

# Package: anicheck (via r-universe)

October 14, 2026

**Type** Package

**Title** An R package for diagnosing movement data quality

**Version** 0.3.0.9004

**Description** An R package for diagnosing movement data quality.

**License** MIT + file LICENSE

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

**BugReports** <https://github.com/animovement/anicheck/issues>

**Encoding** UTF-8

**Language** en-GB

**Depends** R (>= 4.1.0)

**Imports** anicore (>= 0.8.0.9003), cli, lifecycle, rlang, stats, utils

**Suggests** anivis (>= 0.2.1.9003), testthat (>= 3.3.0)

**Additional\_repositories** <https://animovement.r-universe.dev>

**Config/Needs/website** animovement/animovementtemplate

**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 15:13:14 UTC

**RemoteUrl** <https://github.com/animovement/anicheck>

**RemoteRef** HEAD

**RemoteSha** ce7986f9f4c837d69b1d53b8870a1d643025dc64

## Contents

|                                |   |
|--------------------------------|---|
| check_confidence . . . . .     | 2 |
| check_na_gapsize . . . . .     | 3 |
| check_na_timing . . . . .      | 4 |
| check_segment_length . . . . . | 6 |

check\_confidence

*Check the Distribution of Tracking Confidence*

## Description

Summarises the per-frame tracking confidence (or likelihood) of each keypoint. A keypoint with a low median or a long low tail is one the tracker was often unsure about - exactly the points whose coordinates deserve suspicion. This check reduces the confidence column to a per-keypoint distribution, and `plot.check_confidence()` draws it as a violin.

## Usage

```
check_confidence(data, ...)

## Default S3 method:
check_confidence(data, ...)

## S3 method for class 'anipoint'
check_confidence(data, n = 256, ...)
```

## Arguments

|                   |                                                        |
|-------------------|--------------------------------------------------------|
| <code>data</code> | An anipoint (position frame) with a confidence column. |
| <code>...</code>  | Additional arguments (currently unused).               |
| <code>n</code>    | Density grid resolution per keypoint. Default 256.     |

## Details

The distribution is reduced to a compact kernel-density estimate (a fixed-size grid) per keypoint, plus a five-number summary, so the object stays small however long the recording - the violin is drawn straight from the stored density.

This is the data-generating half of the check. The plotting method (`plot.check_confidence()`) lives in **anivis**, mirroring the **performance** / **see** split in **easystats**. (`check_*`) functions are destined for the **anicheck** package; they are kept here for now for convenience.)

## Value

A data frame of class `check_confidence` with one row per (keypoint, density grid point): the identity columns, the confidence value, and its kernel density. A per-keypoint five-number summary (with `n`, mean, sd) and the grouping columns are stored as attributes. Use `summary()` for a trimmed overview.

## See Also

`plot.check_confidence()`

## Examples

```
af <- anicore::as_anipoint(data.frame(
  keypoint = rep(c("head", "tail"), each = 50),
  time = rep(1:50, 2),
  x = rnorm(100),
  y = rnorm(100)
))
af$confidence <- c(rbeta(50, 8, 2), rbeta(50, 2, 5))
check_confidence(af)

# Plot the confidence distributions (drawn by anivis)
plot(check_confidence(af))
```

---

|                  |                                                          |
|------------------|----------------------------------------------------------|
| check_na_gapsize | <i>Check the Distribution of Missing-Value Gap Sizes</i> |
|------------------|----------------------------------------------------------|

---

## Description

Tabulates the *lengths* of the runs of consecutive missing values (NA) in an anipoint - how often a gap of each size occurs. A recording riddled with single-frame dropouts (easy to interpolate) has a very different gap-size profile from one with a few long blackouts (which interpolation cannot rescue), even when their total missing counts match; this check exposes that profile, and `plot.check_na_gapsize()` draws it as a bar chart.

## Usage

```
check_na_gapsize(data, ...)

## Default S3 method:
check_na_gapsize(data, ...)

## S3 method for class 'anipoint'
check_na_gapsize(data, variable = "x", ...)
```

## Arguments

|                       |                                                                                                                                           |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <code>data</code>     | An anipoint (position frame).                                                                                                             |
| <code>...</code>      | Additional arguments (currently unused).                                                                                                  |
| <code>variable</code> | Name(s) of the column(s) whose missingness to track. A frame counts as missing when <i>any</i> named column is NA there. Defaults to "x". |

## Details

This is the data-generating half of the check. The plotting method (`plot.check_na_gapsize()`) lives in **anivis**, mirroring the **performance** / **see** split in easystats. (`check_*()` functions are destined for the **anicheck** package; they are kept here for now for convenience.)

**Value**

A data frame of class `check_na_gapsize` with one row per (group, gap size): the anipoint's grouping columns, the `gap_size` (run length in frames), the number of gaps of that size (`n_gaps`), and the total missing frames they account for (`n_na = gap_size x n_gaps`). Per-group totals and the checked variable(s) are stored as attributes. Use `summary()` for a per-group overview.

**See Also**

`plot.check_na_gapsize()`

**Examples**

```
af <- anicore::as_anipoint(data.frame(
  keypoint = rep(c("head", "tail"), each = 8),
  time = rep(1:8, 2),
  x = c(1, NA, NA, 4, NA, NA, 7, 8, 1, NA, 3, 4, 5, 6, 7, 8)
))
check_na_gapsize(af)
summary(check_na_gapsize(af))

# Plot the gap sizes (drawn by anivis)
plot(check_na_gapsize(af))
```

---

check\_na\_timing

*Check the Timing of Missing Values*

---

**Description**

Finds the runs of consecutive missing values (NA) in an anipoint and returns them as a compact table - **one row per gap**, not per frame. Where the gaps fall (at the start, the end, scattered, or in long bursts) is often more telling than how many there are; this check exposes that timing, and `plot.check_na_timing()` draws it as a missingness strip.

**Usage**

```
check_na_timing(data, ...)

## Default S3 method:
check_na_timing(data, ...)

## S3 method for class 'anipoint'
check_na_timing(data, variable = "x", ...)
```

**Arguments**

|          |                                                                                                                                           |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------|
| data     | An anipoint (position frame).                                                                                                             |
| ...      | Additional arguments (currently unused).                                                                                                  |
| variable | Name(s) of the column(s) whose missingness to track. A frame counts as missing when <i>any</i> named column is NA there. Defaults to "x". |

**Details**

Returning gaps rather than per-frame rows keeps the object (and any plot built from it) tiny even for million-frame recordings: its size scales with the number of gaps, not the length of the data.

This is the data-generating half of the check. The plotting method (`plot.check_na_timing()`) lives in **anivis**, mirroring the **performance** / **see** split in easystats: `check_*`() computes a classed object with `summary()` / `print()` methods, and a `plot.*()` method in the companion package draws it. (`check_*`() functions are destined for the **anichack** package; they are kept here for now for convenience.)

**Value**

A data frame of class `check_na_timing` with one row per missing run: the anipoint's grouping columns (every `variables_what` and context `variables_when` column), the run's start and stop time, and its length in frames. Per-group totals (frame and missing counts, time range), the checked variable(s), the time unit, and the typical time step are stored as attributes for the `summary()`, `print()`, and plotting methods. Use `summary()` for a per-group overview.

**See Also**

`plot.check_na_timing()`, `summary.check_na_timing()`

**Examples**

```
af <- anicore::as_anipoint(data.frame(
  keypoint = rep(c("head", "tail"), each = 6),
  time = rep(1:6, 2),
  x = c(1, NA, NA, 4, 5, 6, 1, 2, 3, 4, 5, 6)
))
check_na_timing(af)
summary(check_na_timing(af))

# Plot when the values are missing (drawn by anivis)
plot(check_na_timing(af))
```

---

check\_segment\_length    *Check how much segments vary from their usual length*

---

## Description

### [Experimental]

Measures how far each segment of a structure strays from its usual length. A keypoint can be confident and present in every frame and still be in the wrong place — a left/right swap, a limb taken from a neighbouring individual, a wrong match that several cameras agree on — and the segments that should keep their length then stretch and shrink. This check reports, per individual and segment, how much the length varies and how often it is far from its reference, and `plot.check_segment_length()` draws the distribution of each segment's length relative to that reference.

## Usage

```
check_segment_length(data, ...)

## Default S3 method:
check_segment_length(data, ...)

## S3 method for class 'anipoint'
check_segment_length(data, structure = NULL, tolerance = 0.3, n = 256, ...)
```

## Arguments

|           |                                                                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| data      | A Cartesian 2D or 3D anipoint with a structure that has segments, attached with <code>anicore::set_structure()</code> .                                                                   |
| ...       | Additional arguments (currently unused).                                                                                                                                                  |
| structure | Name of the structure to use (default NULL). May be omitted when only one of the frame's structures has segments.                                                                         |
| tolerance | A single non-negative number (default 0.3): how far, as a share of the reference length, a segment's length may differ from it before the frame counts as off. 0.3 allows 30% either way. |
| n         | Density grid resolution per track (default 256).                                                                                                                                          |

## Details

The segment lengths per frame are those of `anicore::as_anisegment()`. Each track — each combination of the frame's keys, so each individual and segment (and session or trial, when the frame has them) — is compared with a **reference length**: the structure's length for that segment when it records one, otherwise the median of the segment's length over the track. A structure's length applies to every individual, so leave it NA to let each individual be measured against its own median.

For each track the check reports

- the **spread** of the length, robustly: mad, the median absolute deviation (scaled by 1.4826, so it estimates the standard deviation of normally distributed lengths) in the frame's units, and cv, that MAD divided by the median length — a robust coefficient of variation, which compares segments of different sizes. Both measure how much the length varies, whatever the reference.
- the **share off**: the share of frames whose length differs from the reference by more than tolerance times the reference. It counts the frames with a measured length; a frame missing either end point is neither on nor off.

A segment that is consistently longer than the structure says shows up in the share off and in its median, not in its spread.

The distribution of each track's length relative to its reference is reduced to a kernel-density grid, as `check_confidence()` does, so the object stays small however long the recording. One wild frame (a knee placed on the far side of the arena) would otherwise stretch that grid until the bulk of the distribution falls between two grid points, so relative lengths above  $\max(2, 1 + 2 * \text{tolerance})$  enter the grid at that value. The summary statistics use the lengths as they are.

This is the data-generating half of the check. The plotting method (`plot.check_segment_length()`) lives in **anivis**, mirroring the **performance** / **see** split in easystats.

## Value

A data frame of class `check_segment_length` with one row per (track, density grid point): the grouping columns (the frame's keys, with segment in place of the structure's variable), the relative length value (length divided by the reference), and its kernel density. Stored as attributes:

- **groups**: one row per track, with the reference length and where it came from (`reference_source`, "structure" or "median"), the number of frames with a measured length (`n`), how many of them are off (`n_off`) and their share (`share_off`), the median length, its mad and cv, and the five-number summary of the relative length (`relative_min`, `relative_q25`, `relative_median`, `relative_q75`, `relative_max`).
- **off**: one row per run of consecutive frames off, with its start and stop (in the frame's index), its length in frames (`n_frames`), and the largest relative deviation within it (`max_deviation`, so 0.5 is 50% longer or shorter than the reference) — when the segments are off.
- **structure**, **tolerance**, the relative length above which values enter the density grid at that value (`clamp`), and the grouping columns.

Use `summary()` for one row per track.

## See Also

`plot.check_segment_length()`, `anicore::as_anisegment()`, `anicore::anistructure()`

## Examples

```
af <- anicore::example_anipoint(n_obs = 50, n_individuals = 1) |>
  anicore::set_structure(anicore::example_structure())
chk <- check_segment_length(af)
chk
summary(chk)
```

```
# When the segments were off
attr(chk, "off")

# Plot the segment lengths (drawn by anivis)
plot(chk)
```

# Index

`anicore::anistructure()`, [7](#)  
`anicore::as_anisegment()`, [6](#), [7](#)  
`anicore::set_structure()`, [6](#)  
  
`check_confidence`, [2](#)  
`check_confidence()`, [7](#)  
`check_na_gapsize`, [3](#)  
`check_na_timing`, [4](#)  
`check_segment_length`, [6](#)  
  
`plot.check_confidence()`, [2](#)  
`plot.check_na_gapsize()`, [3](#), [4](#)  
`plot.check_na_timing()`, [4](#), [5](#)  
`plot.check_segment_length()`, [6](#), [7](#)  
  
`summary()`, [2](#), [4](#), [5](#), [7](#)  
`summary.check_na_timing()`, [5](#)