

Package ‘trackclean’

July 1, 2026

Title Tools for Cleaning High-Frequency Real-Time Location Tracking Data

Version 0.1.0

Description Provides data cleaning and preprocessing tools for high-frequency positional data from real-time location tracking systems (UWB, RFID, and similar technologies), with functions for ID mapping, time period marking, data standardization, and two-phase conditional gap interpolation. See Bilevicius (2026) <[doi:10.5281/zenodo.20783488](https://doi.org/10.5281/zenodo.20783488)>.

License CC BY 4.0

URL <https://github.com/tomasbil/trackclean>

BugReports <https://github.com/tomasbil/trackclean/issues>

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Imports dplyr, lubridate, magrittr, readr, rlang, tibble

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Tomas Bilevicius [aut, cre]

Maintainer Tomas Bilevicius <Tomas.Bilevicius@vub.be>

Repository CRAN

Date/Publication 2026-07-01 09:40:02 UTC

Contents

clean_playground_data	2
fix_tag_replacement	4
interpolate_gaps	5
map_ids	6
mark_time_periods	7
standardize_to_seconds	8

clean_playground_data *Clean tracking data (complete pipeline)*

Description

Master function that runs the complete data cleaning pipeline:

1. Map raw IDs to participant IDs
2. Mark analysis and bell time periods
3. Standardize to fixed time intervals
4. Interpolate gaps (two-phase)
5. Optionally export to CSV

Usage

```
clean_playground_data(  
  data,  
  id_mapping,  
  exclude_ids = NULL,  
  analyze_start,  
  analyze_end,  
  bell_start = NULL,  
  bell_end = NULL,  
  unit = "second",  
  time_step = 1,  
  max_gap_small = 10,  
  max_gap_large = NULL,  
  max_position_change = 0.3,  
  output_file = NULL,  
  verbose = TRUE,  
  time_col = "At",  
  x_col = "X",  
  y_col = "Y",  
  raw_id_col = "ID",  
  id_col = "id_code",  
  analyze_col = "Analyze",  
  bell_col = "Bell"  
)
```

Arguments

data	Raw tracking data frame
id_mapping	Path to ID mapping CSV file or mapping data frame
exclude_ids	Vector of raw IDs to exclude from analysis

analyze_start	Start time for analysis period (character or POSIXct)
analyze_end	End time for analysis period (character or POSIXct)
bell_start	Start time for bell period (optional)
bell_end	End time for bell period (optional)
unit	Time interval for standardization, passed to <code>standardize_to_seconds()</code> (default: "second"). Use "2 seconds", "5 seconds", etc. for coarser intervals.
time_step	Expected time step in seconds between consecutive observations after standardization (default: 1). Must match the numeric value of <code>unit</code> , e.g. set <code>time_step = 2</code> when <code>unit = "2 seconds"</code> .
max_gap_small	Maximum gap for phase 1 interpolation in seconds (default: 10)
max_gap_large	Maximum gap for phase 2 interpolation in seconds (default: NULL)
max_position_change	Maximum position change for phase 2 in meters (default: 0.3)
output_file	Path to save cleaned data as CSV (optional)
verbose	Print progress messages (default: TRUE)
time_col	Name of the timestamp column (default: "At")
x_col	Name of the x-coordinate column (default: "X")
y_col	Name of the y-coordinate column (default: "Y")
raw_id_col	Name of the raw device ID column in the input data (default: "ID")
id_col	Name of the output column for standardized participant IDs (default: "id_code")
analyze_col	Name of the analysis period flag column (default: "Analyze")
bell_col	Name of the bell period flag column (default: "Bell")

Value

Cleaned data frame

Examples

```
# Complete pipeline using bundled example data
library(readr)
raw_data <- read_csv(system.file("extdata", "raw_tracking_data.csv",
                                package = "trackclean"))

cleaned_data <- clean_playground_data(
  data = raw_data,
  id_mapping = system.file("extdata", "id_mapping.csv", package = "trackclean"),
  analyze_start = "2025-03-18 11:47:00",
  analyze_end   = "2025-03-18 11:57:00",
  bell_start    = "2025-03-18 11:53:00",
  bell_end      = "2025-03-18 11:58:00"
)
```

fix_tag_replacement *Fix tag replacements in raw data*

Description

Handles cases where a participant's tracking tag was replaced during data collection. Renames observations from the new tag to the original ID and removes invalid observations.

Usage

```
fix_tag_replacement(
  data,
  original_id,
  replacement_id,
  replacement_time,
  time_col = "At",
  id_col = "ID"
)
```

Arguments

data	A data frame with raw tracking data
original_id	The participant's original tag ID
replacement_id	The new tag ID that replaced the original
replacement_time	Time when tag was replaced (POSIXct or character, e.g. "2025-03-18 11:20:00")
time_col	Name of the timestamp column (default: "At")
id_col	Name of the ID column (default: "ID")

Value

Data frame with corrected IDs:

- Observations from replacement_id >= replacement_time are renamed to original_id
- Observations from original_id >= replacement_time are removed
- Observations from replacement_id < replacement_time are removed

Examples

```
raw_data <- data.frame(
  ID = c(159L, 159L, 106L, 106L),
  At = as.POSIXct(c("2025-03-18 11:00:00", "2025-03-18 11:30:00",
    "2025-03-18 11:00:00", "2025-03-18 11:30:00")),
  X = c(1.0, 2.0, 3.0, 4.0),
  Y = c(1.0, 2.0, 3.0, 4.0)
)
```

```

raw_data <- fix_tag_replacement(
  data = raw_data,
  original_id = 159,
  replacement_id = 106,
  replacement_time = "2025-03-18 11:20:00"
)

```

interpolate_gaps *Interpolate gaps in location tracking data (two-phase approach)*

Description

Phase 1: Interpolates small gaps (gap <= max_gap_small seconds) Phase 2: Interpolates larger gaps if position change is <= max_position_change meters

Usage

```

interpolate_gaps(
  data,
  time_col = "At",
  x_col = "X",
  y_col = "Y",
  id_col = "id_code",
  analyze_col = "Analyze",
  time_step = 1,
  max_gap_small = 10,
  max_gap_large = NULL,
  max_position_change = 0.3,
  verbose = TRUE
)

```

Arguments

data	A data frame with location tracking data
time_col	Name of the timestamp column (default: "At")
x_col	Name of x-coordinate column (default: "X")
y_col	Name of y-coordinate column (default: "Y")
id_col	Name of ID column (default: "id_code")
analyze_col	Name of column indicating rows to analyze (default: "Analyze")
time_step	Expected time step in seconds between consecutive observations after standardization (default: 1). Set this to match the unit used in <code>standardize_to_seconds()</code> , e.g. <code>time_step = 2</code> if you standardized to 2-second intervals.
max_gap_small	Maximum gap size for phase 1 in seconds (default: 10)
max_gap_large	Maximum gap size for phase 2 in seconds (default: NULL for no limit)
max_position_change	Maximum position change in meters for phase 2 (default: 0.3)
verbose	Print progress messages (default: TRUE)

Details

Uses linear interpolation: $X_t = X_{start} + (k/gap) * (X_{end} - X_{start})$

Value

Data frame with interpolated coordinates and flags:

- imputed: 1 if row was added in phase 1 (small gaps)
- imputed_large: 1 if row was added in phase 2 (large gaps)
- n_entries: 0 for imputed rows
- standardized: 0 for imputed rows

Examples

```
standardized_data <- data.frame(
  id_code      = c(1L, 1L),
  At           = as.POSIXct(c("2025-03-18 11:00:00", "2025-03-18 11:00:05")),
  X            = c(1.0, 1.0),
  Y            = c(2.0, 2.0),
  Analyze      = c(1L, 1L),
  n_entries    = c(1L, 1L),
  standardized = c(0L, 0L)
)
data_clean <- interpolate_gaps(standardized_data)
```

map_ids

Map raw tracking IDs to standardized child IDs

Description

Map raw tracking IDs to standardized child IDs

Usage

```
map_ids(
  data,
  mapping,
  exclude_ids = NULL,
  raw_id_col = "ID",
  id_col = "id_code",
  analyze_col = "Analyze"
)
```

Arguments

data	A data frame with raw tracking data
mapping	Either: <ul style="list-style-type: none"> • Path to CSV file with columns 'raw_id' and 'child_id' • Data frame with columns 'raw_id' and 'child_id' • Named vector (raw_id = child_id)
exclude_ids	Vector of raw IDs to exclude from analysis (sets Analyze = 0)
raw_id_col	Name of the raw ID column in data (default: "ID")
id_col	Name of the output column for standardized IDs (default: "id_code")
analyze_col	Name of the Analyze column in data (default: "Analyze")

Value

Data frame with added id_code column and updated Analyze column

Examples

```
raw_data <- data.frame(
  ID = c(1L, 2L, 3L),
  At = as.POSIXct("2025-03-18 11:45:00"),
  X = c(1.0, 2.0, 3.0),
  Y = c(1.0, 2.0, 3.0)
)
id_map <- data.frame(raw_id = c(1L, 2L, 3L), child_id = c(5001L, 5002L, 5003L))
data_mapped <- map_ids(raw_data, id_map)
```

mark_time_periods *Mark time periods for analysis and bell time in raw data*

Description

Creates binary columns indicating whether each timestamp falls within specified time periods (e.g., recess period, bell ringing period)

Usage

```
mark_time_periods(
  data,
  time_col = "At",
  analyze_start,
  analyze_end,
  bell_start = NULL,
  bell_end = NULL,
  analyze_col = "Analyze",
  bell_col = "Bell"
)
```

Arguments

<code>data</code>	A data frame with timestamp data
<code>time_col</code>	Name of the timestamp column (default: "At")
<code>analyze_start</code>	Start time for analysis period (POSIXct or character, e.g. "2025-03-18 11:50:00")
<code>analyze_end</code>	End time for analysis period (POSIXct or character)
<code>bell_start</code>	Start time for bell period (POSIXct or character, optional)
<code>bell_end</code>	End time for bell period (POSIXct or character, optional)
<code>analyze_col</code>	Name of column to create for analysis period (default: "Analyze")
<code>bell_col</code>	Name of column to create for bell period (default: "Bell")

Value

Data frame with added binary columns (1 = within period, 0 = outside period)

Examples

```
raw_data <- data.frame(
  At = as.POSIXct(c("2025-03-18 11:00:00", "2025-03-18 12:00:00",
    "2025-03-18 13:00:00")),
  ID = 1:3
)
raw_data <- mark_time_periods(
  raw_data,
  analyze_start = "2025-03-18 10:30:00",
  analyze_end   = "2025-03-18 12:30:00",
  bell_start    = "2025-03-18 11:30:00",
  bell_end      = "2025-03-18 12:30:00"
)
```

standardize_to_seconds

Standardize location data to fixed time intervals

Description

Rounds timestamps to the nearest interval boundary and averages X/Y coordinates if multiple signals fall within the same interval.

Usage

```
standardize_to_seconds(
  data,
  time_col = "At",
  x_col = "X",
  y_col = "Y",
  id_col = "id_code",
```

```
    unit = "second",  
    verbose = TRUE  
  )
```

Arguments

<code>data</code>	A data frame with location tracking data
<code>time_col</code>	Name of the timestamp column (default: "At")
<code>x_col</code>	Name of x-coordinate column (default: "X")
<code>y_col</code>	Name of y-coordinate column (default: "Y")
<code>id_col</code>	Name of ID column (default: "id_code")
<code>unit</code>	Time interval to standardize to, passed to <code>lubridate::floor_date()</code> (default: "second"). Use "2 seconds", "5 seconds", etc. for coarser intervals.
<code>verbose</code>	Print summary statistics (default: TRUE)

Value

Data frame with one row per (id, interval) with:

- X, Y: Averaged coordinates
- n_entries: Number of original signals in that interval
- standardized: 1 if multiple signals were aggregated, 0 if single

Examples

```
raw_data <- data.frame(  
  id_code = c(1L, 1L, 1L),  
  At = as.POSIXct(c("2025-03-18 11:00:00.0", "2025-03-18 11:00:00.5",  
                   "2025-03-18 11:00:01.0")),  
  X = c(1.0, 1.2, 2.0),  
  Y = c(2.0, 2.1, 3.0)  
)  
standardized_data <- standardize_to_seconds(raw_data)
```

Index

`clean_playground_data`, 2
`fix_tag_replacement`, 4
`interpolate_gaps`, 5
`map_ids`, 6
`mark_time_periods`, 7
`standardize_to_seconds`, 8