

Package: sleepR (via r-universe)

October 7, 2026

Title Analyse Activity and Sleep Behaviour

Date 2026-10-07

Version 0.4.0

Description Use behavioural variables to score activity and infer sleep from bouts of immobility. It is primarily designed to score sleep in fruit flies from Drosophila Activity Monitor (TriKinetics) and Ethoscope data. It implements sleep scoring using the "five-minute rule" (Hendricks et al. (2000) <[DOI:10.1016/S0896-6273\(00\)80877-6](https://doi.org/10.1016/S0896-6273(00)80877-6)>), activity classification for Ethoscopes (Geissmann et al. (2017) <[DOI:10.1371/journal.pbio.2003026](https://doi.org/10.1371/journal.pbio.2003026)>) and a new algorithm to detect when animals are dead.

Depends R (>= 3.00), behavr

Imports data.table

Suggests testthat, covr, knitr

License GPL-3

Encoding UTF-8

LazyData true

URL <https://github.com/rethomics/sleepR>

BugReports <https://github.com/rethomics/sleepR/issues>

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

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

Date/Publication 2026-10-07 18:44:58 UTC

RemoteUrl <https://github.com/rethomics/sleepR>

RemoteRef HEAD

RemoteSha 4f9650952909422fd91cf802410d0f4f7dc24572

Contents

bout_analysis	2
curate_dead_animals	3
motion_calibration	4
motion_detectors	6
motion_qc	8
sleep_annotation	9
sleep_rules	11
Index	14

bout_analysis	<i>Find "bouts" in categorical time series</i>
---------------	--

Description

This function is used to find contiguous regions of unique value in a – potentially irregular/heterogeneous – univariate categorical time series.

Usage

```
bout_analysis(var, data)
```

Arguments

var	name of the variable to use from data
data	data.table::data.table containing behavioural variable from or one multiple animals. When it has a key, unique values, are assumed to represent unique individuals (e.g. in a beavr::beavr table). Otherwise, it analysis the data as coming from a single animal. data must have a column t representing time.

Value

an object of the same type as data (i.e. [data.table::data.table](#) or [beavr::beavr](#)). Each row is a specific bout characterised by three columns.

- t – its *onset*
- duration – its length
- <var> – a column with the same name as var. The value of var for this bout.

References

- The relevant [rethomic tutorial section](#) – on sleep analysis

See Also

- [sleep_annotation](#) – to generate a binary sleep variable
- [rle](#) run length encoding function – on which this analysis is based

Examples

```
# Bout analysis on binary variable:
dt <- toy_dam_data()
dt[, moving := activity > 0]
bdt <- bout_analysis(moving,dt)
print(bdt)
# With multiple states
dt <- toy_ethoscope_data()
# we discretise x position in three states: left, middle and right (1/3 each)
dt[, location := as.character( cut(x,
                                breaks = c(0.0, .33, .67, 1.0),
                                labels = c("left", "middle", "right")))]

bdt <- bout_analysis(location, dt)
```

curate_dead_animals	<i>Remove – irrelevant – data after individual have died</i>
---------------------	--

Description

This function detects when individuals have died based on their first (very) long bout of immobility. Following data (which may include spurious artefact of movement) are removed.

Usage

```
curate_dead_animals(
  data,
  moving_var = moving,
  time_window = hours(24),
  prop_immobile = 0.01,
  resolution = 24
)
```

Arguments

data	data.table::data.table containing behavioural variable from or one multiple animals. When it has a key, unique values, are assumed to represent unique individuals (e.g. in a behavr::behavr table). Otherwise, it analysis the data as coming from a single animal. data must have a column t representing time.
moving_var	logical variable in data used to define the moving (alive) state (default is moving)
time_window	window during which to define death (default is one day)
prop_immobile	proportion of immobility that counts as "dead" during time_window (see details)
resolution	how much scanning windows overlap. Expressed as a factor (see details).

Details

This function scans the time series looking for "death" in the right (future) data, within `time_window`. Death is defined as `mean(moving_var) < prop_immobile` within a time window. Moving window start every `time_window/resolution`. `resolution = 1` is fast but means no overlap. The default would score an animal as dead if it does not move more than *one percent of the time* for at least *one day*. All data following a "death" event are removed.

Value

an object of the same type as data (i.e. `data.table::data.table` or `behavr::behavr`).

See Also

- [sleep_annotation](#) – to score movement and sleep

Examples

```
dt1 <- toy_activity_data()
#all movement after day 3 is set at 0
dt1[t > days(3), moving := FALSE]
# one artefact of movement is detected at day 3.5
dt1[t == days(3.5), moving := TRUE]

dt2 <- curate_dead_animals(dt1)
dt3 <- curate_dead_animals(dt1, prop_immobile = 0)
## Not run:
library(ggplot2)
ggplot(data=dt1[, test:=1], aes(t, as.numeric(moving))) +
  geom_line(data=dt1[, test:=1]) +
  geom_line(data=dt2[, test:=2]) +
  geom_line(data=dt3[, test:=3]) +
  facet_grid(test ~ .) +
  scale_x_time()

## End(Not run)
```

motion_calibration *Tracking-noise diagnostics*

Description

[max_velocity_detector](#) calls a time window "moving" when its peak corrected velocity exceeds a threshold. When a motionless animal's tracking noise sits near it, spurious "movements" fragment every immobility bout: a bout of 5 min needs 30 consecutive immobile 10-s windows, so a per-window false positive rate p leaves only $(1 - p)^{30}$ of true rests intact.

Usage

```

pixel_size(x)

default_still_shift(x)

light_phase(t, day_length = 24, lights_off = 12)

find_spikes(
  t,
  x,
  y,
  max_frames = 2,
  min_jump = NULL,
  tolerance = NULL,
  max_span = 2
)

find_still_bins(
  d_small,
  time_window_length = 10,
  context_bins = 3,
  max_shift = NULL
)

```

Arguments

x	x positions of one animal.
t	time in seconds, 0 at lights on.
day_length	length of the day in hours.
lights_off	hour of lights off.
y	y positions.
max_frames	longest displaced run treated as a spike.
min_jump	smallest displacement treated as a jump; NULL is 3 pixels.
tolerance	how close the return must be; NULL is 1 pixel.
max_span	longest time from the last good frame to the return, in seconds.
d_small	binned data of <i>a single animal</i> , with columns t, x, y (mean position per window).
time_window_length	window size in seconds.
context_bins	windows on each side used to judge stillness (3, i.e. 30 s).
max_shift	largest positional change still counted as still, in the units of x/y. NULL uses one pixel (default_still_shift).

Details

These helpers support [motion_qc](#), which reports how often a still animal crosses the fixed threshold. Windows in which the animal is still are found *from position alone* – its position over the half-minute either side of the window stays within one pixel – so the check does not depend on the velocity it checks. Where the fixed threshold fails, score sleep with `sleep_annotation(rule = "k")` ([sleep_rules](#)). The algorithms and defaults are identical to `ethoscopy`'s `motion_calibration` module.

Functions

- `pixel_size()`: one pixel in the units of x: positions normalised to the ROI width (as `scor` loads them) or in pixels.
- `default_still_shift()`: positional tolerance for a still window, 1 pixel in the units of x. Validated against tracker-independent pixel motion on two recordings: at 1 px the calibrated threshold matched the one chosen from the pixels; at 0.5 px it skipped still flies whose tracked position jitters and came out too low.
- `light_phase()`: "light" / "dark" label for each time point.
- `find_spikes()`: tracking spikes: the centroid jumps more than `min_jump` away for up to `max_frames` frames and lands back within tolerance of where it was, within `max_span` seconds – the tracker briefly locking onto something other than the animal. Returns a list of two logical vectors over frames: `velocity` (displaced frames and the return frame) and `position` (displaced frames only).
- `find_still_bins()`: logical vector, TRUE for windows in which the animal did not change position. The median positions before and after the window must agree, and the positions across the whole window must agree once the single highest and lowest window are set aside, so that a glitch in one window is forgiven but pacing back and forth is not.

See Also

[max_velocity_detector](#), [motion_qc](#), [sleep_rules](#)

motion_detectors

Motion detector for Ethocope data

Description

Defines whether a *single animal* is moving according to:

Usage

```
max_velocity_detector(
  data,
  time_window_length,
  velocity_correction_coef = 0.003,
  masking_duration = 6
)
```

```
max_velocity_detector_legacy(data, velocity_threshold = 0.006)
```

```
virtual_beam_cross_detector(data, time_window_length)
```

Arguments

data [data.table::data.table](#) containing behavioural variables of *a single animal* (no id). It must have the columns `xy_dist_log10x1000` (for computing subpixel velocity), `x` (beam cross), `t` and `has_interacted` (whether a stimulus was delivered).

time_window_length number of seconds to be used by the motion classifier. This corresponds to the sampling period of the output data.

velocity_correction_coef an empirical coefficient to correct velocity with respect to variable framerate.

masking_duration number of seconds during which any movement is ignored (velocity is set to 0) after a stimulus is delivered (a.k.a. interaction).

velocity_threshold uncorrected velocity above which an animal is classified as ‘moving’ (for the legacy version).

Details

- Validated and corrected subpixel velocity ([max_velocity_detector](#)), the most rigorous
- Uncorrected subpixel velocity ([max_velocity_detector_legacy](#))
- Crossing a virtual beam in the middle of the region of interest ([virtual_beam_cross_detector](#))

[max_velocity_detector](#) is the default movement classification for real-time ethoscope experiments. It is benchmarked against human-generated ground truth.

These functions are *rarely called directly*, but typically used is in the context of [sleep_annotation](#).

Value

an object of the same type as `data` (i.e. [data.table::data.table](#) or [behavr::behavr](#)) with additional columns:

- `moving` Logical, TRUE iff. motion was detected.
- `beam_crosses` The number of beam crosses (when the animal crosses `x = 0.5` – that is the midpoint of the region of interest) within the time window
- `max_velocity` The maximal velocity within the time window. The resulting data is sampled at a period equals to `time_window_length`.

See Also

- [sleep_annotation](#) – which requires a motion detector

motion_qc

Check how well the fixed movement threshold fits the tracking noise

Description

Run on raw ethoscope data before scoring sleep. For each animal and light phase it reports how often a still animal crosses the fixed threshold (`fp_rate_fixed`) and what fraction of genuine rests of `min_time_immobile` would survive that (`rest_survival_fixed`, $(1 - fp)^{\text{windows}}$). `fp_rate_fixed` above about 0.01 means classic sleep estimates are unreliable; score sleep with `sleep_annotation(rule = "k")` instead. A large `untracked_fraction` matters under either rule: with the default `untracked = "immobile"`, windows where the animal was lost count as still.

Usage

```
motion_qc(
  data,
  time_window_length = 10,
  velocity_correction_coef = 0.003,
  velocity_threshold = 1,
  min_time_immobile = 300,
  day_length = 24,
  lights_off = 12
)
```

Arguments

<code>data</code>	<code>behavr::behavr</code> table (or keyed <code>data.table::data.table</code>) of raw ethoscope data, or the data of a single animal.
<code>time_window_length</code>	window size in seconds.
<code>velocity_correction_coef</code>	an empirical coefficient to correct velocity with respect to variable framerate.
<code>velocity_threshold</code>	the fixed threshold being checked.
<code>min_time_immobile</code>	sleep criterion in seconds.
<code>day_length</code>	length of the day in hours.
<code>lights_off</code>	hour of lights off.

Value

a `data.table::data.table` with one row per animal and phase: `phase`, `n_bins`, `untracked_fraction`, `spike_fraction` (frames that are tracking spikes, see [find_spikes](#)), `n_still`, `fp_rate_fixed`, `rest_survival_fixed`.

See Also

[motion_calibration](#), [sleep_rules](#)

sleep_annotation	<i>Score sleep behaviour from immobility</i>
------------------	--

Description

This function first uses a motion classifier to decide whether an animal is moving during a given time window. Then, it defines sleep as contiguous immobility for a minimum duration.

Usage

```
sleep_annotation(
  data,
  time_window_length = 10,
  min_time_immobile = 300,
  motion_detector_FUN = max_velocity_detector,
  untracked = c("immobile", "break"),
  rule = NULL,
  k = NULL,
  pixel = NULL,
  ...
)

sleep_dam_annotation(data, min_time_immobile = 300)
```

Arguments

data	data.table::data.table containing behavioural variable from or one multiple animals. When it has a key, unique values, are assumed to represent unique individuals (e.g. in a behavr::behavr table). Otherwise, it analysis the data as coming from a single animal. data must have a column t representing time.
time_window_length	number of seconds to be used by the motion classifier. This corresponds to the sampling period of the output data.
min_time_immobile	Minimal duration (in s) of a sleep bout. Immobility bouts longer or equal to this value are considered as sleep.
motion_detector_FUN	function used to classify movement
untracked	how windows with no tracked data enter sleep scoring. "immobile" (default) counts them as immobility, so they can extend or create sleep bouts; "break" ends a bout at them, so sleep is only scored where the animal was seen still. moving is unaffected either way. With rule = "k", "immobile" also measures the step after such windows from the last position seen.

rule	"classic", "k", or "k" with its k ("k3", "k2"). There is no built-in default: without it, <code>getOption("sleepr.sleep_rule")</code> is used, then the environment variable <code>SLEEP_R_SLEEP_RULE</code> , and with none of these <code>sleep_annotation</code> stops with an explanation. Use "classic" to reproduce earlier analyses and "k" for new ones. "classic" scores a window as sleep when no frame passed the movement threshold for <code>min_time_immobile</code> (the 5-minute rule). "k" scores it from walking and sustained movement events, ignoring flickers and isolated micro-movements (see sleep_rules); it matches video ground truth at night, and flies it scores asleep respond to air puffs like sleeping flies. It adds the columns <code>walking</code> , <code>sustained</code> and <code>micro_awake</code> , leaves the classic ones as they are and drops frames the tracker inferred. With <code>untracked = "break"</code> it reproduces the reference rule exactly. It needs 10-s windows and does not take <code>velocity_threshold</code> .
k	with <code>rule = "k"</code> , sustained events within a centred 60-s window that make a window awake. NULL takes it from the rule name ("k2") or uses 3.
pixel	with <code>rule = "k"</code> , one pixel in the units of x/y. NULL infers it (pixel_size): 1 for positions in pixels, 1/500 for positions as a fraction of the ROI width (as <code>scopr</code> loads them).
...	extra arguments to be passed to <code>motion_classifier_FUN</code> .

Details

The default `time_window_length` is 300 seconds – it is also known as the "5-minute rule". `sleep_annotation` is typically used for `ethoscope` data, whilst `sleep_dam_annotation` only works on DAM2 data. These functions are *rarely used directly*, but rather passed as an argument to a data loading function, so that analysis can be performed on the go.

Value

a `beavr::beavr` table similar to data with additional variables/annotations (i.e. moving and asleep). The resulting data will only have one data point every `time_window_length` seconds.

References

- The relevant [rethomic tutorial section](#) – on sleep analysis

See Also

- [motion_detectors](#) – options for the `motion_detector_FUN` argument
- [bout_analysis](#) – to further analyse sleep bouts in terms of onset and length

Examples

```
dt_one_animal <- toy_ethoscope_data(seed=2)
##### Ethoscope, corrected velocity classification #####
sleep_dt <- sleep_annotation(dt_one_animal, rule = "classic", masking_duration=0)
print(sleep_dt)
# We could make a sleep `barcode'
## Not run:
```

```

library(ggplot2)
ggplot(sleep_dt, aes(t,y="Animal 1",fill=asleep)) +
  geom_tile() + scale_x_time()

## End(Not run)
##### Ethoscope, virtual beam cross classification #####
sleep_dt2 <- sleep_annotation(dt_one_animal, rule = "classic",
  motion_detector_FUN=virtual_beam_cross_detector)

## Not run:
library(ggplot2)
ggplot(sleep_dt2, aes(t,y="Animal 1",fill=asleep)) +
  geom_tile() + scale_x_time()

## End(Not run)
##### DAM data, de facto beam cross classification #####
dt_one_animal <- toy_dam_data(seed=7)
sleep_dt <- sleep_dam_annotation(dt_one_animal)
## Not run:
library(ggplot2)
ggplot(sleep_dt, aes(t,y="Animal 1",fill=asleep)) +
  geom_tile() + scale_x_time()

## End(Not run)

```

sleep_rules

Sleep from walking and sustained movement (the k-rule)

Description

The classic rule calls a 10-s window "moving" when any frame's velocity exceeds a threshold, so a single noisy frame breaks a sleep bout. The k-rule asks two questions instead: did the animal walk (its median position moved more than 10 px from the previous window), and did it move in place (at least k sustained movement events started in a centred 60-s window)?

Usage

```

classify_events(
  x,
  y,
  moving,
  tolerance = RETURN_TOLERANCE_PIXELS,
  max_flicker_frames = MAX_FLICKER_FRAMES
)

k_rule_bins(
  t,
  x,
  y,
  xy_dist_log10x1000,

```

```

k = 3,
pixel = 1,
velocity_correction_coef = 0.003,
min_sleep_bins = 30,
untracked = c("immobile", "break")
)

```

Arguments

<code>x, y</code>	positions of one animal, one per frame, in recording order.
<code>moving</code>	logical, TRUE where the frame passes the movement test.
<code>tolerance</code>	how close a flicker must return, in the units of x/y.
<code>max_flicker_frames</code>	longest run of moving frames that can be a flicker.
<code>t</code>	frame times in seconds, in recording order.
<code>xy_dist_log10x1000</code>	the tracker's per-frame displacement.
<code>k</code>	sustained events in the centred 60-s window that make a window awake.
<code>pixel</code>	one pixel in the units of x/y.
<code>velocity_correction_coef</code>	as in max_velocity_detector ; a frame moves when its corrected velocity exceeds 1.
<code>min_sleep_bins</code>	shortest sleep bout, in 10-s windows.
<code>untracked</code>	"immobile" counts windows without frames as still and measures the step after them from the last position seen; "break" never scores them as sleep and counts the window after them as walking (the reference rule).

Details

A movement event is a run of consecutive frames above the classic velocity test. Events in which the position never leaves the pixel it started from ("subpixel"), and those that jump out and land back within a pixel in one or two frames ("flicker"), are tracking noise and are ignored; only the remaining "sustained" events count. Sleep is then 5 minutes or more of windows with neither walking nor such movement.

Windows without frames are handled as in the classic rule. By default (`untracked = "immobile"`) they count as still, and the first window with frames after them is walking only if the animal is found more than 10 px from where it was last seen. Background-subtraction tracking loses still flies, so this is what keeps their sleep. `untracked = "break"` never scores such windows as sleep and reproduces the reference rule exactly.

Evidence: against pixel-motion truth from video, night-time error per animal was 0.03-0.07 with either tracker; and in 136 air-puff runs (1,319 flies), flies the rule scores asleep, by day too, respond to puffs like sleeping flies (real minus sham response +1.2 percentage points asleep by every rule, +3.2 asleep only at $k = 3$, +5.2 awake).

Choose it with `sleep_annotation(rule = "k")`, or declare it once with `options(sleepr.sleep_rule = "k")`; `sleep_annotation` has no default rule. The algorithm and its defaults are identical to `ethoscopy`'s `sleep_rules` module, so both packages score the same data alike.

Functions

- `classify_events()`: split runs of moving frames into events. The frame before a run is its anchor, so a run on the first frame is not an event. An event is subpixel if every frame is exactly at the anchor's position, else a flicker if it has at most `max_flicker_frames` frames and the next frame is back within tolerance of the anchor, else sustained. Returns a list with `start` (row of the first frame) and `class` (0 subpixel, 1 flicker, 2 sustained).
- `k_rule_bins()`: score one animal's frames window by window. Returns a [data.table::data.table](#) with one row per 10-s window from the first to the last window with frames: `t` (window start, s), `has_data`, `walking`, `sustained` (events starting in the window), `micro_awake` and `asleep`.

See Also[sleep_annotation](#)

Index

behavr::behavr, [2–4](#), [7–10](#)
bout_analysis, [2](#), [10](#)

classify_events(sleep_rules), [11](#)
curate_dead_animals, [3](#)

data.table::data.table, [2–4](#), [7–9](#), [13](#)
default_still_shift, [5](#)
default_still_shift
 (motion_calibration), [4](#)

find_spikes, [8](#)
find_spikes(motion_calibration), [4](#)
find_still_bins(motion_calibration), [4](#)

k_rule_bins(sleep_rules), [11](#)

light_phase(motion_calibration), [4](#)

max_velocity_detector, [4](#), [6](#), [7](#), [12](#)
max_velocity_detector
 (motion_detectors), [6](#)
max_velocity_detector_legacy, [7](#)
max_velocity_detector_legacy
 (motion_detectors), [6](#)
motion_calibration, [4](#), [9](#)
motion_detectors, [6](#), [10](#)
motion_qc, [6](#), [8](#)

pixel_size, [10](#)
pixel_size(motion_calibration), [4](#)

rle, [2](#)

sleep_annotation, [2](#), [4](#), [7](#), [9](#), [13](#)
sleep_dam_annotation
 (sleep_annotation), [9](#)
sleep_rules, [6](#), [9](#), [10](#), [11](#)

virtual_beam_cross_detector, [7](#)
virtual_beam_cross_detector
 (motion_detectors), [6](#)