Package: trackclean (via r-universe)

July 3, 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.

License CC BY 4.0

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

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

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

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

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/pak/sysreqs libx11-dev

Repository https://tomasbil.r-universe.dev

Date/Publication 2026-06-21 10:03:57 UTC

RemoteUrl https://github.com/tomasbil/trackclean

RemoteRef HEAD

RemoteSha 3d5a72468ec28dbd092d6a4781b5adc00c01eebe

Contents

`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 standardize_to_seconds() (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 unit, e.g. set time_step = 2 when unit = "2 seconds".
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

```
## Not run:
# 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"
)

# Custom column names for a dataset with different structure
cleaned_data <- clean_playground_data(
  data = raw_data,
  id_mapping = "id_mapping.csv",
  analyze_start = "2025-03-18 11:47:00",
  analyze_end   = "2025-03-18 11:57:00",
  time_col    = "timestamp",
  x_col       = "pos_x",
  y_col       = "pos_y",
  raw_id_col  = "tag_id",
  id_col      = "participant_id",
  analyze_col = "in_window"
)

## End(Not run)
```

`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

```
## Not run:
# Tag 106 replaced tag 159 at 11:20
raw_data <- fix_tag_replacement(
  data = raw_data,
  original_id = 159,
  replacement_id = 106,
  replacement_time = "2025-03-18 11:20:00"
)

## End(Not run)
```

`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 `standardize_to_seconds()`, e.g. `time_step = 2` 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

```
## Not run:
# Default: phase 1 <=10sec, phase 2 unlimited with <=30cm movement
data_clean <- interpolate_gaps(standardized_data)

# Custom thresholds
data_clean <- interpolate_gaps(standardized_data,
                               max_gap_small = 5,
                               max_gap_large = 30,
                               max_position_change = 0.5)

# After 2-second standardization
data_clean <- interpolate_gaps(standardized_data, time_step = 2)

## End(Not run)
```

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: • 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

```
## Not run:
# Using a CSV file (recommended)
data_mapped <- map_ids(raw_data, "id_mapping.csv", exclude_ids = c(67, 72, 80))

# Using a data frame
id_map <- data.frame(raw_id = c(2, 3, 6), child_id = c(5129, 5113, 5222))
data_mapped <- map_ids(raw_data, id_map, exclude_ids = c(67, 72, 80))

## End(Not run)
```

`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

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

Value

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

Examples

```
## Not run:
# Mark analysis period only
raw_data <- mark_time_periods(
  raw_data,
  analyze_start = "2025-03-18 11:50:00",
  analyze_end = "2025-03-18 13:11:00"
)
```

```
# Mark both analysis and bell periods
raw_data <- mark_time_periods(
  raw_data,
  analyze_start = "2025-03-18 11:50:00",
  analyze_end = "2025-03-18 13:11:00",
  bell_start = "2025-03-18 12:30:00",
  bell_end = "2025-03-18 14:00:00"
)

## End(Not run)
```

`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

`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")
`unit`	Time interval to standardize to, passed to `lubridate::floor_date()` (default: `"second"`). Use `"2 seconds"`, `"5 seconds"`, etc. for coarser intervals.
`verbose`	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

```
## Not run:
# Default: 1-second intervals
standardized_data <- standardize_to_seconds(raw_data)

# 2-second intervals
standardized_data <- standardize_to_seconds(raw_data, unit = "2 seconds")

## End(Not run)
```

Index