Package: kaltoa (via r-universe)

May 14, 2026

Title Kalman filtered time-of-arrival positioning

Version 0.3.0.0

Description State-space models for time-of-arrival positioning in underwater acoustic telemetry arrays.

License GPL-3

Imports geometry, mvtnorm, gamlss.dist, extraDistr, lattice, latticeExtra, generics, TMB, RcppEigen, Rcpp, invgamma, abind, MASS

LinkingTo TMB, RcppEigen, Rcpp

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, tictoc, numDeriv, stringr, magrittr, archive, testthat (>= 3.0.0)

VignetteBuilder knitr

Depends R (>= 4.1.0)

Config/testthat/edition 3

Collate 'DetectionTime.R' 'Detections.R' 'DetectionsList.R' 'DetectionsList_attributes.R' 'DetectionsList_diagnostics_connectivity.R' 'DetectionsList_diagnostics_latency.R' 'DetectionsList_generics.R' 'Detections_attributes.R' 'Detections_functions.R' 'Detections_generics.R' 'EmptyDetectionsList.R' 'KaltoaClockDrift.R' 'KaltoaClockDrift_internal.R' 'KaltoaClockOffsets_1.R' 'KaltoaClockOffsets_2.R' 'KaltoaClockSync_1_classes.R' 'KaltoaClockSync_2_methods.R' 'KaltoaClockSync_slots.R' 'class_definitions.R' 'KaltoaProcessModel.R' 'KaltoaKalmanPositions.R' 'KaltoaKalmanPositions_generics.R' 'KaltoaKalmanPositions_methods.R' 'KaltoaKalmanPositions_misc.R' 'KaltoaObservationModel.R'

'KaltoaPMCFit.R' 'KaltoaPMCPositions.R'
'KaltoaPMCPositions_methods.R' 'KaltoaPointPositioning.R'
'KaltoaPointPositioning_PMC.R'
'KaltoaPointPositioning_internal.R' 'KaltoaPointPositions.R'
'KaltoaPointPositions_generics.R' 'KaltoaReceiverPositions.R'
'KaltoaTrackPositioning.R' 'KaltoaTrackPositioning_points.R'
'KaltoaTrackPositioning_toa.R' 'Latency.R' 'Latency_methods.R'
'RcppExports.R' 'Rcpp_extra_docs.R' 'S4.R' 'TOAMatrix.R'
'TOAMatrix_csv.R' 'TOAMatrix_generics.R' 'TOAMatrix_plot.R'
'TOA_internal.R' 'TdoaPositioning.R' 'classes_generic.R'
'color_bar.R' 'ctcrw.R' 'data_frame.R' 'distributions.R'
'find_sync_order.R' 'generic.R' 'kalman_filter.R'
'metaprogramming.R' 'misc.R' 'overview.R' 'package.R'
'permute_clock_drift.R' 'permute_clock_drift_internal.R'
'permute_clock_drift_pair_likelihood.R'
'permute_clock_drift_pair_xcorr.R' 'print_parameters.R'
'sync_pair.R'

Config/pak/sysreqs libjpeg-dev libpng-dev

Repository https://ocean-tracking-network.r-universe.dev

Date/Publication 2026-05-14 16:28:18 UTC

RemoteUrl https://gitlab.com/RTbecard/kaltoa

RemoteRef HEAD

RemoteSha 86e1f26e2d0ded2a625cc6562d0e0f5c39165a19

Contents

`ctcrw` *Continuous-time correlated random walk (CTCRW) functions*

Description

A set of functions for calculating the state (position and velocity) covariance for a Ornstein-Uhlenbeck velocity process.

Usage

```
# Position variance
ctcrw_var_pos(alpha, beta, delta)

# Velocity variance
ctcrw_var_vel(alpha, beta, delta)

# Position/velocity covariance
ctcrw_cov(alpha, beta, delta)
```

Arguments

alpha	Variance parameter of CTCRW process.
beta	Correlation parameter of CTCRW process.
delta	Time interval since previous state (seconds).

Details

These functions are used internally for generating the CTCRW process covariance matrices in Kalman filters.

---

ddetect *Detection error distribution*

Description

A mixture of a Gaussian, dnorm, and generalized normal distribution, dgnorm, to capture direct and reflected transmission residuals, respectively. See Campbell et al. (2025) for details.

Usage

```
ddetect(x, sigma, phi, beta = 32, log = F)
```

Arguments

x	A vector of detection time residuals.
sigma	The standard deviation of the direct residuals (seconds).
phi	The maximum value for detection error (seconds). Setting phi to high values allows the capture of large positive outliers resulting from reflected transmissions.
beta	Shape parameter for generalized normal distribution, dgnorm.
log	When TRUE, return log density.

References

- Campbell, J. A., Shry, S. J., Lundberg, P., Calles, O., Hölker, F. (2025) **A population Monte Carlo model for underwater acoustic telemetry positioning in reflective environments**. *Methods Ecology and Evolution*. 16(4):775-785.

detection_error *Get sync-tag detection error*

Description

Get sync-tag detection error

Usage

```
detection_error.SyncTagLatency(x, transmission_speed = 1500)

detection_error.Latency(x, transmission_speed = 1500)

# Generic function
detection_error(x, ...)
```

Arguments

x	A Latency or SyncTagLatency object.
transmission_speed	sync-tag transmission speed (m/s).

Detections *Detections object*

Description

Holds the detections data from a single telemetry receiver along with useful metadata.

Usage

```
## Crete new Detections object
det <- Detections(name, detect_times, detect_id, type = NA, x, y,
    send_id, depth = NA, sync_emissions, aux)

## Get attribute: emit_id
emit_id(det)

## Set attribute: emit_id
emit_ids(det) <- value

## Get attribute: depth
depths(det)

## Set attribute: depth
```

```
depths(det) <- value

## Get attribute: position
positions(det)

## Set attribute: position
positions(det) <- value

## Get display timezone
tzone(object = det)

## Set display timezone
tzone(object = det) <- value

## Show summary information
summary(det)

## Combine Detections objects from same receiver
rbind(det_1, det_2)

## Subset rows
subset(det, subset)

## Add offset(s) to detection times
det + x

## Subtract offset(s) from detection times
det - x

## Convert Detections to data.frame
df = as.data.frame(x = det, strftime = TRUE)

## Convert data.frame to Detections object
det = as.Detections(x = df, name, coord, depth, emit_id)
```

Arguments

`detect_times`	A DetectionTime object.
`detect_id`	A vector of tag ids for the corresponding detection times and sync tag emissions.
`type`	Optional parameter for describing the type of tag detected. Useful for distinguishing between Vemco PPM and HR tags which can share the same `detect_id`.
`coords`	A vector holding the x & y coordinates of the receiver in meters.
`emit_id`	The `detect_id` of this receiver's sync-tags as they appear on other receivers in the array.
`depth`	Optional depth in meters of this receiver.
`sync_emissions`	A logical vector indicating which detections are sync-tag emission events from that receiver.

aux	A data.frame holding auxiliary data to be attached to the detections table. For example, diagnostic data such as signal and background noise levels.
subset	See base::subset.
strftime	When TRUE, detection times will be converted into a string representation.

Details

Detections objects are the core class used by kaltoa preprocessing functions. They contain all the detections for a given receiver along with important metadata for preprocessing and positioning. Sync-tag emissions marked by the sync_tags column are used for clock correction functions.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Detections: Detections-generics, remove_double_detections()

Examples

```
# ------ Create example data
df_example <- as.data.frame(exdata_positioning[[1]])
head(df_example)

# ------ Create detections object
det <- Detections(
  detect_times = iso2DetectionTime(df_example$POSIX_UTC, tz = "UTC"),# Detection times
  detect_id = df_example$detect_id,# Recorded tag ID for detections
  type = df_example$type,# Differentiate HR and PPM detections
  coords = c(481472, 6243509),# Receiver coordinates in UTM meters
  emit_id = "62041",# Detection ID of sync-tag attached to this receiver
  name = "1",# Name of receiver
  depth = 3,# Depth in meters
  sync_emissions = df_example$type %in% c("PPM_SELF", "HR_SELF"))# Sync-tag emissions

# Set display timezone
tzone(det) <- "CET"

# Preview data
head(det)

# ------ Remove double detections
det_filt <- remove_double_detections(det, dd_thresh = 0.1)

# ------ Show summary info
summary(det_filt)

# ------ Subset detections
start <- as.POSIXct("2021-05-08 20:00:00")
end <- as.POSIXct("2021-05-08 21:00:00")
types <- c("PPM")
det_sub <- subset(det_filt, second >  start & second < end & type %in% types)
```

```
# ------ Add an offset of 0.5 seconds to all detection times
det_offset <- det_filt + 0.5

# ------ Export to csv
fname <- tempfile(fileext = ".csv.gz")
fh <- gzfile(fname, open = "w")
write.csv(x = as.data.frame(det), file = fh, row.names = F)
close(fh)

# ------ Import from csv
fh <- gzfile(fname, open = "r")
df_det <- read.csv(file = fh)
det_import <- as.Detections(
  df_det, name = "1", coords = c(481472, 6243509),
  depth = NaN, emit_id = "62041")
close(fh)
```

`Detections-generics` *Detections generic functions*

Description

Generic methods and slot access functions for Detections objects.

Usage

```
## Indexing
det[i]
det[i, j]

## Replacing
det[i, j] <- value

head(det, n = 6L)

tail(det, n = 6L)
```

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Detections: `Detections()`, `remove_double_detections()`

DetectionsList *Create a DetectionsList object*

Description

A DetectionsList holds all the detections data for a receiver array. This object is used to apply clock drift/offset and receiver position corrections. Finally, TOAMatrixs can be extracted from DetectionsList objects for running positioning models.

Usage

```
## Create DetectionsList from Detections objects
detlst <- DetectionsList(...)

## Get table of attributes for DetectionsList object
attribute_table(detlst)

## Get attributes: emit_ids
emit_ids(detlst)

## Set attributes: emit_ids
emit_ids(detlst) <- x

## Get attributes: depths
depths(detlst)

## Set attributes: depths
depths(detlst) <- x

## Get attibutes: positions
positions(detlst)

## Set attribtes: positions
positions(detlst) <- x

## Convert a list of Detections objects to a DetectionsList
detlst <- as.DetectionsList(x)

## Plot locations of receivers
plot(detlst, ...)

## Show a summary detections info
summary(detlst)
```

See Also

Other DetectionsList: detlst_apply(), subset.DetectionsList()

Examples

```
# Extract detections objects
det_1 <- exdata_positioning[[1]]
det_2 <- exdata_positioning[[2]]
det_3 <- exdata_positioning[[3]]
det_4 <- exdata_positioning[[4]]
det_5 <- exdata_positioning[[5]]
det_6 <- exdata_positioning[[6]]

# Remake DetectionsList
detlst <- DetectionsList(det_1, det_2, det_3, det_4, det_5, det_6)
detlst

# Convert a 'list of Detections' into a DetectionsList object
ldet <- list(det_1, det_2, det_3, det_4, det_5, det_6)
detlst <- as.DetectionsList(ldet)
detlst

# Remove double detections from array
detlst <- detlst_apply(detlst, remove_double_detections, dd_thresh = 0.05)

# ------ Viewing data
# Show summary of array detections data
summary(detlst)
# Plot map of array
plot(detlst)
# Return a matrix of array positions
positions(detlst)

# ------ Diagnostics
# Count sync-tag detections across the array
sync_conn <- SyncTagConnectivity(detlst)
# Plot a conenctivity map
plot(sync_conn, type = "map")
# Plot connectivity matrices
plot(sync_conn, type = "matrix")

# ------ Export DetectionsList as tar archived csv files
require(archive)
# Create temp directory
tmp_dir <- tempfile()
dir.create(tmp_dir)
# Write array attributes to csv
write.csv(
  file = file.path(tmp_dir, "attributes.csv"),
  x = attribute_table(exdata_positioning), row.names = FALSE)
# Write detections tables to csv files
for(det in exdata_positioning){
  fname <- sprintf("Detections-%s.csv", attr(det, "name"))
  write.csv(
    file = file.path(tmp_dir, fname),
    x = det, row.names = F)
```

```
}
# Archive directory
f_archv <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archv, dir = tmp_dir, full.names = FALSE)
# Delete temp folder
unlink(tmp_dir, recursive = T)
```

DetectionTime *Crete a DetectonTime object*

Description

Each detection time is stored as a pair of values. An integer number giving the number of seconds since 1970-01-01 (POSIX formatted times), and a double, giving the additional number of milliseconds. Typically, DetectionTime objects are created with the function iso2DetectionTime.

Usage

```
## Create DetectionsTime object
det_time = DetectionTime(s, ms)

## Convert ISO timestamp into DetectionTime
iso_time = iso2DetectionTime(x = det_time, tz = Sys.timezone())

## Convert DetectionTime into ISO timestamp
det_time = DetectionTime2iso(x = iso_time, tz = Sys.timezone())

## Add offset (second)
det_time + y

## Subtract offset (seconds)
det_time - y

## Convert POSIX seconds into DetectionTime
as.DetectionTime(x)
```

Arguments

s	A vector of POSIX seconds.
ms	A vector of milliseconds.
iso_time	A vector of strings with the format "2020-01-01 00:25:00.073876".
det_time	A DetectionTime object.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
# Make example data.frame
example <- data.frame(
  timestamp = sprintf("%s.%03i",
    strftime(format = "%FT%T", exdata_positioning[[1]]$second),
    as.integer(exdata_positioning[[1]]$millisecond)),
  detect_id =  exdata_positioning[[1]]$detect_id,
  type = exdata_positioning[[1]]$type)
head(example)

# Load from ISO date/time stamps
det_time <- iso2DetectionTime(example$timestamp, tz = "CET")
head(det_time)

# Convert back to ISO format
iso_stamp <- DetectionTime2iso(det_time)
head(iso_stamp)

# Example formats for time stamps, note the timezones
iso2DetectionTime("2021-05-08 18:05:23.948", tz = "CET")
iso2DetectionTime("2021-05-08T18:05:23.948", tz = "UTC")

# Manually create DetectionsTime from POSIXct values
x <- DetectionTime(s = as.POSIXct("2021-05-08 18:05:23", tz = "CET"), ms = 350)
print(x)

# Convert POSIXct times into DetectionTimes
x <- as.POSIXct(
  c("2021-05-08 18:05:23.333", "2021-05-08 18:08:37.214"), tz = "CET")
y <- as.DetectionTime(x)
print(y)

# Find difference (in seconds) between detection times
head(det_time - det_time[5,])

# Add offset to detection times
head(det_time + 1.5)
```

`detlst_apply` *lapply wrapper for DetectionsList objects*

Description

This will apply the lapply function, but return a DetectionsList object.

Usage

```
detlst_apply(X, FUN, ...)
```

Arguments

X	A DetectionsList object.
FUN	Function to apply to all DetectionsList objects.
...	Arguments passed to FUN.

See Also

Other DetectionsList: DetectionsList, subset.DetectionsList()

Examples

```
# Extract detections objects
det_1 <- exdata_positioning[[1]]
det_2 <- exdata_positioning[[2]]
det_3 <- exdata_positioning[[3]]
det_4 <- exdata_positioning[[4]]
det_5 <- exdata_positioning[[5]]
det_6 <- exdata_positioning[[6]]

# Remake DetectionsList
detlst <- DetectionsList(det_1, det_2, det_3, det_4, det_5, det_6)
detlst

# Convert a 'list of Detections' into a DetectionsList object
ldet <- list(det_1, det_2, det_3, det_4, det_5, det_6)
detlst <- as.DetectionsList(ldet)
detlst

# Remove double detections from array
detlst <- detlst_apply(detlst, remove_double_detections, dd_thresh = 0.05)

# ------ Viewing data
# Show summary of array detections data
summary(detlst)
# Plot map of array
plot(detlst)
# Return a matrix of array positions
positions(detlst)

# ------ Diagnostics
# Count sync-tag detections across the array
sync_conn <- SyncTagConnectivity(detlst)
# Plot a conenctivity map
plot(sync_conn, type = "map")
# Plot connectivity matrices
plot(sync_conn, type = "matrix")

# ------ Export DetectionsList as tar archived csv files
require(archive)
# Create temp directory
tmp_dir <- tempfile()
```

```
dir.create(tmp_dir)
# Write array attributes to csv
write.csv(
  file = file.path(tmp_dir, "attributes.csv"),
  x = attribute_table(exdata_positioning), row.names = FALSE)
# Write detections tables to csv files
for(det in exdata_positioning){
  fname <- sprintf("Detections-%s.csv", attr(det, "name"))
  write.csv(
    file = file.path(tmp_dir, fname),
    x = det, row.names = F)
}
# Archive directory
f_archv <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archv, dir = tmp_dir, full.names = FALSE)
# Delete temp folder
unlink(tmp_dir, recursive = T)
```

`dgnorm` *Generalized normal distribution*

Description

An implementation of Nadarajah's (2005) generalized normal distribution.

Usage

```
dgnorm(x, mu, alpha, beta, log = F)

qgnorm(p, mu, alpha, beta)
```

Arguments

`x`	Vector of values to calculate density for.
`mu`	Location parameter.
`alpha`	Scale parameter
`beta`	Shape parameter.
`log`	When `TRUE`, return the log density.
`p`	Vector of probabilities to calculate quantiles for.

References

- Nadarajah, S. 2005. **A generalized normal distribution**. *Journal of Applied Statistics*. 32 (7): 685–694.

`fill_sync_order` *Create a sync_order data.frame*

Description

Convenience functions for creating a sync_order data.frame.

Usage

```
## Automatically create sync_order data.frame for array
find_sync_order(detlst, controllers)

## Return a sync_order data.frame with a single receiver pair.
sync_pair(controller, target, toa_ct = T, toa_tc = T, tdoa_ct = T)

## Combine receiver pairs into a sync_order data.frame
sync_pair("rec_a", "rec_b") + sync_pair("rec_a", "rec_c")
```

Arguments

`detlst`	A DetectionsList object.
`controller`	The name of the receiver which serves as the reference clock in a paired synchronization.
`target`	The name of the receiver to apply clock corrections to.
`toa_ct`	Use time-of-arrival sync-tag transmissions emitted from the controller and detected on the the target.
`toa_tc`	Use time-of-arrival sync-tag transmissions emitted from the target and detected on the the controller.
`tdoa_ct`	Use time-difference-of-arrival sync-tag transmissions emitted from the rest of the array and detected on the controller and target.
`controllers`	A vector holding the names fo the controller receivers.

Details

Clock drift in a receiver array is corrected by applying a series of pair-wise synchronizations. `sync_pair()` can be used to specify a receiver pair where the target receiver clock is corrected to match that of the controller receiver's. A suitable sync_order data.frame should have a list of receiver pairs that covers all clocks in the array (see SyncTagConnectivity). Note that the order is important and clock synchronizations will be carried out in row-order.

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.
```

```
# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
```

```
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
```

```
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))
```

```
# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

`fit_clock_drift` *Estimate clock drift values*

Description

A mixed-effects model (implemented in TMB) will be used to estimate the drift values for each receiver pair listed in the sync_order.

Usage

```
## Fit clock drift values
model_drift <- fit_clock_drift(
  clocksync, sigma_det = 0.2, sigma_drift = 0.1,
  grouping_thresh, initial_clock_drift, quiet = F, silent = T,
  fix_sigma_det = F, fix_sigma_drift = F,  fix_transmission_latency = F)

## S3 method for class 'KaltoaClockDriftList'
clock_drift(object, ...)

## S3 method for class 'KaltoaClockDriftList'
print(x, ...)

## Plot fitted drift values
plot(x = model_drift, rate = F)
```

Arguments

`clocksync`	A KaltoaClockSync model.
`sigma_det`	Starting value for the standard deviation of the direct detection error (seconds). This will be fitted by the model.
`sigma_drift`	Starting value for the standard deviation of the clock drift (seconds of drift per hour of time). This will be fitted by the model.
`initial_clock_drift`	A matrix of initial clock drift values where rows are the receiver pairs and columns are the clock times. You can use `find_drift` to get these values. When missing, the clock drift values stored in `clocksync` will be used.
`quiet`	When `FALSE`, the time to fit each receiver pair will be reported.

Details

Clock drift is treated as a Gaussian random walk over time that follows a linear trend line. Drift values are estimated by aligning sync-tag transmissions between the receiver pairs specified in the `sync_order` data frame. Note the order of synchronization will affect the resulting clock drift values. Make sure to choose a `sync_order` where all receivers in the array are included and paired receivers have good connectivity (see SyncTagConnectivity).

The detection error will be modeled as a mixture of Gaussian and generalized normal distributions to capture direct and reflected transmissions, respectively. Hence, there is no need to remove reflected sync-tag detections before applying this function. See ddetect for details on the mixture distribution used.

It is not known which detection events resulted from any particular emission event. Hence, detection events are assumed to have originated from any nearby emission event. The detection error density is then summed over all of these nearby emission events.

Clock drift values are treated as random effects and fitted with the TMB package. It's critical to provide good `initial_clock_drift` values for the model to converge. We suggest using permute_clock_drift to get `initial_clock_drift` values within a resolution of at least 0.1 seconds.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Kaltoa-clock-sync: `KaltoaClockSync`, `KaltoaClockSync-slots`, `fit_clock_offsets()`, `fit_receiver_positions()`, `plot.KaltoaClockSync()`

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
```

```
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
```

```
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
```

```
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

`fit_clock_offsets` *Estimate clock offsets*

Description

Weighted least squares regression is used to find clock offsets for array receivers. The resulting transmission latency values will be fitted to match the distances between the receivers.

Usage

```
fit_clock_offsets(clocksync, pairs, estimate_transmission_speed = F)
```

Arguments

clocksync	A KaltoaClockSync model.
pairs	An optional two column data.frame holding the receiver pairs to use to estimate offsets. When missing, all combinations of receiver pairs will be used.
estimate_transmission_speed	When TRUE, the speed of tag transmission will be estimated.

Details

For each receiver pair, the inner 95% of latency values are kept, removing extreme outliers. Next, the detection error distribution ddetect is fitted to the remaining latency values. Note that the presence of reflected transmissions should have a negligible effect as they are captured by the large positive component of the detection error distribution. The mean latency of direct sync-tag transmissions is returned for each pair along with the precision of the estimate. The latter will be used as the weighting term (inverse variance) in the least squares.

Weighted least squares regression is then applied to fit receiver offset values. The fitted offset values will minimize the difference between the mean latencies between the receiver pairs and their predicted values (based on distances between receiver pairs and the speed of tag transmission).

Optionally, the speed of tag transmission can be fitted while running the least squares model.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Kaltoa-clock-sync: KaltoaClockSync, KaltoaClockSync-slots, fit_clock_drift(), fit_receiver_positions(
plot.KaltoaClockSync()

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
```

```
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
```

```
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
```

```
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

`fit_receiver_positions` *Estimate receiver positions*

Description

Sync-tag transmissions are utilized to estimate the true positions of the array receivers.

Usage

```
fit_receiver_positions(
  clocksync,
  ref_receivers,
  adj_receivers,
  estimate_transmission_speed = T,
  estimate_gps_error = F
)
```

Arguments

clocksync	A KaltoaClockSync object.
ref_receivers	Vector of receiver names to use as reference points—receivers with known locations.
adj_receivers	Vector of receiver names to estimate new positions for.
estimate_transmission_speed	When TRUE the speed of sound transmission will be estimated.
pairs	Optional two column data.frame holding the pairs of receiver names to use sync-tag latency values from. When missing, all receiver pair combinations will be used.

Details

Similar to fit_clock_offsets, for each receiver pair the mean direct transmission latency and standard error of the mean will be fitted using the detection error distribution ddetect. A set of reference receivers with known positions will be specified by either `ref_receivers` or `adj_receivers`. Next, the remaining receiver positions will be fitted to minimize the difference between the mean and expected latency values for each pair.

It is advisable to set at least 2 receivers as reference points.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Kaltoa-clock-sync: `KaltoaClockSync`, `KaltoaClockSync-slots`, `fit_clock_drift()`, `fit_clock_offsets()`, `plot.KaltoaClockSync()`

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
```

```
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
```

```
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
```

```
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
```

```
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

`fit_transmission_time` *Fit a transmission time model to a Latency object*

Description

A mixed detection error distribution, ddetect, will be fitted to the latency object. The mean transmission latency will be returned along with the standard deviation of the direct detection error.

Usage

```
fit_transmission_time(lat, phi = 1, beta = 32, remove_outliers = T)
```

Arguments

`lat`	A Latency object.
`phi, beta`	see ddetect.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Latency: `Latency()`, `plot.Latency()`

Examples

```
# ------ Clear double detections from 2 receivers
# Emitting receiver
det_emis <- remove_double_detections(
  exdata_positioning[["1"]], dd_thresh = 0.5)
# Detecting receiver
det_recv <- remove_double_detections(
  exdata_positioning[["2"]], dd_thresh = 0.5)

# ------ Generate sync-tag latency values for receiver pair
lat <- Latency(det_emis, det_recv, group_thresh = 0.5)
head(lat)

# Show clock drift over time
plot(lat)
# Plot delta latency values
plot(lat, delta = TRUE)
```

get_toa *Get time-of-arrival matrix from receiver array*

Description

For a given tag ID, this will return a TOAMatrix that can be used for positioning.

Usage

```
get_toa(
  detlst,
  tag_id,
  group_thresh = 0.5,
  quietly = F,
  transmission_speed = 1500
)
```

Arguments

detlst	A DetectionsList object
tag_id	A character string holding the tag ID to return arrival times for.
group_thresh	Threshold in seconds for grouping detections to the same emission event.
quietly	When TRUE, no messages will be printed.

Value

A TOAMatrix object.

See Also

Other TOAMatrix: TOAMatrix, TOAMatrix-generics, plot.TOAMatrix()

Examples

```
# ------ Make time-of-arrival matrix from DetectionsList
toa <- get_toa(exdata_positioning,
  tag_id = "HR-53883",
  group_thresh = 0.5)

# ------ Summary info
summary(toa)
head(toa)

# Plot diagnostics
plot(toa, type = "hist")
plot(toa, type = "diag")

# Subset by diagnostic data
```

```
toa_sub_1 <- subset(toa, detections >= 4 & period <= 0.05)
head(toa_sub_1)
plot(toa_sub_1, type = "diag")


# Subset receivers
toa_sub_2 <- subset(toa, select = c("1", "2", "3", "4"))
head(toa_sub_2)

# ------ Export as csv file
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(x = toa, file = fh, row.names = F)
close(fh)

# ------ Import from csv
fh <- gzfile(fname, open = "r")
df_toa <- read.csv(file = fh, check.names = F)
det_import <- as.TOAMatrix(
  df_toa, tag_id = "HR-53883", receiver_positions = positions(toa),
  transmission_speed = 1420)
close(fh)
```

`KaltoaClockSync` *Create a KaltoaClockSync model*

Description

KaltoaClockSync models hold corrections for clock drift and receiver locations. The synchronize function can then be used to apply the corrections to a DetectionsList object.

See KaltoaClockSync-slots for generic methods and slot access functions.

Usage

```
# Initialize a KaltoaClockSync model.
sync <- KaltoaClockSync(
    sync_order, clock_times, clock_drift, clock_offset,
    detlst, max_transmission_latency = 500/1500,
    beta = 32, grouping_thresh = 0.5,
    transmission_speed = 1500)

## Apply KaltoaClockSync corrections to DetectionsList
synchronize(detlst, clocksync,
  clock_drift = T, clock_offsets = T, position_offsets = T)
```

Arguments

`sync_order`	A data.frame holding the order of paired receiver synchronizations to carry out for drift corrections. This can be created with the sync_pair function.

`clock_times` A vector of POSIXct times to estimate clock drift values at.

`clock_drift` A matrix of clock drift values. This can be manually specified or set automatically using the permute_clock_drift and fit_clock_drift functions.

`detlst` A DetectionsList holding sync-tag emissions and detections across the array.

`phi` The maximum possible detection error resulting from a reflected transmission. This is used in the detection error distribution, ddetect. A sensible starting value is to take the maximum expected transmission distance of a tag emission within the study area divided by the transmission speed.

`beta` The beta parameter for the generalized normal distribution. Higher values result in steeper edges and a better approximation to a random uniform distribution. See ddetect and dgnorm.

`grouping_thresh`
The threshold in seconds to use when matching sync-tag emissions to their resulting detections. Setting this to half the minimum sync-tag emission interval is a recommended starting value. This is used by the fit_clock_drift and plot.KaltoaClockSync functions.

`transmission_speed`
The speed of signal transmission (meters per second).

`clock_offsets` A vector of clock offset values. This can be manually specified or set later using fit_clock_offsets.

Details

KaltoaClockSync objects hold three types of corrections that can be applied to DetectionsList objects. Each type can be set manually (see KaltoaClockSync-slots) or with automatic fitting functions.

- **clock drift**: For each receiver pair listed in `sync_order`, each value in `clock_times` will have a matching drift correction value. These can be initially estimated using permute_clock_drift and then further refined with fit_clock_drift.
- **clock offsets**: Each receiver has a single offset value. These values are used to align the sync-tag transmission latencies with the known distances between receivers across the entire array. These can be set automatically with fit_clock_offsets.
- **receiver position offsets**: Each receiver has an x and y coordinate offset, correcting for errors in the provided locations of receivers. fit_receiver_positions will automatically fit these values.

A typical workflow involves initializing a KaltoaClockSync object and then fitting clock drift, clock offset, and position offset values — in this order. See the examples below.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Kaltoa-clock-sync: `KaltoaClockSync-slots`, `fit_clock_drift()`, `fit_clock_offsets()`, `fit_receiver_positions()`, `plot.KaltoaClockSync()`

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
```

```
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')
```

```
# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
```

```
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

KaltoaClockSync-slots *KaltoaClockSync generic functions*

Description

Generic methods and slot access functions for KaltoaClockSync objects.

Usage

```
# Get clock drift values
clock_drift(object)

# Set clock drift values
clock_drift(object) <- value

# Get clock times
clock_times(x)

# Get clock offset values
clock_offsets(x)

# Set clock drift values
clock_offsets(x) <- value

# Get receiver positions
positions(x)

# Set receiver positions
positions(x) <- value

# Get receiver position offsets
positions_offsets(x)

# Set receiver position offsets
```

```
positions_offsets(x) <- value

# Get tag transmission speed
transmission_speed(x)

# Set tag transmission speed
transmission_speed(x) <- value

# Get receiver-pair sync order
sync_order(x)
```

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Kaltoa-clock-sync: `KaltoaClockSync`, `fit_clock_drift()`, `fit_clock_offsets()`, `fit_receiver_positions()`, `plot.KaltoaClockSync()`

KaltoaKalmanPositions *KaltoaKalmanPositions object*

Description

This object holds the position estimates and error covariances returned by KaltoaTrackPositioning methods.

See KaltoaKalmanPositions-generics for generic methods and slot access functions.

Usage

```
# Manually create a KaltoaKalmanPositions object.
track <- KaltoaKalmanPositions(
  states, covariance,
  fcn_state_transition, fcn_state_covariance,
  times, start_time = NULL)

# Plot track positions
plot(x = track, xlim, ylim, legend = T, add = F, ...)

# Apply RTS smoother to state estimates.
smooth(track)

# Predict states for new emission times
predict(track, times_out)
```

Arguments

`times_out`	A vector of absolute POSIXct times, or relative numeric times in seconds.

Details

State estimates and covariances are returned as arrays where the final dimension indexes *a priori*, *a posteriori*, and *RTS smoothed* values.

The slots `fcn_state_transition` and `fcn_state_covariance` are functions with the form `function(delta_i){...}`, where `delta_i` is the time period between the current and previous state estimates. Here, any model parameters must be hard coded into the functions.

Typically, the user should not need to create this object themselves.

Slots

`states` An (`n_states` x `n_positions` x 3) array holding state estimates.

`covariance` An (`n_states` x `n_states` x `n_positions` x 3) array error covariance matrices.

`fcn_state_transition` A function for creating a state transition matrix.

`fcn_state_covariance` A function for creating a state covariance matrix.

`times` The relative time (in seconds) of each position estimate.

`start_time` POSIXct indicating the absolute start time.

`smoothed` When `TRUE`, RTS-smoothed state estimates are available.

Examples

```
# Apply EK-TOA positioning model

# ------ Load positioning data
# We'll use only Innovasea PPM tags here, as they are less prone to reflections
# then the HR tags.
toa <- get_toa(exdata_positioning, tag_id = c("PPM-53883"),
    group_thresh = 0.5, transmission_speed= 1420)

# Preview TOAMatrix
summary(toa)
plot(toa)

# ------ Set parameters for component models
# Define a CTCRW process model.
t_05 <- 15      # Seconds until velocity correlation falls below 0.05
sigma_vel <- 0.2     # SD of uncorrelated velocity distribution
beta = -log(0.05)/t_05
alpha = sigma_vel*sqrt(2*beta)
proc_model <- KaltoaProcess(
  type = "ctcrw", alpha = alpha, beta = beta)
# Visualize model
plot(proc_model)

# Make time-of-arrival observation model
obs_sigma <- 0.002
obsv_model <- KaltoaObservation(sigma = obs_sigma)
# Visualize model
plot(obsv_model)
```

```
# ------ Apply EK-TOA positioning model
# Apply EK-TOA positioning
track <- KaltoaTrackPositioning(
  x = subset(toa, detections >= 2),
  observation_model = obsv_model,
  process_model = proc_model)
# Plot results
plot(track)
points(positions(toa), pch = 17, col = 'red')

# Apply RTS smoother
track <- smooth(track)
# Plot results
plot(track)
points(positions(toa), pch = 17, col = 'red')

# Predict positions along a discrete time-series
times <- emission_times(track)
times_out <- seq(floor(times[1]), ceiling(times[length(times)]), by = 10)
track_interp <- predict(object = track, times_out = times_out)

# Plot results
plot(track_interp)
points(positions(toa), pch = 17, col = 'red')
points(positions(track), col = "blue", pch = 4)

# --- Export to csv
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(as.data.frame(as.KaltoaPointPositions(track)), file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

`KaltoaKalmanPositions-generics`
KaltoaKalmanPositions generic functions

Description

Generic methods and slot access functions for KaltoaKalmanPositions objects.

Usage

```
# Extract state estimates
```

```
states(track)

# Extract state error covariance matrices
covariance(track)

# Extract emission times
emission_times(track)

# Extract start time
start_time(track)

# Extract position estimates
positions(track)

# Indexing
track[i]

# Convert to KaltoaPointPositions
as.KaltoaPointPositions(track)
```

See Also

See Kaltoa-overview to quickly find the functions you need.

Other KaltoaKalmanPositions: `KaltoaPointPositions-generics`

`KaltoaObservation` *Create KaltoaObservation model*

Description

This object holds the detection error parameters used in positioning models.

Usage

```
# Create PObservation model
obsv <- KaltoaObservation(type = "gaussian",
  max_transmission_latency = 500/1500, ...)

# Plot detection error density function
plot(x = obsv, add = F, xlim, ylim, ...)
```

Arguments

`type`	The type of observation model. Gaussian, "gaussian", or a mixture, "mixed".
`max_transmission_latency`	The maximum possible transmission latency for an emission in seconds.
`...`	See details for the required parameters of each movement model type.

Details

`type` is used to choose from a Gaussian or mixed detection error distribution. The mixed distribution (see Campbell et al., 2005) is a combination of a Gaussian and a smoothed approximation to a uniform random for capturing direct and reflected detections, respectively.

Each model type requires the following parameters to be set *a priori*.

Gaussian

- `sigma` Standard deviation.

Mixture

- `sigma` Standard deviation of the Gaussian component.
- `beta` Shape of generalized normal component. Higher values give steeper edges, more closely approximating a uniform distribution.

The mixture distribution captures reflected detections by modelling them as a generalized normal distribution that ranges from `0` to `max_transmission_latency`. Here, `beta` controls the shape of this component distribution. Make sure to choose a `beta` which doesn't allow the generalized normal to overshoot the negative side of the Gaussian component, but also has slightly smoothed edges to aid in model fitting. We suggest starting out with `beta = 32` then using plot.KaltoaObservationModel to visually check the shape of the resulting detetion error distribution.

The function ddetect is used to calculate the density of the mixture distribution.

References

- Campbell, J. A., Shry, S. J., Lundberg, P., Calles, O., Hölker, F. (2025) **A population Monte Carlo model for underwater acoustic telemetry positioning in reflective environments**. *Methods Ecology and Evolution*. 16(4):775-785.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
c = 1500# (m/s) speed of tag transmission
max_trns_lat <- 500 / c# (s) maximum expected transmission latency
# Here, we assume a transmission will go no further
# then 500 meters.

sigma <- 0.002# (s) SD of detection error.

# --- Gaussian detection error
# This is suitable when no reflected transmissions
# are present
obsv_gaus <- KaltoaObservation(type = "gaussian",
  max_transmission_latency = max_trns_lat,
  sigma = sigma)
print(obsv_gaus)
plot(obsv_gaus)
```

```
# --- Mix of Gaussian and generalized normal
# This is suitable when reflected transmissions are present
obsv_gaus_1 <- KaltoaObservation(type = "mix",
  max_transmission_latency = max_trns_lat,
  sigma = sigma, beta = 32)
plot(obsv_gaus_1)

# Compare the effect of changing beta
obsv_gaus_2 <- KaltoaObservation(type = "mix",
  max_transmission_latency = max_trns_lat,
  sigma = sigma, beta = 128)
plot(obsv_gaus_1, xlim = c(-0.01, 0.03), ylim = c(0,10))
plot(obsv_gaus_2, xlim = c(-0.01, 0.03), ylim = c(0,10))
```

`KaltoaPMCPositions` *KaltoaPMCPositions object*

Description

This object holds the position estimates and error covariances returned by KaltoaPointPositioning. This can be further post-processed with KaltoaTrackPositioning.

Usage

```
# Convert to KaltoaPointPositions
as.KaltoaPointPositions(x, iter)

# Return the probability that a detection results
# from a reflected transmission.
p_reflect(x)

# Get start time
start_time(x)

# Indexing
x[pos]
```

Arguments

`iter`	Select particles from a particular iteration of the algorithm.

Slots

`pmc_results` A list holding PMC-TOA particles and weights for each iteration of the algorithm.

`iter` Number of algorithm iterations.

`n_particles` Number of particles per iteration.

`pertubation_sigma` Standard deviation of the perturbation kernel.

`observation_model` A KaltoaObservation object holding the detection error distribution.

`timer` Number of seconds for the algorithm to run.

`weighting_method` Either `"PMC"` or `"DM-PMC"`.

`toa` A TOAMatrix object holding the relative detection times for each emission.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other KaltoaPMCPositions: `plot.KaltoaPMCPositions()`

Examples

```
# We'll use a population Monte Carlo (PMC) algorithm to estimate the (1)
# location of the tag in our array and (2) the probability that a given
# detection is an outlier resulting from a reflected transmission path.

# Load detections and hydrophone data
# We'll extract PPM and HR detections, then combine them
# into a single TOAMatrix
toa <- c(
  get_toa(exdata_positioning, tag_id = c("HR-53883"),
          group_thresh = 0.5, transmission_speed = 1420),
  get_toa(exdata_positioning, tag_id = c("PPM-53883"),
    group_thresh = 0.5, transmission_speed = 1420))

# Create observation model (receiver detection error)
# This will capture both direct-path detections and reflected detections,
# the latter appearing as large positive outliers
obsv_model <- KaltoaObservation(
  type = "mixed",  max_transmission_latency = 500/1500,
  sigma = 0.002,# (m/s) Stationary velocity
  beta = 32)# Shape parameter
# Preview distribution
plot(obsv_model)
plot(obsv_model, xlim = c(-0.01, 0.03), ylim = c(0, 10))

# ------ Position a single emission
# Use automatic initial positions
# (particles will be sampled from a convex polygon around the array)
model_points <- KaltoaPointPositioning(
  toa = toa[103,],# Time-of-arrival for single emission
  observation_model = obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)
```

```
# --- Specify initial positions within 25m of any receiver
require(sf)
init_pos <- st_multipoint(positions(toa)[,-3]) |>
  st_buffer(dist = 25) |>
  st_sample(type = "hexagonal", size = 200) |>
  st_coordinates()
plot(init_pos, asp = 1, col = 'grey')
points(positions(toa), pch = 17, col = "red")
# Apply model
model_points <- KaltoaPointPositioning(
  toa = toa[103,], init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# ------ Position all emissions
model_points <- KaltoaPointPositioning(
  toa = subset(toa, detections >= 4),
  init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 3)
# Show results
model_points
plot(model_points)

# ------ Apply Post-processing
# Get movement parameters
t_05 = 15; sigma_vel = 0.2
beta = -log(0.05) / t_05
alpha = sigma_vel * sqrt( 2 * beta )
# Initialize process model
proc_model <- KaltoaProcess(type = "ctcrw",
  alpha = alpha, beta = beta)
# Plot process model
plot(proc_model)

# --- Apply Kalman filter
model_track <- KaltoaTrackPositioning(
  model_points, process_model = proc_model) |>
  smooth()
plot(model_track)
points(positions(toa), pch = 17, col = 'red')

# --- Predict positions along a discrete time-series
times <- emission_times(model_track)
```

```
times_out <- seq(floor(times[1]), ceiling(times[length(times)]), by = 10)
track_interp <- predict(object = model_track, times_out = times_out)
# Plot results
plot(track_interp)
points(positions(model_track), col = "blue", pch = 4)
points(positions(toa), pch = 17, col = "red")

# --- Export to csv
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(track_interp, file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

`KaltoaPointPositioning` *Estimate point positions*

Description

Estimate tag locations using a population Monte Carlo positioning model.

Usage

```
KaltoaPointPositioning(
  toa,
  observation_model,
  init_positions,
  n_particles = 500,
  iter = 5,
  weighting_method = "PMC",
  perturbation_sigma,
  depths = NULL,
  quietly = F
)
```

Arguments

`toa`	A TOAMatrix object.
`observation_model`	A KaltoaObservation object.
`init_positions`	A 2 (or 3) x n matrix of initial tag locations. When missing, `n_particles` locations will be uniformly sampled from a convex polygon surrounding the receiver array.

`n_particles` Number of particles to use in the PMC-TOA algorithm.

`iter` Number of iterations for the PMC-TOA algorithm.

`weighting_method`
The weighting method, `"PMC"` or `"DM-PMC"`. The latter is more accurate but processing times will scale poorly when using large numbers of particles.

`perturbation_sigma`
The standard deviation of the perturbation kernel. When missing, will be set to double the value of detection error standard deviation.

Details

PMC-TOA can solve positions using arbitrary detection error distributions. When provided with a `"mixed"` KaltoaObservation model, reasonably good positions can be estimated in reflective environments where late arriving detection outliers are present.

The resulting KaltoaKalmanPositions object can be further passed onto KaltoaTrackPositioning to refine the position estimates.

When all receivers in the TOAMatrix have depth values, 3D positioning will automatically be applied.

Supplemental depth data can be provided to the argument `depths`. In this case, the provided depth values will be assumed to be true. For unknown depths `NA` values can be provided, causing depth values to be estimated for those positions.

Value

A KaltoaKalmanPositions object.

References

- Campbell, J. A., Shry, S. J., Lundberg, P., Calles, O., Hölker, F. (2025) **A population Monte Carlo model for underwater acoustic telemetry positioning in reflective environments**. *Methods Ecology and Evolution*. 16(4):775-785.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
# We'll use a population Monte Carlo (PMC) algorithm to estimate the (1)
# location of the tag in our array and (2) the probability that a given
# detection is an outlier resulting from a reflected transmission path.

# Load detections and hydrophone data
# We'll extract PPM and HR detections, then combine them
# into a single TOAMatrix
toa <- c(
  get_toa(exdata_positioning, tag_id = c("HR-53883"),
          group_thresh = 0.5, transmission_speed = 1420),
  get_toa(exdata_positioning, tag_id = c("PPM-53883"),
```

```
    group_thresh = 0.5, transmission_speed = 1420))

# Create observation model (receiver detection error)
# This will capture both direct-path detections and reflected detections,
# the latter appearing as large positive outliers
obsv_model <- KaltoaObservation(
  type = "mixed",  max_transmission_latency = 500/1500,
  sigma = 0.002,# (m/s) Stationary velocity
  beta = 32)# Shape parameter
# Preview distribution
plot(obsv_model)
plot(obsv_model, xlim = c(-0.01, 0.03), ylim = c(0, 10))

# ------ Position a single emission
# Use automatic initial positions
# (particles will be sampled from a convex polygon around the array)
model_points <- KaltoaPointPositioning(
  toa = toa[103,],# Time-of-arrival for single emission
  observation_model = obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# --- Specify initial positions within 25m of any receiver
require(sf)
init_pos <- st_multipoint(positions(toa)[,-3]) |>
  st_buffer(dist = 25) |>
  st_sample(type = "hexagonal", size = 200) |>
  st_coordinates()
plot(init_pos, asp = 1, col = 'grey')
points(positions(toa), pch = 17, col = "red")
# Apply model
model_points <- KaltoaPointPositioning(
  toa = toa[103,], init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# ------ Position all emissions
model_points <- KaltoaPointPositioning(
```

```
  toa = subset(toa, detections >= 4),
  init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 3)
# Show results
model_points
plot(model_points)

# ------ Apply Post-processing
# Get movement parameters
t_05 = 15; sigma_vel = 0.2
beta = -log(0.05) / t_05
alpha = sigma_vel * sqrt( 2 * beta )
# Initialize process model
proc_model <- KaltoaProcess(type = "ctcrw",
  alpha = alpha, beta = beta)
# Plot process model
plot(proc_model)

# --- Apply Kalman filter
model_track <- KaltoaTrackPositioning(
  model_points, process_model = proc_model) |>
  smooth()
plot(model_track)
points(positions(toa), pch = 17, col = 'red')

# --- Predict positions along a discrete time-series
times <- emission_times(model_track)
times_out <- seq(floor(times[1]), ceiling(times[length(times)]), by = 10)
track_interp <- predict(object = model_track, times_out = times_out)
# Plot results
plot(track_interp)
points(positions(model_track), col = "blue", pch = 4)
points(positions(toa), pch = 17, col = "red")

# --- Export to csv
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(track_interp, file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

KaltoaPointPositions *KaltoaPointPositions object*

Description

Object holding point position estimates and covariances.

See KaltoaPointPositions-generics for generic methods and slot access functions.

Usage

```
# Create new object
kpp <- KaltoaPointPositions(states, covariance, start_time, times = NULL)

# Plot positions
plot(x = kpp, add = F, t = 'b', legend = T, xlim, ylim, ...)

## S3 method for class 'data.frame'
as.KaltoaPointPositions(object, tz, ...)
```

Arguments

states	Matrix holding state estimates.
covariance	Array holding state estimate covariances.
start_time	POSIXct value giving the absolute start time.
times	Vector of relative emission times in seconds.

Slots

`states` Matrix holding state estimates.

`covariance` Array holding covariances for state estimates.

`start_time` POSIXct holding the start time.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
# We'll use a population Monte Carlo (PMC) algorithm to estimate the (1)
# location of the tag in our array and (2) the probability that a given
# detection is an outlier resulting from a reflected transmission path.

# Load detections and hydrophone data
# We'll extract PPM and HR detections, then combine them
# into a single TOAMatrix
toa <- c(
  get_toa(exdata_positioning, tag_id = c("HR-53883"),
          group_thresh = 0.5, transmission_speed = 1420),
  get_toa(exdata_positioning, tag_id = c("PPM-53883"),
    group_thresh = 0.5, transmission_speed = 1420))

# Create observation model (receiver detection error)
```

```
# This will capture both direct-path detections and reflected detections,
# the latter appearing as large positive outliers
obsv_model <- KaltoaObservation(
  type = "mixed",  max_transmission_latency = 500/1500,
  sigma = 0.002,# (m/s) Stationary velocity
  beta = 32)# Shape parameter
# Preview distribution
plot(obsv_model)
plot(obsv_model, xlim = c(-0.01, 0.03), ylim = c(0, 10))

# ------ Position a single emission
# Use automatic initial positions
# (particles will be sampled from a convex polygon around the array)
model_points <- KaltoaPointPositioning(
  toa = toa[103,],# Time-of-arrival for single emission
  observation_model = obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# --- Specify initial positions within 25m of any receiver
require(sf)
init_pos <- st_multipoint(positions(toa)[,-3]) |>
  st_buffer(dist = 25) |>
  st_sample(type = "hexagonal", size = 200) |>
  st_coordinates()
plot(init_pos, asp = 1, col = 'grey')
points(positions(toa), pch = 17, col = "red")
# Apply model
model_points <- KaltoaPointPositioning(
  toa = toa[103,], init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# ------ Position all emissions
model_points <- KaltoaPointPositioning(
  toa = subset(toa, detections >= 4),
  init_positions = init_pos,
  observation_model =  obsv_model,
```

```
  n_particles = 200, iter = 3)
# Show results
model_points
plot(model_points)

# ------ Apply Post-processing
# Get movement parameters
t_05 = 15; sigma_vel = 0.2
beta = -log(0.05) / t_05
alpha = sigma_vel * sqrt( 2 * beta )
# Initialize process model
proc_model <- KaltoaProcess(type = "ctcrw",
  alpha = alpha, beta = beta)
# Plot process model
plot(proc_model)

# --- Apply Kalman filter
model_track <- KaltoaTrackPositioning(
  model_points, process_model = proc_model) |>
  smooth()
plot(model_track)
points(positions(toa), pch = 17, col = 'red')

# --- Predict positions along a discrete time-series
times <- emission_times(model_track)
times_out <- seq(floor(times[1]), ceiling(times[length(times)]), by = 10)
track_interp <- predict(object = model_track, times_out = times_out)
# Plot results
plot(track_interp)
points(positions(model_track), col = "blue", pch = 4)
points(positions(toa), pch = 17, col = "red")

# --- Export to csv
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(track_interp, file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

`KaltoaPointPositions-generics`

KaltoaPointPositions generic functions

Description

Generic methods and slot access functions for KaltoaPointPositions objects.

Usage

```
# Indexing
x[i]

# Combine positions
c(...)

# Get positions
positions(x)

# Get positions
positions(x)

# Indexing
states(x)

# Indexing
covariance(x)

# Get start time
start_time(x)

# Indexing
emission_times(x)

# Add values to TOAMatrix
append(toa_mat, values)
```

See Also

See Kaltoa-overview to quickly find the functions you need.

Other KaltoaKalmanPositions: KaltoaKalmanPositions-generics

KaltoaProcess *Create KaltoaProcess model*

Description

KaltoaProcess model objects hold the animal movement parameters for state-spece positioning models.

Usage

```
KaltoaProcess(type = "rw", ...)

## S3 method for class 'KaltoaProcessModel'
plot(x, add = F, ...)
```

Arguments

`type`	The type of movement model. Random walk, `"rw"`, or continuous-time correlated random walk, `"ctcrw"`.
...	See details for the required parameters of each movement model type.

Details

`type` is used to choose from a standard Gaussian random walk (RW), which only takes step length into consideration, or a continuous-time correlated random walk (CTCRW, Johnson el al. 2008), which models movement as a correlated velocity process. In most cases we suggest choosing CTCRW, as even a small amount of velocity correlation can reduce "stationary jitter" in tracks where positioning error is misattributed to erratic movements around a fixed point in space.

Each model type requires the following parameters to be set *a priori*.

Random Walk:

- `sigma` The standard deviation of the step length.

Continuous-time correlated random walk

- `alpha`, `beta`. These parameters define the variance and correlation of the velocity process.

For CTCRW models, α and β can be chosen by first selecting the stationary velocity standard deviation σ_{vel} and the time until the velocity correlation falls below 0.05, t_{05}. Then

$$\beta = \log(0.05) \cdot t_{05}^{-1},$$

and

$$\alpha = \sigma_{\text{vel}} \cdot \sqrt{2 \cdot \beta}.$$

Note that the CTCRW process is applied to each spatial axis (x & y) independently. When we consider movement as velocity vectors in the horizontal plane, the magnitude of the stationary velocity vector takes on the shape of Rayleigh distribution, where σ_{vel} is the mode of that distribution.

References

- Johnson, D. S., London, J. M., Lea, M., Durban, J. W. (2008) **Continuous-time correlated random walk model for animal telemetry data**. *Ecology*, 89(5):1208-1215.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
# --- Gaussian random walk
proc_rw <- KaltoaProcess(type = "rw", sigma = 0.2)
print(proc_rw)

# --- Continuous-time correlated random walk
# Time until velocity correlation falls below 0.05
```

```
t_05 = 5# seconds
# Stationary velocity SD
sigma_vel = 0.2# meter per second
# Create model
beta = -log(0.05) / t_05
alpha = sigma_vel * sqrt( 2 * beta )
proc_ctcrw <- KaltoaProcess(type = "ctcrw", alpha = alpha, beta = beta)
print(proc_ctcrw)
plot(proc_ctcrw)
```

`KaltoaReceiverPositions` *Create a KaltoaReceiverPositions model*

Description

Provided there is no clock drift, this model will estimate the true receiver positions in an array.

Usage

```
KaltoaReceiverPositions(
  detlst,
  ref_receivers,
  c = 1500,
  group_thresh = 1,
  phi = 500/1500,
  beta = 32,
  estimate_transmission_speed = T,
  estimate_gps_error = T
)
```

Arguments

`detlst`	A DetectionsList object.
`ref_receivers`	A vector holding the names of a pair of receivers to be used as reference locations for the positioning model.
`c`	Speed of tag transmission.
`group_thresh`	Period in seconds used to group sync tag emission/detection pairs.
`phi, beta`	See ddetect.
`init_sd`	The standard deviation of the initial position estimates, in meters. This restricts receiver offsets so they cant result in a array configuration wildy different than the inital positions.

Details

For receiver positions to be estimated, clock drift must be removed from the DetectionsList (see KaltoaClockSync).

Sync-tag transmission latency times will be measured for each receiver pair in the array. Using the mean and standard deviation of these latency values, the `nlminb` function will be used to estimate position offsets that minimize the difference between the expected and observed transmission latencies. Standard errors taken from the Hessian are reported for each estimated position offset.

A pair of reference receivers must be specified with `ref_receivers`. Some trial and error may be necessary to find an ideal pair of reference receivers that results in suitable corrections.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
```

```
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
```

```
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
```

```
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

`KaltoaTrackPositioning`

Estimate track positions

Description

A generic function that applies a Kalman filter to position or time-of-arrival data. The filter estimates positions by making assumptions about the underlying animal movement process that generated the data.

Usage

```
# S3 Generic
KaltoaTrackPositioning(x, process_model, ...)

# Apply Kalman filter to point position data
KaltoaTrackPositioning.KaltoaPointPositions(x, process_model)

# Apply Kalman filter to PMC_TOA estimates
```

```
KaltoaTrackPositioning.KaltoaPMCPositions(x, process_model)

# Apply Kalman filter directly to time-of-arrival data
KaltoaTrackPositioning.TOAMatrix(x, observation_model, process_model)
```

Arguments

`x`	A KaltoaPointPositions, KaltoaPMCPositions, or TOAMatrix object.
`process_model`	A KaltoaProcessModel object holding the animal movement parameters.
`observation_model`	For TOAMatrix objects
`depths`	An optional vector of known depth values. Unknown depths can be provided as `NA` values and will be estimated.

Details

When depth data is available in `x`, 3D positioning will be automatically applied. Otherwise, 2D positioning is used.

When applied to a TOAMatrix, EK-TOA positioning (Campbell et al., *in review*) will be applied. Here, σ_{emis} is set to double the value of `maximum_transmission_latency` and the initial values for each emission time are set to the first detection time of each emission.

For 3D positioning of a TOAMatrix, if some or all depths are known beforehand (from pressure sensor data, for example) these can be provided with the argument `depths`. Here, `NA` values can be used to indicate depth values that are not known *a priori* and need to be estimated. The *a priori* depth values are treated as truth with no observation error. The estimated depth values are are fitted by a Gaussian error distribution with a mean and standard deviation equal to half of the depth value of the deepest receiver. This restriction prevents depth estimates that are far above the water surface or below the deepest receiver. Note that negative depth values (position estimates above the water surface) are possible.

When applied to a KaltoaPointPositions or KaltoaPMCPositions object, a Kalman filter will be used to refine the position estimates, making use of the positioning error covariance matrices for each point position.

Value

A KaltoaKalmanPositions object.

References

- Campbell, J. A., Ellings, J., Lundberg, P., Mawer, R., Pauwels, I., Hölker, F. 2025. **An extended Kalman filter for large-volume path positioning of aquatic animals within acoustic telemetry arrays**. *In review*.

Examples

```
# Apply EK-TOA positioning model

# ------ Load positioning data
```

```
# We'll use only Innovasea PPM tags here, as they are less prone to reflections
# then the HR tags.
toa <- get_toa(exdata_positioning, tag_id = c("PPM-53883"),
    group_thresh = 0.5, transmission_speed= 1420)

# Preview TOAMatrix
summary(toa)
plot(toa)

# ------ Set parameters for component models
# Define a CTCRW process model.
t_05 <- 15      # Seconds until velocity correlation falls below 0.05
sigma_vel <- 0.2     # SD of uncorrelated velocity distribution
beta = -log(0.05)/t_05
alpha = sigma_vel*sqrt(2*beta)
proc_model <- KaltoaProcess(
  type = "ctcrw", alpha = alpha, beta = beta)
# Visualize model
plot(proc_model)

# Make time-of-arrival observation model
obs_sigma <- 0.002
obsv_model <- KaltoaObservation(sigma = obs_sigma)
# Visualize model
plot(obsv_model)

# ------ Apply EK-TOA positioning model
# Apply EK-TOA positioning
track <- KaltoaTrackPositioning(
  x = subset(toa, detections >= 2),
  observation_model = obsv_model,
  process_model = proc_model)
# Plot results
plot(track)
points(positions(toa), pch = 17, col = 'red')

# Apply RTS smoother
track <- smooth(track)
# Plot results
plot(track)
points(positions(toa), pch = 17, col = 'red')

# Predict positions along a discrete time-series
times <- emission_times(track)
times_out <- seq(floor(times[1]), ceiling(times[length(times)]), by = 10)
track_interp <- predict(object = track, times_out = times_out)

# Plot results
plot(track_interp)
points(positions(toa), pch = 17, col = 'red')
points(positions(track), col = "blue", pch = 4)

# --- Export to csv
```

```
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(as.data.frame(as.KaltoaPointPositions(track)), file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

`Latency` *Create a latency object.*

Description

For pairs of receivers with sync-tags, show the measured transmission latency. Transmission latency is the time between when a signal is emitted from one receiver and detected on another. Latency objects are used to measure and correct array clock drift and offset.

Usage

```
Latency(det_emit, det_recv, group_thresh = 1)

## S3 method for class 'Latency'
x[...]

## S3 method for class 'Latency'
summary(lat)

## S3 method for class 'Latency'
x + y

## S3 method for class 'Latency'
x - y
```

Arguments

`det_emit`	The Detections object of the emitting receiver.
`det_recv`	The Detections object of the detecting receiver.
`group_thresh`	Sync-tag emissions/detections falling within this period will be grouped.

Details

Latency values are generated by pairing sync-tag detections on `det_recv` with emissions on `det_emit`. Emission-detection pairs are made if they both fall within `group_thresh` seconds of each other.

Note that double detections can cause issues here, as each detection is assumed to match with a single emission. remove_double_detections can easily remove problematic double detections from Detections objects.

The columns titled `second` and `millisecond` hold the time recorded on the detecting receiver.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Latency: `fit_transmission_time()`, `plot.Latency()`

Examples

```
# ------ Clear double detections from 2 receivers
# Emitting receiver
det_emis <- remove_double_detections(
  exdata_positioning[["1"]], dd_thresh = 0.5)
# Detecting receiver
det_recv <- remove_double_detections(
  exdata_positioning[["2"]], dd_thresh = 0.5)

# ------ Generate sync-tag latency values for receiver pair
lat <- Latency(det_emis, det_recv, group_thresh = 0.5)
head(lat)

# Show clock drift over time
plot(lat)
# Plot delta latency values
plot(lat, delta = TRUE)
```

`offset_correction_fit` *Calculate clock offsets for a receiver array.*

Description

Calculate clock offsets for a receiver array.

Usage

```
offset_correction_fit(latency, X, W, range, c = 1490)
```

offset_correction_param *Generate input parameters for clock offset correction.*

Description

Generate input parameters for clock offset correction.

Usage

```
offset_correction_param(lat_lst, remove_outliers = T, phi, beta)
```

permute_clock_drift *Estimate initial drift values*

Description

These functions are used to provide initial drift values to KaltoaClockSync objects. Make sure to preview the returned values before using them.

Usage

```
drift <- permute_clock_drift(
sync_model, c = 1500, quiet = F, init_drift,
clock_resolution = 0.1, clock_window = 60*60*2,
max_drift = 60*5,
sigma = clock_resolution*5,
sigma_det = 0.1, phi = 1000/1500, beta = 36,
remove_outliers = T, fit_missing = T))

## Show measured drift values
print(drift)

## Plot inital drift values
plot(x = drift, rate = F)

## Return matrix of drift values
clock_drift(object = drift)
```

Arguments

sync_model	A KaltoaClockSync object.
quiet	When FALSE, a progress bar will be displayed.
init_drift	An optional matrix of initial drift values to center the clock_window around.

`clock_resolution`
: The time resolution to measure the clock drift. Smaller values will increase processing times.

`clock_window`
: Width of the clock window in seconds.

`max_drift`
: Clock drift values will be tested within the range of `init_drift` +/- `max_drift`.

`sigma`
: The standard deviation of the Gaussian kernel (seconds). Make sure this is larger than `clock_resolution`.

`sigma_det`, `phi`, `beta`
: Parameters for the detection error distribution, (see ddetect). These parameters are used when `method = "likelihood"`.

`phi`
: The standard deviation of the Gaussian kernel (seconds). Make sure this is larger than `clock_resolution`.

`fit_missing`
: When TRUE, linear regression will be used to interpolate/extrapolate `NaN` clock drift values.

`rate`
: When `TRUE`, drift rate is plotted.

`c`
: The speed of tag transmission in m/s.

Details

Clock drift values are estimated for each clock time and receiver pair combination with a brute-force permutation algorithm. Hence, finer clock resolutions will scale linearly with processing times.

The permutation approach consists of extracting the sync-tag emissions and detections between a receiver pair around a particular clock time (spanning a period of `clock_window`). The emissions are arranged into a discrete time-series and detections are arrange into a second time-series. Both time-series have a resolution of `clock_resolution`.

Next, all possible drift values are permuted over, providing time offsets to the detections time-series. The drift value which results in the emissions and detections time-series being most closely aligned in time is selected for. This is repeated for each clock time and receiver pair combination

There are two methods for evaluating how well aligned the emissions and detections time series are.

- `method = "xcorr"` will mark each emission and detection event as a Gaussian kernel in their respective time-series. Cross-correlation will be applied and the drift value resulting in the largest correlation score will be selected.
- `method = "likelihood"` will draw a mixed detection error distribution around each emission event in the time-series. Detection events will then be sampled from here and the likelihood that any detection event originated from any emission event will be calculated. The drift value providing the maximum lilelihood will be selected

`"xcorr"` is preferred when emission events may be missing from the dataset while `"likelihood"` is preferred when many reflected detections are expected to be present. Each method utilized convolution in the Fourier domain and gives comparable run-times.

Outlier detection will fit a robust linear regression (MASS::rlm) to a receiver pairs' drift values. Those that fall outside the 1-99th percentile of the fitted residual distribution will be marked as `NaN` values and replaced with linear predictions.

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
```

```
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')
```

```
# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
```

```
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)

# Delete tmp directory
unlink(tmp_path)
```

`plot.KaltoaClockSync` *Plot KaltoaClockSync object*

Description

Clock corrections will be applied to the DetectionsList which will be passed onto SyncTagLatency and plotted. This can be used to quickly verify receiver clock and position corrections.

Usage

```
## S3 method for class 'KaltoaClockSync'
plot(
  x,
  clock_drift = T,
  clock_offsets = T,
  position_offsets = T,
  only_sync_order = F,
  transmission_speed,
  ...
)
```

Arguments

`x`	A KaltoaClockSync object.
`clock_drift`	When `T`, clock drift corrections will be applied to the plotted sync-tag data.
`clock_offsets`	When `T`, clock offset corrections will be applied to the plotted sync-tag data.
`position_offsets`	When `T` receiver position corrections will be applied to the plotted sync-tag data.
`only_sync_order`	When `T`, only latency values specified in the `sync_order` table will be plotted. This is useful for checking how well your clock drift correction models worked.
`...`	Additional arguments will be passed to plot.SyncTagLatency.

See Also

Other Kaltoa-clock-sync: KaltoaClockSync, KaltoaClockSync-slots, fit_clock_drift(), fit_clock_offsets(), fit_receiver_positions()

---

plot.KaltoaPMCPositions *Plot PMC-TOA results*

Description

For single positions, all particles of an iteration will be plotted. This is a useful visual diagnostic to verify that the PMC-TOA algorithm is converging correctly. For multiple positions, tracks with confidence ellipses will be plotted.

Usage

```
## S3 method for class 'KaltoaPMCPositions'
plot(x, iter, i, xlim, ylim, add = F, ...)
```

Arguments

x	A KaltoaPMCPositions object.
iter	The iteration to plot.
i	The particle index.
xlim, ylim, add, ...	parameters passed to plot.

See Also

Other KaltoaPMCPositions: KaltoaPMCPositions

Examples

```
# We'll use a population Monte Carlo (PMC) algorithm to estimate the (1)
# location of the tag in our array and (2) the probability that a given
# detection is an outlier resulting from a reflected transmission path.

# Load detections and hydrophone data
# We'll extract PPM and HR detections, then combine them
# into a single TOAMatrix
toa <- c(
  get_toa(exdata_positioning, tag_id = c("HR-53883"),
          group_thresh = 0.5, transmission_speed = 1420),
  get_toa(exdata_positioning, tag_id = c("PPM-53883"),
    group_thresh = 0.5, transmission_speed = 1420))

# Create observation model (receiver detection error)
# This will capture both direct-path detections and reflected detections,
```

```
# the latter appearing as large positive outliers
obsv_model <- KaltoaObservation(
  type = "mixed",  max_transmission_latency = 500/1500,
  sigma = 0.002,# (m/s) Stationary velocity
  beta = 32)# Shape parameter
# Preview distribution
plot(obsv_model)
plot(obsv_model, xlim = c(-0.01, 0.03), ylim = c(0, 10))

# ------ Position a single emission
# Use automatic initial positions
# (particles will be sampled from a convex polygon around the array)
model_points <- KaltoaPointPositioning(
  toa = toa[103,],# Time-of-arrival for single emission
  observation_model = obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# --- Specify initial positions within 25m of any receiver
require(sf)
init_pos <- st_multipoint(positions(toa)[,-3]) |>
  st_buffer(dist = 25) |>
  st_sample(type = "hexagonal", size = 200) |>
  st_coordinates()
plot(init_pos, asp = 1, col = 'grey')
points(positions(toa), pch = 17, col = "red")
# Apply model
model_points <- KaltoaPointPositioning(
  toa = toa[103,], init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 4)
# Show results
model_points
plot(model_points)

# --- View particle convergence
plot(model_points, iter = 1)
plot(model_points, iter = 2)
plot(model_points, iter = 3)

# ------ Position all emissions
model_points <- KaltoaPointPositioning(
  toa = subset(toa, detections >= 4),
  init_positions = init_pos,
  observation_model =  obsv_model,
  n_particles = 200, iter = 3)
```

```
# Show results
model_points
plot(model_points)

# ------ Apply Post-processing
# Get movement parameters
t_05 = 15; sigma_vel = 0.2
beta = -log(0.05) / t_05
alpha = sigma_vel * sqrt( 2 * beta )
# Initialize process model
proc_model <- KaltoaProcess(type = "ctcrw",
  alpha = alpha, beta = beta)
# Plot process model
plot(proc_model)

# --- Apply Kalman filter
model_track <- KaltoaTrackPositioning(
  model_points, process_model = proc_model) |>
  smooth()
plot(model_track)
points(positions(toa), pch = 17, col = 'red')

# --- Predict positions along a discrete time-series
times <- emission_times(model_track)
times_out <- seq(floor(times[1]), ceiling(times[length(times)]), by = 10)
track_interp <- predict(object = model_track, times_out = times_out)
# Plot results
plot(track_interp)
points(positions(model_track), col = "blue", pch = 4)
points(positions(toa), pch = 17, col = "red")

# --- Export to csv
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(track_interp, file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

`plot.Latency` *Plot latency values.*

Description

Plot latency values.

Usage

```
## S3 method for class 'Latency'
plot(lat, delta = F, ...)
```

Arguments

lat	A Latency object.
delta	When TRUE, latency differences are plotted instead (normalized by hour periods). This is useful for identifying reflected detections.
...	Additional parameters are passed to plot(...)

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Latency: Latency(), fit_transmission_time()

---

plot.SyncTagLatency *Plot sync-tag latency for entire array*

Description

Plot sync-tag latency for entire array

Usage

```
plot(x, type = "latency", hist = F, transmission_speed, ...)
```

Arguments

x	A SyncTagLatency object.
type	A partial match to "latency" or "error", selecting the type of plot.
hist	When TRUE, return a histogram instead of a line plot.
...	Extra arguments to pass to xyplot(...) or histogram(...).
transmission_speed	Required for error plots..

See Also

See Kaltoa-overview to quickly find the functions you need.

Other SyncTagLatency: SyncTagLatency()

plot.TOAMatrix *Plot diagnostics for time-of-arrival values*

Description

type = "histogram" shows binned detections over time for each receiver while type = "diagnostic" shows the the emission period (last minus first detection) and number of detections per emission event.

Usage

```
## S3 method for class 'TOAMatrix'
plot(x, type = "hist", ...)
```

Arguments

x	TOAMatrix object.
type	A character value partially matching either 'histogram' or 'diagnostic'.
...	Remaining arguments passed to the respective plotting functions.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other TOAMatrix: TOAMatrix, TOAMatrix-generics, get_toa()

remove_double_detections *Remove double detections*

Description

For multiple detections from the same tag arriving within a short span, remove_double_detections will keep only the first recorded detection. This is a simple filter for quickly removing obvious multi-path reflections.

Usage

```
remove_double_detections(det, dd_thresh = 0.05)
```

Arguments

det	A Detections object.
dd_thresh	Threshold in seconds used to filter out double detections. A detection that occurs within this many seconds of a previous detection is removed.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other Detections: `Detections()`, `Detections-generics`

`subset.DetectionsList` *Subset a DetectionsList object*

Description

The subset expression is passed onto the function subset.Detections for each item in the list.

Usage

```
## S3 method for class 'DetectionsList'
subset(x, ...)
```

Arguments

`x`	A DetectionsList object.
`...`	Arguments passed to subset.Detections.

See Also

Other DetectionsList: `DetectionsList`, `detlst_apply()`

Examples

```
# Extract detections objects
det_1 <- exdata_positioning[[1]]
det_2 <- exdata_positioning[[2]]
det_3 <- exdata_positioning[[3]]
det_4 <- exdata_positioning[[4]]
det_5 <- exdata_positioning[[5]]
det_6 <- exdata_positioning[[6]]

# Remake DetectionsList
detlst <- DetectionsList(det_1, det_2, det_3, det_4, det_5, det_6)
detlst

# Convert a 'list of Detections' into a DetectionsList object
ldet <- list(det_1, det_2, det_3, det_4, det_5, det_6)
detlst <- as.DetectionsList(ldet)
detlst

# Remove double detections from array
detlst <- detlst_apply(detlst, remove_double_detections, dd_thresh = 0.05)

# ------ Viewing data
```

```
# Show summary of array detections data
summary(detlst)
# Plot map of array
plot(detlst)
# Return a matrix of array positions
positions(detlst)

# ------ Diagnostics
# Count sync-tag detections across the array
sync_conn <- SyncTagConnectivity(detlst)
# Plot a conenctivity map
plot(sync_conn, type = "map")
# Plot connectivity matrices
plot(sync_conn, type = "matrix")

# ------ Export DetectionsList as tar archived csv files
require(archive)
# Create temp directory
tmp_dir <- tempfile()
dir.create(tmp_dir)
# Write array attributes to csv
write.csv(
  file = file.path(tmp_dir, "attributes.csv"),
  x = attribute_table(exdata_positioning), row.names = FALSE)
# Write detections tables to csv files
for(det in exdata_positioning){
  fname <- sprintf("Detections-%s.csv", attr(det, "name"))
  write.csv(
    file = file.path(tmp_dir, fname),
    x = det, row.names = F)
}
# Archive directory
f_archv <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archv, dir = tmp_dir, full.names = FALSE)
# Delete temp folder
unlink(tmp_dir, recursive = T)
```

`SyncTagConnectivity.DetectionsList`

Sync-tag diagnostics for DetectionsList objects

Description

Sync-tag detections will be counted between unique receiver pairs and a diagnostic object will be returned that can be plotted. Detections here refers to the number of sync-tag emissions from one receiver that are detected on another. Connectivity is the sum of those detections in both directions between a pair of receivers. Examining detection/connectivity plots is helpful when trying to determine an optimal `sync_order` data.frame for KaltoaClockSync objects.

Usage

```
## Plot map of sync-tag connectivity per receiver
plot(sync_conn, type = "map", labels = T, name)

## Plot matrix of sync-tag connectivity between receiver pairs
plot(sync_conn, type = "matrix", sync_order, labels = T, name)

## Print counts of sync-tag detections
summary(sync_conn)

## Create sync-tag connectivity matrix
sync_conn = SyncTagConnectivity(object)
```

Arguments

object	A DetectionsList object.
type	Character specifying "map" or "matrix".
labels	When TRUE, receiver names will be plotted.
connectivity	When TRUE, connectivity will be plotted for matrix plots.
sync_order	A sync_order object (see sync_pair). When provided, the sync_order will be annotated on "matrix" type plots.
name	An optional character string. When given, connectivity scores will be calculated only from transmissions involving these receivers.

Details

For "map" plots, receivers will be color coded according to their connectivity summed across the entire array. The connectivity score for a single receiver is the number of times it has detected another sync-tag plus the number of times its own sync tag has been detected by other receivers.

For "matrix" plots, either detections or connectivity can be plotted. Detections are the number of times one receiver was detected be another, while connectivity is the total number of times both receivers were detected by each other.

See Also

See Kaltoa-overview to quickly find the functions you need.

Examples

```
# This example will walk through how to use
# sync-tags to synchronize the receiver clocks
# in an array.

# ------ Initialize a ClockSync object
# Plot receiver positions.
plot(exdata_clocksync)
# Show receiver connectivity
# (number of sync-tag detections between each receiver pair)
```

```
conn <- SyncTagConnectivity(exdata_clocksync)
plot(conn, type = "map")
plot(conn, type = "matrix")

# Create a data.frame holding the order
#  of paired clock synchronizations
# We'll use the center receiver as our reference clock.
sync_ordr = sync_pair(
  controller = "3",
  target =  c("1", "2", "4", "5", "6"))
sync_ordr

# Make a vector of times to calculate clock
# drift corrections for.  Hourly drift corrections
# are a good starting point.
clk_times <- seq(
  as.POSIXct("2021-05-08 18:00:00", tz = "CET"),
  as.POSIXct("2021-05-09 06:00:00", tz = "CET"), by = 3600)
# Create the model
sync_model <- KaltoaClockSync(
  sync_order = sync_ordr, # Order of pair-wise drift calibrations
  detlst = exdata_clocksync,# Raw sync-tag data
  clock_times = clk_times, # Times to calculate drift corrections at
  phi = 500/1500, # Largest possible (positive) detection error
  grouping_thresh = 0.5, # threshold to group sync-tag emissions to detections
  transmission_speed = 1480,# Speed of transmission
  max_drift = 60) # largest possible clock drift
# Show sync_order
sync_order(sync_model)
# Plot sync-order connectivity
plot(conn, type = "matrix", sync_order = sync_ordr)

# Plot alternative sync_order (multiple control receivers)
sync_ordr_alt = sync_pair(
  controller = "1", target =  "3") |>
  fill_sync_order(detlst = exdata_clocksync)
sync_ordr_alt
plot(conn, type = "matrix", sync_order = sync_ordr_alt)

# ------ Get initial clock drift values
# This is a simple, low-resolution measure
#  of clock drift
# (via cross correlation)
drft_init_1 <- permute_clock_drift(
  sync_model = sync_model,
  method = "xcor",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_1)
# Plot pairwise-drift over time
plot(drft_init_1)
# Show drift rate over time
```

```
plot(drft_init_1, rate = T)

# (via detection likelihood)
drft_init_2 <- permute_clock_drift(
  sync_model = sync_model,
  method = "li",
  clock_resolution = 0.1,
  clock_window = 3600*1)
# Show drift values and correlation scores
print(drft_init_2)
# Plot pairwise-drift over time
plot(drft_init_2)

# Plot sync-tag latency with initial drift values
clock_drift(sync_model) <- drft_init_2
plot(sync_model)

# ------ Apply estimate hourly clock drift
# (mixed model in TMB)
model_drift <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,# (s) Standard deviation of direct detection error
  sigma_drift = 0.1)# (s/h) Standard deviation of hourly clock drift
model_drift
plot(model_drift)
# Show drift values
# Plot the updated sync-tag latency plots
clock_drift(sync_model) <- model_drift
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# --- Fit with fixed clock drift error
model_drift_2 <- fit_clock_drift(
  sync_model,
  sigma_det = 0.1,
  sigma_drift = 0.01, fix_sigma_drift = T,
  initial_clock_drift = clock_drift(drft_init_2))
model_drift_2
clock_drift(sync_model) <- model_drift_2
plot(sync_model, type = 'error')
plot(sync_model, type = 'latency')

# ------ Estimate clock offsets
# (Weighted least squares)
# This will align the transmission times with the
# distances between receiver pairs.
clock_drift(sync_model) <- model_drift
model_clock_offsets <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
model_clock_offsets
clock_offsets(sync_model) <- model_clock_offsets
# Plot the updated sync-tag latency plots
```

```
plot(sync_model, type = 'error')

# ------ Estimate true receiver positions
model_position_offsets <- fit_receiver_positions(
  clocksync = sync_model,
  estimate_transmission_speed = T, estimate_gps_error = F,
  ref_receivers = c("2", "5"))
# Plot newly fitted receiver positions
plot(model_position_offsets)
model_position_offsets
# Plot the updated sync-tag latency plots
position_offsets(sync_model) <- model_position_offsets
plot(sync_model, type = 'error')

# ------ Refit offsets with new receiver positions
clock_offsets(sync_model) <- fit_clock_offsets(
  clocksync = sync_model,
  estimate_transmission_speed = T)
plot(sync_model, type = 'error')

# Apply receiver clock and position corrections to DetectionsList
detlst_corr <- synchronize(exdata_clocksync, sync_model)
plot(SyncTagLatency(detlst_corr, group_thresh = 0.5),
  transmission_speed = 1420, type = "error")

# ------  Save KaltoaClockSync to csv tables
require(archive)
# Create temp directory
tmp_path <- tempfile()
dir.create(tmp_path)

# write clock times
writeLines(con = file.path(tmp_path, "clock_times_UTC.txt"),
  paste(strftime(fmt = "%Y-%m-%d %H:%M:%S", tz = "UTC",
    clock_times(sync_model)), collapse = ", "))
# Write clock offsets
writeLines(con = file.path(tmp_path, "clock_offsets.txt"),
  paste(sprintf(fmt = "%.5f",
    clock_offsets(sync_model)), collapse = ", "))
# Write sync_order
write.csv(file = file.path(tmp_path, "sync_order.csv"),
  sync_order(sync_model), row.names = F)
# Write drift values
write.csv(file = file.path(tmp_path, "clock_drift.csv"),
  clock_drift(sync_model))
# Write position offsets
write.csv(file = file.path(tmp_path, "position_offests.csv"),
  position_offsets(sync_model))

# Archive directory
f_archive <- tempfile(fileext = ".tar.gz")
archive_write_dir(archive = f_archive, dir = tmp_path)
```

```
# Delete tmp directory
unlink(tmp_path)
```

SyncTagLatency *Calculate sync-tag transmission latency within a receiver array.*

Description

This is a useful diagnostic tool for identifying which receiver pairs have reflected transmissions and validating clock drift corrections. Latency objects will be created for every receiver pair and the output can be visualized with the `plot()` function.

Usage

```
SyncTagLatency(detlst, group_thresh = 1)
```

Arguments

`detlst`	A list holding `detections` objects.
`group_thresh`	Threshold in seconds for grouping sync-tag detection-emission pairs.
`...`	Additional arguments passed to Latency.

Value

A data.frame holding the transmission

See Also

See Kaltoa-overview to quickly find the functions you need.

Other SyncTagLatency: `plot.SyncTagLatency()`

Examples

```
# ------ Show sync-tag latency diagnostics
synctag_lat <- SyncTagLatency(exdata_positioning, group_thresh = 0.1)
# Show diagnostics as line plots
plot(synctag_lat)
plot(synctag_lat, transmission_speed = 1400)
plot(synctag_lat, type = 'error', transmission_speed = 1420)
# Show diagnostics as histograms
plot(synctag_lat, hist = T, nint = 50, transmission_speed = 1420)
plot(synctag_lat, type = 'error', hist = T, nint = 50, transmission_speed = 1420)

# ------ Advanced plotting options
# Plot first 9 panels only, restrict histograms to +/- 0.002 error
plot(synctag_lat[1:9], type = 'error', transmission_speed = 1420,
  hist = T, subset = abs(error) < 0.002)
# Plot panels in 3x3 across multiple pages
```

```
# restrict ylim to -0.002 to 0.004 error
plot(synctag_lat, type = 'error', transmission_speed = 1420,
  layout = c(3,3), ylim = c(-0.002, 0.004))
# Restrict latency values
plot(synctag_lat, type = 'latency', transmission_speed = 1420,
  subset = latency > -0.001 & latency < 0.05)
# see ?xyplot for more details on available plotting arguments.
```

`TdoaPositioning` *Time-difference-of-arrival positioning*

Description

Estimate positions from a time-of-arrival matrix.

Usage

```
TdoaPositioning(toa, remove_outliers = TRUE, sigma_det = 0.001)
```

Arguments

`toa`	A TOAMatrix of detection times..
`remove_outliers`	When TRUE, large outliers will be set to NA.
`sigma_det`	The standard deviation of the Gaussian detection error. This is used for calculating positioning error.
`c`	Speed of signal transmission.

Details

This function uses Smith and Abel's (1987) closed form solutions for TDOA positioning (spherical interpolation) and absolute positioning error is calculated according to the methods described in Campbell et al. (*in preparation*) and further examined in Campbell et al. (2005).

Note that the returned emission times are simply the first detection time for each emission event.

Value

A KaltoaPointPositions object holding the results.

References

- Smith, J, and J Abel. (1987) **Closed-Form Least-Squares Source Location Estimation from Range-Difference Measurements**. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(12):1661–69.
- Campbell, J. A., Lundberg, P., Hölker, F. (*in preparation*) **Absolute measures of time-difference-of-arrival positioning error in underwater acoustic telemetry setups**.
- Campbell, J. A., Shry, S. J., Lundberg, P., Calles, O., Hölker, F. (2005) **A population Monte Carlo model for underwater acoustic telemetry positioning in reflective environments**. *Methods in Ecology and Evolution*, 16:775-785.

Examples

```
# Get TOAMatrix
# We'll only use PPM tags to avoid reflections
toa <- get_toa(exdata_positioning, tag_id = c("PPM-53883"),
  group_thresh = 0.5)

# We'll assume all receivers have a
# detection error standard deviation of 2ms.
# This will be used to calculate positioning
# uncertainty
sigma_det <- 0.002

# Solve and plot positions
tdoa_pos <- TdoaPositioning(
  toa = toa,          # Time-of-arrival matrix
  remove_outliers = T,# Large detection outliers will be automatically removed
  sigma_det = sigma_det)  # Detection error SD
# Plot results
plot(tdoa_pos)
points(positions(toa), pch = 17, col = 'red')

# ------ Apply Post-processing
# Get movement parameters
t_05 = 15; sigma_vel = 0.2
beta = -log(0.05) / t_05
alpha = sigma_vel * sqrt( 2 * beta )
# Initialize process model
proc_model <- KaltoaProcess(type = "ctcrw",
  alpha = alpha, beta = beta)
# Plot process model
plot(proc_model)

# --- Apply Kalman filter & smoother
tdoa_smooth <- KaltoaTrackPositioning(
  x = tdoa_pos, process_model = proc_model) |>
  smooth()
plot(tdoa_smooth)
points(positions(toa), pch = 17, col = 'red')

# --- Export to csv
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(as.KaltoaPointPositions(tdoa_smooth), file = fh, row.names = F)
close(fh)

# --- Import from csv
fh <- gzfile(fname, open = "r")
df_points <- read.csv(file = fh)
close(fh)
points_import <- as.KaltoaPointPositions(df_points)
```

toa_h *Time-of-Arrival Observation Model*

Description

Functions for calculating the travel time between the tag and receivers, `toa_h`, and the respective Jacobian `toa_H`. The Jacobian can be used as the observation model for time-of-arrival positioning in a extended Kalman filter.

Usage

```
toa_h(pos, hydro, c = 1500)

toa_H(hydro, pos, c = 1500)
```

Arguments

`pos`	Vector or Matrix holding the coordinates of the emitting tag(s).
`hydro`	Matrix holding the coordinates of the receiving hydrophones
`c`	Propagation speed of sound in water.

TOAMatrix *Time-of-arrival matrix*

Description

This objects holds the time-of-arrival data for array detections along with useful metadata. get_toa can be used to automatically generate `TOAMatrix` objects from DetectionsList objects.

See TOAMatrix-generics for generic methods and attribute access functions.

Usage

```
# Create TOAMatrix object
toa <- TOAMatrix(data, start_time = 0, tag_id = NA,
    receiver_positions, receiver_depths, transmission_speed = 1500)

# Create TOAMatrix diagnostics object
toa_diag <- diagnostics(toa)

## Convert TOAMatrix to data.frame
df_toa = as.data.frame(x = toa)

## Convert data.frame to TOAMatrix
as.TOAMatrix(x = df_toa, tag_id, receiver_positions, transmission_speed)
```

Arguments

`data`	A matrix holding relative time-of-arrival values in seconds. Column names should hold receiver ids and each row should indicate a unique tag emission.
`start_time`	A POSIXct datetime that matches 0 seconds in the `data` matrix. Example: '2020-03-01 12:03:22'
`tag_id`	String indicating the ID of the tag in the TOA matrix. If the matrix holds data from multiple tags, the convention is to set this to 'Multiple tags'.
`receiver_positions`	A matrix holding the (x, y) coordinates of each array receiver.
`receiver_depths`	A vector holding the depths of each array receiver.
`transmission_speed`	The speed of tag transmission in meters per second.

Details

The `TOAMatrix` stores relative time-of arrival data. The absolute times are calculated by summing the relative times with the provided `start_time`. Note that `start_time` is automatically rounded to the nearest second.

Value

A TOAMatrix object.

See Also

See Kaltoa-overview to quickly find the functions you need.

Other TOAMatrix: `TOAMatrix-generics`, `get_toa()`, `plot.TOAMatrix()`

Examples

```
# ------ Make time-of-arrival matrix from DetectionsList
toa <- get_toa(exdata_positioning,
  tag_id = "HR-53883",
  group_thresh = 0.5)

# ------ Summary info
summary(toa)
head(toa)

# Plot diagnostics
plot(toa, type = "hist")
plot(toa, type = "diag")

# Subset by diagnostic data
toa_sub_1 <- subset(toa, detections >= 4 & period <= 0.05)
head(toa_sub_1)
plot(toa_sub_1, type = "diag")
```

```
# Subset receivers
toa_sub_2 <- subset(toa, select = c("1", "2", "3", "4"))
head(toa_sub_2)

# ------ Export as csv file
fname <- tempfile()
fh <- gzfile(fname, open = "w")
write.csv(x = toa, file = fh, row.names = F)
close(fh)

# ------ Import from csv
fh <- gzfile(fname, open = "r")
df_toa <- read.csv(file = fh, check.names = F)
det_import <- as.TOAMatrix(
  df_toa, tag_id = "HR-53883", receiver_positions = positions(toa),
  transmission_speed = 1420)
close(fh)
```

TOAMatrix-generics *TOAMatrix generic functions*

Description

Generic methods and attribute access functions for TOAMatrix objects.

Usage

```
# Convert to a matrix object
as.matrix(toa_mat)

# Print first n rows
head(toa_mat, n)

# Print last n rows
tail(toa_mat, n)

# Index TOAMatrix rows
toa_mat[i]

# Index TOAMatrix rows & coklumns
toa_mat[i, j]

# Get start time
start_time(toa_mat)

# Get transmission speed
transmission_speed(toa_mat)
```

```
# Set transmission speed
transmission_speed(toa_mat) <- x

# Combine and sort multiple TOAMatrix objects.
c(toa_mat, toa_mat_2)

# Print summary information
summary(toa_mat, receiver_positions = T)

# Return receiver positions
positions(toa_mat)

# Return receiver depths
depths(toa_mat)

# Subset TOAMatrix using diagnostics
subset(toa_mat, subset, select)

# Add values to TOAMatrix
append(toa_mat, values)
```

See Also

See Kaltoa-overview to quickly find the functions you need.

Other TOAMatrix: `TOAMatrix`, `get_toa()`, `plot.TOAMatrix()`

`weighted.cov` *Weighted covariance*

Description

Calculate the weighted covariance for a matrix.

Usage

```
weighted.cov(x, w)
```

Arguments

`x`	Matrix to calculate covariance for.
`w`	vector of weights.

Index