

Package: anivis (via r-universe)

October 14, 2026

Type Package

Title An R package for visualizing movement data and diagnostics

Version 0.2.1.9003

Description An R package for visualizing movement data and diagnostics.

License MIT + file LICENSE

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

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

Depends R (>= 4.1.0)

Encoding UTF-8

Language en-GB

Imports lifecycle, rlang, ggplot2 (>= 3.5.0), patchwork, marquee, hms,
cli, grDevices, stats, anicore (>= 0.8.0.9004)

Suggests anicheck, animetric (>= 0.5.0.9006), testthat (>= 3.0.0)

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

Config/Needs/website animovement/animovementtemplate, knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-10-14 15:22:41 UTC

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

RemoteRef HEAD

RemoteSha 628cf29de6124e693d33a82e7444db5e2f8e2f38

Contents

as_plot_data 2

geom_event_point 4

geom_event_state 6

material_colors 7

okabeito_colors 8

palette_animovement 9

palette_material 10

palette_okabeito 10

plot.anievent 11

plot.anipoint 12

plot.anivis_check_confidence 12

plot.anivis_check_na_gapsize 13

plot.anivis_check_na_timing 15

plot.anivis_check_segment_length 16

plot_circular 17

plot_events 20

plot_timeseries 21

plot_trajectory 23

plots 24

scale_material 25

scale_okabeito 27

theme_animovement 28

theme_imputets 29

Index 31

as_plot_data	<i>Prepare a Check Result for Plotting</i>
--------------	--

Description

Coerces a check object — such as the result of `check_na_timing()` (from the `anichack` package) — into the plot-ready data frame that its `plot()` method draws from. This is the staging step that sits between the diagnostic computation (the `check_*()` function) and the `ggplot`: it reshapes the natural, analysis-friendly columns of the check into the exact aesthetic contract a geom needs (a single grouping factor, ordered fill levels, the right level ordering, and so on).

Usage

```
as_plot_data(x, ...)
```

Default S3 method:

```
as_plot_data(x, ...)
```

S3 method for class 'check_confidence'

```
as_plot_data(x, ..., clip = 0.02)
```

```
## S3 method for class 'check_na_gapsize'
as_plot_data(
  x,
  ...,
  ranked_by = c("occurrence", "total"),
  limit = 10,
  include_total = TRUE
)

## S3 method for class 'check_na_timing'
as_plot_data(x, ..., measure = c("percent", "count"), n_intervals = NULL)

## S3 method for class 'check_segment_length'
as_plot_data(x, ..., clip = 0.02)
```

Arguments

<code>x</code>	A check object.
<code>...</code>	Additional arguments passed to methods.
<code>clip</code>	For <code>as_plot_data.check_confidence()</code> : density floor (fraction of each keypoint's peak) below which the violin is cut. Default <code>0.02</code> .
<code>ranked_by, limit, include_total</code>	For <code>as_plot_data.check_na_gapsize()</code> : order bars by "occurrence" or "total", keep the top <code>limit</code> per group, and whether to include the total-NAs series.
<code>measure</code>	For <code>as_plot_data.check_na_timing()</code> : "percent" (default, share of each interval) or "count" (number of frames).
<code>n_intervals</code>	For <code>as_plot_data.check_na_timing()</code> : number of time intervals to bin into (default <code>NULL</code> uses Sturges' rule on the largest group).

Details

It is the anivis analogue of `data_plot()` in the **see** package. Most users never call it directly — `plot()` calls it for you. Reach for it when you want the precise rows a plot is built from, to inspect them or to assemble a custom chart by hand.

Methods exist for check objects with a single canonical plot (one object class maps to one figure). Object types that can be plotted several different ways — an `aniframe`, which feeds both `plot_trajectory()` and `plot_timeseries()` — deliberately do not have a single method, since there would be nothing for it to dispatch on to choose between those shapes.

`as_plot_data.check_confidence()` turns the per-keypoint density grid into closed violin polygons: each keypoint sits at an integer `y` position, and its density is mirrored either side along the confidence (`x`) axis (width-normalised so every violin has the same maximum width). Grid points below `clip` x the peak are dropped, so thin tails and bimodal-bridging necks disappear and a split distribution becomes separate polygons. keypoint is the `x`-axis category; any other identity that varies (e.g. individual) collapses into a group factor for faceting. The `x` positions, a per-keypoint median / quartile overlay, and facet ride along as attributes. Returns a frame classed `anivis_check_confidence_data`.

`as_plot_data.check_na_gapsize()` reshapes the gap-size table into the long, ranked form the imputeTS-style bar chart needs: one row per (group, gap size, series), where series is occurrence (and total when `include_total`), value the count, and key a `reorder_within`-style factor ("`<n>NA-gap___ <group>`") ordered by `ranked_by` so each facet sorts independently. Returns a frame classed `anivis_check_na_gapsize_data`.

`as_plot_data.check_na_timing()` reconstructs, from the compact gap table, the missing / present frame counts per time interval — one row per (group, interval, status). A single interval width (in frames) is chosen for all groups (Sturges' rule by default) so bars line up across panels, and each gap's overlap with each interval is counted, so no per-frame data is needed. `value` is the share (measure = "percent") or count ("count"), width the interval's span in time units, and the frame interval size rides along as an attribute. Returns a frame classed `anivis_check_na_timing_data`.

`as_plot_data.check_segment_length()` turns the per-track density grid of relative lengths into closed violin polygons, as for `check_confidence()`: each segment sits at an integer y position, the structure's first segment at the top, with its density mirrored either side along the relative-length (x) axis and cut below `clip` x its peak. `segment` is always the y axis; any other key that varies (an individual, a session) collapses into a group factor for faceting. A track that was never measured has nothing to draw and is left out. The y positions, a per-track overlay of the median and quartiles (capped at the check's `clamp`, like the grid) and the share off, the tolerance and the x positions of its guides, `clamp` and whether any track exceeded it (`clamped`), and facet ride along as attributes. Returns a frame classed `anivis_check_segment_length_data`.

Value

A data frame classed for the corresponding plot method (for example `anivis_check_na_timing_data`).

See Also

`plot.anivis_check_na_timing()`

Examples

```
af <- anicore::example_anipoint(n_obs = 50, n_individuals = 1, n_keypoints = 3)
af$xc[c(5:8, 20, 31:40)] <- NA

# The data the plot is drawn from, e.g. to build a custom plot from it
as_plot_data(anicore::check_na_timing(af))
```

`geom_event_point`

Draw point events as dots or raster ticks

Description

A ggplot2 layer that renders instantaneous events at each event's start time, with one row per label (or channel, ...) on the y axis. Two styles are available via `style`:

Usage

```
geom_event_point(
  mapping = NULL,
  data = NULL,
  ...,
  style = c("point", "raster"),
  size = 1.8,
  height = 0.7,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Aesthetic mapping. Merged on top of the defaults, so you only need to supply the aesthetics you want to override.
data	A data frame (typically an <code>anievent</code>). If <code>NULL</code> , inherits from the parent <code>ggplot()</code> call. Either way, rows with <code>type != "point"</code> are dropped from this layer.
style	Either <code>"point"</code> (default, a dot per event) or <code>"raster"</code> (a vertical tick per event).
size	Dot size, for <code>style = "point"</code> . Default 1.8.
height	Tick height in y-axis units (1 = the spacing between adjacent rows), for <code>style = "raster"</code> . Default 0.7.
na.rm, show.legend, inherit.aes, ...	Passed to <code>ggplot2::layer()</code> .

Details

- `"point"` (default) — a dot at each event time.
- `"raster"` — a short vertical tick at each event time, the classic spike-raster look (more standard for e.g. neuron spike trains).

When the input has a type column (e.g. an `anicore::anievent()`), rows with `type != "point"` are dropped — the point half of the state/point pair complementing `geom_event_state()`.

The default mapping is only `aes(x = start)` — every other aesthetic is up to the caller. Common patterns:

- `aes(colour = label)` — colour each mark by behaviour.
- `aes(y = label)` (or `y = channel`, `y = subject`, ...) — spread marks across rows. With no `y` mapping all marks collapse onto a single line.

Opinionated wrappers (themes, colour scales, sensible defaults) are intended to live one layer up; this geom is deliberately bare.

This is a thin wrapper around `ggplot2::GeomPoint` (`"point"`) or `ggplot2::GeomSegment` (`"raster"`), with a `setup_data` step that supplies a default `y` when none is mapped.

Value

A ggplot2 layer.

Examples

```
library(ggplot2)
af <- anicore::example_anipoint(n_obs = 6, n_individuals = 1, n_keypoints = 1)
af$behaviour <- rep(c("walk", "rest"), each = 3)
ev <- anicore::to_anievent(anicore::set_variables(af, event = list(state = "behaviour")))
ggplot(ev, aes(x = start, y = channel)) +
  geom_event_point()
```

geom_event_state	<i>Draw state events as horizontal bars</i>
------------------	---

Description

A ggplot2 layer that renders state (durative) events as rectangles spanning each event's start -> stop on the x axis, with one row per label on the y axis. When the input has a type column (e.g. an `anicore::anievent()`), rows with type != "state" are dropped.

Usage

```
geom_event_state(
  mapping = NULL,
  data = NULL,
  ...,
  height = 0.7,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Aesthetic mapping. Merged on top of the defaults, so you only need to supply the aesthetics you want to override.
data	A data frame (typically an <code>anievent</code>). If NULL, inherits from the parent <code>ggplot()</code> call. Either way, rows with type != "state" are dropped from this layer.
height	Vertical extent of each bar, in y-axis units (1 = the spacing between adjacent label rows). Default 0.7.
na.rm, show.legend, inherit.aes, ...	Passed to <code>ggplot2::layer()</code> .

Details

The default mapping is only `aes(xmin = start, xmax = stop)` — every other aesthetic is up to the caller. Common patterns:

- `aes(fill = label)` — colour each bout's interior by behaviour.
- `aes(colour = label)` — outline each bout (keeps sub-pixel-wide bouts visible at any zoom).
- `aes(y = label)` (or `y = channel`, `y = subject`, ...) — spread bouts across rows. With no `y` mapping all bouts collapse onto a single line.
- `+ ggplot2::facet_wrap(~ channel, scales = "free_y", ncol = 1)` — split channels into their own panels.

Opinionated wrappers (themes, colour scales, sensible defaults) are intended to live one layer up; this geom is deliberately bare.

This is a thin wrapper around [ggplot2::GeomRect](#): a subclass adds a `setup_data` step that derives `ymin / ymax` from `y + height`, so a discrete `y` scale keeps the label rows ranked correctly (including under per-panel scales = "free_y").

Value

A `ggplot2` layer.

Examples

```
library(ggplot2)
af <- anicore::example_anipoint(n_obs = 6, n_individuals = 1, n_keypoints = 1)
af$behaviour <- rep(c("walk", "rest"), each = 3)
ev <- anicore::to_anievent(anicore::set_variables(af, event = list(state = "behaviour")))
ggplot(ev, aes(xmin = start, xmax = stop, y = channel, fill = label)) +
  geom_event_state()
```

material_colors

Material design colours

Description

Extract one or more colours from the Material design palette. Vendored from the [see package](#).

Usage

```
material_colors(...)
```

Arguments

... Character names of colours to extract. If none are given, the full palette is returned.

Value

A named character vector of hex colours.

Examples

```
material_colors()
material_colors("red", "blue")
```

okabeito_colors

Okabe-Ito colours

Description

Extract one or more colours from the colour-blind-safe Okabe-Ito palette. `oi_colors()` is an alias. Vendored from the `see` package.

Usage

```
okabeito_colors(..., original_names = FALSE, black_first = FALSE, amber = TRUE)

oi_colors(..., original_names = FALSE, black_first = FALSE, amber = TRUE)
```

Arguments

<code>...</code>	Character names of colours to extract (e.g. "orange"). If none are given, the full ordered palette is returned.
<code>original_names</code>	Use the original Okabe & Ito colour names (e.g. "sky blue", "vermillion") rather than the plain-language ones.
<code>black_first</code>	Place black first in the returned palette.
<code>amber</code>	Use the slightly darkened "amber" yellow (#F5C710) instead of the original bright yellow (#F0E442), for better legibility.

Value

A named character vector of hex colours.

Examples

```
okabeito_colors()
oi_colors("blue", "orange")
```

palette_animovement	<i>Per-group colour palette for an aniframe</i>
---------------------	---

Description

Builds a named character vector mapping trajectory group keys to colours. The colour scheme adapts to how many identity (what) and temporal context (when, such as session or trial) levels the aniframe's grouping columns ([anicare::get_keys\(\)](#)) have:

Usage

```
palette_animovement(data, palette = "Dark 3", single_hue = "#3A6FB0")
```

Arguments

data	An aniframe.
palette	Name of a qualitative palette accepted by grDevices::hcl.colors() . Default "Dark 3".
single_hue	Colour used when neither what nor when has more than one level.

Details

- **single** (one of each, or none): a single hue.
- **what** (multiple what, one when): one qualitative hue per what level.
- **when** (one what, multiple when): one qualitative hue per when level.
- **matrix** (multiple of both): one hue per what level, then a lighter-to-darker shade of that hue per when level.

Names of the returned vector match the group keys returned by the internal grouping logic — the redundant axis is dropped when only one side varies.

Value

A named character vector of hex colours.

Examples

```
af <- anicare::example_anipoint(n_obs = 10, n_individuals = 2, n_keypoints = 1)
palette_animovement(af)
```

palette_material	<i>Material design colour palette</i>
------------------	---------------------------------------

Description

Palette generator for the Material design colours. Vendored from the see package.

Usage

```
palette_material(palette = "contrast", reverse = FALSE, ...)
```

Arguments

palette	One of "full", "ice", "gradient", "rainbow", "contrast", "light" or "complement".
reverse	Reverse the colour order.
...	Additional arguments passed to the underlying palette retriever.

Value

A function that returns n hex colours.

Examples

```
pal <- palette_material()
pal(3)
```

palette_okabeito	<i>Okabe-Ito colour palette</i>
------------------	---------------------------------

Description

Palette generator for the Okabe-Ito colours, for use with ggplot2 scales. Vendored from the see package.

Usage

```
palette_okabeito(palette = "full", reverse = FALSE, order = 1:9)
```

Arguments

palette	One of "full", "black_first", "full_original" or "black_original".
reverse	Reverse the colour order.
order	Integer vector (within 1:9) selecting and reordering colours.

Value

A function that returns n hex colours.

Examples

```
pal <- palette_okabeito()
pal(3)
```

plot.anievent

Plot an anievent Object

Description

Creates a visualization of state and point events from an `anicore::anievent()` using `geom_event_state()` and `geom_event_point()`. Returns a patchwork object that can be combined with additional plots. The drawing itself is done by `plot_events()`.

Usage

```
## S3 method for class 'anievent'
plot(x, ..., mode = c("light", "dark"))
```

Arguments

x	An anievent object.
...	Additional arguments passed to underlying plot functions.
mode	Either "light" (default) or "dark"; passed to <code>theme_animovement()</code> .

Value

A patchwork object.

See Also

`plot_events()`

Examples

```
events <- anicore::anievent(
  individual = 1L,
  channel = c("behaviour", "behaviour", "call"),
  label = c("REM", "wake", "alarm"),
  start = c(3, 14, 4.5),
  stop = c(9, 19, 4.5)
)
plot(events)
```

plot.anipoint	<i>Plot an anipoint Object</i>
---------------	--------------------------------

Description

[Experimental]

Creates a visualization of movement data stored in an `anicore::anipoint()`. Returns a patchwork object that can be combined with additional plots.

Today the figure is the trajectory alone, drawn by `plot_trajectory()`. What `plot()` shows may change without a deprecation cycle, for example to add speed and course traces beneath the path. Call `plot_trajectory()` directly when you need exactly that plot.

Usage

```
## S3 method for class 'anipoint'
plot(x, ..., mode = c("light", "dark"))
```

Arguments

x	An anipoint object.
...	Additional arguments passed to underlying plot functions.
mode	Either "light" (default) or "dark"; passed to <code>plot_trajectory()</code> .

Value

A patchwork object.

Examples

```
af <- anicore::example_anipoint(n_obs = 20, n_individuals = 2, n_keypoints = 1)
plot(af)
```

plot.anivis_check_confidence	<i>Plot the Distribution of Tracking Confidence</i>
------------------------------	---

Description

Renders a `check_confidence()` result (from the `anichack` package) as a horizontal violin per keypoint of its tracking confidence (keypoints on the y axis, confidence on the x), with a point at the median and a line spanning the inter-quartile range. A violin sitting far to the right and tight is a reliably tracked keypoint; a left-shifted body or a long low tail flags one the tracker was often unsure about. With several individuals, each gets its own panel.

Usage

```
## S3 method for class 'anivis_check_confidence'
plot(x, ..., clip = 0.02, mode = c("light", "dark"))
```

Arguments

x	A check_confidence object (from the anicheck package).
...	Additional arguments (currently unused).
clip	Cut each violin where its density falls below this fraction of its peak, so thin tails and the neck bridging a bimodal distribution are removed and the violin shows only where data actually is. Default 0.02; set 0 to keep the full density.
mode	Either "light" (default) or "dark"; passed to theme_imputets() .

Details

The violins are drawn from the kernel-density grid stored in the check object (via `geom_polygon`), so no raw values are needed. Styling matches the other check plots ([theme_imputets\(\)](#), horizontal-only gridlines). The plot is built from an intermediate frame of class `anivis_check_confidence_data` produced by [as_plot_data\(\)](#) — the staging step that mirrors `data_plot()` in [see](#).

Value

A ggplot object.

See Also

[as_plot_data\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 50, n_individuals = 1, n_keypoints = 3)
plot(anicheck::check_confidence(af))
```

plot.anivis_check_na_gapsize

Plot the Occurrence of Missing-Value Gap Sizes

Description

Renders a `check_na_gapsize()` result (from the anicheck package) as a horizontal bar chart of how often each gap size occurs — styled after `ggplot_na_gapsize()` from the imputeTS package (Moritz & Gatscha, GPL-3). For every gap length it draws the number of such gaps (indianred) and, optionally, the total missing frames they account for (steelblue, the gap size times its count). Bars are ranked so the most common (or most costly) gaps sit on top; only the top limit are shown. With more than one group, each gets its own panel.

Usage

```
## S3 method for class 'anivis_check_na_gapsize'
plot(
  x,
  ...,
  ranked_by = c("occurrence", "total"),
  limit = 10,
  include_total = TRUE,
  mode = c("light", "dark")
)
```

Arguments

x	A check_na_gapsize object (from the anicheck package).
...	Additional arguments (currently unused).
ranked_by	Order bars by "occurrence" (default, gap frequency) or "total" (resulting missing frames).
limit	Maximum number of gap sizes (bars) to show per group, keeping the top-ranked. Default 10.
include_total	Whether to add the steelblue "resulting NAs" bar alongside the occurrence bar. Default TRUE.
mode	Either "light" (default) or "dark"; passed to theme_animovement() .

Details

The plot is built from an intermediate frame of class `anivis_check_na_gapsize_data` produced by [as_plot_data\(\)](#) — the staging step that mirrors `data_plot()` in [see](#).

Value

A ggplot object.

References

Moritz, S., & Bartz-Beielstein, T. (2017). imputeTS: Time series missing value imputation in R. *The R Journal*, 9(1), 207-218. doi:[10.32614/RJ2017009](#).

See Also

[as_plot_data\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 50, n_individuals = 1, n_keypoints = 3)
af$x[c(5:8, 20, 31:40)] <- NA
plot(anicheck::check_na_gapsize(af))
```

plot.anivis_check_na_timing

Plot the Distribution of Missing Values Over Time

Description

Renders a `check_na_timing()` result (from the `anichack` package) as a stacked bar chart of missing (NA) versus present values across successive time intervals — styled after `ggplot_na_distribution2()` from the `imputeTS` package (Moritz & Gatscha, GPL-3). Each bar is one interval of the recording; its split shows how much of that stretch was missing, so a run of tall “NA” bars marks a blackout while an even sprinkle marks scattered dropouts. With more than one group, each gets its own stacked panel.

Usage

```
## S3 method for class 'anivis_check_na_timing'
plot(
  x,
  ...,
  measure = c("percent", "count"),
  n_intervals = NULL,
  mode = c("light", "dark")
)
```

Arguments

<code>x</code>	A <code>check_na_timing</code> object (from the <code>anichack</code> package).
<code>...</code>	Additional arguments (currently unused).
<code>measure</code>	Either "percent" (default, share of each interval) or "count" (number of frames).
<code>n_intervals</code>	Number of intervals to bin into. Default (NULL) uses Sturges' rule on the largest group's frame count.
<code>mode</code>	Either "light" (default) or "dark"; passed to <code>theme_imputets()</code> .

Details

The interval counts are reconstructed from the compact gap table in the check object (no per-frame data needed). Missing uses the `imputeTS` `indianred`, present the `imputeTS` `steelblue`, with the two named in the coloured subtitle (so no legend is needed). Styling is `theme_imputets()`. The plot is built from an intermediate frame of class `anivis_check_na_timing_data` produced by `as_plot_data()` — the staging step that mirrors `data_plot()` in `see`.

Value

A `ggplot` object.

References

Moritz, S., & Bartz-Beielstein, T. (2017). imputeTS: Time series missing value imputation in R. *The R Journal*, 9(1), 207-218. doi:[10.32614/RJ2017009](https://doi.org/10.32614/RJ2017009).

See Also

[as_plot_data\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 50, n_individuals = 1, n_keypoints = 3)
af$x[c(5:8, 20, 31:40)] <- NA
plot(anicheck::check_na_timing(af))
```

plot.anivis_check_segment_length

Plot how much segments vary from their usual length

Description

[Experimental]

Renders a `check_segment_length()` result (from the `anicheck` package) as a horizontal violin per segment of its length relative to its reference (segments on the y axis, relative length on the x), with a point at the median and a line spanning the inter-quartile range. A solid line marks the reference (1) and dashed lines the check's tolerance either side of it, and the share of frames beyond them is written at the right of each segment that has any. A segment that keeps its length is a narrow violin on the solid line; a wide one, or one reaching past the dashed lines, is a segment the tracking gets wrong. Segments run top to bottom in the structure's order, and with several individuals, each gets its own panel.

The violins are drawn from the kernel-density grid stored in the check object (via `geom_polygon`), so no raw lengths are needed. The check enters relative lengths above its clamp (2, unless the tolerance is large) into that grid at the clamp, so a segment with a few wild frames shows a bump at the right edge rather than stretching the axis; the plot's caption says so when it happens. Styling matches the other check plots ([theme_imputets\(\)](#), vertical gridlines only). The plot is built from an intermediate frame of class `anivis_check_segment_length_data` produced by [as_plot_data\(\)](#) — the staging step that mirrors `data_plot()` in [see](#).

Usage

```
## S3 method for class 'anivis_check_segment_length'
plot(x, ..., clip = 0.02, mode = c("light", "dark"))
```

Arguments

x	A check_segment_length object (from the anicheck package).
...	Additional arguments (currently unused).
clip	Cut each violin where its density falls below this fraction of its peak (default 0.02), so thin tails and the neck bridging a bimodal distribution are removed. Set 0 to keep the full density.
mode	Either "light" (default) or "dark"; passed to theme_impurities() .

Value

A ggplot object.

See Also

[as_plot_data\(\)](#), [plot.anivis_check_confidence\(\)](#)

Examples

```
af <- anicore::example_anipoint(n_obs = 100, n_individuals = 2) |>
  anicore::set_structure(anicore::example_structure())
plot(anicheck::check_segment_length(af))
```

plot_circular	<i>Plot the Distribution of an Angle</i>
---------------	--

Description**[Experimental]**

Draws an angular column of an aniframe as a circular histogram, or rose diagram: one wedge per bin of directions, its size the share of the group's angles that fall in the bin. Use it for any angle the frame stores in its angular unit, such as course from [animetric::add_kinematics\(\)](#) or a 2D orientation (heading) declared with [animetric::add_orientation\(\)](#).

The plot takes its conventions from the frame rather than assuming them:

- **Unit:** the column is read in the frame's `unit_angle` ([anicore::get_metadata\(\)](#)) and binned in radians ([anicore::angle_to_rad\(\)](#)). The angular axis is labelled in degrees or radians to match, and `binwidth` is given in the same unit. A frame declaring no angular unit is read as radians.
- **Zero and sense:** 0 points along +x, drawn to the right, and angles count from x toward y. Whether that is counter-clockwise or clockwise on the page comes from [anicore::get_angle_direction\(\)](#): a frame with y pointing up is drawn counter-clockwise, and one with y pointing down, such as image coordinates, clockwise. A frame whose axis directions are unknown is drawn counter-clockwise, the sense of `atan2()`.

- **Range:** angles are wrapped onto the circle before binning, so signed angles in $(-\pi, \pi]$, angles in $[0, 2\pi)$ and unwrapped angles all give the same plot, and a distribution straddling π is not split. The axis is labelled in the range the column uses: signed when it holds negative angles, otherwise from 0 to a full turn.

The bins are centred on multiples of the bin width, starting at 0, so with the default 36 bins of 10 degrees the four directions along the axes each fall in the middle of a bin.

Wedges show the share of each group's angles, or with weight, the share of its total weight. They start from an empty inner circle, `inner_radius` of the way out, which keeps the narrow tips of the wedges from crowding the centre; set it to 0 for a classic rose. By default a wedge's area, rather than its length, is proportional to its share, as in `circular::rose.diag()` and `tectonicr::rose()`: a wedge of share n reaches from the inner radius r_0 out to $\sqrt{r_0^2 + k * n}$, with k set so that the largest reaches the edge. With `equal_area = FALSE` its length is proportional to its share instead, which makes peaks stand out more but exaggerates them. The radial axis is labelled in shares either way. Missing angles and weights are left out.

Weighting is useful for directions of travel, such as course: when an animal is nearly still, its direction of travel is set by tracking jitter, and those frames can swamp the ones where it is going somewhere. Weighting by speed counts each frame by how far it moved.

With `show_mean = TRUE`, a thin line marks each group's circular mean direction (`anicores::circ_mean()`, weighted when weight is given), running from the inner circle to the edge. It shows the direction only, not how concentrated the angles are around it. When the angles are close to uniform, the mean resultant length is small and the mean direction is poorly defined, so read the line together with the wedges.

Groups are the frame's trajectory groups (`anicores::get_keys()`), coloured with `palette_animovement()` as in `plot_timeseries()`. The layout argument chooses how they are arranged:

- "facet" (default): each group gets its own panel, filled in its colour – `facet_grid()` when both an identity and a temporal context (such as trial) vary, otherwise `facet_wrap()`.
- "inline": all groups share one panel, each drawn as an outline in its colour, with a legend when there is more than one group.

Usage

```
plot_circular(data, ...)

## Default S3 method:
plot_circular(
  data,
  variable = NULL,
  ...,
  bins = NULL,
  binwidth = NULL,
  weight = NULL,
  equal_area = TRUE,
  inner_radius = 0.25,
  show_mean = TRUE,
  layout = c("facet", "inline"),
  mode = c("light", "dark"),
```

```
palette = "Dark 3"
)
```

Arguments

<code>data</code>	An aniframe object.
<code>...</code>	Additional arguments (currently unused).
<code>variable</code>	Name of the angular column to plot, a single string. Required (no auto-detection).
<code>bins</code>	Number of bins round the full circle, a single whole number. Defaults to 36, bins of 10 degrees, unless <code>binwidth</code> is given.
<code>binwidth</code>	Width of each bin, in the frame's angular unit, as an alternative to <code>bins</code> . It must divide a full turn into a whole number of bins, for example 15 in a frame in degrees or $\pi / 6$ in one in radians.
<code>weight</code>	Name of a numeric column to weight each angle by, such as "speed", or NULL (default) to count each angle once. Weights must not be negative.
<code>equal_area</code>	Whether a wedge's area (TRUE, default) or its length (FALSE) is proportional to its share.
<code>inner_radius</code>	Radius of the empty circle in the middle, as a fraction of the plot's radius, from 0 (a classic rose) up to but not including 1. Default 0.25.
<code>show_mean</code>	Whether to draw a line at each group's circular mean direction. Default TRUE.
<code>layout</code>	Either "facet" (default, one panel per group) or "inline" (all groups in one panel, as outlines).
<code>mode</code>	Either "light" (default) or "dark"; passed to <code>theme_animovement()</code> .
<code>palette</code>	Name of a qualitative palette accepted by <code>grDevices::hcl.colors()</code> ; controls the hue family across groups.

Value

A ggplot object.

See Also

`plot_timeseries()` for the same column against time.

Examples

```
af <- anicore::example_anipoint(n_obs = 200, n_individuals = 2, n_keypoints = 1)
af$course <- atan2(c(0, diff(af$y)), c(0, diff(af$x)))
plot_circular(af, variable = "course")

# Fewer, wider bins, with the groups overlaid
plot_circular(af, variable = "course", binwidth = pi / 6, layout = "inline")

# Course from animetric, and a heading declared from two keypoints, in a
# frame storing angles in degrees
kin <- anicore::example_anipoint(n_obs = 200, n_individuals = 1) |>
```

```

anicore::convert_unit_angle("deg") |>
animetric::add_orientation(
  from = "shoulder_right",
  to = "head",
  level = "keypoint"
) |>
animetric::add_kinematics()

# Each frame counted by how far it moved
plot_circular(kin, variable = "course", weight = "speed")
plot_circular(kin, variable = "heading", show_mean = FALSE)

```

plot_events

Plot State and Point Events

Description

Creates a ggplot that overlays state events (durative bouts, drawn as horizontal bars by `geom_event_state()`) and point events (instantaneous occurrences, drawn as dots by `geom_event_point()`) on a shared time axis. Each behaviour label becomes a row on the y axis.

Usage

```

plot_events(data, ...)

## S3 method for class 'anievent'
plot_events(
  data,
  ...,
  layout = c("facet", "inline"),
  point_style = c("point", "raster"),
  mode = c("light", "dark")
)

## Default S3 method:
plot_events(
  data = NULL,
  point = NULL,
  ...,
  point_style = c("point", "raster"),
  mode = c("light", "dark")
)

```

Arguments

<code>data</code>	An anievent object, or a data frame of state events for the default method.
<code>...</code>	Additional arguments (currently unused).

layout	For <code>plot_events.anievent()</code> : either "facet" (default, one y row per label with channels faceted) or "inline" (an ethogram — one y row per channel with labels in a colour legend).
point_style	How point events are drawn: "point" (default, dots) or "raster" (vertical ticks, the classic spike-raster look). Passed to <code>geom_event_point()</code> .
mode	Either "light" (default) or "dark"; passed to <code>theme_animovement()</code> .
point	For the default method only: an optional data frame of point events. Ignored by <code>plot_events.anievent()</code> .

Details

Two dispatches:

- `plot_events.anievent()` — auto-detects which rows are states and which are points by reading the type column. The layout argument chooses how channels and labels are arranged:
 - "facet" (default): each label is a y-axis row, and multiple channels become facet rows (with free y-scales). Any variables_what column with more than one level adds a facet column per identity — `facet_grid()` when both vary, otherwise `facet_wrap()`.
 - "inline" (ethogram): each channel is a single y-axis row and label moves to a colour legend, so all channels share one panel.
- `plot_events.default()` — for callers without an `anievent`. Pass the state events as the first arg and the point events via `point`; either may be NULL. No automatic faceting.

Value

A ggplot object.

Examples

```
af <- anicore::example_anipoint(n_obs = 6, n_individuals = 1, n_keypoints = 1)
af$behaviour <- rep(c("walk", "rest"), each = 3)
ev <- anicore::to_anievent(anicore::set_variables(af, event = list(state = "behaviour")))
plot_events(ev)
```

plot_timeseries

Plot a Variable as a Time Series

Description

Plots one or more numeric variables from an aniframe against time, with one line per trajectory group (every grouping column of the frame, `anicore::get_keys()`: its identity columns and its temporal context). Use it for any per-frame measure — speed, acceleration, a confidence score, a temperature channel, or any other derived column.

Usage

```
plot_timeseries(data, ...)

## Default S3 method:
plot_timeseries(
  data,
  variable = NULL,
  ...,
  layout = c("inline", "facet"),
  mode = c("light", "dark"),
  palette = "Dark 3"
)
```

Arguments

<code>data</code>	An aniframe object.
<code>...</code>	Additional arguments (currently unused).
<code>variable</code>	Name(s) of the column(s) to plot on the y axis — a character vector. Required (no auto-detection). Several names draw a stacked panel per variable.
<code>layout</code>	Either "inline" (default, all groups in one panel) or "facet" (one panel per group).
<code>mode</code>	Either "light" (default) or "dark"; passed to <code>theme_animovement()</code> .
<code>palette</code>	Name of a qualitative palette accepted by <code>grDevices::hcl.colors()</code> ; controls the hue family across groups.

Details

Time is read from the frame's index column (`aniframe::get_index()`), so it need not be called time.

Pass several names to `variable` to draw a panel per variable, **vertically stacked** with a shared x axis and a single shared legend (via `plots()`); stacking makes it easy to compare their shape over time. A single variable returns a plain ggplot.

The layout argument chooses how groups are arranged, mirroring `plot_events()`:

- "inline" (default): all groups share one panel as coloured lines, with a legend when there is more than one group.
- "facet": each group gets its own panel — `facet_grid()` when both an identity and a temporal context (such as trial) vary, otherwise `facet_wrap()` with the panels stacked in rows so the shared x axis lines up.

Colours come from `palette_animovement()`, matching `plot_trajectory()`. The x axis uses the same time-unit handling as `plot_events()` — HH:MM:SS for true time units, raw frames otherwise.

Value

A ggplot object for a single variable, or a stacked patchwork for several.

Examples

```
af <- anicore::example_anipoint(n_obs = 20, n_individuals = 2, n_keypoints = 1)
plot_timeseries(af, variable = "x")
```

plot_trajectory	<i>Plot Movement Trajectory</i>
-----------------	---------------------------------

Description

Creates a ggplot of the x-y trajectory from an anipoint. One path is drawn per trajectory group, where a group is the combination of the frame's grouping columns ([anicore::get_keys\(\)](#)): its identity columns and its temporal context, such as session or trial.

Usage

```
plot_trajectory(data, ...)

## Default S3 method:
plot_trajectory(data, ..., mode = c("light", "dark"), palette = "Dark 3")
```

Arguments

data	An anipoint object.
...	Additional arguments (currently unused).
mode	Either "light" (default) or "dark"; passed to theme_animovement() .
palette	Name of a qualitative palette accepted by grDevices::hcl.colors() ; controls the hue family across grouping levels.

Details

The positions are read from the columns the frame declares for its x and y axes ([anicore::get_axes\(\)](#)), and time from its index column ([anicore::get_index\(\)](#)), so neither has to be named x, y or time. The axes are labelled by role. A three-dimensional frame is drawn in its x-y plane. A frame without x and y axes, such as a polar one, is an error.

Colours adapt to the dataset shape:

- **single** trajectory (no grouping): the line is coloured continuously by time using the Material gradient scale ([scale_colour_material_c\(\)](#)), shown as a time colour bar.
- **what-only** or **when-only** grouping (one varying axis): each line gets its own hue from a qualitative palette, with time mapped to alpha so the line fades in from start to end — shown in a time legend.
- **matrix** grouping (both axes vary): each line is solid, coloured by the hue × shade matrix from [palette_animovement\(\)](#) — hue per what, shade per when; time reads from the start/end markers.

Every trajectory is annotated with a filled circle at its first point and a filled triangle at its last point, identified in a start/end legend. Gaps from missing data are bridged with a dashed line so the path stays traceable.

Value

A ggplot object.

Examples

```
af <- anicore::example_anipoint(n_obs = 20, n_individuals = 2, n_keypoints = 1)
plot_trajectory(af)
```

plots

Arrange multiple plots side by side

Description

A thin convenience wrapper around **patchwork** for composing several plots into a single figure with optional sub-figure tags, a shared legend, and an overall title / subtitle / caption. Vendored from the `see` package so anivis plots can be combined without reaching for patchwork directly.

Usage

```
plots(
  ...,
  n_rows = NULL,
  n_columns = NULL,
  guides = NULL,
  axes = NULL,
  axis_titles = NULL,
  tags = FALSE,
  tag_prefix = NULL,
  tag_suffix = NULL,
  tag_sep = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  theme = NULL
)
```

Arguments

... Plots to combine, or a single list of plots.

n_rows, n_columns Number of rows / columns in the layout. By default patchwork chooses a roughly square grid.

guides	How to treat legends: "collect" to pool identical legends into one, "keep" to leave them in place. Passed to <code>patchwork::wrap_plots()</code> .
axes, axis_titles	How to treat duplicated axes / axis titles: "collect", "collect_x", "collect_y" to share them across aligned panels, or "keep" (default). Passed to <code>patchwork::wrap_plots()</code> .
tags	Sub-figure tags. TRUE labels panels A, B, C, ...; a character vector such as <code>c("1", "2")</code> or "i" chooses another sequence; FALSE (default) adds none.
tag_prefix, tag_suffix, tag_sep	Strings placed before / after each tag, and the separator between nested tag levels.
title, subtitle, caption	Overall annotations for the combined figure.
theme	A ggplot2 theme applied to the annotations (e.g. the title).

Value

A patchwork object.

Examples

```
library(ggplot2)
p1 <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
p2 <- ggplot(mtcars, aes(factor(cyl))) + geom_bar()
plots(p1, p2, tags = TRUE)
```

scale_material	<i>Material design colour scales</i>
----------------	--------------------------------------

Description

Colour and fill scales using the Material design palette, vendored from the `see` package. With `discrete = FALSE` the palette is interpolated into a smooth gradient — the default "gradient" palette runs blue to orange, the warm-to-cool ramp suited to continuous data.

Usage

```
scale_colour_material(
  palette = NULL,
  discrete = TRUE,
  reverse = FALSE,
  aesthetics = "colour",
  ...
)
```

```
scale_color_material(
  palette = NULL,
```

```

    discrete = TRUE,
    reverse = FALSE,
    aesthetics = "colour",
    ...
)

scale_fill_material(
  palette = NULL,
  discrete = TRUE,
  reverse = FALSE,
  aesthetics = "fill",
  ...
)

scale_colour_material_d(palette = NULL, ...)
scale_color_material_d(palette = NULL, ...)
scale_colour_material_c(palette = NULL, ...)
scale_color_material_c(palette = NULL, ...)
scale_fill_material_d(palette = NULL, ...)
scale_fill_material_c(palette = NULL, ...)

```

Arguments

palette	Palette name (see palette_material()). Defaults to "contrast" when discrete and "gradient" when continuous.
discrete	Treat the scale as discrete (TRUE) or continuous (FALSE).
reverse	Reverse the colour order.
aesthetics	Aesthetics this scale applies to.
...	Passed to the underlying ggplot2 scale.

Value

A ggplot2 scale.

Examples

```

library(ggplot2)
ggplot(mtcars, aes(mpg, wt, colour = factor(cyl))) +
  geom_point() +
  scale_colour_material()

```

scale_okabeito	<i>Okabe-Ito colour scales</i>
----------------	--------------------------------

Description

Colour-blind-safe qualitative colour and fill scales using the Okabe-Ito palette, vendored from the `see` package. `*_oi()` are shorter aliases.

Usage

```
scale_colour_okabeito(  
  palette = "full",  
  reverse = FALSE,  
  order = 1:9,  
  aesthetics = "colour",  
  ...  
)
```

```
scale_color_okabeito(  
  palette = "full",  
  reverse = FALSE,  
  order = 1:9,  
  aesthetics = "colour",  
  ...  
)
```

```
scale_fill_okabeito(  
  palette = "full",  
  reverse = FALSE,  
  order = 1:9,  
  aesthetics = "fill",  
  ...  
)
```

```
scale_colour_oi(  
  palette = "full",  
  reverse = FALSE,  
  order = 1:9,  
  aesthetics = "colour",  
  ...  
)
```

```
scale_color_oi(  
  palette = "full",  
  reverse = FALSE,  
  order = 1:9,  
  aesthetics = "colour",  
  ...  
)
```

```

    ...
  )

  scale_fill_oj(
    palette = "full",
    reverse = FALSE,
    order = 1:9,
    aesthetics = "fill",
    ...
  )

```

Arguments

palette	One of "full", "black_first", "full_original" or "black_original".
reverse	Reverse the colour order.
order	Integer vector (within 1:9) selecting and reordering colours.
aesthetics	Aesthetics this scale applies to.
...	Passed to <code>ggplot2::discrete_scale()</code> .

Value

A ggplot2 scale.

Examples

```

library(ggplot2)
ggplot(mtcars, aes(mpg, wt, colour = factor(cyl))) +
  geom_point() +
  scale_color_okabeito()

```

theme_animovement	<i>Animovement ggplot2 Theme</i>
-------------------	----------------------------------

Description

The default animovement theme, with light and dark variants. It builds on `ggplot2::theme_linedraw()` — fine black panel border, thin gridlines — but swaps linedraw's black facet strips for the usual light-grey strips with dark text, and adds the plain enlarged title, generous axis-title margins and title positioning of `see::theme_modern()`. Further themes with their own names may be added over time.

Usage

```

theme_animovement(mode = c("light", "dark"), base_size = 11, base_family = "")

theme_animovement_light(base_size = 11, base_family = "")

theme_animovement_dark(base_size = 11, base_family = "")

```

Arguments

mode	Either "light" (default) or "dark".
base_size	Base font size.
base_family	Base font family.

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  theme_animovement()
```

theme_imputets	<i>imputeTS-style Theme for Check Plots</i>
----------------	---

Description

A larger-font variant of `theme_animovement()` tuned for the diagnostic check plots (`plot.anivis_check_na_timing()`, `plot.anivis_check_na_gapsize()`, `plot.anivis_check_confidence()`), styled after the `imputeTS` package figures. Compared with `theme_animovement()` it uses a larger base size, places the title directly above the panel (not over the axis), renders the subtitle as markdown via **marquee** — so words like NA / non-NA can be coloured inline with the `.na` / `.nona` classes — and puts the legend at the bottom.

Usage

```
theme_imputets(mode = c("light", "dark"), base_size = 13)
```

Arguments

mode	Either "light" (default) or "dark".
base_size	Base font size. Default 13.

Value

A ggplot2 theme object.

References

Moritz, S., & Bartz-Beielstein, T. (2017). `imputeTS`: Time series missing value imputation in R. *The R Journal*, 9(1), 207-218. doi:[10.32614/RJ2017009](https://doi.org/10.32614/RJ2017009).

See Also[theme_animovement\(\)](#)**Examples**

```
library(ggplot2)
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  theme_imputets()
```

Index

`anicore::angle_to_rad()`, 17
`anicore::anievent()`, 5, 6, 11
`anicore::anipoint()`, 12
`anicore::circ_mean()`, 18
`anicore::get_angle_direction()`, 17
`anicore::get_axes()`, 23
`anicore::get_index()`, 22, 23
`anicore::get_keys()`, 9, 18, 21, 23
`anicore::get_metadata()`, 17
`as_plot_data`, 2
`as_plot_data()`, 13–17

`geom_event_point`, 4
`geom_event_point()`, 11, 20, 21
`geom_event_state`, 6
`geom_event_state()`, 5, 11, 20
`ggplot2::discrete_scale()`, 28
`ggplot2::GeomPoint`, 5
`ggplot2::GeomRect`, 7
`ggplot2::GeomSegment`, 5
`ggplot2::layer()`, 5, 6
`ggplot2::theme_linedraw()`, 28
`grDevices::hcl.colors()`, 9, 19, 22, 23

`material_colors`, 7

`oi_colors(okabeito_colors)`, 8
`okabeito_colors`, 8

`palette_animovement`, 9
`palette_animovement()`, 18, 22, 23
`palette_material`, 10
`palette_material()`, 26
`palette_okabeito`, 10
`patchwork::wrap_plots()`, 25
`plot.anievent`, 11
`plot.anipoint`, 12
`plot.anivis_check_confidence`, 12
`plot.anivis_check_confidence()`, 17, 29
`plot.anivis_check_na_gapsizes`, 13
`plot.anivis_check_na_gapsizes()`, 29
`plot.anivis_check_na_timing`, 15
`plot.anivis_check_na_timing()`, 4, 29
`plot.anivis_check_segment_length`, 16
`plot_circular`, 17
`plot_events`, 20
`plot_events()`, 11, 22
`plot_timeseries`, 21
`plot_timeseries()`, 3, 18, 19
`plot_trajectory`, 23
`plot_trajectory()`, 3, 12, 22
`plots`, 24
`plots()`, 22

`scale_color_material(scale_material)`, 25
`scale_color_material_c(scale_material)`, 25
`scale_color_material_d(scale_material)`, 25
`scale_color_oi(scale_okabeito)`, 27
`scale_color_okabeito(scale_okabeito)`, 27
`scale_colour_material(scale_material)`, 25
`scale_colour_material_c(scale_material)`, 25
`scale_colour_material_c()`, 23
`scale_colour_material_d(scale_material)`, 25
`scale_colour_oi(scale_okabeito)`, 27
`scale_colour_okabeito(scale_okabeito)`, 27
`scale_fill_material(scale_material)`, 25
`scale_fill_material_c(scale_material)`, 25
`scale_fill_material_d(scale_material)`, 25
`scale_fill_oi(scale_okabeito)`, 27
`scale_fill_okabeito(scale_okabeito)`, 27

scale_material, [25](#)
scale_okabeito, [27](#)

theme_animovement, [28](#)
theme_animovement(), [11](#), [14](#), [19](#), [21–23](#), [29](#),
 [30](#)
theme_animovement_dark
 (theme_animovement), [28](#)
theme_animovement_light
 (theme_animovement), [28](#)
theme_imputets, [29](#)
theme_imputets(), [13](#), [15–17](#)