Package 'araponga'

October 27, 2026

Title Estimate 3D Orientations from 2D Landmarks

Version 2.0.0

Description
Estimates possible 3D orientations of directed objects from the 2D image coordinates of two landmarks and any additional information about the imaging geometry or object. Because a 2D image generally does not uniquely determine a 3D orientation, the package returns sets of compatible pitch, yaw, and view-elevation angles rather than a single unconstrained estimate.

License GPL (>= 3)

URL https://github.com/jocateme/araponga

BugReports https://github.com/jocateme/araponga/issues

Depends R (>= 4.1.0)

Imports graphics

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author João C T Menezes [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-4156-2934>)

Maintainer João C T Menezes <jocateme@gmail.com>

Repository CRAN

Date/Publication 2026-10-27 00:10:02 UTC

Contents

conventions 2
deg2rad 3
find.3d 3

full_length.from.camera 8
full_length.from.reference 9
pitch2d.w.error 11
plot.angles 12
project2d.from.3d 13
project2d.from.xy 16
rotate3d 17
summarize.yaws 18
trim.yaws 20

Index **22**

`conventions` *Plot angle conventions*

Description

Quick visual reminders of the angle conventions used by `araponga`.

Usage

```
conventions(type = c("pitch", "yaw", "view_elevation"))
```

Arguments

`type`	Character vector specifying which convention plot(s) to draw. Choices are `"pitch"`, `"yaw"`, and `"view_elevation"`. Default plots all.

Value

No return value.

See Also

`plot.angles()`

Examples

```
conventions()

conventions("yaw")
```

deg2rad *Convert degrees to radians, and vice versa*

Description

`deg2rad()` takes angles in degrees and returns them converted in radians.

`rad2deg()` takes angles in radians and returns them converted in degrees.

Usage

```
deg2rad(deg)

rad2deg(rad)
```

Arguments

`deg`	Numeric vector of angles in degrees.
`rad`	Numeric vector of angles in radians.

Value

A numeric vector of angles, in radians or degrees. Length matches `deg` or `rad`.

Examples

```
deg2rad(c(0, 90, 180))

rad2deg(c(0, pi/2, pi))
```

find.3d *Find 3D orientations compatible with observed 2D constraints*

Description

Evaluate combinations of candidate 3D pitch, yaw, and view elevation and return those whose predicted 2D projection is compatible with supplied observations. `find.pitch()` and `find.yaw()` are convenience wrappers for finding compatible 3D pitch and yaw orientations, respectively.

Usage

```
find.3d(
  observed2d = NULL,
  find = "all",
  label_error = NULL,
  full_length = NULL,
  pitch2d = NULL,
  length2d = NULL,
  candidate_pitches = seq(-90, 90, default_step),
  candidate_yaws = seq(-180 + default_step, 180, default_step),
  candidate_view_elevations = seq(-90, 90, default_step),
  default_step = 1,
  max_combinations = 1e+08
)

find.yaw(
  observed2d = NULL,
  label_error = NULL,
  full_length = NULL,
  pitch2d = NULL,
  length2d = NULL,
  candidate_pitches = seq(-90, 90, default_step),
  candidate_yaws = seq(-180 + default_step, 180, default_step),
  candidate_view_elevations = seq(-90, 90, default_step),
  paired = FALSE,
  default_step = 1,
  max_combinations = 1e+08
)

find.pitch(
  observed2d = NULL,
  label_error = NULL,
  full_length = NULL,
  pitch2d = NULL,
  length2d = NULL,
  candidate_pitches = seq(-90, 90, default_step),
  candidate_yaws = seq(-180 + default_step, 180, default_step),
  candidate_view_elevations = seq(-90, 90, default_step),
  paired = FALSE,
  default_step = 1,
  max_combinations = 1e+08
)
```

Arguments

observed2d	Optional one-row `data.frame` of class `araponga2d` returned by `project2d.from.xy()`. Must be supplied with `label_error` and cannot be combined with `pitch2d` or `length2d`.

`find` Character vector specifying which 3D angle(s) to return. One or more of "pitch", "yaw", and "view_elevation", or "all" to return all three. Default is "all".

`label_error` Non-negative numeric scalar specifying the maximum labeling error in each landmark coordinate, in the same units as the coordinates in `observed2d` (e.g., pixels). Required when `observed2d` is supplied and not allowed with `pitch2d` or `length2d`.

`full_length` Optional positive numeric vector giving the possible unprojected (no foreshortening) length of the directed vector, in the same units as `length2d` or the coordinates in `observed2d`. Its minimum and maximum define a continuous allowed range.

`pitch2d` Optional numeric vector of projected 2D pitch angles, in degrees in the interval (-180, 180]. The smallest continuous angular interval containing the supplied values is treated as the allowed `pitch2d` interval. Cannot be combined with `observed2d`.

`length2d` Optional non-negative numeric vector of projected 2D lengths. Its minimum and maximum define a continuous allowed range. Requires `full_length` and cannot be combined with `observed2d`.

`candidate_pitches`
Numeric vector of candidate 3D pitch angles, in degrees in the interval (-180, 180]. Convention: `90` = pointed up, `0` = horizontally aligned, `-90` = pointed down. By default, a [-90, 90] grid is evaluated at `default_step` resolution. Values outside [-90, 90] may be supplied explicitly when an extended pitch representation is desired.

`candidate_yaws` Numeric vector of candidate yaw angles, in degrees in the interval (-180, 180]. Convention: `0` = pointed right, `90` = pointed straight away, `-90` = pointed straight toward, `180` = pointed left. By default, the entire (-180, 180] grid is evaluated at `default_step` resolution.

`candidate_view_elevations`
Numeric vector of candidate camera elevation angles relative to the object, in degrees in the interval [-90, 90]. Convention: `-90` = seen from straight below, `0` = eye level, `90` = seen from straight above. By default, the entire [-90, 90] grid is evaluated at `default_step` resolution.

`default_step` Positive numeric scalar specifying the step size, in degrees, used to generate default candidate-angle vectors. Must evenly divide 180 to ensure full coverage of default candidate ranges. Default is `1` (integers). Has no effect on candidate vectors supplied explicitly by the user.

`max_combinations`
Positive numeric scalar specifying the maximum total number of candidate pitch-yaw-view elevation combinations to evaluate, in order to prevent excessive memory use and computation. Default is `1e8`. Searches exceeding this value stop before candidate combinations are evaluated. Set to `Inf` to disable this limit.

`paired` Logical scalar used by `find.pitch()` and `find.yaw()`. If `TRUE`, returns a `data.frame` of yaws mapped to pitches; if `FALSE` (default), a vector of yaws or pitches.

Details

For each unique combination of candidate pitch, yaw, and view elevation, `find.3d()` and wrappers use `project2d.from.3d()` to predict the corresponding 2D projection. Candidate orientations are retained when that projection is compatible with all supplied observational constraints.

The function supports two main ways of specifying the observed 2D projection.

The preferred route is to supply `observed2d` together with `label_error`. If `full_length` is not known, the landmarks constrain projected pitch only. If `full_length` is also supplied, both projected pitch and length are used jointly, preserving the relationship between them under landmark labeling uncertainty.

Alternatively, projected measurements may be supplied directly through `pitch2d` and/or `length2d`. This is useful when landmark coordinates are unavailable or when the user already has suitable 2D constraints. `length2d` requires `full_length`. When both `pitch2d` and `length2d` are supplied directly, they are treated as independent constraints.

Candidate angles supplied explicitly are evaluated exactly and are not rounded. For candidate vectors left at their defaults, `default_step` determines the resolution of the search.

The default pitch range is [-90, 90], which provides a conventional down-to-up representation. Explicit candidate pitches may extend over (-180, 180]. Such extended pitch values should be used intentionally because they necessarily have equivalent representations involving a different yaw (e.g., a pitch of 135° at a yaw of 10° is equivalent to a pitch of 45° at the opposite yaw of -170°).

Value

A `data.frame` containing the requested subset of:

pitch numeric: 3D pitch angles, in degrees.

yaw numeric: yaw angles, in degrees.

view_elevation numeric: view elevation angles, in degrees.

Each row represents a candidate 3D orientation compatible with all supplied observational constraints. If no combinations are compatible, the returned `data.frame` has zero rows.

For `find.pitch(..., paired = FALSE)` and `find.yaw(..., paired = FALSE)`, a numeric vector of unique compatible pitch or yaw angles is returned.

See Also

`project2d.from.3d()`, `project2d.from.xy()`, `pitch2d.w.error()`

Examples

```
# construct a hypothetical observation from a known 3D orientation
predicted <- project2d.from.3d(
  pitch = 30,
  yaw = 20,
  view_elevation = 10,
  full_length = 100
)
```

```
observed <- project2d.from.xy(
  x_tip = predicted$dx,
  y_tip = predicted$dy,
  x_base = 0,
  y_base = 0
)

# landmark route: constrain projected direction only
find.3d(
  observed2d = observed,
  label_error = 1,
  candidate_pitches = 25:35,
  candidate_yaws = 15:25,
  candidate_view_elevations = 5:15
)

# landmark route with full-length information: constrain projected direction and length jointly
find.3d(
  observed2d = observed,
  label_error = 1,
  full_length = c(98, 102),
  candidate_pitches = 25:35,
  candidate_yaws = 15:25,
  candidate_view_elevations = 5:15
)

# direct constraints
find.3d(
  pitch2d = predicted$pitch2d + c(-1, 1),
  length2d = predicted$length2d + c(-2, 2),
  full_length = c(98, 102),
  candidate_pitches = 25:35,
  candidate_yaws = 15:25,
  candidate_view_elevations = 5:15
)

# convenience wrappers
find.pitch(
  observed2d = observed,
  label_error = 1,
  candidate_yaws = 15:25,
  candidate_view_elevations = 5:15
)

find.yaw(
  observed2d = observed,
  label_error = 1,
  candidate_pitches = 25:35,
  candidate_view_elevations = 5:15
)
```

full_length.from.camera *Estimate full length from camera parameters*

Description

Estimates the unforeshortened length of an object in image pixels from its physical length, distance from the camera, and camera parameters.

Usage

```
full_length.from.camera(
  length_physical,
  distance,
  image_width,
  focal_length = NULL,
  sensor_width = NULL,
  field_of_view = NULL
)
```

Arguments

length_physical	Positive numeric vector specifying the physical length of the object. When length(length_physical) > 1, the range of supplied values is treated as the range of possible physical lengths.
distance	Positive numeric vector specifying the distance from the camera to the object, in the same units as length_physical. When length(distance) > 1, the range of supplied values is treated as the range of possible distances.
image_width	Positive numeric vector specifying the image width, in pixels, corresponding to the supplied camera parameters and image scale. When length(image_width) > 1, the range of supplied values is treated as the range of possible image widths.
focal_length	Positive numeric vector specifying camera focal length. Must be in the same units as sensor_width. When length(focal_length) > 1, the range of supplied values is treated as the range of possible focal lengths. Required when field_of_view is not supplied.
sensor_width	Positive numeric vector specifying sensor width. Must be in the same units as focal_length. When length(sensor_width) > 1, the range of supplied values is treated as the range of possible sensor widths. Required when field_of_view is not supplied.
field_of_view	Positive numeric vector specifying horizontal field of view, in degrees in the interval (0, 180). When length(field_of_view) > 1, the range of supplied values is treated as the range of possible fields of view. Required when focal_length and sensor_width are not supplied.

Details

Camera geometry can be specified either by `focal_length` and `sensor_width`, or by `field_of_view`. These alternatives are mutually exclusive.

Input vectors represent ranges of possible values rather than paired observations. The returned interval contains the minimum and maximum `full_length` values compatible with all supplied ranges, allowing inputs to vary independently within those ranges.

The calculation assumes scaled orthographic projection: image scale is determined at the supplied camera-to-object distance and treated as constant across the object.

Value

Numeric vector of length 2 giving the minimum and maximum compatible `full_length` values, in pixels.

See Also

`find.3d()`

Examples

```
# focal length and sensor width
full_length.from.camera(length_physical = 10,
                        distance = 1000,
                        image_width = 2000,
                        focal_length = 50,
                        sensor_width = 20)

# equivalent camera geometry specified by field of view
full_length.from.camera(length_physical = 10,
                        distance = 1000,
                        image_width = 2000,
                        field_of_view = rad2deg(2 * atan(20 / (2 * 50))))

# propagate uncertainty in physical length and distance
full_length.from.camera(length_physical = c(9, 11),
                        distance = c(900, 1100),
                        image_width = 2000,
                        focal_length = 50,
                        sensor_width = 20)
```

---

full_length.from.reference

Estimate full length from a reference segment

Description

Estimates the unforeshortened length of a target object in image pixels from a reference segment of known physical length.

Usage

```
full_length.from.reference(length_physical, reference_physical, reference_px)
```

Arguments

`length_physical`	Positive numeric vector specifying the physical length of the target object. When `length(length_physical) > 1`, the range of supplied values is treated as the range of possible physical lengths.
`reference_physical`	Positive numeric vector specifying the physical length of the reference segment, in the same units as `length_physical`. When `length(reference_physical) > 1`, the range of supplied values is treated as the range of possible reference lengths.
`reference_px`	Positive numeric vector specifying the length of the reference segment in image pixels. When `length(reference_px) > 1`, the range of supplied values is treated as the range of possible pixel lengths.

Details

Input vectors represent ranges of possible values rather than paired observations. The returned interval contains the minimum and maximum `full_length` values compatible with all supplied ranges, allowing inputs to vary independently within those ranges.

The reference segment should be approximately perpendicular to the camera's viewing axis and at approximately the same distance from the camera as the target object. Foreshortening of the reference segment or differences in camera distance can bias the estimated `full_length`.

Value

Numeric vector of length 2 giving the minimum and maximum compatible `full_length` values, in pixels.

See Also

`find.3d()`

Examples

```
full_length.from.reference(length_physical = 10,
                           reference_physical = 20,
                           reference_px = 100)

# propagate uncertainty in target and reference measurements
full_length.from.reference(length_physical = c(9, 11),
                           reference_physical = c(18, 22),
                           reference_px = c(90, 110))
```

`pitch2d.w.error` *Calculate 2D pitch uncertainty from landmark labeling uncertainty*

Description

Calculate the range of projected 2D pitch angles compatible with uncertainty in the locations of two landmarks.

Usage

```
pitch2d.w.error(observed2d, label_error)
```

Arguments

`observed2d`	One-row `data.frame` of class `araponga2d`, as returned by `project2d.from.xy()`.
`label_error`	Non-negative numeric scalar specifying the maximum labeling error in each landmark coordinate, in the same units as the coordinates in `observed2d` (e.g., pixels).

Details

`label_error` is applied independently to each landmark coordinate. The function returns the smallest continuous interval of projected 2D pitch angles compatible with that uncertainty.

If the uncertainty is large enough that all projected directions are possible, the returned interval spans the full circle.

Value

A named list describing the smallest continuous angular interval compatible with the specified labeling error, with components:

- `from`: Starting angle of the interval, in degrees.
- `to`: Ending angle of the interval, in degrees.
- `width`: Width of the interval, in degrees.
- `wrap`: Logical indicating whether the interval crosses the `180`/`-180` boundary.
- `all`: An empty list, included for consistency with `summarize.yaws()`.

If the landmark error region contains the origin in its interior, all 2D pitch angles are possible and the returned interval has `from` = `-180`, `to` = `180`, and `width` = `360`.

See Also

`project2d.from.xy()`, `summarize.yaws()`

Examples

```
# Projected vector from hypothetical pixel coordinates
observed <- project2d.from.xy(
  x_tip = 10, y_tip = 8,
  x_base = 11, y_base = 12
)

# Pitch interval compatible with ±1 pixel labeling uncertainty
pitch2d.w.error(observed, label_error = 1)

# Pitch interval compatible with ±5 pixel labeling uncertainty (full circle returned)
pitch2d.w.error(observed, label_error = 5)
```

`plot.angles` *Visualize angles*

Description

Plotting helper for quick visual inspection of angles, in particular pitches, yaws, and view elevations.

Usage

```
## S3 method for class 'angles'
plot(
  angles = NULL,
  type = NULL,
  facing = c("right", "left"),
  col = NULL,
  main = NULL,
  labels = TRUE,
  add = FALSE
)
```

Arguments

`angles`	Set(s) of angles to be plotted, in degrees in the range (-180, 180]. If a numeric vector, one plot is generated. If a `list` or a `data.frame`, `length(angles)` plots are generated, customized when `names(angles)` match `"pitch"`, `"yaw"`, or `"view_elevation"`. Compatible with outputs from `find.3d()`, `find.pitch()`, and `find.yaw()`.
`type`	When `angles` is a vector, optional character choice between `"pitch"`, `"yaw"`, and `"view_elevation"` that determines the kind of visualization to be generated. If `NULL` (default), a generic plot is generated.
`facing`	Character choice between `"right"` (default) or `"left"`: which direction the object should be facing for `pitch` and `view_elevation`.

col	Optional vector of colors to draw angle segments in each plot. Recycled to match length(angles) when angles is not a vector.
main	Optional character vector: titles for each plot. Recycled to match length(angles) when angles is not a vector.
labels	Logical scalar (default TRUE): should inset labels be plotted?
add	Logical scalar (default FALSE): should angle segments be added to the current plotting device?

Value

No return value.

See Also

find.3d()

Examples

```
# calls with a vector
## pitch between 60 and 70
plot.angles(60:70, "pitch")
## view_elevation between -30 and -40
plot.angles(-30:-40, "view_elevation")

# call with named list
list <- list(pitch = 10:30, yaw = -100:-45, view_elevation = -10:-15)
plot.angles(list)

# call with output from find.3d
## pitches and yaws that project to 10° (± 1° error) if seen from 15° below
df <- find.3d(
  pitch2d = 9:11,
  find = c("pitch", "yaw"),
  candidate_view_elevations = -15
  )
plot.angles(df)
```

---

project2d.from.3d *Project a 3D directed vector into two dimensions*

Description

Calculate the projected 2D pitch and length (relative or absolute) produced by combinations of 3D pitch, yaw, and view elevation.

Usage

```
project2d.from.3d(pitch, yaw, view_elevation, full_length = NULL, plot = FALSE)
```

Arguments

`pitch`	Numeric vector: vertical orientation of the object relative to the horizontal plane, in degrees, in the interval \(-180, 180]. Convention: `90` = pointed up, `0` = horizontally aligned, `-90` = pointed down.
`yaw`	Numeric vector: horizontal orientation around the vertical axis, in degrees in the interval (-180, 180]. Convention: `0` = pointed right, `90` = pointed straight away, `-90` = pointed straight toward, `180` = pointed left.
`view_elevation`	Numeric vector: camera elevation relative to the object, in degrees in the interval [-90, 90]. Convention: `-90` = seen from straight below, `0` = eye level, `90` = seen from straight above.
`full_length`	Optional positive numeric vector giving the unprojected (no foreshortening) length of the directed vector. Values may be expressed in any linear unit. The returned `length2d` is expressed in the same unit. Scalar is recycled; otherwise must have the same length as the angle arguments.
`plot`	Logical scalar. `TRUE` draws a diagnostic plot with original and rotated axes, projected 2D pitch and, if `full_length` is supplied, projected 2D length. Plotting is only supported when all angle arguments have length 1.

Details

For the rotation matrix

$$R = R_x(e)R_y(y)R_z(p),$$

where p, y, and e are respectively pitch, yaw, and view elevation in radians (as in `rotate3d()`), the projected components of the directed vector are

$$f_x = R_{1,1} = \cos(y)\cos(p)$$

and

$$f_y = R_{2,1} = \cos(e)\sin(p) + \sin(e)\sin(y)\cos(p).$$

These are returned as `dx_factor` and `dy_factor`, respectively. Projected 2D pitch is then

$$p_2 = \operatorname{atan2}(f_y, f_x)$$

and projection factor is

$$q = \sqrt{f_x^2 + f_y^2}.$$

If `full_length` L is supplied, the projected components are

$$dx = Lf_x$$

and

$$dy = Lf_y,$$

giving projected length

$$L_2 = Lq.$$

`full_length` therefore need not represent a physical length. For example, a pixel length measured from the same object under a known orientation with `projection_factor == 1` may be supplied as a reference length, provided image scale is unchanged. In that case, `length2d`, `dx`, and `dy` are returned in pixels.

`pitch`, `yaw`, and `view_elevation` are evaluated elementwise and must have the same length.

Value

A `list` containing:

- `pitch2d`: projected 2D pitch angle(s), in degrees in the interval (-180, 180].
- `projection_factor`: projected length relative to `full_length`, ranging from 0 to 1.
- `dx_factor`, `dy_factor`: projected x and y components per unit `full_length`.
- `length2d`: if `full_length` is supplied, projected length in the same units.
- `dx`, `dy`: if `full_length` is supplied, projected x and y components in the same units.

When the vector is effectively parallel to the viewing axis, `projection_factor` is zero and 2D pitch is undefined (`NA`).

See Also

`rotate3d()`, `project2d.from.xy()`, `find.3d()`

Examples

```
# projected 2D pitch of object pointed up 15º and 30º toward camera, seen from 20º below
project2d.from.3d(
  pitch = 15,
  yaw = -30,
  view_elevation = -20,
  plot = TRUE
)

# also predict projected length from a 100-pixel reference length
project2d.from.3d(
  pitch = 15,
  yaw = -30,
  view_elevation = -20,
  full_length = 100,
  plot = TRUE
)

# vectorized usage
```

```
project2d.from.3d(
  pitch = c(15, 15, 15),
  yaw = c(0, -30, 0),
  view_elevation = c(0, 0, -30),
  full_length = 100
)
```

`project2d.from.xy` *Calculate a projected 2D vector from two landmarks*

Description

Calculate the projected pitch and length of the 2D directed vector defined by a base landmark and a tip landmark.

Usage

```
project2d.from.xy(x_tip, y_tip, x_base, y_base, plot = FALSE)
```

Arguments

`x_tip`, `x_base`	Numeric vectors: image *x*-coordinate(s) of tip and base of object, with *x* increasing right.
`y_tip`, `y_base`	Numeric vectors: image *y*-coordinate(s) of the tip and base of objects, with *y* increasing up.
`plot`	Logical scalar. `TRUE` draws a diagnostic plot with base-tip segment and calculated 2D pitch angle. Plotting is only supported for single base-tip pair.

Details

The landmarks define the directed vector

$$(dx, dy) = (x_{tip} - x_{base}, y_{tip} - y_{base})$$

with projected 2D pitch

$$p_2 = \operatorname{atan2}(dy, dx)$$

and projected 2D length

$$L_2 = \sqrt{dx^2 + dy^2}.$$

`length2d` is expressed in the same units as the landmark coordinates. For image coordinates in pixels, for example, `length2d` is returned in pixels.

Coordinate arguments may have length 1 or a common length greater than 1. Length-1 arguments are recycled.

Value

A `data.frame` of class `araponga2d`, with one row per base-tip pair and the following columns:

- `x_tip`, `y_tip`, `x_base`, `y_base`: supplied landmark coordinates.
- `pitch2d`: projected 2D pitch angle(s), in degrees in the interval (-180, 180].
- `length2d`: Euclidean distance(s) between base and tip, in the same units as the supplied coordinates.

When tip and base coincide, `length2d` is zero and `pitch2d` is undefined (`NA`).

The returned object can be supplied as `observed2d` to `find.3d()` and wrappers.

See Also

`pitch2d.w.error()`, `project2d.from.3d()`

Examples

```
# scalar examples (plots)
project2d.from.xy(1, 0, 0, 0, plot = TRUE) # pointed right -> 0°
project2d.from.xy(-1, 0, 0, 0, plot = TRUE) # pointed left -> 180°
project2d.from.xy(0, 1, 0, 0, plot = TRUE) # pointed up -> 90°
project2d.from.xy(0, -1, 0, 0, plot = TRUE) # pointed down -> -90°

# vectorised usage (no plot)
x_tips  <- c(1, 0, -1)
y_tips  <- c(0, 1, 0)
x_bases <- c(0, 0, 0)
y_bases <- c(0, 0, 0)
project2d.from.xy(x_tips, y_tips, x_bases, y_bases)
```

---

rotate3d *Create and apply 3D rotations*

Description

Create a 3×3 rotation matrix corresponding to sequential Euler rotations about Z, then Y, then X (i.e., ZYX or pitch–yaw–roll sequence).

Usage

```
rotate3d(pitch = 0, yaw = 0, roll = 0)

Rz(pitch = 0)

Ry(yaw = 0)

Rx(roll = 0)
```

Arguments

`pitch`	Numeric scalar: angle of rotation about the Z axis, in degrees. Defaults to `0` (no rotation).
`yaw`	Numeric scalar: angle of rotation about the Y axis, in degrees. Defaults to `0` (no rotation).
`roll`	Numeric scalar: angle of rotation about the X axis, in degrees. Defaults to `0` (no rotation).

Details

`Rz()`, `Ry()`, and `Rx()` construct elementary rotation matrices about the Z, Y, and X axes, respectively. `rotate3d()` combines these as

$$R = R_x(r)R_y(y)R_z(p),$$

where p, y, and r are pitch, yaw, and roll. When the resulting matrix is applied to a column vector, the rotations therefore occur in the order pitch, then yaw, then roll.

Angles are not restricted to a particular interval because rotations are periodic; any finite value in degrees is accepted.

In `project2d.from.3d()`, the same X-axis rotation represented here by `roll` is used to incorporate view elevation. This represents relative viewing geometry and should not be confused with image-plane camera roll.

Value

A numeric 3×3 matrix (class `matrix`).

Examples

```
# rotate 45 about z, then 30 about y, then -80 about x
rotate3d(45, 30, -80)

# rotate 30 degrees about x
Rx(30)
```

`summarize.yaws` *Summarize yaw angles into the smallest continuous interval*

Description

Given a set of yaw angles (or any set of angles that wrap around ± 180), `summarize.yaws()` returns the endpoints and width of the smallest continuous angular interval that contains all supplied angles.

Usage

```
summarize.yaws(yaws, plot = FALSE, tie_action = c("all", "error"))
```

Arguments

yaws	Numeric vector of yaw angles, in degrees, in the interval (-180, 180].
plot	Logical scalar. TRUE draws a diagnostic plot with individual candidate angles and the returned smallest continuous interval that contains them. Works only when length(yaws) > 1.
tie_action	Choice between "all" and "error" on how to proceed when multiple smallest continuous intervals exist. "all" (default) issues a warning and returns all possible intervals (see Value), while "error" issues an error.

Value

A named list with components:

from numeric scalar: interval start (clockwise-most endpoint), in degrees.

to numeric scalar: interval end (counterclockwise-most endpoint), in degrees.

width numeric scalar: angular width of the shortest enclosing arc, in degrees.

wrap logical: TRUE if the interval crosses the ±180 boundary.

all list containing the output (from, to, width, wrap) for all alternative intervals, if any

Examples

```
# simple non-wrapping interval
summarize.yaws(-60:60, plot = TRUE)

# a wrapping interval (points around ±180)
summarize.yaws(c(-175:-170, 171:179), plot = TRUE)


# singleton (no plotting)
summarize.yaws(30)

# more than one interval possible (returns warning)
summarize.yaws(c(-90, 90))

# output from find.yaw()
yaws <- find.yaw(
  pitch2d = 14:16,
  candidate_view_elevations = -45,
  candidate_pitches = 10:20
  )
summarize.yaws(yaws, plot = TRUE)
```

`trim.yaws` *Trim candidate yaw sets by directional separation*

Description

Trim two candidate yaw sets to angles that have at least one compatible partner in the other set, given minimum and maximum directional separation constraints.

Usage

```
trim.yaws(ccw_yaws, cw_yaws, min_sep, max_sep, plot = FALSE)
```

Arguments

`ccw_yaws`	Numeric vector: candidate yaw angles known/expected to be **counterclockwise** of `cw_yaws` (degrees, in the interval (-180, 180]).
`cw_yaws`	Numeric vector: candidate yaw angles known/expected to be **clockwise** of `ccw_yaws` (degrees, in the interval (-180, 180]).
`min_sep`	Numeric scalar: minimum directional angular separation in degrees, satisfying `0 <= min_sep < 360`.
`max_sep`	Numeric scalar: maximum directional angular separation in degrees, satisfying `min_sep <= max_sep <= 360`.
`plot`	Logical scalar. `TRUE` draws diagnostic plots with retained (blue) and excluded (red) angles for each set.

Details

Given two sets of candidate yaw angles, `trim.yaws()` retains angles that have at least one compatible partner in the other set. Directional separation is measured counterclockwise from `cw_yaws` to `ccw_yaws` and is represented in the interval [0, 360).

For a pair of angles c from `ccw_yaws` and w from `cw_yaws`, their directional separation is

$$(c - w) \bmod 360.$$

A pair is compatible when this separation is greater than or equal to `min_sep` and less than or equal to `max_sep`. Each returned angle therefore has at least one compatible partner in the other returned set.

Value

A `list` with elements:

trimmed_ccw_yaws numeric: subset of `ccw_yaws` that have at least one matching `cw_yaws`.

trimmed_cw_yaws numeric: subset of `cw_yaws` that have at least one matching `ccw_yaws`.

Examples

```
# hypothetical candidate yaw sets:
a <- 10:80
b <- 30:90

# we know `a` to be CCW of `b` by up to 180°
trim.yaws(a, b, 0, 180, plot = TRUE)

# we know `a` to be CCW of `b` by 30 to 45°
trim.yaws(a, b, 30, 45, plot = TRUE)

# if no mutual partners exist, both returned vectors become empty
trim.yaws(ccw_yaws = c(-10), cw_yaws = c(130), min_sep = 0, max_sep = 180)
```

Index

conventions, 2

deg2rad, 3

find.3d, 3
find.3d(), *9*, *10*, *12*, *13*, *15*, *17*
find.pitch *(*find.3d*)*, 3
find.pitch(), *12*
find.yaw *(*find.3d*)*, 3
find.yaw(), *12*
full_length.from.camera, 8
full_length.from.reference, 9

pitch2d.w.error, 11
pitch2d.w.error(), *6*, *17*
plot.angles, 12
plot.angles(), *2*
project2d.from.3d, 13
project2d.from.3d(), *6*, *17*, *18*
project2d.from.xy, 16
project2d.from.xy(), *4*, *6*, *11*, *15*

rad2deg *(*deg2rad*)*, 3
rotate3d, 17
rotate3d(), *14*, *15*
Rx *(*rotate3d*)*, 17
Ry *(*rotate3d*)*, 17
Rz *(*rotate3d*)*, 17

summarize.yaws, 18
summarize.yaws(), *11*

trim.yaws, 20