

Package: araponga (via r-universe)

July 2, 2026

Title Estimate 3D Orientations from 2D Landmarks

Version 1.0.1

Description Estimates possible 3D orientations of an object from the 2D image coordinates of two landmarks and an approximate camera-object elevation angle. Because a projected 2D angle can be compatible with multiple 3D pitch, yaw, and view-elevation angles, the package returns compatible sets of orientations rather than a single unconstrained estimate.

License GPL (>= 3)

Depends R (>= 4.1.0)

Imports arrow, dplyr, graphics, rlang

Suggests curl, httr, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

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

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

VignetteBuilder knitr

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 <https://cran.r-universe.dev>

Date/Publication 2026-07-02 18:10:02 UTC

RemoteUrl <https://github.com/cran/araponga>

RemoteRef HEAD

RemoteSha 25867782d7daebbbc60bca7e87cb7c43de6bbfe1

Contents

conventions	2
deg2rad	3
download.simdata	3
find.3d	4
pitch2d.from.3d	7
pitch2d.from.xy	8
pitch2d.w.error	9
plot.angles	10
rotate3d	12
summarize.yaws	13
trim.yaws	14
Index	16

conventions	<i>Plot angle conventions</i>
-------------	-------------------------------

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	<i>Convert degrees to radians, and vice versa</i>
---------	---

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))
```

download.simdata	<i>Download the precomputed simulation dataset</i>
------------------	--

Description

Download the precomputed simulation dataset used by [find.3d\(\)](#) and related functions.

Usage

```
download.simdata(overwrite = FALSE, quiet = FALSE)
```

Arguments

overwrite Logical scalar. If TRUE, re-download the archive even if a cached copy is already present.

quiet Logical scalar. If TRUE, suppress download progress and nonessential messages where supported by the underlying downloader.

Details

The dataset is stored under `tools::R_user_dir("araponga", "cache")`. If the archive is already available locally and `overwrite = FALSE`, the cached dataset is reused. See [find.3d\(\)](#) for how the dataset was constructed.

Value

An invisible character scalar giving the normalized path to the extracted simulation dataset directory.

find.3d	<i>Find combinations of 3D orientations that project to an observed 2D pitch</i>
---------	--

Description

Using precomputed simulation data, recover the combinations of yaw, pitch, and view elevations (at 1° resolution) that produce a given observed 2D pitch. `find.pitch()` and `find.yaw()` are convenient wrappers for two common uses: finding 3D pitch and yaw, respectively.

Usage

```
find.3d(
  pitch2d,
  find = NULL,
  candidate_view_elevations = NULL,
  candidate_pitches = NULL,
  candidate_yaws = NULL,
  label_error,
  label_nsamp = 625,
  sim_download = FALSE
)
```

```
find.yaw(
  pitch2d,
  candidate_view_elevations = NULL,
  candidate_pitches = NULL,
  candidate_yaws = NULL,
  paired = FALSE,
  label_error,
  label_nsamp = 625,
  sim_download = FALSE
)
```

```
find.pitch(
  pitch2d,
  candidate_pitches = NULL,
  candidate_yaws = NULL,
```

```

candidate_view_elevations = NULL,
paired = FALSE,
label_error,
label_nsamp = 625,
sim_download = FALSE
)

```

Arguments

pitch2d	Either a numeric scalar returned by <code>pitch2d.from.xy()</code> or a numeric vector of >1 candidate 2D pitch angles (e.g., as returned by <code>pitch2d.w.error()</code>), in degrees in the interval (-180, 180].
find	Character vector. Which angle(s) to return between "pitch", "yaw", "view_elevation", and "pitch2d". Default (NULL) returns all.
candidate_view_elevations	Optional numeric vector: known or candidate camera elevation angle(s) 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. Default is NULL (all view elevations considered). Provided values are rounded to the nearest integer.
candidate_pitches	Optional numeric vector: known or candidate (3D) pitch angle(s), in degrees, in the interval [-90, 90]. Convention: 90 = pointed up, 0 = horizontally aligned, -90 = pointed down. Default is NULL (all pitches considered). Provided values are rounded to the nearest integer.
candidate_yaws	Optional numeric vector: known or candidate yaw angle(s), in degrees, in the interval (-180, 180]. Convention: 0 = pointed right, 90 = pointed straight away, -90 = pointed straight toward, 180 = pointed left. Default is NULL (all yaws considered). Provided values are rounded to the nearest integer.
label_error	Positive numeric scalar specifying the error (\pm pixels) used to perturb landmark coordinates. Passed internally to <code>pitch2d.w.error()</code> . Required if <code>length(pitch2d) = 1</code> .
label_nsamp	Positive integer scalar specifying the approximate number of grid combinations to evaluate. Passed internally to <code>pitch2d.w.error()</code> . Required if <code>length(pitch2d) = 1</code> .
sim_download	Logical scalar. If TRUE, the function will attempt to download the precomputed simulation dataset automatically if it is not found in the local cache. Default is FALSE (CRAN-friendly).
paired	Logical scalar. If TRUE, a data.frame of yaws mapped to pitches will be returned; if FALSE (default), a vector of yaws or pitches.

Details

The lookup is based on a precomputed Blender simulation grid. A 3D pyramid was generated and rotated by pitch, then yaw, then roll (equivalently to what `rotate3d()` does) for all 11,793,960 integer combinations of pitch (-90:90), yaw (-179:180), and roll (-90:90). For each combination the base midpoint and tip were projected to 2D pixel coordinates with an orthographic camera. The cached simulation table stores the resulting projected geometry and derived 2D pitch values;

`find.3d()` (and wrappers) then filter that table for combinations compatible with the requested 2D pitch and any optional candidate constraints.

Value

A data.frame with all or a subset of the following columns:

pitch integer: pitch angles (degrees) compatible with the provided arguments.

yaw integer: yaw angles (degrees) compatible with the provided arguments.

pitch2d numeric: projected 2D pitch angles (degrees) compatible with the provided arguments.

view_elevation integer: elevation angles (degrees) compatible with the provided arguments.

If no matches are found, the returned data.frame has zero rows.

For `find.pitch(..., paired = FALSE)` and `find.yaw(..., paired = FALSE)`, an integer vector of pitch or yaw angles compatible with the provided arguments.

See Also

[pitch2d.from.xy\(\)](#), [pitch2d.w.error\(\)](#), [rotate3d\(\)](#), [download.simdata\(\)](#)

Examples

```
# test that dataset is downloaded before running examples
sim_path <- file.path(
  tools::R_user_dir("araponga", "cache"),
  "sim_data_parquet"
)

if(dir.exists(sim_path)){

# hypothetical pitch2d from coordinates
p2d <- pitch2d.from.xy(10, 1, -12, 20)

# pitches that project to `p2d` ( $\pm 2$  pixel error) if seen from  $30^\circ$  ( $\pm 1^\circ$  error) below
find.pitch(
  p2d,
  candidate_view_elevations = -29:-31,
  label_error = 2,
  sim_download = FALSE
)

# same returned values as from:
find.3d(
  p2d,
  find = "pitch",
  candidate_view_elevations = -29:-31,
  label_error = 2,
  sim_download = FALSE
)

# similar to above, now given yaw between  $5^\circ$  and  $15^\circ$ 
```

```

find.pitch(
  pitch2d = p2d,
  candidate_view_elevations = -29:-31,
  candidate_yaws = 5:15,
  label_error = 2,
  sim_download = FALSE
)

# yaws that project to `p2d` ( $\pm 2$  pixel error) if seen from  $10^\circ$  ( $\pm 2^\circ$  error) below
find.yaw(
  p2d,
  candidate_view_elevations = -8:-12,
  label_error = 2
)

} else {
message(
  "Run download.simdata() first to enable this example."
)
}

```

pitch2d.from.3d	<i>Calculate projected 2D pitch angle from orientations and view elevation</i>
-----------------	--

Description

Compute the 2D pitch produced by 3D orientations (pitch, yaw) when viewed from a specified elevation.

Usage

```
pitch2d.from.3d(pitch, yaw, view_elevation, plot = FALSE)
```

Arguments

pitch	Numeric scalar: vertical angle relative to horizontal plane, in degrees, in the interval $[-90, 90]$. Convention: $90 =$ pointed up, $0 =$ horizontally aligned, $-90 =$ pointed down.
yaw	Numeric scalar: horizontal angle 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 scalar: camera elevation angle 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.
plot	Logical scalar. TRUE draws a diagnostic plot with original and rotated axes, and calculated 2D pitch angle.

Details

The function constructs the full rotation matrix R using `rotate3d(pitch, yaw, view_elevation)` and computes the projected 2D pitch p_2 as

$$p_2 = \text{atan2}(R_{2,1}, R_{1,1})$$

Value

Numeric scalar: 2D pitch angle, in degrees, in the interval $(-180^\circ, 180^\circ]$. Convention: 0 points right, 90 points up, -90 points down, 180 points left.

See Also

[rotate3d\(\)](#), [pitch2d.from.xy\(\)](#)

Examples

```
# object pointed up 15 degrees
pitch2d.from.3d(15, 0, 0, plot = TRUE)
# object pointed up 15 and 30 degrees toward camera
pitch2d.from.3d(15, -30, 0, plot = TRUE)
# object pointed up 15 degrees, looked at from 30 degrees below
pitch2d.from.3d(15, 0, -30, plot = TRUE)
```

pitch2d.from.xy

Calculate projected 2D pitch from two landmarks

Description

Compute the 2D pitch angle defined by two landmarks: a tip point and a base point.

Usage

```
pitch2d.from.xy(x_tip, y_tip, x_base, y_base, plot = FALSE)
```

Arguments

<code>x_tip, x_base</code>	Numeric scalars or vectors: image x-coordinate(s) of tip and base of object, with x increasing right.
<code>y_tip, y_base</code>	Numeric scalars or vectors: image y-coordinate(s) of the tip and base of objects, with y increasing up.
<code>plot</code>	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 calculation uses the two-landmark vector

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

and returns

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

Value

Numeric vector of 2D pitch angles, in degrees, in the interval $(-180^\circ, 180^\circ]$. Convention: 0° points right, 90° points up, -90° points down, 180° points left.

Scalar outputs also include an "xy" attribute containing the input coordinates, which makes it directly compatible with `find.3d()`.

See Also

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

Examples

```
# scalar examples (plots)
pitch2d.from.xy(1, 0, 0, 0, plot = TRUE) # pointed right -> 0°
pitch2d.from.xy(-1, 0, 0, 0, plot = TRUE) # pointed left -> 180°
pitch2d.from.xy(0, 1, 0, 0, plot = TRUE) # pointed up -> 90°
pitch2d.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)
pitch2d.from.xy(x_tips, y_tips, x_bases, y_bases)
```

pitch2d.w.error

Simulate error in 2D pitch calculation from labeling error

Description

`pitch2d.w.error()` simulates the effect of landmark-labeling uncertainty on a 2D pitch value returned by `pitch2d.from.xy()`.

Usage

```
pitch2d.w.error(
  pitch2d,
  label_error,
  label_nsamp = 625,
  add_boundaries = FALSE
)
```

Arguments

pitch2d	Numeric scalar returned by <code>pitch2d.from.xy()</code> .
label_error	Positive numeric scalar specifying the error (\pm pixels) used to perturb each landmark coordinate.
label_nsamp	Positive integer scalar specifying the approximate number of grid combinations to evaluate. Default 625 uses a 5-point grid for each of the four coordinates, for $5^4 = 625$ total combinations.
add_boundaries	Logical scalar (default FALSE). If TRUE, boundary angles 0, 90, -90, and 180 are added when the simulated set spans them. This is useful when working with the returned angles individually, since these boundary values can behave differently from interior values.

Value

Numeric vector of unique simulated 2D pitch angles, in degrees, in the interval $(-180, 180]$.

See Also

[pitch2d.from.xy\(\)](#)

Examples

```
# A pitch value computed from landmark coordinates
p2d <- pitch2d.from.xy(x_tip = 100, y_tip = 80, x_base = 110, y_base = 120)

# Simulate the effect of  $\pm 1$  pixel labeling uncertainty
pitch2d.w.error(p2d, label_error = 1)

#' # Simulate the effect of  $\pm 5$  pixel labeling uncertainty
pitch2d.w.error(p2d, 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

# test that dataset is downloaded before running example
sim_path <- file.path(

```

```
tools::R_user_dir("araponga", "cache"),
"sim_data_parquet"
)

if(dir.exists(sim_path)){
## pitches and yaws that project to 10° (± 1° error) if seen from 15° below
df <- find.3d(
  9:11,
  find = c("pitch", "yaw"),
  candidate_view_elevations = -15,
  sim_download = FALSE
)
plot.angles(df)
} else {
message(
  "Run download.simdata() first to enable this example."
)
}
```

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 in degrees for rotation about the Z axis. Defaults to 0 (no rotation).

yaw Numeric scalar: angle in degrees for rotation about the Y axis. Defaults to 0 (no rotation).

roll Numeric scalar: angle in degrees for rotation about the X axis. Defaults to 0 (no rotation).

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	<i>Summarize yaw angles into the smallest continuous interval</i>
----------------	---

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

<code>yaws</code>	Numeric vector of yaw angles, in degrees, in the interval $(-180, 180]$.
<code>plot</code>	Logical scalar. TRUE draws a diagnostic plot with individual candidate angles and the returned smallest continuous interval that contains them. Works only when <code>length(yaws) > 1</code> .
<code>tie_action</code>	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()

# test that dataset is downloaded before running example
sim_path <- file.path(
  tools::R_user_dir("araponga", "cache"),
  "sim_data_parquet"
)

if(dir.exists(sim_path)){
  yaws <- find.yaw(
    pitch2d = 14:16,
    candidate_view_elevations = -45,
    candidate_pitches = 10:20,
    sim_download = FALSE
  )
  summarize.yaws(yaws, plot = TRUE)
} else {
  message(
    "Run download.simdata() first to enable this example."
  )
}

```

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	Non-negative numeric scalar: the minimum angular separation (degrees) allowed between a value in one set and a partner in the other set.
max_sep	Non-negative numeric scalar: the maximum angular separation (degrees) allowed between a value in one set and a partner in the other set.
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()` removes elements from each set that do not have at least one partner in the other set that is at least `min_sep` and up to `max_sep` clockwise (for `ccw_yaws` set) or counterclockwise (for `cw_yaws`) from it. In other words, each retained `ccw_yaws` angle will be $\geq \text{min_sep}$ and $\leq \text{max_sep}$ counterclockwise of at least one `cw_yaws` angle. Analogously, each retained `cw_yaws` angle will be $\geq \text{min_sep}$ and $\leq \text{max_sep}$ clockwise of at least one `ccw_yaws` angle.

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](#)

download.simdata, [3](#)

download.simdata(), [6](#)

find.3d, [4](#)

find.3d(), [3](#), [4](#), [9](#), [11](#)

find.pitch (find.3d), [4](#)

find.pitch(), [11](#)

find.yaw (find.3d), [4](#)

find.yaw(), [11](#)

pitch2d.from.3d, [7](#)

pitch2d.from.3d(), [9](#)

pitch2d.from.xy, [8](#)

pitch2d.from.xy(), [5](#), [6](#), [8–10](#)

pitch2d.w.error, [9](#)

pitch2d.w.error(), [5](#), [6](#), [9](#)

plot.angles, [10](#)

plot.angles(), [2](#)

rad2deg (deg2rad), [3](#)

rotate3d, [12](#)

rotate3d(), [5](#), [6](#), [8](#)

Rx (rotate3d), [12](#)

Ry (rotate3d), [12](#)

Rz (rotate3d), [12](#)

summarize.yaws, [13](#)

trim.yaws, [14](#)