate conversion: [cartesian_to_phi\(\)](#), [cartesian_to_rho\(\)](#), [cartesian_to_theta\(\)](#), [polar_to_x\(\)](#), [polar_to_y\(\)](#)

Examples

```
spherical_to_z(1, pi / 4)

# Non-finite input propagates as NA
spherical_to_z(c(1, NA), c(pi / 4, pi / 4))
```

transform_euler_to_quaternion

Convert orientation between Euler angles and quaternions

Description

[Experimental]

Motion-capture software often exports 3D orientation as three Euler angles, but an `anipoint` stores 3D orientation as a unit quaternion (`animovement/anicore#46`), which has no gimbal lock and no wraparound. These functions are where the two meet, and where the Euler convention is stated once.

- `transform_euler_to_quaternion()` adds quaternion columns computed from the Euler columns, declares them as the frame's orientation, and records the convention (`euler_sequence`, `euler_intrinsic`).
- `transform_quaternion_to_euler()` adds Euler columns computed from the declared orientation, as a derived view for reporting or plotting. It uses the recorded convention unless another is given.

Angles are read and written in the frame's `unit_angle`.

Usage

```
transform_euler_to_quaternion(
    data,
    euler,
    sequence,
    intrinsic,
    names = c("qw", "qx", "qy", "qz")
)

transform_quaternion_to_euler(
    data,
    sequence = NULL,
    intrinsic = NULL,
    names = c("euler_1", "euler_2", "euler_3")
)
```

Arguments

<code>data</code>	A 3D <code>anipoint</code> .
<code>euler</code>	The three Euler angle columns, in sequence order.
<code>sequence, intrinsic</code>	The Euler convention; see <code>quat_from_euler()</code> . For <code>transform_quaternion_to_euler()</code> , NULL uses the convention the frame recorded.
<code>names</code>	Names for the new columns.

Value

data with the new columns; for transform_euler_to_quaternion(), with where\$orientation declared.

Examples

```
df <- data.frame(
  time = 1:3, x = 0, y = 0, z = 0,
  yaw = c(0, 0.1, 0.2), pitch = 0, roll = c(0, 0, 0.1)
)
af <- anicore::as_anipoint(df) |>
  transform_euler_to_quaternion(
    c("yaw", "pitch", "roll"),
    sequence = "ZYX",
    intrinsic = TRUE
  )
anicore::get_variables(af, "where", "orientation")
```

transform_to_egocentric

Transform coordinates to an egocentric reference frame

Description

Places the subject at the centre of its own coordinate system: translating onto a reference member, and then, if asked, rotating it to face a fixed direction. Positions then describe the subject's own geometry rather than where it happened to be in the arena, which is what makes poses comparable across moments and individuals.

The rotation comes from one of two places. Alignment points are members whose positions define the axes. Alternatively, align = "orientation" uses the subject's declared orientation (where\$orientation), for data that has one – FicTrac, rigid-body motion capture, a centroid with a heading – whether or not it also has keypoints to align on.

Translation alone re-centres without changing orientation. Rotation alone is rotate_coords(), which turns the frame about the coordinate origin rather than about the subject. A declared orientation is turned with the positions, as rotate_coords() describes.

Usage

```
transform_to_egocentric(
  data,
  to,
  align = NULL,
  level = NULL,
  align_perpendicular = FALSE
)
```

Arguments

<code>data</code>	An aniframe in a Cartesian coordinate system.
<code>to</code>	A value of <code>level</code> to place at the origin.
<code>align</code>	Optionally, how to rotate. Two or three values of <code>level</code> define the axes: two give a direction, and in 3D a third fixes the roll about it. [Experimental] "orientation" turns each subject to face +x by its declared orientation; see below. This option is experimental, and may change without a deprecation cycle. Omitted, the frame is re-centred and left as it was oriented.
<code>level</code>	The identity variable to and <code>align</code> name members of. Defaults to the frame's only one; a frame declaring several has to be told.
<code>align_perpendicular</code>	Put the primary axis across the target rather than along it.

Value

An aniframe centred on `to`, with `reference_frame` set to "egocentric".

Aligning by orientation

Each subject at each moment is turned by the inverse of its own orientation, so that it faces +x. In 2D that is a rotation by -yaw, after which its yaw is 0. In 3D it is the inverse of its quaternion, after which its body axes lie along the coordinate axes and its orientation is the identity, (1, 0, 0, 0). With `align_perpendicular`, it faces across instead, as for alignment points.

Orientation is declared per row, so the subject's is read from its `to` member: the member it is centred on is the one it is turned to face with. Any other member's orientation is turned with it, and so ends up relative to the `to` member's. To centre on one member and face by another's orientation, align on the second and then `translate_coords()` onto the first; translating changes no orientation.

A moment whose `to` member has no orientation cannot be aligned, and its positions and orientation come back NA rather than unrotated.

A single value can never be alignment points, which come in twos and threes, so "orientation" is unambiguous even when `level` has a member of that name: `c("orientation", "head")` aligns on that member.

See Also

`translate_coords()` and `rotate_coords()`, which this combines.

Other coordinate transforms: `rotate_coords()`, `translate_coords()`

Examples

```
af <- anicore::example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 3)

# The head becomes the origin, and the head-neck axis points forward
transform_to_egocentric(
  af,
  to = "head",
  align = c("head", "neck"),
```

```

    level = "keypoint"
  )

  # Re-centre without reorienting
  transform_to_egocentric(af, to = "head", level = "keypoint")

  # Face the way the declared heading says, whatever the keypoints do
  heading <- anicore::as_anipoint(data.frame(
    time = rep(1:2, each = 2),
    keypoint = c("centroid", "head"),
    x = c(0, 1, 5, 5),
    y = c(0, 1, 5, 6),
    yaw = rep(c(pi / 4, pi / 2), each = 2)
  )) |>
  anicore::set_variables(where = list(
    position = c(x = "x", y = "y"),
    orientation = c(yaw = "yaw")
  ))
  transform_to_egocentric(heading, to = "centroid", align = "orientation")

```

translate_coords

Translate coordinates in Cartesian space

Description

Moves the origin: to a fixed offset, or onto a member of the frame's identity — a keypoint, an animal, or whatever level the frame declares. Translating onto a member is how coordinates are made relative to the subject rather than the arena.

Usage

```
translate_coords(data, to = NULL, level = NULL, by = NULL)
```

Arguments

data	An aniframe in a Cartesian coordinate system.
to	A value of level to place at the origin. All other coordinates become relative to it.
level	The identity variable to is a member of. Defaults to the frame's only one; a frame declaring several has to be told.
by	Named numeric giving a fixed offset per axis role, e.g. <code>c(x = 100, y = 50)</code> . The offset moves the <i>origin</i> , as to does, so the coordinates shift by the negative of it: <code>c(x = 100)</code> moves every point 100 to the left. Mutually exclusive with to.

Value

An aniframe with translated coordinates.

See Also

Other coordinate transforms: [rotate_coords\(\)](#), [transform_to_egocentric\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 3, n_individuals = 1, n_keypoints = 3)

# Everything becomes relative to the head, which sits at the origin
translate_coords(af, to = "head", level = "keypoint")

# Or move the origin by a fixed amount, which shifts the coordinates the
# other way: x becomes x - 100
translate_coords(af, by = c(x = 100, y = 50))
```

Index

- * **coordinate conversion**
 - cartesian_to_phi, [2](#)
 - cartesian_to_rho, [3](#)
 - cartesian_to_theta, [4](#)
 - polar_to_x, [8](#)
 - polar_to_y, [8](#)
 - spherical_to_z, [16](#)
- * **coordinate systems**
 - map_to_cartesian, [5](#)
 - map_to_cylindrical, [6](#)
 - map_to_polar, [6](#)
 - map_to_spherical, [7](#)
- * **coordinate transforms**
 - rotate_coords, [14](#)
 - transform_to_egocentric, [18](#)
 - translate_coords, [20](#)

anicores::wrap_angle(), [2](#)

atan2(), [2](#)

cartesian_to_phi, [2](#)

cartesian_to_phi(), [3](#), [4](#), [6–9](#), [16](#)

cartesian_to_rho, [3](#)

cartesian_to_rho(), [3](#), [4](#), [8](#), [9](#), [16](#)

cartesian_to_theta, [4](#)

cartesian_to_theta(), [3](#), [8](#), [9](#), [16](#)

map_to_cartesian, [5](#)

map_to_cartesian(), [6](#), [7](#)

map_to_cylindrical, [6](#)

map_to_cylindrical(), [5](#), [7](#)

map_to_polar, [6](#)

map_to_polar(), [5–7](#)

map_to_spherical, [7](#)

map_to_spherical(), [5–7](#)

polar_to_x, [8](#)

polar_to_x(), [3](#), [4](#), [9](#), [16](#)

polar_to_y, [8](#)

polar_to_y(), [3](#), [4](#), [8](#), [16](#)

quat_angular_velocity(quat_slerp), [12](#)

quat_conjugate(quaternions), [13](#)

quat_continuous(quat_slerp), [12](#)

quat_distance(quaternions), [13](#)

quat_from_axis_angle, [9](#)

quat_from_euler, [10](#)

quat_from_euler(), [14](#), [17](#)

quat_from_matrix
(quat_from_axis_angle), [9](#)

quat_from_matrix(), [11](#)

quat_from_vectors, [11](#)

quat_mean(quat_slerp), [12](#)

quat_multiply(quaternions), [13](#)

quat_normalise(quaternions), [13](#)

quat_rotate(quaternions), [13](#)

quat_slerp, [12](#)

quat_slerp(), [14](#)

quat_to_axis_angle
(quat_from_axis_angle), [9](#)

quat_to_euler(quat_from_euler), [10](#)

quat_to_matrix(quat_from_axis_angle), [9](#)

quaternions, [9–11](#), [13](#), [13](#), [15](#)

rotate_coords, [14](#)

rotate_coords(), [14](#), [18](#), [19](#), [21](#)

spherical_to_z, [16](#)

spherical_to_z(), [3](#), [4](#), [8](#), [9](#)

transform_euler_to_quaternion, [17](#)

transform_quaternion_to_euler
(transform_euler_to_quaternion),
[17](#)

transform_to_egocentric, [18](#)

transform_to_egocentric(), [15](#), [21](#)

translate_coords, [20](#)

translate_coords(), [15](#), [19](#)