Package: glatos (via r-universe)

July 8, 2024

Type Package

Title A package for the Great Lakes Acoustic Telemetry Observation System

Description Functions useful to members of the Great Lakes Acoustic Telemetry Observation System https://glatos.glos.us; many more broadly relevant to simulating, processing, analysing, and visualizing acoustic telemetry data.

Version 0.7.3

Date 2024-04-09

Depends R (>= 3.5.0)

Imports av, data.table, fasterize, fasttime, gdalUtilities, geodist, gdistance, jsonlite, knitr, lubridate, magrittr, methods, plotrix, plyr, purrr, raster, readxl, rmarkdown, sf, tibble, tidyr

Suggests gganimate, gifsky, GISTools, png, tint

URL https://github.com/ocean-tracking-network/glatos

BugReports https://github.com/ocean-tracking-network/glatos/issues

License GPL-2

LazyLoad yes

LazyData true

RoxygenNote 7.3.1

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/ocean-tracking-network/glatos

RemoteRef HEAD

RemoteSha f455938a81dc095863f5394b99e3623a387878fe

Contents

`abacus_plot` *Plot detection locations of acoustic transmitters over time*

Description

Plot detection locations of acoustic transmitters over time.

Usage

```
abacus_plot(
  det,
  location_col = "glatos_array",
  locations = NULL,
  show_receiver_status = NULL,
  receiver_history = NULL,
  out_file = NULL,
  x_res = 5,
  x_format = "%Y-%m-%d",
  outFile = NULL,
  ...
)
```

Arguments

det	A glatos_detections object (e.g., produced by read_glatos_detections) containing detections to be plotted. *OR* A data frame containing detection data with at least two columns, one of which must be named 'detection_timestamp_utc', described below, and another column containing a location grouping variable, whose name is specified by location_col (see below). The following column must appear in det: detection_timestamp_utc Detection timestamps; MUST be of class POSIXct.
location_col	A character string indicating the column name in det that will be used as the location grouping variable (e.g. "glatos_array"), in quotes.
locations	An optional vector containing the locations location_col to show in the plot. Plot order corresponds to order in the vector (from bottom up). Should correspond to values in location_col, but can contain values that are not in the det data frame (i.e., can use this option to plot locations fish were not detected).
show_receiver_status	DEPCRECATED. No longer used. A logical value indicating whether or not to display receiver status behind detection data (i.e., indicate when receivers were in the water). If show_receiver_status == TRUE, then a receiver_history data frame (receiver_history) must be supplied. Default is FALSE.
receiver_history	An optional glatos_receivers object (e.g., produced by read_glatos_receivers) containing receiver history data for plotting receiver status behind the detection data when receiver_history is not NULL. *OR* An optional data frame containing receiver history data for plotting receiver status behind the detection data. The following column must be present: deploy_date_time Receiver deployment timestamps; MUST be of class POSIXct. recover_date_time Receiver recovery timestamps; MUST be of class POSIXct. **a grouping column whose name is specified by** location_col See above.
out_file	An optional character string with the name (including extension) of output image file to be created. File extension will determine type of file written. For example, "abacus_plot.png" will write a png file to the working directory. If NULL (default) then the plot will be printed to the default plot device. Supported extensions: png, jpeg, bmp, and tiff.
x_res	Resolution of x-axis major tick marks. If numeric (e.g., 5 (default value), then range of x-axis will be divided into that number of equally-spaced bins; and will be passed to length.out argument of seq.Date. If character, then value will be passed to by argument of seq.Date. In that case, a character string, containing one of "day", "week", "month", "quarter" or "year". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s". E.g., "10 days", "weeks", "4 weeks", etc. See seq.Date.
x_format	Format of the x-axis tick mark labels (major ticks only; minor ticks are not supported). Default is "%Y-%m-%d". Any valid strptime specification should work.

outFile	Deprecated. Use out_file instead.
...	Other plotting arguments that pass to plot, points (e.g., col, lwd, type). Use cex.main to set title character size, and col.main to set title color. If xlim is specified, it must be a two-element vector of POSIXct.

Details

NAs are not allowed in any of the two required columns.

The locations vector is used to control which locations will appear in the plot and in what order they will appear. If no locations vector is supplied, the function will plot only those locations that appear in the det data frame and the order of locations on the y-axis will be alphebetical from top to bottom.

By default, the function does not distinguish detections from different transmitters and will therefore plot all transmitters the same color. If more than one fish is desired in a single plot, a vector of colors must be passed to the function using the 'col =' argument. The color vector must be the same length as the number of rows in the detections data frame or the colors will be recycled.

Plotting options (i.e., line width and color) can be changed using optional graphical parameters http://www.statmethods.net/advgraphs/parameters.html that are passed to "points" (see ?points).

Value

An image to the default plot device or a file containing the image if out_file is specified.

Author(s)

T. R. Binder, edited by A. Dini

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
 package = "glatos")
det <- read_glatos_detections(det_file)

#subset one transmitter
det2 <- det[det$animal_id == 153, ]

#plot without control table and main tile and change color to red
abacus_plot(det2, locations=NULL,
  main = "TagID: 32054", col = "red")

#example with locations specified
abacus_plot(det2, locations=c("DRF", "DRL", "FMP", "MAU", "PRS", "RAR",
   "DRM", "FDT"), main = "TagID: 32054", col = "red")

#plot with custom y-axis label and lines connecting symbols
abacus_plot(det2, main = "TagID: 32054", type = "o", pch = 20,  col = "red")

#plot with custom x-axis resolution - 10 bins
```

```
abacus_plot(det2, main = "TagID: 32054", x_res = 10)

#plot with custom x-axis resolution - monthly bins
abacus_plot(det2, main = "TagID: 32054", x_res = "month")

#plot with custom x-axis resolution - 8-week bins
abacus_plot(det2, main = "TagID: 32054", x_res = "8 weeks")

#plot with custom x-axis format
abacus_plot(det2, main = "TagID: 32054", x_res = "months", x_format = "%b-%y")

#plot with custom x axis limits
xLim <- as.POSIXct(c("2012-01-01", "2014-01-01"), tz = "UTC")
abacus_plot(det2, main = "TagID: 32054", xlim = xLim)

#example with receiver locations
# get example receiver location data
rec_file <- system.file("extdata", "sample_receivers2.csv",
 package = "glatos")
rec <- read_glatos_receivers(rec_file)

abacus_plot(det2, locations=c("DRF", "DRL", "FMP", "MAU", "PRS", "RAR",
 "DRM", "FDT"), receiver_history = rec,
 main = "TagID: 32054", col = "red")

#example with grey box plotted in background (using panel.first)

#set time range covered by rectangle
rect_x_rng <- as.POSIXct(c("2012-07-31", "2013-04-15"), tz = "UTC")
#get number of unique locations (y-axis)
n_locs <- length(unique(det2$glatos_array))

#plot as grey box in background
abacus_plot(det2, locations=NULL,
 main = "TagID: 32054", col = "red",
 panel.first = rect(rect_x_rng[1], 1, rect_x_rng[2], n_locs, col = "grey",
   border = NA))
```

`adjust_playback_time` *Modify playback time of video*

Description

Speed up or slow down playback of video

Usage

```
adjust_playback_time(
  scale_factor = 1,
```

```
  input,
  output_dir = getwd(),
  output = "new.mp4",
  overwrite = FALSE,
  diagnostic_mode = FALSE
)
```

Arguments

scale_factor	multiplicative factor changes duration of video playback. See details.
input	character, path to video file (any file type supported by av::av_encode_video; e.g., *.mp4, *.wmv, etc)
output_dir	character, output directory, default is working directory
output	character, output file name
overwrite	logical, default is overwrite = TRUE
diagnostic_mode	Logical (default = FALSE). If true, returns FFMPEG output.

Details

A simple wrapper for av::av_encode_video.

adjust_playback_time adjusts playback speed of video. scale_factor controls the magnitude of speed-up or slow-down by modifying the presentation timestamp of each video frame. Values of scale_factor < 1 speed up playback and > 1 slow down playback. In addition to changing playback, function can change output format by specifying a different file extension in output.

Value

One video animation will be written to output_dir and the path and name of output file with be returned.

Note

Input argument 'ffmpeg' was removed in glatos version 0.7.0.

Author(s)

Todd Hayden, Tom Binder, Chris Holbrook

Examples

```
## Not run:

# load example frames
frames <- system.file("extdata", "frames", package = "glatos")

# make video animation
out_file <- file.path(tempdir(), "animation_av.mp4")
make_video(input_dir = frames,
```

```
            input_ext = ".png",
            output = out_file)

  # slow video down by a factor of 10
  path <- file.path(tempdir(), "animation_av.mp4")
  adjust_playback_time(scale_factor = 10,
                       input = path,
                       output_dir = tempdir(),
                       output = "animation_av_slow.mp4",
                       diagnostic_mode = FALSE,
                       overwrite = TRUE)

  # slow video down by a factor of 10 and change format of output video
  adjust_playback_time(scale_factor = 10,
                       input = path,
                       output_dir = tempdir(),
                       output = "animation_av_slow.wmv",
                       diagnostic_mode = FALSE,
                       overwrite = TRUE)

  # speed up video
  adjust_playback_time(scale_factor = 0.5,
                       input = path,
                       output_dir = tempdir(),
                       output = "animation_av_fast.mp4",
                       diagnostic_mode = FALSE,
                       overwrite = TRUE)

  ## End(Not run)
```

`aggregate_total_no_overlap` *The function below aggregates timedelta of first_detection and last_detection, excluding overlap between detections. Any overlap between two detections is converted to a new detection using the earlier first_detection and the latest last_detection. If the first_detection and last_detection are the same, a timedelta of one second is assumed.*

Description

The function below aggregates timedelta of first_detection and last_detection, excluding overlap between detections. Any overlap between two detections is converted to a new detection using the earlier first_detection and the latest last_detection. If the first_detection and last_detection are the same, a timedelta of one second is assumed.

Usage

```
aggregate_total_no_overlap(detections)
```

Arguments

detections	• data frame pulled from the compressed detections CSV

`aggregate_total_with_overlap` *The function below aggregates timedelta of first_detection and last_detection of each detection into a final timedelta then returns a float of the number of days. If the first_detection and last_detection are the same, a timedelta of one second is assumed.*

Description

The function below aggregates timedelta of first_detection and last_detection of each detection into a final timedelta then returns a float of the number of days. If the first_detection and last_detection are the same, a timedelta of one second is assumed.

Usage

```
aggregate_total_with_overlap(detections)
```

Arguments

detections	-data frame pulled from the compressed detections CSV

`calc_collision_prob` *Estimate probability of collision for telemetry transmitters*

Description

Estimate (by simulation) probability of collision for co-located telemetry transmitters with pulse-period-modulation type encoding

Usage

```
calc_collision_prob(
  delayRng = c(60, 180),
  burstDur = 5,
  maxTags = 50,
  nTrans = 10000
)
```

Arguments

delayRng	A 2-element numeric vector with minimum and maximum delay (time in seconds from end of one coded burst to beginning of next).
burstDur	A numeric scalar with duration (in seconds) of each coded burst (i.e., pulse train).
maxTags	A numeric scalar with maximum number of co-located transmitters (within detection range at same time).
nTrans	A numeric scalar with the number of transmissions to simulate for each co-located transmitter.

Details

Calculates the detection probability associated with collision, given delay range (delayRng), burst duration (burstDur), maximum number of co-located tags (maxTags), and number of simulated transmission per tag (nTrans). The simulation estimates detection probability due only to collisions (i.e., when no other variables influence detection probability) and assuming that all tags are co-located at a receiver for the duration of the simulation.

Value

A data frame containing summary statistics:

nTags	Number of tags within detection range at one time
min	Minimum detection probability among simulated tags
q1	First quartile of detection probabilities among simulated tags
median	Median detection probability among simulated tags
q3	Third quartile of detection probabilities among simulated tags
max	Maximum detection probability among simulated tags
mean	Mean detection probability among simulated tags
expDetsPerHr	Expected number of detections per hour assuming perfect detection probability, given the number of tags within detection range
totDetsPerHr	Observed number of detections per hour for a given number of tags
effDelay	Eeffective delay of the transmitter after incorporating collisions
detsPerTagPerHr	Mean number of detections per hour per tag

Author(s)

C. Holbrook (cholbrook@usgs.gov) and T. Binder

References

For application example, see:

Binder, T.R., Holbrook, C.M., Hayden, T.A. and Krueger, C.C., 2016. Spatial and temporal variation in positioning probability of acoustic telemetry arrays: fine-scale variability and complex interactions. Animal Biotelemetry, 4(1):1.
http://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-016-0097-4

Examples

```
#parameters analagous to Vemco tag, global coding, 45 s nominal delay
foo <- calc_collision_prob(delayRng = c(45, 90), burstDur = 5.12, maxTags = 50,
    nTrans = 10000)

#plot probabilities of detection
plot(med~nTags, data=foo, type='p', pch=20, ylim=c(0,1),
  b

#plot probability of collision by subtracting detection probability from 1
plot((1 - med)~nTags, data=foo, type='p', pch=20, ylim=c(0,1),
  xlab="# of transmitters within range", ylab="Probability of collision")
```

---

check_cross_boundary *Check if track crosses polygon boundary*

Description

Check if track crosses polygon boundary

Usage

```
check_cross_boundary(path, boundary, EPSG)
```

---

check_in_polygon *Check if in polygon*

Description

Check if in polygon

Usage

```
check_in_polygon(points, polygon, EPSG)
```

convert_glatos_to_att *Convert detections and receiver metadata to a format that ATT accepts.*

Description

Convert glatos_detections and glatos_receiver objects to ATT for compatibility with the Animal Tracking Toolbox (https://github.com/vinayudyawer/ATT).

Usage

```
convert_glatos_to_att(
  detectionObj,
  receiverObj,
  crs = sp::CRS("+init=epsg:4326")
)
```

Arguments

detectionObj	a list from read_glatos_detections
receiverObj	a list from read_glatos_receivers
crs	a sp::CRS object with geographic coordinate system for all spatial information (latitude/longitude). If none provided or crs is not recognized, defaults to WGS84.

Details

This function takes 2 lists containing detection and reciever data and transforms them into one list containing 3 tibble::tibble objects. The input that AAT uses to get this data product is located here: https://github.com/vinayudyawer/ATT/blob/master/README.md and our mappings are found here: https://gitlab.oceantrack.org/GreatLakes/glatos/issues/83 in a comment by Ryan Gosse.

Value

a list of 3 tibble::tibbles containing tag dectections, tag metadata, and station metadata, to be ingested by VTrack/ATT

Author(s)

Ryan Gosse

Examples

```
#-----------------------------------------------
# EXAMPLE #1 - loading from the vignette data

library(glatos)
wal_det_file <- system.file("extdata", "walleye_detections.csv",
     package = "glatos")
walleye_detections <- read_glatos_detections(wal_det_file) # load walleye data

rec_file <- system.file("extdata", "sample_receivers.csv",
     package = "glatos")
rcv <- read_glatos_receivers(rec_file) # load receiver data

ATTdata <- convert_glatos_to_att(walleye_detections, rcv)
```

`convert_otn_erddap_to_att` *Convert detections, transmitter, receiver, and animal metadata to a format that ATT accepts.*

Description

Convert `glatos_detections` and transmitter, receiver, and animal metadata from the OTN ERDDAP to `ATT` format for use in the Animal Tracking Toolbox (https://github.com/vinayudyawer/ATT).

Usage

```
convert_otn_erddap_to_att(
  detectionObj,
  erdTags,
  erdRcv,
  erdAni,
  crs = sf::st_crs(4326)
)
```

Arguments

`detectionObj`	a data frame from `read_glatos_detections`
`erdTags`	a data frame with tag release data from the OTN ERDDAP
`erdRcv`	a data frame with receiver station data from the OTN ERDDAP
`erdAni`	a data frame with animal data from the OTN ERDDAP
`crs`	an object of class `crs` (see sf::st_crs) with geographic coordinate system for all spatial information (latitude/longitude). If none provided or `crs` is not recognized, defaults to WGS84.

Details

This function takes 4 data frames containing detection, and ERDDAP data from the tags, receivers, and animals tables, and transforms them into 3 `tibble::tibble` objects inside of a list. The input that AAT uses to get this data product is located here: https://github.com/vinayudyawer/ATT/blob/master/README.md and our mappings are found here: https://github.com/ocean-tracking-network/glatos/issues/75 in a comment by Ryan Gosse. The OTN ERDDAP instance is here: https://members.oceantrack.org/erddap/tabledap/index.html but please note that this only contains public data.

Value

a list of 3 tibble::tibbles containing tag dectections, tag metadata, and station metadata, to be ingested by VTrack/ATT

Author(s)

Ryan Gosse

Examples

```
#-------------------------------------------------
# EXAMPLE #1 - loading from the OTN ERDDAP + vignettes

library(glatos)

#get path to example files from OTN ERDDAP
ani_erd_file <- system.file("extdata", "otn_aat_animals.csv",
                            package = "glatos")
animals <- read.csv(ani_erd_file) # load the CSVs from ERDDAP

tags_erd_file <- system.file("extdata", "otn_aat_tag_releases.csv",
                            package = "glatos")
tags <- read.csv(tags_erd_file)

rcv_erd_file <- system.file("extdata", "otn_aat_receivers.csv",
                            package = "glatos")
stations <- read.csv(rcv_erd_file)

#Remove first row; (blank or metadata about the column)
animals <- animals[-1,]
tags <- tags[-1,]
stations <- stations[-1,]

#get blue shark example data
shrk_det_file <- system.file("extdata", "blue_shark_detections.csv",
     package = "glatos")
blue_shark_detections <- read_otn_detections(shrk_det_file) # load shark data

ATTdata <- convert_otn_erddap_to_att(blue_shark_detections,
                                     tags, stations, animals)
```

`convert_otn_to_att` *Convert detections, tagging metadata, and deployment metadata to a format that ATT accepts.*

Description

Convert `glatos_detections`, OTN tagging metadata and OTN deployment metadata to `ATT` format for use in the Animal Tracking Toolbox (https://github.com/vinayudyawer/ATT).

Usage

```
convert_otn_to_att(
  detectionObj,
  taggingSheet,
  deploymentObj = NULL,
  deploymentSheet = NULL,
  timeFilter = TRUE,
  crs = sf::st_crs(3426)
)
```

Arguments

`detectionObj`	a data frame from `read_otn_detections`
`taggingSheet`	a data frame from `prepare_tag_sheet`
`deploymentObj`	a data frame from `read_otn_deployments`
`deploymentSheet`	a data frame from `prepare_deploy_sheet`
`timeFilter`	Whether the data should be filtered using the deployment and recovery/last download times of receivers. Defaults to TRUE, if not all receiver metadata is available, this should be set to FALSE otherwise there will be data loss.
`crs`	a object of class `crs` (see sf::st_crs with geographic coordinate system for all spatial information (latitude/longitude). If none provided or `crs` is not recognized, defaults to WGS84 (EPSG:4326).

Details

This function takes 3 data frames containing detections, tagging metadata, and deployment metadata from either `read_otn_deployments` or `prepare_deploy_sheet` and transforms them into 3 `tibble::tibble` objects inside of a list. The input that AAT uses to get this data product is located here: https://github.com/vinayudyawer/ATT/blob/master/README.md and our mappings are found here: https://github.com/ocean-tracking-network/glatos/issues/75 in a comment by Ryan Gosse.

Value

a list of 3 tibble::tibbles containing tag dectections, tag metadata, and station metadata, to be ingested by VTrack/ATT

Author(s)

Ryan Gosse

Examples

```
#-------------------------------------------------
# EXAMPLE #1 - loading from Deployment Object

library(glatos)

dets_path <- system.file("extdata", "blue_shark_detections.csv",
                         package = "glatos")
deploy_path <- system.file("extdata", "hfx_deployments.csv",
                           package = "glatos")
tag_path <- system.file("extdata", "otn_nsbs_tag_metadata.xls",
                        package = "glatos")

dets <- read_otn_detections(dets_path)
tags <- prepare_tag_sheet(tag_path, 5, 2)
deploy <- read_otn_deployments(deploy_path)

ATTdata <- convert_otn_to_att(dets, tags, deploymentObj = deploy)

#-------------------------------------------------
# EXAMPLE #2 - loading from Deployment Sheet

library(glatos)

dets_path <- system.file("extdata", "blue_shark_detections.csv",
                         package = "glatos")
deploy_path <- system.file("extdata", "hfx_deploy_simplified.xlsx",
                           package = "glatos")
tag_path <- system.file("extdata", "otn_nsbs_tag_metadata.xls",
                        package = "glatos")

dets <- read_otn_detections(dets_path)
tags <- prepare_tag_sheet(tag_path, 5, 2)
deploy <- prepare_deploy_sheet(deploy_path, 1, 1)

ATTdata <- convert_otn_to_att(dets, tags, deploymentSheet = deploy)
```

---

crw *Simulate a correlated random walk*

Description

Simulate a random walk as series of equal-length steps with turning angles drawn from a normal distribution.

Usage

```
crw(
  theta = c(0, 5),
  stepLen = 10,
  initPos = c(0, 0),
  initHeading = 0,
  nsteps = 10000
)
```

Arguments

theta	A 2-element numeric vector with turn angle parameters (theta1 = mean; theta2 = sd) from normal distribution.
stepLen	A numeric scalar with total distance moved in each step.
initPos	A 2-element numeric vector with nital position (initPos1=x, initPos2=y).
initHeading	A numeric scalar with initial heading in degrees.
nsteps	A numeric scalar with number of steps to simulate.

Details

First, nsteps turn angles are drawn from a normal distribution. Second, the cumulative sum of the vector of turn angles defines the heading within each step. The x and y component vectors in each are then calculated and summed to obtain the simualted path.

Value

A two-column data frame containing:

x	x coordinates
y	y coordinates

Note

Adapted from code provided by Tom Binder.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
foo <- crw(theta=c(0,5), stepLen=10, initPos=c(0,0), initHeading=0,
  nsteps=10)
plot(foo,type="o",pch=20,asp=c(1,1))
```

crw_in_polygon *Simulate a correlated random walk inside a polygon*

Description

Uses crw to simulate a random walk as series of equal-length steps with turning angles drawn from a normal distribution inside a polygon.

Usage

```
crw_in_polygon(
  polyg,
  theta = c(0, 10),
  stepLen = 100,
  initPos = c(NA, NA),
  initHeading = NA,
  nsteps = 30,
  inputCRS = NA,
  cartesianCRS = NA,
  sp_out = TRUE,
  show_progress = TRUE
)
```

Arguments

polyg	A spatial polygon object of class sf or sfc containing POLYGON or MULTIPOLYGON features (but SpatialPolygonsDataFrame and SpatialPolygons are also accepted); *OR* A polygon defined as data frame or matrix with numeric columns x and y.
theta	A 2-element numeric vector with turn angle parameters (theta1 = mean; theta2 = sd), in degrees, from normal distribution.
stepLen	A numeric scalar with total distance moved in each step, in meters.
initPos	A 2-element numeric vector with initial position (initPos1=x, initPos2=y) in same coordinate reference system as polyg.
initHeading	A numeric scalar with initial heading in degrees. E.g., 0 = North; 90 = East, 180 = South, 270 = West; etc.
nsteps	A numeric scalar with number of steps to simulate.
inputCRS	A crs object or numeric EPSG code of coordinate system of input polyg. Only used if polyg does not contain a crs. If missing, then polyg is assumed in an arbitrary Cartesian (projected) system with base unit of one meter.
cartesianCRS	Coordinate reference system used for simulations. Must be a Cartesian (projected) coordinate system. Must be given when input CRS is non-Cartesian (e.g., long-lat); optional otherwise. See Note.

sp_out	Logical. If TRUE (default) then output is an sf object. If FALSE, then output is a data.frame.
show_progress	Logical. Progress bar and status messages will be shown if TRUE (default) and not shown if FALSE.

Details

If initPos = NA, then a starting point is randomly selected within the polygon boundary. A path is simulated forward using crw. Initial heading is also randomly selected if initHeading = NA. When a step crosses the polygon boundary, a new heading for that step is drawn and the turn angle standard deviation is enlarged slightly for each subsequent point that lands outside the polygon.

If input polyg object is a data.frame with x and y columns and sp_out = TRUE, then output object coordinate system is defined by inputCRS. Coordinate system on output will be same as input if polyg contains a valid CRS.

Value

When sp_out = TRUE, an sf object containing one POINT feature for each vertex in the simulated path.
OR
When sp_out = FALSE, a two-column data frame containing:

x	x coordinates
y	y coordinates

in the same units as polyg.

Note

The path is constructed in segments based on the minimum distance between the previous point and the closest polygon boundary.

Simulations are conducted within the coordinate system specified by argument cartesianCRS.

EPSG 3175 (cartesianCRS = 3175) is recommended projected coordinate system for the North American Great Lakes Basin and St. Lawrence River system. https://spatialreference.org/ref/epsg/nad83-great-lakes-and-st-lawrence-albers/.

Author(s)

C. Holbrook <cholbrook@usgs.gov>

See Also

crw, transmit_along_path, detect_transmissions

Examples

```
# Example 1 - data.frame input
mypolygon <- data.frame(x = c(-50,-50, 50, 50), y = c(-50,50,50,-50))

path_df <- crw_in_polygon(mypolygon, theta = c(0, 20), stepLen = 10,
  initPos=c(0,0), initHeading=0, nsteps=50, sp_out = FALSE)

class(path_df) #note object is data.frame

plot(path_df, type = "o", pch = 20, asp = c(1,1),
  xlim = range(mypolygon$x), ylim = range(mypolygon$y))

polygon(mypolygon, border = "red")


# Example 2 - data.frame input; input CRS specified
mypolygon <- data.frame(x = c(-84,-85, -85, -84),
                        y = c(45, 44, 45, 45))
path_df <- crw_in_polygon(mypolygon,
                          theta = c(0, 20),
                          stepLen = 1000,
                          initPos = c(-84.75, 44.75),
                          initHeading = 0,
                          nsteps = 50,
                          inputCRS = 4326,
                          cartesianCRS = 3175,
                          sp_out = FALSE)
plot(path_df, type = "o", pch = 20, asp = c(1,1),
  xlim = range(mypolygon$x), ylim = range(mypolygon$y))
class(path_df) #note object is data.frame
polygon(mypolygon, border = "red")


# Example 3 - sf POLYGON input
data(great_lakes_polygon)

#simulate in great lakes polygon
path_sf <- crw_in_polygon(great_lakes_polygon,
                          theta = c(0, 25),
                          stepLen = 10000,
                          initHeading = 0,
                          nsteps = 100,
                          cartesianCRS = 3175)

#plot
plot(sf::st_geometry(great_lakes_polygon),
                     col = "lightgrey",
                     border = "grey")
points(sf::st_coordinates(path_sf), type = "o", pch = 20, col = "red")

#zoom in
plot(sf::st_geometry(great_lakes_polygon), col = "lightgrey",
```

```
  xlim = sf::st_bbox(path_sf)[c("xmin", "xmax")],
  ylim = sf::st_bbox(path_sf)[c("ymin", "ymax")])
points(sf::st_coordinates(path_sf),type="o", pch = 20, col = "red")


# Example 4 - SpatialPolygonsDataFrame input
data(greatLakesPoly)

#simulate in great lakes polygon
path_sp <- crw_in_polygon(greatLakesPoly,
                          theta = c(0, 25),
                          stepLen = 10000,
                          initHeading = 0,
                          nsteps = 100,
                          cartesianCRS = 3175,
                          sp_out = TRUE)

#plot
plot(sf::st_as_sfc(greatLakesPoly), col = "lightgrey", border = "grey")
points(sf::st_coordinates(sf::st_as_sf(path_sp)), type = "o", pch = 20,
                          col = "red")

#zoom in
plot(sf::st_as_sfc(greatLakesPoly), col = "lightgrey", border = "grey",
  xlim = sp::bbox(sf::st_coordinates(sf::st_as_sf(path_sp)))[1,],
  ylim = sp::bbox(sf::st_coordinates(sf::st_as_sf(path_sp)))[2,])
points(sf::st_coordinates(sf::st_as_sf(path_sp)), type = "o", pch = 20,
       col = "red")
```

`detection_bubble_plot` *Plot number of tagged animals or detections on a map*

Description

Make bubble plots showing the number of fish detected across a defined set of receiver locations.

Usage

```
detection_bubble_plot(
  det,
  location_col = "glatos_array",
  receiver_locs = NULL,
  map = NULL,
  out_file = NULL,
  background_ylim = c(41.3, 49),
  background_xlim = c(-92.45, -75.87),
  symbol_radius = 1,
  col_grad = c("white", "red"),
  scale_loc = NULL
)
```

Arguments

det	A glatos_detections object (e.g., produced by read_glatos_detections). *OR* a data frame containing detection data with four columns described below and one column containing a location grouping variable, whose name is specified by location_col (see below). The following four columns must appear in det, except deploy_lat and deploy_lon are not needed if receiver_locs is specified: animal_id Individual animal identifier; character. detection_timestamp_utc Timestamps for the detections (MUST be of class 'POSIXct'). deploy_lat Latitude of receiver deployment in decimal degrees, NAD83. deploy_long Longitude of receiver deployment in decimal degrees, NAD83.
location_col	A character string indicating the column name in det (and receiver_locs if specified) that will be used as the location grouping variable (e.g. "glatos_array"), in quotes.
receiver_locs	An optional data frame containing receiver data with the two columns ('deploy_lat', 'deploy_long') described below and one column containing a location grouping variable, whose name is specified by location_col (see above). The following two columns must appear in receiver_locs: • deploy_lat Latitude of receiver deployment in decimal degrees, NAD83. • deploy_long Longitude of receiver deployment in decimal degrees, NAD83.
map	An optional sp or sf spatial object that can by plotted with using plot to be included as the background for the plot. If NULL, then the example Great Lakes polygon object (data(great_lakes_polygon)) will be used.
out_file	An optional character string with the name (including extension) of output file created. File extension will determine type of file written. For example, "BubblePlot.png" will write a png file to the working directory. If NULL (default) then the plot will be printed to the default plot device. Supported extensions: png, jpeg, bmp, and tiff.
background_ylim	A two-element numeric vector that defines minimum and maximum extents of the viewable plot area along the y-axis (i.e., longitude).
background_xlim	A two-element numeric vector that defines minimum and maximum extents of the viewable plot area along the x-axis (i.e., latitude).
symbol_radius	Radius of each "bubble" on the plot in units of percent of x-axis scale. Default value = 1 (i.e., 1 percent of x-axis).
col_grad	A two-element character vector indicating the start and end colors of the gradient scale used to color-code "bubbles".
scale_loc	An optional 4-element numeric vector, to be passed to plotrix::color.legend, indicating the plotting location of the legend in the same units as map. Elements in the vector are the lower left and upper right coordinates of the rectangle of colors (i.e., c(xleft, ybottom, xright, ytop)). If scale_loc = NULL (default), the legend is plotted along the left edge of the plot.

Details

Data are summarized using summarize_detections.

If `receiver_locs` is specified (not NULL) then the plot will show all receivers in `receiver_locs` including any that detected none of the transmitters in `det`. Although this is helpful to view locations where fish were *not* detected, the user will usually want to take care to include only receivers that were in the water during the period of interest. If you are using a glatos receiver locations file to specify location for plotting, you will likely want to filter the receiver data by depoyment and receovery dates to exclude deployments that occured outside of the period of interest.

"col_grad" is used in a call to colorRampPalette, which will accept a vector containing any two colors return by colors as character strings.

Value

A data frame produced by `glatos::summarize_detections(det, location_col = location_col, receiver_locs = receiver_locs, summ_type = "location")`

If not out_file is specified, then an image is printed to the default plot device. If out_file is specified, then an image of specified type is written to `out_file`.

Author(s)

T. R. Binder, edited by A. Dini

See Also

`summarize_detections()`

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
  package = "glatos")
det <- read_glatos_detections(det_file)

#call with defaults
detection_bubble_plot(det, map = great_lakes_polygon)

#change symbol size and color
detection_bubble_plot(det, symbol_radius = 2, col_grad = c("grey90", "grey10"))

#Add all receivers

# get path to example receiver file
rec_file <- system.file("extdata", "sample_receivers.csv",
  package = "glatos")
rec <- read_glatos_receivers(rec_file)

detection_bubble_plot(det, receiver_locs = rec)


#' #Subset receivers to include on receivers that were deployed during the
```

```
#' detection interval.

# get path to example receiver file
rec_file <- system.file("extdata", "sample_receivers.csv",
  package = "glatos")
rec <- read_glatos_receivers(rec_file)

first <- min(det$detection_timestamp_utc) # time of first detection
last <- max(det$detection_timestamp_utc) # time of last detection

# Subset receiver deployments oustide the detection period.
# !is.na(rec$recover_date_time) eliminates receivers that have been
# deployed but not yet recovered.
plot_rec <- rec[rec$deploy_date_time < last &
                rec$recover_date_time > first &
                !is.na(rec$recover_date_time),]

detection_bubble_plot(det, receiver_locs = plot_rec)
```

`detection_events` *Classify discrete events in detection data*

Description

Reduce detection data into discrete detection events, defined by movement between receivers (or receiver groups, depending on location), or sequential detections at the same location that are separated by a user-defined threshold period of time.

Usage

```
detection_events(
  det,
  location_col = "glatos_array",
  time_sep = Inf,
  condense = TRUE
)
```

Arguments

`det`	A `glatos_detections` object (e.g., produced by read_glatos_detections). *OR* a data frame containing detection data with four columns described below and one column containing a location grouping variable, whose name is specified by `location_col` (see below). The following four columns must appear in `det`: `animal_id` Individual animal identifier; character. `detection_timestamp_utc` Detection timestamps; MUST be of class POSIXct. `deploy_lat` Latitude of receiver deployment in decimal degrees, NAD83.

deploy_long Longitude of receiver deployment in decimal degrees, NAD83.

location_col A character string indicating the column name in det that will be used as the location grouping variable (e.g. "glatos_array"), in quotes.

time_sep Amount of time (in seconds) that must pass between sequential detections on the same receiver (or group of receivers, depending on specified location) before that detection is considered to belong to a new detection event. The default value Inf, will not define events based on elapsed time (only when location changes).

condense A logical indicating if the result should be a condensed data frame (condense = TRUE; default value) with one event per row, or the input data frame with new event data columns added condense = TRUE.

Details

mean_latitude and mean_longitude columns in the output dataframe are the mean GPS locations for the detections comprising that detection event. For example, if the a fish was detected at 3 receiver stations in a glatos_array and glatos_array was selected as the location, then GPS location for that event will be the mean of the latitude and longitude for those three receiver stations (weighted based on the number of detections that occurred on each station).

Value

A data.table or tibble object (if input is either type; output class to match input) or data.frame otherwise. Structure depends on value of condense argument:

If condense = TRUE, a data.frame, data.table, or tibble with the following columns:

event Unique event identifier.

individual Unique 'animal_id'.

location Unique 'location'.

mean_latitude Mean latitude of detections comprising each event.

mean_longitude Mean longitude of detections comprising each event.

first_detection
The time of the first detection in a given detection event.

last_detection The time of the last detection in a given detection event.

num_detections The total number of detection that comprised a given detection event.

res_time_sec The elapsed time in seconds between the first and last detection in a given event.

If condense = FALSE, a data.frame, data.table, or tibble matching the input data frame det with the following columns added:

time_diff Lagged time difference in seconds between successive detections of each animal_id.

arrive Flag (0 or 1) representing the first detection in each event.

depart Flag (0 or 1) representing the last detection in each event.

event Integer representing the event number.

Author(s)

T. R. Binder, T. A. Hayden, C. M. Holbrook

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")
det <- read_glatos_detections(det_file)

filt0 <- detection_events(det) #no time filter

#7-day filter
filt_7d <- detection_events(det , time_sep = 604800)

#7-day filter but return do not condense result
filt_7d <- detection_events(det , time_sep = 604800, condense = FALSE)
```

---

detect_transmissions *Simulate detection of transmitter signals in a receiver network*

Description

Simulates detection of transmitter signals in a receiver network based on detection range curve (detection probability as a function of distance), location of transmitter, and location of receivers.

Usage

```
detect_transmissions(
  trnsLoc = NA,
  recLoc = NA,
  detRngFun = NA,
  trnsColNames = list(time = "time", x = "x", y = "y"),
  recColNames = list(x = "x", y = "y"),
  inputCRS = NA,
  sp_out = TRUE,
  show_progress = TRUE
)
```

Arguments

trnsLoc	A data frame with location (two numeric columns) and time (numeric or POSIXct column) of signal transmissions. *OR* An object of class sf or sfc containing POINT features (geometry column) and time (see colNames). (SpatialPointsDataFrame is also allowed.)

recLoc	A data frame with coordinates (two numeric columns) of receiver locations. *OR* An object of class sf or sfc containing a POINT feature (geometry column) for each receiver. (SpatialPointsDataFrame is also allowed.)
detRngFun	A function that defines detection range curve; must accept a numeric vector of distances (in meters) and return a numeric vector of detection probabilities at each distance.
trnsColNames	A named list containing the names of columns in trnsLoc with timestamps (default is "time") and coordinates (defaults are "x" and "y") of signal transmissions. Location column names are ignored if trnsLoc is a spatial object with a geometry column.
recColNames	A named list containing the names of columns in recLoc with coordinates of receiver locations (defaults are "x" and "y"). Location column names are ignored if recLoc is a spatial object with a geometry column.
inputCRS	Defines the coordinate reference system (object of class crs or numeric EPSG code) if crs is missing from inputs trnsLoc or recLoc; ignored if input trnsLoc and recLoc are of class sf, sfc, or SpatialPointsDataFrame).
sp_out	Logical. If TRUE (default) then output is an sf object. If FALSE, then output is a data.frame.
show_progress	Logical. Progress bar and status messages will be shown if TRUE (default) and not shown if FALSE.

Details

Distances between each signal transmission and receiver are calculated using geodist (measure = "haversine") if input crs is geographic (i.e., longitude, latitude) and using simple Euclidean distances if input crs is Cartesian (e.g., UTM). If crs is missing, the an arbitrary Cartesian coordinate system with base unit of 1 meter is assumed. Computation time is fastest if coordinates are are in a Cartesian (projected) coordinate system and slowest if coordinates are in a geographic (long lat) coordinate system.

The probability of detecting each signal on each receiver is determined from the detection range curve. Detection of each signal on each receiver is determined stochastically by draws from a Bernoulli distribution with probability p (detection prob.).

This function was written to be used along with transmit_along_path.

Value

When sp_out = TRUE, an sf object containing one POINT feature with coordinates of each receiver and transmission location of each simulated detection (i.e., two geography columns names rec_geometry and trns_geometry) and the the following columns:

trns_id	Unique signal transmission ID.
rec_id	Unique receiver ID.
time	Elapsed time.

When sp_out = FALSE, a data.frame with columns containing coordinates of receiver locations of each simulation detection:

rec_x	Receiver x coordinate.
rec_y	Receiver y coordinate.
trns_x	Transmitter x coordinate at time of transmission.
trns_y	Transmitter y coordinate at time of transmission.

and the columns described above.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

See Also

transmit_along_path to simulate transmissions along a path (i.e., create trnsLoc).

Examples

```
#Example 1 - data.frame input (make a simple path in polygon)

mypoly <- data.frame(x = c(0, 0, 1000, 1000),
                     y = c(0, 1000, 1000, 0))

mypath <- crw_in_polygon(mypoly,
                         stepLen = 100,
                         nsteps = 50,
                         sp_out = FALSE)

plot(mypath, type = "l", xlim = c(0, 1000), ylim = c(0, 1000))

#add receivers
recs <- expand.grid(x = c(250, 750), y = c(250, 750))
points(recs, pch = 15, col = "blue")

#simulate tag transmissions
mytrns <- transmit_along_path(mypath, vel = 2.0, delayRng = c(60, 180),
                              burstDur = 5.0, sp_out = FALSE)
points(mytrns, pch = 21) #add to plot

#Define detection range function (to pass as detRngFun)
# that returns detection probability for given distance
# assume logistic form of detection range curve where
#   dm = distance in meters
#   b = intercept and slope
pdrf <- function(dm, b=c(0.5, -1/120)){
  p <- 1/(1+exp(-(b[1]+b[2]*dm)))
  return(p)
}
pdrf(c(100,200,300,400,500)) #view detection probs. at some distances
```

```
#simulate detection
mydtc <- detect_transmissions(trnsLoc = mytrns,
                              recLoc = recs,
                              detRngFun = pdrf,
                              sp_out = FALSE)

points(mydtc[, c("trns_x", "trns_y")], pch = 21, bg = "red")

#link transmitter and receiver locations for each detection\
with(mydtc, segments(x0 = trns_x,
                     y0 = trns_y,
                     x1 = rec_x,
                     y1 = rec_y,
                     col = "red"))


#Example 2 - spatial (sf) input

data(great_lakes_polygon)

set.seed(610)
mypath <- crw_in_polygon(great_lakes_polygon,
                         stepLen = 100,
                         initPos = c(-83.7, 43.8),
                         initHeading = 0,
                         nsteps = 50,
                         cartesianCRS = 3175)

plot(sf::st_geometry(mypath), type = "l")

#add receivers
recs <- expand.grid(x = c(-83.705, -83.70),
                    y = c(43.810, 43.815))
points(recs, pch = 15, col = "blue")

#simulate tag transmissions
mytrns <- transmit_along_path(mypath, vel = 2.0, delayRng = c(60, 180),
                              burstDur = 5.0)
points(sf::st_coordinates(mytrns), pch = 21) #add to plot

#Define detection range function (to pass as detRngFun)
# that returns detection probability for given distance
# assume logistic form of detection range curve where
#   dm = distance in meters
#   b = intercept and slope
pdrf <- function(dm, b=c(2, -1/120)){
  p <- 1/(1+exp(-(b[1]+b[2]*dm)))
  return(p)
}
pdrf(c(100,200,300,400,500)) #view detection probs. at some distances

#simulate detection
mydtc <- detect_transmissions(trnsLoc = mytrns,
```

```
                              recLoc = recs,
                              detRngFun = pdrf)

#view transmissions that were detected
sf::st_geometry(mydtc) <- "trns_geometry"

points(sf::st_coordinates(mydtc$trns_geometry), pch = 21, bg = "red")

#link transmitter and receiver locations for each detection
segments(x0 = sf::st_coordinates(mydtc$trns_geometry)[,"X"],
         y0 = sf::st_coordinates(mydtc$trns_geometry)[,"Y"],
         x1 = sf::st_coordinates(mydtc$rec_geometry)[,"X"],
         y1 = sf::st_coordinates(mydtc$rec_geometry)[,"Y"],
         col = "red")
```

`false_detections` *False detection filter*

Description

Identify possible false detections based on "short interval" criteria (e.g., GLATOS 'min_lag') .

Usage

```
false_detections(det, tf, min_lag_col = "min_lag", show_plot = FALSE, ...)
```

Arguments

`det`	A `glatos_detections` object (e.g., produced by read_glatos_detections). *OR:* A data frame with one column containing 'min_lag' which for each detection record, is the smallest time (in seconds) to the next closest detection (either previous or subsequent) of the same transmitter on the same receiver. The name of the column containing 'min_lag' can be specified via `min_lag_col`; see below). *OR* (*if* `min_lag` *is missing*) A data farme containing detection data with the four columns described below. In that case, `min_lag` will be calculated using min_lag). **detection_timestamp_utc** Detection timestamps; MUST be of class POSIXct. **transmitter_codespace** A character string with transmitter code space (e.g., "A69-1061" for Vemco PPM coding"). **transmitter_id** A character string with transmitter ID code (e.g., "1363" for Vemco PPM coding"). **receiver_sn** A character vector with unique receiver serial number.
`tf`	A number indicating the time threshold (in seconds; e.g., Pincock's (2012) "short interval") for identifying possible false detections.
`min_lag_col`	A character string containing the name of the column in `det` that contains 'min_lag'.

show_plot	Indicates if a plot should be displayed showing the proportion of detections that exceed min_lag from min_lag = 1 to min_lag = 5 * tf.
...	Additional arguments passed to plot.

Details

Detections are identified as potentially false when `min_lag > tf`.

A new column (`passed_filter`), indicating if each record (row) passed the filter, is added to the input data frame.

This function was written specifically with GLATOS standard detection export in mind, but if `min_lag` is absent and `min_lag_col` is not specified, then `min_lag` will be calculated using min_lag.

A common rule of thumb for choosing `tf` for VEMCO PPM encoded transmitters is 30 times the nominal delay (e.g., 3600 s for a transmitter with a 120 s nominal delay) - see Pincock (2012).

When `show_plot = TRUE` then the plot may be used to assess sensitivity of the proportion of detections removed to the choice of `tf`.

Value

A data frame consisting of `det` with an additional column 'passed_filter' indicating if each detection did (1) or did not (0) pass the criteria.

Author(s)

T. R. Binder, edited by A. Dini

References

Pincock, D.G., 2012. False detections: what they are and how to remove them from detection data. Vemco Division, Amirix Systems Inc., Halifax, Nova Scotia.
http://www.vemco.com/pdf/false_detections.pdf

Simpfendorfer, C.A., Huveneers, C., Steckenreuter, A., Tattersall, K., Hoenner, X., Harcourt, R. and Heupel, M.R., 2015. Ghosts in the data: false detections in VEMCO pulse position modulation acoustic telemetry monitoring equipment. Animal Biotelemetry, 3(1), p.55.
https://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-015-0094-z

See Also

min_lag

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")
det <- read_glatos_detections(det_file)

det <- false_detections(det, 3600)
head(det)
```

```
#plot sensitivity to tf
det <- false_detections(det, 3600, show_plot = TRUE)
```

`flynn_island_polygon` *An sf POLYGON object with coastline of Flynn Island*

Description

An sf POLYGON object with coastline of Flynn Island; and island within Higgins Lake, Michigan. Used as an example of a polygon representing a body of land (as opposed to water body).

Usage

```
flynn_island_polygon
```

Format

An object of class `sf` (inherits from `data.frame`) with 1 rows and 2 columns.

Author(s)

Chris Holbrook

`flynn_island_transition` *A transition object for Flynn Island for testing make_transition*

Description

A transition object, created from `flynn_island_polygon` for testing `make_transition`.

Usage

```
system.file("testdata", "flynn_island_transition.rds", package = "glatos")
```

Format

A list comprised of a TransitionLayer and RasterLayer (see `make_transition`).

Filename

flynn_island_transition.rds

Author(s)

Chris Holbrook

get_days — *Determines which calculation method to use for the residency index.*

Description

Wrapper method for the calulation methods above.

Usage

```
get_days(dets, calculation_method = "kessel", time_interval_size = "1 day")
```

Arguments

dets	• data frame pulled from the detection events
calculation_method	• determines which method above will be used to count total time and location time
time_interval_size	• size of time interval

glatos — *An R package for the Great Lakes Acoustic Telemetry Observation System*

Description

glatos is an R package with functions useful to members of the Great Lakes Acoustic Telemetry Observation System (https://glatos.glos.us). Functions may be generally useful for processing, analyzing, simulating, and visualizing acoustic telemetry data, but are not strictly limited to acoustic telemetry applications.

Package status

This package is in early development and its content is evolving. To access the package or contribute code, join the project at (https://github.com/ocean-tracking-network/glatos). If you encounter problems or have questions or suggestions, please post a new issue or email cholbrook@usgs.gov (maintainer: Chris Holbrook).

Installation

Installation instructions can be found at https://github.com/ocean-tracking-network/glatos/wiki/installation-instructions.

Data loading and processing

read_glatos_detections and read_otn_detections Fast data loading from standard GLATOS and OTN data files to a single structure that is compatible with other glatos functions.

read_glatos_receivers and read_otn_deployments Reads receiver location histories from standard GLATOS and OTN data files to a single structure that is compatible with other glatos functions.

read_glatos_workbook Reads project-specific receiver history and fish taggging and release data from a standard glatos workbook file.

read_vemco_tag_specs Reads transmitter (tag) specifications and operating schedule.

real_sensor_values Converts 'raw' transmitter sensor (e.g., depth, temperature) to 'real'-scale values (e.g., depth in meters) using transmitter specification data (e.g., from read_vemco_tag_specs).

Filtering and summarizing

min_lag Facilitates identification and removal of false positive detections by calculating the minimum time interval (min_lag) between successive detections.

false_detections Removes potential false positive detections using "short interval" criteria (see min_lag).

detection_events Distills detection data down to a much smaller number of discrete detection events, defined as a change in location or time gap that exceeds a threshold.

summarize_detections Calculates number of fish detected, number of detections, first and last detection timestamps, and/or mean location of receivers or groups, depending on specific type of summary requested.

residence_index calculates the relative proportion of time spent at each location.

REI calculates the relative activity at each receiver based on number of unique species and individual animals.

Visualization and data exploration

abacus_plot Useful for exploring movement patterns of individual tagged animals through time.

detection_bubble_plot Useful for exploring distribution of tagged individuals among receivers.

interpolate_path, make_frames, and make_video Interpolate spatio-temporal movements, between detections, create video frames, and stitch frames together to create animated video file using FFmpeg software.

adjust_playback_time Modify playback speed of videos and optionally convert between video file formats. Requires FFmpeg.

Simulation functions for system design and evaluation

calc_collision_prob Estimates the probability of collisions for pulse-position-modulation type co-located telemetry transmitters. This is useful for determining the number of fish to release or tag specifications (e.g., delay).

receiver_line_det_sim Simulates detection of acoustic-tagged fish crossing a receiver line (or single receiver). This is useful for determining optimal spacing of receviers in a line and tag specifications (e.g., delay).

crw_in_polygon, transmit_along_path, and detect_transmissions Individually simulate random fish movement paths within a water body (crw_in_polygon: a random walk in a polygon), tag signal transmissions along those paths (transmit_along_path: time series and locations of transmissions based on tag specs), and detection of those transmittions by receivers in a user-defined receiver network (detect_transmissions: time series and locations of detections based on detection range curve). Collectively, these functions can be used to explore, compare, and contrast theoretical performance of a wide range of transmitter and receiver network designs.

Convert glatos data objects to other package classes

convert_glatos_to_att Converts glatos_detections and glatos_receiver objects to ATT for compatibility with the Animal Tracking Toolbox(https://github.com/vinayudyawer/ATT) and the VTrack package.

convert_otn_erddap_to_att Converts glatos_detections and transmitter, receiver, and animal metadata from the OTN ERDDAP to ATT format for compatibility with the Animal Tracking Toolbox(https://github.com/vinayudyawer/ATT) and the VTrack package.

---

glatos-defunct *Defunct functions in glatos*

Description

These functions are gone, no longer available.

Details

- `check_dependencies`: Removed in glatos 0.7.0.
- `install_ffmpeg`: Removed in glatos 0.7.0.
- `make_video_ffmpeg`: Removed in glatos 0.7.0. Use `make_video` instead.

---

glatos_animals *Constructor function for the class glatos_animals*

Description

Constructor function for the class glatos_animals. Currently barebones and only used inside read_glatos_workbook.

Usage

```
glatos_animals(x)
```

Arguments

x	A data.frame or data.table created from a standard GLATOS workbook file.

Value

A data.frame of class `glatos_animals`:

Note

This function may be developed in the future to dictate conversion constuction from a data frame.

`glatos_detections` *Constructor function for the class glatos_detections*

Description

Constructor function for the class glatos_detections. Currently barebones and only used inside read_glatos_detections and read_otn_detections.

Usage

```
glatos_detections(x)
```

Arguments

`x`	A data.frame or data.table created from a standard glatos detection file.

Value

A data.frame of class `glatos_detections`:

Note

This function may be developed in the future to dictate conversion constuction from a data frame.

`glatos_receivers` *Constructor function for the class glatos_receivers*

Description

Constructor function for the class glatos_receivers. Currently barebones and only used inside read_glatos_receivers.

Usage

```
glatos_receivers(x)
```

Arguments

`x`	A data.frame or data.table created from a standard glatos receiver_location file.

Value

A data.frame of class `glatos_receivers`:

Note

This function may be developed in the future to dictate conversion constuction from a data frame.

`greatLakesPoly` *Deprecated A SpatialPolygonDataFrame with Great Lakes coastline and some major tributaries.*

Description

A SpatialPolygonDataFrame with Great Lakes coastline and some major tributaries. This is used as a default map background in several glatos functions.

Usage

```
greatLakesPoly
```

Format

An object of class `SpatialPolygonsDataFrame` with 4 rows and 8 columns.

Details

This dataset is deprecated and will be removed in a future version. Use `great_lakes_polygon` instead.

Author(s)

Todd Hayden

`greatLakesTrLayer` *A TransitionLayer object that only allows transitions to occur within water of the Great Lakes Basin.*

Description

A TransitionLayer object that only allows transitions to occur within water (i.e., prohibits movement onto land). This dataset was developed for non-linear interpolation of fish movement paths from telemetry data and is used by default in interpolate_path.

Usage

```
greatLakesTrLayer
```

Format

An object of class `TransitionLayer` of dimension 692 x 504 x 1.

Author(s)

Todd Hayden

See Also

interpolate_path, gdistance

`great_lakes_polygon` *An sf POLYGON object with Great Lakes coastline and some major tributaries.*

Description

Created from `greatLakesPoly`. This is used as a default map background in several glatos functions.

Usage

```
great_lakes_polygon
```

Format

An object of class `sf` (inherits from `data.frame`) with 4 rows and 9 columns.

Author(s)

Todd Hayden (coerced to sf by C. Holbrook)

`higgins_lake_polygon` *An sf POLYGON object with coastline of Higgins Lake*

Description

An sf POLYGON object with coastline of Higgins Lake, Michigan. Used as an example of a polygon representing a water body.

Usage

```
higgins_lake_polygon
```

Format

An object of class `sf` (inherits from `data.frame`) with 1 rows and 2 columns.

Author(s)

Chris Holbrook

`higgins_lake_transition` *A transition object for Higgins Lake for testing make_transition*

Description

A transition object, created from `higgins_lake_polygon` for testing `make_transition`.

Usage

```
system.file("testdata", "higgins_lake_transition.rds", package = "glatos")
```

Format

A list comprised of a TransitionLayer and RasterLayer (see `make_transition`).

Filename

higgins_lake_transition.rds

Author(s)

Chris Holbrook

`interpolate_path` *Interpolate new positions within a spatiotemporal path data*

Description

Interpolate new positions within a spatiotemporal path data set (e.g., detections of tagged fish) at regularly-spaced time intervals using linear or non-linear interpolation.

Usage

```
interpolate_path(
  det,
  trans = NULL,
  start_time = NULL,
  int_time_stamp = 86400,
  lnl_thresh = 0.9,
  out_class = NULL,
  show_progress = TRUE
)
```

Arguments

det	An object of class glatos_detections or data frame containing spatiotemporal data with at least 4 columns containing 'animal_id', 'detection_timestamp_utc', 'deploy_long', and 'deploy_lat' columns.
trans	An optional transition matrix with the "cost" of moving across each cell within the map extent. Must be of class TransitionLayer. A transition layer may be created from a polygon shapefile using make_transition.
start_time	specify the first time bin for interpolated data. If not supplied, default is first timestamp in the input data set. Must be a character string that can be coerced to 'POSIXct' or an object of class 'POSIXct'. If character string is supplied, timezone is automatically set to UTC.
int_time_stamp	The time step size (in seconds) of interpolated positions. Default is 86400 (one day).
lnl_thresh	A numeric threshold for determining if linear or non-linear interpolation shortest path will be used.
out_class	Return results as a data.table or tibble. Default returns results as data.frame. Accepts data.table or tibble.
show_progress	Logical. Progress bar and status messages will be shown if TRUE (default) and not shown if FALSE.

Details

Non-linear interpolation uses the gdistance package to find the shortest pathway between two locations (i.e., receivers) that avoid 'impossible' movements (e.g., over land for fish). The shortest non-linear path between two locations is calculated using a transition matrix layer that represents the 'cost' of an animal moving between adjacent grid cells. The transition matrix layer (see gdistance) is created from a polygon shapefile using make_transition or from a RasterLayer object using transition. In make_transition, each cell in the output transition layer is coded as water (1) or land (0) to represent possible (1) and impossible (0) movement paths.

Linear interpolation is used for all points when trans is not supplied. When trans is supplied, then interpolation method is determined for each pair of sequential observed detections. For example, linear interpolation will be used if the two geographical positions are exactly the same and when the ratio (linear distance:non-linear distance) between two positions is less than lnl_thresh. Non-linear interpolation will be used when ratio is greater than lnl_thresh. When the ratio of linear distance to non-linear distance is greater than lnl_thresh, then the distance of the non-linear path needed to avoid land is greater than the linear path that crosses land. lnl_thresh can be used to control whether non-linear or linear interpolation is used for all points. For example, non-linear interpolation will be used for all points when lnl_thresh > 1 and linear interpolation will be used for all points when lnl_thresh = 0.

All linear interpolation is done by codestats::approx with argument ties = "ordered" controlling how tied x values are handled. See approxfun().

Value

A dataframe with animal_id, bin_timestamp, latitude, longitude, and record_type.

Author(s)

Todd Hayden, Tom Binder, Chris Holbrook

Examples

```
#-------------------------------------------------
# EXAMPLE #1 - simple interpolate among lakes

# get polygon of the Great Lakes
data(great_lakes_polygon) #glatos example data
plot(sf::st_geometry(great_lakes_polygon), xlim = c(-92, -76))

# make sample detections data frame
pos <- data.frame(
   animal_id=1,
   deploy_long=c(-87,-82.5, -78),
   deploy_lat=c(44, 44.5, 43.5),
   detection_timestamp_utc=as.POSIXct(c("2000-01-01 00:00",
     "2000-02-01 00:00", "2000-03-01 00:00"), tz = "UTC"))

#add to plot
points(deploy_lat ~ deploy_long, data = pos, pch = 20, cex = 2, col = 'red')

# interpolate path using linear method
path1 <- interpolate_path(pos)
nrow(path1) #now 61 points
sum(path1$record_type == "interpolated") #58 interpolated points

#add linear path to plot
points(latitude ~ longitude, data = path1, pch = 20, cex = 0.8, col = 'blue')

# load a transition matrix of Great Lakes
# NOTE: This is a LOW RESOLUTION TransitionLayer suitable only for
#       coarse/large scale interpolation only. Most realistic uses
#       will need to create a TransitionLayer; see ?make_transition.
data(greatLakesTrLayer) #glatos example data; a TransitionLayer

# interpolate path using non-linear method (requires 'trans')
path2 <- interpolate_path(pos, trans = greatLakesTrLayer)

# add non-linear path to plot
points(latitude ~ longitude, data = path2, pch = 20, cex = 1,
       col = 'green')

# can also force linear-interpolation with lnlThresh = 0
path3 <- interpolate_path(pos, trans = greatLakesTrLayer, lnl_thresh = 0)

# add new linear path to plot
points(latitude ~ longitude, data = path3, pch = 20, cex = 1,
          col = 'magenta')

#-------------------------------------------------
```

```
# EXAMPLE #2 - walleye in western Lake Erie
## Not run:


# get example walleye detection data
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")
det <- read_glatos_detections(det_file)

# take a look
head(det)

# extract one fish and subset date
det <- det[det$animal_id == 22 &
           det$detection_timestamp_utc > as.POSIXct("2012-04-08") &
           det$detection_timestamp_utc < as.POSIXct("2013-04-15") , ]

# get polygon of the Great Lakes
data(great_lakes_polygon) #glatos example data; an sf object

# convert polygon to terra::spatVector
great_lakes_polygon <- terra::vect(great_lakes_polygon)

# crop polygon to western Lake Erie
maumee <-  terra::crop(great_lakes_polygon,
                       y = terra::ext(-83.7, -82.5, 41.3, 42.4))


plot(maumee, col = "grey")
points(deploy_lat ~ deploy_long, data = det, pch = 20, col = "red",
  xlim = c(-83.7, -80))


make transition layer object
tran <- make_transition3(sf::st_as_sf(maumee), res = c(0.1, 0.1))

# plot to check output
plot(tran$rast, xlim = c(-83.7, -82.0), ylim = c(41.3, 42.7))
plot(maumee, add = TRUE)

# not high enough resolution- bump up resolution, will take some time
tran1 <- make_transition3(sf::st_as_sf(maumee), res = c(0.001, 0.001))

# plot to check resolution- much better
plot(tran1$rast, xlim = c(-83.7, -82.0), ylim = c(41.3, 42.7))
plot(maumee, add = TRUE)


# add fish detections to make sure they are "on the map"
# plot unique values only for simplicity
foo <- unique(det[, c("deploy_lat", "deploy_long")])
points(foo$deploy_long, foo$deploy_lat, pch = 20, col = "red")
```

```
# call with "transition matrix" (non-linear interpolation), other options
# note that it is quite a bit slower than linear interpolation
pos2 <- interpolate_path(det,
                         trans = tran1$transition,
                         out_class = "data.table")

plot(maumee, col = "grey")
points(latitude ~ longitude, data = pos2, pch=20, col='red', cex=0.5)


## End(Not run)
```

`interval_count` *The function below takes a detection events data frame and determines the number of time bins in which detections were observed and returns the cumulative time covered by all bins, in days. Interval (bin) size is determined by the 'time_interval_size' argument.*

Description

For each event (row in detection events data frame), the function sequences from first_detection to last_detection by time_interval_size, then counts the number of unique intervals.

Usage

```
interval_count(detections, time_interval_size)
```

Arguments

`detections` • data frame from detection_events (condensed = TRUE)

`time_interval_size`
time increment string as in seq.Date 'by' argument

`kml_to_csv` *KML To CSV Conversion*

Description

Function for extracting features (points, lines, polygons) from kml files and writing them to csv files.

Usage

```
kml_to_csv(filePath, type = c("points", "lines", "polygons"))
```

Arguments

filePath	The pathname for the kml file you wish to convert.
type	Optional character string indicating the type(s) of feature(s) to read from the kml file. Valid values are c("points", "lines", and "polygons").

Details

kmz files are not supported. Make sure exports from Google earth are saved as kml. Or extract (unzip) kml from kmz.

Value

A csv file (same name as input filePath but with csv extension) is written to directory containing input filePath with five columns

name Feature name

feature_type Feature type

seq Sequential position in feature

longitude Longitude

latitude Latitude

altitude Altitude

Examples

```
#Get example kml with two polygons
kml_file <- system.file("inst/extdata", "example_polygons.kml",
                        package = "glatos")

kml_to_csv(kml_file)
```

---

kml_workbook *Make a KML or KMZ file of receiver and animal release locations*

Description

Convert standard GLATOS receiver location and animal release data to a KML (or optionally KMZ) file (e.g., for viewing in Google Earth). (NOTE: EARLY DEVELOPMENT VERSION).

Usage

```
kml_workbook(
  wb = NULL,
  wb_file = NULL,
  receiver_locs = NULL,
  animals = NULL,
  kmz = FALSE,
  show_ongoing_recs = TRUE,
  end_date = NULL,
  out_file = NULL,
  wb_version = NULL,
  ...
)
```

Arguments

wb	A glatos_workbook object created by read_glatos_workbook.
wb_file	A character string with path and name of workbook in standard GLATOS format (*.xlsm). If only file name is given, then the file must be located in the working directory. File must be a standard GLATOS file (e.g., *xxxxx_GLATOS_YYYYMMDD.xlsm*) submitted via GLATOSWeb Data Portal http://glatos.glos.us.
receiver_locs	**not yet implemented**
animals	**not yet implemented**
kmz	logical; If TRUE, a KMZ file (zipped KML file) will be created. Default value is FALSE.
show_ongoing_recs	Indicates if ongoing stations (missing recovery timestamp) should be included in result.
end_date	End date (e.g. "YYYY-MM-DD") to be used for any ongoing stations (if showOngoing == T). Defaults to current system time.
out_file	File name (path optional) of output file. If path not specified then file will be written to working directory. Extension is not checked against kmz. Required if wb_file is NULL. If not specified and wb_file is given, then file will be written to file with name matching wb_file.
wb_version	An optional character string with the workbook version number. Passed to read_glatos_workbook when input is wb_file.
...	optional arguments that influence kml/kmz features. Curently only two options: labelSize A numeric scalar with the size of placemark labels (only shown when placemark is highlighted by user). iconSize A numeric scalar with the size of placemark icons.

Details

Receiver locations will be visible between deployment and recovery timestamps at each location. Release locations will be displayed when the display window includes the date of release.

Value

A KML (and optionally, KMZ) file, written to the directory that contains the input GLATOS workbook, or `out_file` otherwise. Path to output file is returned.

Author(s)

C. Holbrook <cholbrook@usgs.gov>

Examples

```
## Not run:
#get path to example GLATOS Data Workbook
wb_file <- system.file("extdata",
"walleye_workbook.xlsm", package = "glatos")

#read workbook directly
kml_workbook(wb_file = wb_file)

#now with bigger label and point and out_file
kml_workbook(wb_file = wb_file, labelSize = 20, iconSize = 1,
  out_file = "bigger.kml")

#read workbook directly; output kmz
kml_workbook(wb_file = wb_file, kmz = TRUE)

#get path to example GLATOS Data Workbook
wb <- read_glatos_workbook(wb_file)
kml_workbook(wb = wb, kmz = TRUE, out_file = "bigger.kmz")

## End(Not run)
```

---

lamprey_tracks *Sea Lamprey positions from Lake George, St. Marys River, 2012*

Description

Sea Lamprey positions from a positional acoustic telemetry array in Lake George, North Channel of the St. Marys River during the 2012 spawning year.

Usage

```
lamprey_tracks
```

Format

A data frame with 21043 rows and 14 variables:

DETECTEDID transmitter identifier (channel, frequency, code space, and ID code)

DATETIME position timestamp, in UTC

X,Y horizontal and vertical position on local grid, in meters

D assumed depth at time of detection, in meters (NOT from depth/pressure sensor)

LAT,LON position latitude and longitude, decimal degrees (west is negative); CRS: WGS84

n ?

HPE horizontal position error; calculated by VEMCO

HPEm horizontal position error, in meters; calculated by VEMCO

TEMP temperature at time of detection (from temperature sensor)

DEPTH depth at time of detection (from pressure sensor)

ACCEL acceleration at time of detection (from accelerometer)

DRX receivers that detected the associated transmission

Details

Data were collected as part of the GLATOS project SMRSL http://glatos.glos.us/home/project/SMRSL

Positions were calculated using the Vemco Positioning System.

Source

Chris Holbrook, US Geological Survey (cholbrook@usgs.gov)

`lonlat_to_utm` *Convert geographic positions to UTM*

Description

Convert geographic positions to UTM

Usage

```
lonlat_to_utm(lonlat)
```

make_frames *Create an animated video of spatiotemporal path data*

Description

Create a set of frames (png image files) showing geographic location data (e.g., detections of tagged fish or interpolated path data) at discrete points in time on top of a Great Lakes shapefile and optionally stitches frames into a video animation (mp4 file).

Usage

```
make_frames(
  proc_obj,
  recs = NULL,
  out_dir = getwd(),
  background_ylim = c(41.3, 49),
  background_xlim = c(-92.45, -75.87),
  show_interpolated = TRUE,
  tail_dur = 0,
  animate = TRUE,
  ani_name = "animation.mp4",
  frame_delete = FALSE,
  overwrite = FALSE,
  preview = FALSE,
  bg_map = NULL,
  show_progress = TRUE,
  ...
)
```

Arguments

proc_obj	A data frame created by interpolate_path() function or a data frame containing 'animal_id', 'bin_timestamp', 'latitude', 'longitude', and 'record_type'
recs	An optional data frame containing at least four columns with receiver 'deploy_lat', 'deploy_long', 'deploy_date_time', and 'recover_date_time'. Other columns in object will be ignored. Default column names match GLATOS standard receiver location file (e.g., 'GLATOS_receiverLocations_yyyymmdd.csv').
out_dir	A character string with file path to directory where individual frames for animations will be written. Default is working directory.
background_ylim	Vector of two values specifying the min/max values for y-scale of plot. Units are degrees.
background_xlim	Vector of two values specifying the min/max values for x-scale of plot. Units are degrees.

show_interpolated
: Boolean. Default (TRUE) include interpolated points.

tail_dur
: contains the duration (in same units as proc_obj$bin_timestamp; see interpolate_path()) of trailing points in each frame. Default value is 0 (no trailing points). A value of Inf will show all points from start.

animate
: Boolean. Default (TRUE) creates video animation by calling make_video() with output = ani_name. Default values are used for all other arguments. See Details below.

ani_name
: Character string with name and extension of animation output video file. Full path is optional. If file name only (no path), then the output video is written to 'out_dir' (same as images). To write to working directory, use "./" prefix (e.g., ani_name = "./animation.mp4". If animate = TRUE, the path and filename are passed to make_video().

frame_delete
: Boolean. Default (frame_delete = TRUE) delete individual image frames after animation is created

overwrite
: Overwite the animation (output video) file if it already exists. Default (overwrite = FALSE) prevents file from being overwritten and will result in error if the file exists. Passed to make_video() if animate = TRUE.

preview
: write first frame only. Useful for checking output before processing large number of frames. Default preview = FALSE

bg_map
: A sf points, lines, or polygons object. Spatial sp objects will be converted to sf

show_progress
: Logical. Progress bar and status messages will be shown if TRUE (default) and not shown if FALSE.

...
: Optional graphing parameters for customizing elements of fish location points, receiver location points, timeline, and slider (moves along the timeline). See also **Details** and **Note** sections.

Details

To customize fish location points (from proc_obj): Add any argument that can be passed to points. The following values will create the default plot:

- cex: symbol size; default = 2
- col: symbol color; default = "blue"
- pch: symbol type; default = 16

To customize receiver location points (from recs): Add prefix recs. to any argument that can be passed to points. The following values will create the default plot:

- recs.cex: symbol size; default = 1.5
- recs.pch: symbol type; default = 16

To customize timeline: Add add prefix timeline. to any argument of axis. Note all elements of the timeline except the sliding symbol (see 'slider' below) are created by a call to axis. The following values will create the default plot:

- timeline.at: a sequence with locations of labels (with first and last being start and end) along x-axis; in units of longitude; by default this will center the timeline with five equally-spaced labels in the middle 80% of background_xlim.

- `timeline.pos`: location along the y-axis; in units of latitude; by default this will place the timeline up from the bottom 6% of the range of `background_ylim`
- `timeline.labels`: text used for labels; default = `format(labels, "\%Y-\%m-\%d")`, where labels are values of proc_obj$bin_timestamp
- `timeline.col`: color of line; default = "grey70"
- `timeline.lwd`: width of line; default = 20 times the aspect ratio of the plot device
- `timeline.cex.axis`:size of labels; default = 2

To customize time slider (symbol that slides): Add prefix `timeline.` to any argument that can be passed to points. The following values will create the default plot:

- `timeslider.bg`: a single value with symbol bg color; default = "grey40"
- `timeslider.cex`: a single value with symbol size; default = 2
- `timeslider.col`: a single value with symbol type; default = "grey20"
- `timeslider.pch`: a single value with symbol type; default = 21

To customize parameters controlled by `par`: Add prefix `par.` to any argument that can be passed to par. Note that `par.mar` controls whitespace behind default timeslider. The following values will create the default plot:

- `par.oma` plot outer margins; default = c(0,0,0,0)
- `par.mar` plot inner margins; default = c(6,0,0,0)

If `animate = TRUE` then the animation output file name (`ani_name` argument) will be passed to the `output` argument in `make_video()`. Default values for all other `make_video()` arguments will be used. Note that the default frame rate is 24 frames per second (`framerate` argument in `make_video()`), which will determine the total length (duration) of the output video. For example, a video containing 240 images (frames) will run for 10 seconds at these default parameters. Note that output video duration, dimensions (size), and other ouput video characteristics can be modified by calling `make_video()` directly. To do this, set `animate = FALSE` and then use `make_video()` to create a video from the resulting set of images.

Value

Sequentially-numbered png files (one for each frame) and one mp4 file will be written to `out_dir`.

Note

Customizing plot elements with input argument ... The option to allow customization of plot elements with input argument ... provides a great deal of flexibility, but users will need to be familiar with each associated graphics functions (e.g., axis for timeline arguments). We expect that this will require some trial and error and that input argument `preview = TRUE` will be useful while exploring optional plot arguments.

Author(s)

Todd Hayden, Tom Binder, Chris Holbrook

Examples

```
## Not run:

# load detection data
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")
dtc <- read_glatos_detections(det_file)

# take a look
head(dtc)

# load receiver location data
rec_file <- system.file("extdata",
  "sample_receivers.csv", package = "glatos")
recs <- read_glatos_receivers(rec_file)

# call with defaults; linear interpolation
pos1 <- interpolate_path(dtc)

# make frames, preview the first frame
myDir <- paste0(getwd(),"/frames1")
make_frames(pos1, recs=recs, out_dir=myDir, preview = TRUE)

# make frames but not animation
myDir <- paste0(getwd(),"/frames2")
make_frames(pos1, recs=recs, out_dir=myDir, animate = FALSE)

# make sequential frames, and animate.
# change default color of fish markers to red and change marker and size.
myDir <- paste0(getwd(), "/frames3")
make_frames(pos1, recs=recs, out_dir=myDir, animate = TRUE,
            ani_name = "animation3.mp4", col="red", pch = 16, cex = 3)

# make sequential frames, animate, add 5-day tail
myDir <- paste0(getwd(), "/frames4")
make_frames(pos1, recs=recs, out_dir=myDir, animate = TRUE,
            ani_name = "animation4.mp4", tail_dur=5)

# make animation, remove frames.
myDir <- paste0(getwd(), "/frames5")
make_frames(pos1, recs=recs, out_dir=myDir, animate = TRUE,
            ani_name = "animation5.mp4", frame_delete = TRUE)


## End(Not run)
```

---

make_transition *Deprecated Create transition layer from polygon shapefile*

Description

Create transition layer for interpolate_path from polygon shapefile.

Usage

```
make_transition(
  in_file,
  output = "out.tif",
  output_dir = NULL,
  res = c(0.1, 0.1),
  invert = FALSE,
  all_touched = TRUE
)
```

Arguments

in_file	A sf, SpatialPolygonsDataFrame object, or a character string with file path to polygon shapefile (with extension of *.shp). Default arguments assume the polygon represents a water body. If the polygon represents a land mass, then invert will need to be set to TRUE.
output	character, name of output file with .tif extension
output_dir	character, directory where output file will be written. If NULL (default), then files will be written to temporary directory that will be deleted after R session is closed (see tempdir).
res	two element vector that specifies the x and y dimension of output raster cells. Units of res are same as input shapefile.
invert	logical. Passes into gdal_rasterize. If true, it will return the inverse of the raster object it would normally return. This can be useful if the polygon passed represents a landmass rather than a body of water, such as the ones generated by GADM.
all_touched	logical. If TRUE (default) then any pixel touched by polygon 'in_file' will be coded as water in the output. Alternatively, pixel must be at least 50% covered by polygon to be coded as water.

Details

make_transition uses gdal_rasterize to convert a polygon shapefile into a raster layer and geocorrected transition layer interpolate_path. Raster cell values on land = 0 and water = 1. Function also writes a geotiff file (*.tif) of the input shapefile to the ouput directory. Both raster layer and geotif output have the same extents and geographic projection as input shapefile. Function requires that gdal is working on computer. To determine if gdal is installed on your computer, see gdal_rasterize.

Returned objects will be projected in longlat WGS84 (i.e., CRS("+init=epsg:4326"). If the input object is not recognizable in epsg:4326 then transformation will be attempted and a warning will tell the user this was done. Input shapefile must include an optional *.prj file that specifies the geographic projection.

Output transition layer is corrected for projection distortions using `gdistance::geoCorrection`. Adjacent cells are connected by 16 directions and transition function returns 0 (land) for movements between land and water and 1 for all over-water movements.

Value

A list with two elements:

transition a geo-corrected transition raster layer where land = 0 and water=1 (see `gdistance`)

rast rasterized input layer of class `raster`

Additonally, rasterized version of input shapefile (*.tif extension) is written to computer at `output_dir`

Note

This function has been deprecated and will be removed from the next version of `glatos`. Use `make_transition3` instead.

Author(s)

Todd Hayden, Tom Binder, Chris Holbrook

Examples

```
## Not run:

#Example 1 - read from sf polygon
# use example polygon for Great lakes

library(sf) #for loading great_lakes_polygon
library(raster) # for plotting rasters

# Get polygon of the Great Lakes
data(great_lakes_polygon) #glatos example data; an sf polygons object

# Make transition layer
tst <- make_transition(great_lakes_polygon, res = c(0.1, 0.1))

# plot raster layer
# notice land = 1, water = 0
plot(tst$rast)

#compare to polygon
plot(sf::st_geometry(great_lakes_polygon), add = TRUE)


#Example 2 - read from SpatialPolygonsDataFrame
# use example polygon for Great lakes

library(raster) # for plotting rasters

#get polygon of the Great Lakes
```

```
data(greatLakesPoly) #glatos example data; a SpatialPolygonsDataFrame

# make_transition layer
tst <- make_transition(greatLakesPoly, res = c(0.1, 0.1))

# plot raster layer
# notice land = 1, water = 0
plot(tst$rast)

#compare to polygon
plot(greatLakesPoly, add = TRUE)

# increase resolution and repeat if needed

#-----------------------------------------
#Example 3 - read from ESRI Shapefile
# path to polygon shapefile
poly <- system.file("extdata", "shoreline.zip", package = "glatos")
poly <- unzip(poly, exdir = tempdir())

# make_transition layer
tst <- make_transition(poly[grepl("*.shp", poly)], res = c(0.1, 0.1))

# plot raster layer
# notice land = 0, water = 1
raster::plot(tst$rast)

# plot transition layer
raster::plot(raster::raster(tst$transition))

# increase resolution- this may take some time...
tst1 <- make_transition(poly[grepl("*.shp", poly)], res = c(0.01, 0.01))

# plot raster layer
raster::plot(tst1$rast)

# plot transition layer
raster::plot(raster::raster(tst1$transition))

## End(Not run)
```

`make_transition2` *Deprecated Create transition layer from a spatial polygon*

Description

Create transition layer for interpolate_path from SpatialPolygonsDataFrame.

Usage

```
make_transition2(
  poly,
  res = c(0.1, 0.1),
  extent_out = NULL,
  x_lim = NULL,
  y_lim = NULL
)
```

Arguments

poly	A spatial polygon object of class SpatialPolygonsDataFrame.
res	two element vector that specifies the x and y dimension of output raster cells. Units of res are same as input polygon.
extent_out	An optional Extent object (see extent) that determines the extent of the output objects. Output extent will default to extent of input object poly if extent_out, and x_lim/y_lim are NULL (default).
x_lim	An optional two-element vector with extents of x axis.
y_lim	An optional two-element vector with extents of x axis.

Details

make_transition uses rasterize to convert a SpatialPolygonsDataFrame into a raster layer, and geo-corrected transition layer transition. Raster cell values on land = 0 and water = 1.

output transition layer is corrected for projection distortions using geoCorrection. Adjacent cells are connected by 16 directions and transition function returns 0 (land) for movements between land and water and 1 for all over-water movements.

Value

A list with two elements:

transition a geo-corrected transition raster layer where land = 0 and water=1 (see gdistance)

rast rasterized input layer of class raster

Note

This function has been deprecated and will be removed from the next version of glatos. Use make_transition3 instead.

Author(s)

Todd Hayden, Tom Binder, Chris Holbrook

See Also

make_transition

Examples

```
library(raster) # for plotting rasters

# get polygon of the Great Lakes
data(greatLakesPoly) #glatos example data; a SpatialPolygonsDataFrame

# make_transition layer
tst <- make_transition2(greatLakesPoly, res = c(0.1, 0.1))

# plot raster layer
# notice land = 1, water = 0
plot(tst$rast)

# plot transition layer
plot(raster(tst$transition))

## Not run:
# increase resolution- this may take some time...
tst1 <- make_transition2(greatLakesPoly, res = c(0.01, 0.01))

# plot raster layer
plot(tst1$rast)

# plot transition layer
plot(raster(tst1$transition))

## End(Not run)
```

---

make_transition3 *Create transition layer from polygon shapefile*

Description

Create transition layer for interpolate_path from polygon shapefile.

Usage

```
make_transition3(poly, res = c(0.1, 0.1), receiver_points = NULL, epsg = 3175)
```

Arguments

poly	A spatial polygon object of class SpatialPolygonsDataFrame or a sf::sf() object with a geometry column of polygon and/or multipolygon objects.
res	two element vector that specifies the x and y dimension of output raster cells. Units of res are same as input shapefile.

`receiver_points`
: A SpatialPointsDataFrame object that contains coordinates of receivers dataset or a "glatos_receivers" object.

`epsg`
: coordinate reference code that describes projection used for map calculation and rasterization. Defaults to NAD83/Great Lakes and St. Lawrence Albers.

Details

`make_transition` uses fasterize to convert a polygon shapefile into a raster layer and geo-corrected transition layer interpolate_path. Raster cell values on land equal 1 cells in water equal 0. Output is a two-object list containing the raster layer and transition layer. Both objects have the same extents and geographic projection as input shapefile.

@details If receiver_points is provided, any receiver not in water is buffered by the distance from the receiver to the nearest water. This allows all receivers to be coded as in water if the receiver is on land.

Poly object is transformed into planer map projection specified by epsg argument for calculation of transition object if receiver_points is provided. Output is projected to WGS84 (epsg- 4326).

output transition layer is corrected for projection distortions using `gdistance::geoCorrection`. Adjacent cells are connected by 16 directions and transition function returns 0 (land) for movements between land and water and 1 for all over-water movements.

Value

A list with two elements:

transition a geo-corrected transition raster layer where land = 0 and water=1 (see `gdistance`)

rast rasterized input layer of class `raster`

Additonally, rasterized version of input shapefile (*.tif extension) is written to computer at `output_dir`

Author(s)

Todd Hayden, Tom Binder, Chris Holbrook

Examples

```
#Example 1 - read from SpatialPolygonsDataFrame
# use example polygon for Great lakes

library(sp) #for loading greatLakesPoly
library(raster) # for plotting rasters

#get polygon of the Great Lakes
data(greatLakesPoly) #glatos example data; a SpatialPolygonsDataFrame

# make_transition layer
tst <- make_transition3(greatLakesPoly, res = c(0.1, 0.1))

# plot raster layer
# notice land = 1, water = 0
```

```
plot(tst$rast)

#compare to polygon
plot(greatLakesPoly, add = TRUE)

#Example 2 - read from ESRI Shapefile and include receiver file
# to account for any receivers outside of great lakes polygon

# path to polygon shapefile
poly <- system.file("extdata", "shoreline.zip", package = "glatos")
poly <- unzip(poly, exdir = tempdir())
poly <- sf::st_read(poly[grepl("*.shp", poly)])

# read in glatos receivers object
rec_file <- system.file("extdata", "sample_receivers.csv", package="glatos")
recs <- read_glatos_receivers(rec_file)

# change a coordinate to on-land to show impact...
recs[1, "deploy_lat"] <- recs[1,"deploy_lat"]+4

# make_transition layer
tst <- make_transition3(poly, res = c(0.1, 0.1), receiver_points = recs)

# plot raster layer
# notice the huge circle rasterized as "water"  north of Lake Superior.
# This occurred because we had a "receiver" deployed at that locations
raster::plot(tst$rast)
points(recs$deploy_long, recs$deploy_lat, col = "red", pch = 20)

# plot transition layer
raster::plot(raster::raster(tst$transition))

Example 3- transition layer of Lake Huron only with receivers

# read polygon shapefile
poly <- system.file("extdata", "shoreline.zip", package = "glatos")
poly <- unzip(poly, exdir = tempdir())
poly <- sf::st_read(poly[grepl("*.shp", poly)])


# transform to great lakes projection
poly <- sf::st_transform(poly, crs = 3175)

# set attribute-geometry relationship to constant.
# this avoids error when cropping
sf::st_agr(poly) = "constant"

# crop Great lakes polygon file
poly <- sf::st_crop(x = poly, xmin = 829242.55, ymin = 698928.27,
                              xmax = 1270000.97, ymax = 1097196.15)

# read in glatos receivers object
rec_file <- system.file("extdata", "sample_receivers.csv", package="glatos")
```

```
recs <- read_glatos_receivers(rec_file)

# extract receivers in "HECWL" project
# all receiver stations except one is in Lake Huron
recs <- recs[recs$glatos_project == "HECWL",]

# remove two stations not in Lake Huron
recs <- recs[!recs$glatos_array %in% c("MAU","LVD"),]

# convert recs to simple feature  object (sf)
recs <- sf::st_as_sf(recs, coords = c("deploy_long", "deploy_lat"), crs = 4326 )

# transform to great lakes projection
recs <- sf::st_transform(recs, crs = 3175)

# check by plotting
plot(sf::st_geometry(poly), col = NA)
plot(sf::st_geometry(recs), col = "red", add = TRUE)

# create slightly higher resolution transition layer
tst1 <- make_transition3(poly, res = c(0.01, 0.01), receiver_points = recs)

# plot raster layer
raster::plot(tst1$rast)
 plot(sf::st_transform(sf::st_geometry(recs), crs = 4326), add = TRUE, col = "red", pch = 20)

# plot transition layer
raster::plot(raster::raster(tst1$transition))
```

---

make_video *Create video from sequence of still images*

Description

Stitch a sequence of images into a video animation. A simple wrapper for av::av_encode_video.

Usage

```
make_video(
  input_dir = getwd(),
  input_ext = ".png",
  output = "animation.mp4",
  duration = NULL,
  start_frame = 1,
  end_frame = NULL,
  size = NULL,
  overwrite = FALSE,
```

```
  verbose = FALSE,
  ...
)
```

Arguments

input_dir	directory containing images, default is working directory.
input_ext	character, file extension of images to be stitched into a video. All images must have same extension, width, and height. Each imaged will be positioned in the video in alphabetical order by image file name.
output	character, output video file name. See details.
duration	integer, output video duration in seconds. If NULL (default) then this will be determined by the number of input frames and the framerate (default is 24 frames per second). E.g., a video containing 240 frames at default 24 fps will be 10 seconds long. See details.
start_frame	integer, start frame. Defaults to start=1.
end_frame	integer, end frame. Defaults to end_frame = NULL (i.e., last frame).
size	integer vector with width and height of output video in pixels. Ignored if vfilter is passed via ....
overwrite	logical, overwrite existing output file? (default = FALSE)
verbose	logical, show output from av::av_encode_video? Default = FALSE.
...	optional arguments passed to av::av_encode_video. Such as framerate, vfilter, codec.

Details

This function was overhauled in glatos v 0.4.1 to simplify inputs and to no longer require an external program (ffmpeg.exe). As a result input arguments have changed, as described above. Starting with glatos v 0.7.0, any calls to make_video using the arguments from glatos v 0.4.0 or earlier will fail.

make_video is a simple wrapper of av::av_encode_video. It is intended to allow creation of videos from images (frames) created by glatos::make_frames as simple as possible. More advanced features of av, can be used by including any argument of av::av_encode_video in the call to make_video, or by calling av::av_encode_video directly. More information about the av package is available at https://cran.r-project.org/web/packages/av/index.html and https://docs.ropensci.org/av/.

A directory of sequenced image files (.png, .jpeg) are passed to input_dir argument. The input_ext argument specifies the type of files to be stitched into a video. The images passed to the function must all have the same size, height, and format.

Function can create .mp4, .mov, .mkv, .flv .wmv, or .mpeg animations. Format of created animation is determined by file extension of output.

If start_frame or end_frame are specified, then only frames within the specified range will be included in the output video.

If duration is specified, then the output framerate will be determined by the number of input frames and the framerate (default is 24 frames per second). E.g., a video of 10 second duration containing 240 frames will have an output frame rate of 24 fps. In some cases (when number of frames is small) the number of frames may not divide evently into the specified duration, so the output duration may

differ from that specified. If the output frame rate exceeds 30 fps, then a warning will alert the user that some individual frame content may not be visible to users. Video duration may also be controlled by setting the `framerate` argument of av::av_encode_video. See ... above.

Value

One video animation will be written disk and the path and file name will be returned.

Author(s)

Todd Hayden, Chris Holbrook

Examples

```
## Not run:

# load frames
frames <- system.file("extdata", "frames", package = "glatos")

# make .mp4 video
make_video(input_dir = frames,
           input_ext = ".png",
           output = file.path(tempdir(), "animation1.mp4"))

# set duration to 10 seconds
make_video(input_dir = frames,
           input_ext = ".png",
           output = file.path(tempdir(), "animation2.mp4"),
           duration = 10)

# set size of ouput video
make_video(input_dir = frames,
           input_ext = ".png",
           output = file.path(tempdir(), "animation3.mp4"),
           size = c(320, 240))

# start animation on frame 10, end on frame 20
make_video(input_dir = frames,
           input_ext = ".png",
           output = file.path(tempdir(), "animation_4.mp4"),
           start_frame = 10,
           end_frame = 20)

# make move backwards- start animation of frame 20 and end on frame 10
make_video(input_dir = frames,
           input_ext = ".png",
           output = file.path(tempdir(), "animation_5.mp4"),
           start_frame = 20,
           end_frame = 10)

# make .wmv video
make_video(input_dir = frames,
```

```
            input_ext = ".png",
            output = file.path(tempdir(), "animation1.wmv"))


#--- Examples using more advanced features of av_encode_video

# resize output video by specifying a scale filter
make_video(input_dir = frames,
            input_ext = ".png",
            output = file.path(tempdir(), "animation_6.mp4"),
            vfilter = "scale=320:240")

# slow the video by 10 times
make_video(input_dir = frames,
            input_ext = ".png",
            output = file.path(tempdir(), "animation_7.mp4"),
            vfilter = "setpts=10*PTS")

# slow video by 10 times and scale to 320x240 resolution
make_video(input_dir = frames,
            input_ext = ".png",
            output = file.path(tempdir(), "animation_8.mp4"),
            vfilter = "scale=320:240, setpts=10*PTS")


## End(Not run)
```

`min_lag` *Calculate 'min_lag' for identifying potential false positive detections*

Description

Calculate minimum time interval (min_lag) between successive detections and add to detection data set for identifying potential false detections.

Usage

```
min_lag(det)
```

Arguments

`det`	A `glatos_detections` object (e.g., produced by read_glatos_detections). *OR* a data frame containing detection data with the following columns: **detection_timestamp_utc** Detection timestamps; MUST be of class POSIXct. **transmitter_codespace** A character string with transmitter code space (e.g., "A69-1061" for Vemco PPM coding"). **transmitter_id** A character string with transmitter ID code (e.g., "1363" for Vemco PPM coding"). **receiver_sn** A character vector with unique receiver serial number.

Details

min_lag is loosely based on the the "short interval" described by Pincock (2012) and replicates the min_lag column in the standard glatos detection export file. In this case (GLATOS), min_lag is defined for each detection as the shortest interval (in seconds) between either the previous or next detection (whichever is closest) of the same transmitter code (defined here as combination of transmitter_codespace and transmitter_id) on the same receiver.

A new column (min_lag) is added to the input dataframe that represents the time (in seconds) between the current detection and the next detection (either before or after) of the same transmitter on the same receiver. This function replicates the 'min_lag' column included in the standard glatos export.

Value

A column min_lag (defined above) is added to input object.

Author(s)

Chris Holbrook, Todd Hayden, Angela Dini

References

Pincock, D.G., 2012. False detections: what they are and how to remove them from detection data. Vemco Division, Amirix Systems Inc., Halifax, Nova Scotia.
http://www.vemco.com/pdf/false_detections.pdf

See Also

false_detections

Examples

```
# load example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")
det <- read_glatos_detections(det_file)

# rename existing min_lag column
colnames(det)[colnames(det) == "min_lag"] <- "min_lag.x"

# calculate min_lag
det <- min_lag(det)

head(det)
```

otn_aat_animals *Example animal data from the OTN ERDDAP*

Description

An example animal data file from the OTN ERDDAP

Usage

```
system.file("extdata", "otn_aat_animals.csv", package = "glatos")
```

Format

CSV

Filename

otn_aat_animals.csv

Source

Ryan Gosse, Ocean Tracking Network

otn_aat_receivers *Example station data from the OTN ERDDAP*

Description

An example receiver station data file from the OTN ERDDAP

Usage

```
system.file("extdata", "otn_aat_receivers.csv", package = "glatos")
```

Format

CSV

Filename

otn_aat_receivers.csv

Source

Ryan Gosse, Ocean Tracking Network

otn_aat_tag_releases *Example tag release data from the OTN ERDDAP*

Description

An example tag release data file from the OTN ERDDAP

Usage

```
system.file("extdata", "otn_aat_tag_releases.csv", package = "glatos")
```

Format

CSV

Filename

otn_aat_tag_releases.csv

Source

Ryan Gosse, Ocean Tracking Network

point_offset *Identify new location based on distance and bearing from another*

Description

Calculates latitude and longitude for new point that is x meters away at bearing y from a geographic location (Longitude, Latitude). uses "destPoint" function from "geosphere" package and calculations are based on great circle distances.

Usage

```
point_offset(
  lon = NA,
  lat = NA,
  offsetDist = NA,
  offsetDir = NA,
  distUnit = "m"
)
```

Arguments

lon	vector of longitudes (dd) to calculate offset points
lat	vector of latitudes (dd) to calculate offset points
offsetDist	vector of distances to calculate offset point (meters or feet)
offsetDir	vector of directions to calculate point from starting point. Options are NA,"N", "NNE", "NE", "ENE", "E", "ESE", "SE", "SSE", "S", "SSW", "SW", "WSW", "W", "WNW", "NW", "NNW"
distUnit	specify meters or ft ("m" or "ft")

Examples

```
lat <- rep(44.0, 17)
lon <- rep(-83.0, 17)

offsetDir <- c(NA,"N", "NNE", "NE", "ENE", "E", "ESE", "SE", "SSE", "S",
"SSW", "SW", "WSW", "W", "WNW", "NW", "NNW")

offsetDist <- seq(100, 1700, by = 100)
distUnit <- 'm'

point_offset(lon, lat, offsetDist, offsetDir, distUnit)
```

---

position_heat_map *Position Heat Maps*

Description

Create heat maps to display the spatial distribution of acoustic telemetry positions. Most useful when used on data with high spatial resultion, such as VPS positional telemetry data.

Usage

```
position_heat_map(
  positions,
  projection = "LL",
  fish_pos_int = "fish",
  abs_or_rel = "absolute",
  resolution = 10,
  interval = NULL,
  x_limits = NULL,
  y_limits = NULL,
  utm_zone = NULL,
  hemisphere = "N",
  legend_gradient = "y",
  legend_pos = c(0.99, 0.2, 1, 0.8),
```

```
  output = "plot",
  folder = "position_heat_map",
  out_file = NULL
)
```

Arguments

positions	A dataframe containing detection data with at least the following 4 columns: DETECTEDID Individual animal identifier; character. DATETIME Date-time stamps for the positions (MUST be of class 'POSIXct') LAT Position latitude. LON Position longitude.
projection	A character string indicating if the coordinates in the 'positions' dataframe are geographic (projection = "LL") or projected/Cartesian(projection = "UTM"). Used to convert coordinates between latitude/longitude in decimal degrees ("LL"; e.g., 45.98753) and UTM. Valid arguments are "LL" (latitude/longitude) and "UTM". If projection=="UTM", then utm_zone and 'hemisphere arguments must also be supplied.
fish_pos_int	A character string indicating whether output will display number of fish or number of positions occuring in each cell of the grid. Valid arguments are c("fish", "positions", "intervals"). Default is "fish". If fish_pos_interval == "intervals", then argument "interval" must be supplied.
abs_or_rel	A character string indicating whether output will display values as absolute value (i.e, the actual number of fish, positions, or intervals) or as relative number (relative to total number of fish detected). Valid arguments are c("absolute", "relative"). Default is "absolute".
resolution	A numeric value indicating the spatial resolution (in meters) of the grid system used to make the heat maps. Default is 10 m.
interval	A numeric value indicating the duration (in seconds) of time bin (in seconds) for use in calculating number of intervals fish were resident in a grid cell (i.e., a surrogate for amount of time spent in each cell of the grid). If interval==NULL (default), than raw number of positions is calculated. This value is only used when fish_pos_int == "intervals'.
x_limits	An optional 2-element numeric containing limits of x axis. If x_limits == NULL (default), then it is determined from the extents of the data.
y_limits	An optional 2-element numeric containing limits of y axis. If y_limits == NULL (default), then it is determined from the extents of the data.
utm_zone	An interger value between 1 and 60 (inclusive) indicating the primary UTM zone of the detection data. Required and used only when projection == "UTM". Default is NULL (i.e. assumes detection data are in projection == LL by default).
hemisphere	A character string indicating whether detection data are in the northern or southern hemisphere. Required and used only when projection == "UTM". Valid values are c("N", "S"). Default is "N".

`legend_gradient`
: A character string indicating the orientation of the color legend; "y" = vertical, "x" = horizontal, "n" indicates that no legend should be drawn. Default is "y".

`legend_pos`
: A numeric vector indicating the location of the color legend as a portion of the total plot area (i.e., between 0 and 1). Only used if 'legend_gradient" in not "n". Default is c(0.99, 0.2, 1.0, 0.8), which puts the legend along the right hand side of the plot.

`output`
: An optional character string indicating how results will be displayed visually. Options include: 1) a plot in the R device window ("plot"), 2) a .png image file ("png"), or 3) a .kmz file ("kmz") for viewing results as an overlay in Google Earth. Accepted values are c("plot", "png", "kmz"). Default value is "plot".

`folder`
: A character string indicating the output folder. If path is not specified then `folder` will be created in the working directory. Default is "position_heat_map".

`out_file`
: A character string indicating base name of output files (if `output` = `"png"` or `"kmz"`). If `out_file` is a path, all but last part is ignored (via `basename`). Any file extension is also ignored (via `tools::file_path_sans_ext`).

Details

When an 'interval' argument is supplied, the number of unique fish x interval combinations that occurred in each grid cell is calculated instead of raw number of positions. For example, in 4 hours there are a total of 4 1-h intervals. If fish 'A' was positioned in a single grid cell during 3 of the 4 intervals, than the number of intervals for that fish and grid combination is 3. Intervals are determined by applying the findInterval function (base R) to a sequence of timestamps (class: POSIXct) created using seq(from = min(positions, DATETIME), to = min(positions, DATETIME), by = interval), where interval is the user-assigned interval duration in seconds. Number of intervals is a more robust surrogate than number of positions for relative time spent in each grid in cases where spatial or temporal variability in positioning probability are likely to significantly bias the distribution of positions in the array.

Calculated values (i.e., fish, positions, intervals) can be returned as absolute or relative, which is specified using the abs_or_rel argument; "absolute" is the actual value, "relative" is the absolute value divided by the total number of fish appearing in the 'positions' dataframe. Units for plots: fish = number of unique fish (absolute) or \ 'positions' dataframe (relative); positions = number of positions (absolute) or mean number of positions per fish in 'positions' dataframe (relative); intervals = number of unique fish x interval combinations (absolute) or mean number of unique fish x interval combinations per fish in 'positions' dataframe (relative).

Value

A list object containing 1) a matrix of the calculated values (i.e., fish, positions, intervals), with row and column names indicating location of each grid in UTM, 2) a character string specifying the UTM zone of the data in the matrix, 3) the bounding box of the data in UTM, 4) and the bounding box of the data in latitude (Y) and longitude (X), 5) a character string displaying the function call (i.e., a record of the arguments passed to the function).

In addition, the user specifies an image output for displaying the heat map. Options are a "plot" (displayed in R), "png" (png file saved to specified folder), and "kmz" for viewing the png image as an overlay in Google Earth (kmz file saved to specified folder).

Author(s)

Thomas R. Binder

Examples

```
data(lamprey_tracks)
phm <- position_heat_map(lamprey_tracks)
```

---

prepare_deploy_sheet *Loads the OTN receiver deployment metadata sheet to prepare it for use in* convert_otn_to_att

---

Description

Loads the OTN receiver deployment metadata sheet to prepare it for use in convert_otn_to_att

Usage

```
prepare_deploy_sheet(
  path,
  header_line = 5,
  sheet_name = 1,
  combine_arr_stn = TRUE
)
```

Arguments

path	the path to the deployment sheet
header_line	what line the headers are on
sheet_name	the sheet name or number containing the metadata
combine_arr_stn	whether or not to to join the station and array columns. Format depends on OTN node

Details

The function takes the path to the deployment sheet, what line to start reading from, and what sheet in the excel file to use. It converts column names to be used by convert_otn_to_att.

Value

a data.frame created from the excel file.

Author(s)

Ryan Gosse

Examples

```
#-------------------------------------------------
# EXAMPLE #1 - loading from NSBS simplified Deployments

library(glatos)
deploy_path <- system.file("extdata", "hfx_deploy_simplified.xlsx",
                           package = "glatos")

deploy <- prepare_deploy_sheet(deploy_path,
                               header_line = 1,
                               sheet_name = 1)
```

`prepare_tag_sheet` *Loads the OTN tagging metadata sheet to prepare it for use in* `convert_otn_to_att`

Description

Loads the OTN tagging metadata sheet to prepare it for use in `convert_otn_to_att`

Usage

```
prepare_tag_sheet(path, header_line = 5, sheet_name = 2)
```

Arguments

`path`	the path to the tagging sheet
`header_line`	what line the headers are on
`sheet_name`	the sheet name or number containing the metadata

Details

The function takes the path to the tagging sheet, what line to start reading the headers from, and what sheet in the excel file to use. It converts column names to be used by `convert_otn_to_att`.

Value

a data.frame created from the excel file.

Author(s)

Ryan Gosse

Examples

```
#-------------------------------------------------
# EXAMPLE #1 - loading from NSBS tagging

library(glatos)
tag_path <- system.file("extdata", "otn_nsbs_tag_metadata.xls",
                        package = "glatos")

tags <- prepare_tag_sheet(tag_path, 5, 2)
```

---

range_detection *Detection range data set*

Description

Sample detection range data set from Lake Superior.

Usage

```
range_detection
```

Format

A data frame with 58309 rows and 30 variables

Details

Data from a stationary detection range test conducted in 2018. Data are in standard GLATOS detection export format and are intened to accompany detecton range analysis vignette.

Source

F. Zomer, T. Hayden

---

raw_lamprey_workbook *Raw GLATOS Workbook from St. Marys River Sea Lamprey project*

Description

A completed GLATOS workbook from St. Marys River Sea Lamprey project.

Usage

```
system.file("extdata", "SMRSL_GLATOS_20140828.xlsm", package="glatos")
```

Format

A macro-enabled Microsoft Excel workbook file (*.xlsm) with six worksheets:

project project code, principal investigator and contact

locations descriptions of receiver array locations

proposed proposed receiver deployment locations and dates

deployment receiver deployment data (what, where, when, how)

recovery receiver recovery data (what, where, when, how)

tagging animal collection, tagging, and recovery data

Filename

SMRSL_GLATOS_20140828.xlsm

Author(s)

Chris Holbrook

Source

http://glatos.glos.us/home/project/SMRSL

`raw_walleye_detections` *Zipped GLATOS detection file from Huron Erie Corridor Walleye project*

Description

An example detection file

Usage

```
system.file("extdata", "walleye_detections.zip", package="glatos")
```

Format

A zipped walleye detection file in detection file format 1.3:

Filename

walleye_detections.zip

Author(s)

Todd Hayden

Source

http://glatos.glos.us/home/project/HECWL

---

read_glatos_detections *Read data from a GLATOS detection file*

Description

Read data from a standard GLATOS detection (csv) file and return a data.frame of class `glatos_detections`.

Usage

```
read_glatos_detections(det_file, version = NULL)
```

Arguments

`det_file`	A character string with path and name of detection file in standard GLATOS format (*.csv). If only file name is given, then the file must be located in the working directory. File must be a standard GLATOS file (e.g., *xxxxx_detectionsWithLocs_yyyymmdd_hhmmss.c* submitted via GLATOSWeb Data Portal https://glatos.glos.us.
`version`	An optional character string with the glatos file version number. If NULL (default value) then version will be determined by evaluating file structure. The only allowed values currently are `NULL` and `"1.3"`. Any other values will trigger an error.

Details

Data are loaded using fread and timestamps are coerced to POSIXct using fastPOSIXct. All times must be in UTC timezone per GLATOS standard.

Column `animal_id` is considered a required column by many other functions in this package, so it will be created if any records are `NULL`. When created, it will be constructed from `transmitter_codespace` and `transmitter_id`, separated by '-'.

Value

A data.frame of class `glatos_detections`.

Author(s)

C. Holbrook <cholbrook@usgs.gov>

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")

#note that code above is needed to find the example file
#for real glatos data, use something like below
#det_file <- "c:/path_to_file/HECWL_detectionsWithLocs_20150321_132242.csv"

det <- read_glatos_detections(det_file)
```

`read_glatos_receivers` *Read data from a GLATOS receiver location file*

Description

Read data from a standard GLATOS receiver location (csv) file and return a data.frame of class `glatos_receivers`.

Usage

```
read_glatos_receivers(rec_file, version = NULL)
```

Arguments

`rec_file`	A character string with path and name of receiver location file in standard GLATOS format (*.csv). If only file name is given, then the file must be located in the working directory. File must be a standard GLATOS file (e.g., *GLATOS_receiverLocations_yyyymmdd_xx* obtained from GLATOSWeb Data Portal `http://glatos.glos.us`.
`version`	An optional character string with the GLATOS file version number. If NULL (default value) then version will be determined by evaluating file structure. The only allowed values currently are `NULL` and `"1.0"`. Any other values will trigger an error.

Details

Data are loaded using fread and timestamps are coerced to POSIXct using fastPOSIXct. All timestamps must be 'YYYY-MM-DD HH:MM' format and in UTC timezone per GLATOS standard.

Value

A data.frame of class `glatos_receivers`.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
#get path to example receiver_locations file
rec_file <- system.file("extdata",
  "sample_receivers.csv", package = "glatos")

#note that code above is needed to find the example file
#for real glatos data, use something like below
#rec_file <- "c:/path_to_file/GLATOS_receiverLocations_20150321_132242.csv"

rcv <- read_glatos_receivers(rec_file)
```

---

read_glatos_workbook *Read data from a GLATOS project workbook*

Description

Read data from a GLATOS project workbook (xlsm file) and return a list of class `glatos_workbook`.

Usage

```
read_glatos_workbook(wb_file, read_all = FALSE, wb_version = NULL)
```

Arguments

`wb_file`	A character string with path and name of workbook in standard GLATOS format (*.xlsm). If only file name is given, then the file must be located in the working directory. File must be a standard GLATOS file (e.g., *xxxxx_GLATOS_YYYYMMDD.xlsm*) submitted via GLATOSWeb Data Portal http://glatos.glos.us.
`read_all`	If TRUE, then all columns and sheets (e.g., user-created "project-specific" columns or sheets) in the workbook will be imported. If FALSE (default value) then only columns and sheets in the standard GLATOS workbook will be imported (project-specific columns will be ignored.)
`wb_version`	An optional character string with the workbook version number. If NULL (default value) then version will be determined by evaluating workbook structure. Currently, the only allowed values are `NULL` and `"1.3"`. Any other values will trigger an error.

Details

In the standard glatos workbook (v1.3), data in workbook sheets 'Deployment', 'Recovery', and 'Location' are merged on columns 'GLATOS_PROJECT', 'GLATOS_ARRAY', 'STATION_NO', 'CONSECUTIVE_DEPLOY_NO', AND 'INS_SERIAL_NO' to produce the output data frame `receivers`. Data in workbook sheets 'Project' and 'Tagging' are passed through to new data frames named 'project' and 'animals', respectively, and data from workbook sheet 'Proposed' is not included in result. If `read_all = TRUE` then each sheet in workbook will be included in result.

Data are read from the input file using read_excel in the 'readxl' package. If `read_all = TRUE` then the type of data in each user-defined column (and sheet) will be 'guessed' by read_excel. Therefore, if `read_all = TRUE` then the structure of those columnns should be carefully reviewed in the result. See read_excel for details.

Column `animal_id` is considered a required column by many other functions in this package, so it will be created if any records are `NULL`. When created, it will be constructed from `tag_code_space` and `tag_id_code`, separated by '-'.

Timezone attribute of all timestamp columns (class `POSIXct`) in output will be "UTC" and all 'glatos-specific' timestamp and timezone columns will be omitted from result.

Value

A list of class `glatos_workbook` with three elements (described below) containing data from the standard GLATOS Workbook sheets. If `read_all = TRUE`, then additional elements will be added with names corresponding to non-standard sheet names.

metadata A list with data about the project and workbook.

animals A data frame of class `glatos_animals` with data about tagged animals.

receivers A data frame of class `glatos_receivers` with data about telemetry receivers.

Note

On warnings and errors about date and timestamp formats. Date and time columns are sometimes stored as text in Excel. When those records are loaded by this function, there are two possible outcomes.

1. If the records are formatted according to the GLATOS Data Dictionary specification (e.g., "YYYY-MM-DD" for dates and "YYYY-MM-DD HH:MM" for timestamps; see https:\glatos.glos.us) those records should be properly loaded into R, but the user is encouraged to verify that they were loaded correctly, so a warning points the user to those records in the workbook. Users may want to format as custom date in the workbook to avoid warnings in the future.

2. If the format of a date-as-text column is not consistent with GLATOS specification, then no data will be loaded and an error will alert the user to this condition.

   On cells with locked formatting in Excel: Occasionally the format of a cell in Excel will be locked. In those cases, it is sometimes possible to force date formatting in Excel by (1) highlighting the columns that need reformatting, (2) select 'Text-to-columns' in the 'Data' menu, (3) select 'Delimited' and 'next', (4) uncheck all delimiters and 'next', (5) choose 'Date: YMD' in the 'Column data format' box, and (6) 'Finish'.

Author(s)

C. Holbrook <cholbrook@usgs.gov>

See Also

read_excel

Examples

```
#get path to example GLATOS Data Workbook
wb_file <- system.file("extdata",
  "walleye_workbook.xlsm", package = "glatos")

#note that code above is needed to find the example file
#for real glatos data, use something like below
#wb_file <- "c:/path_to_file/HECWL_GLATOS_20150321.csv"

wb <- read_glatos_workbook(wb_file)
```

---

read_otn_deployments *Read data from a OTN deployment file*

Description

Read data from a standard OTN deployment (csv) file and return a data.frame of class `glatos_receivers`.

Usage

```
read_otn_deployments(
  deployment_file,
  deploy_date_col = "deploy_date",
  recovery_date_col = "recovery_date",
  last_download_col = "last_download"
)
```

Arguments

`deployment_file`

A character string with path and name of deployment file in OTN deployment format (*.csv). If only file name is given, then the file must be located in the working directory.

@param deploy_date_col A character string representing the column name containing deploy_date data. Defaults to "deploy_date".

@param recovery_date_col A character string representing the column name containing recovery_date. Defaults to "recovery_date."

@param last_download_col A character string representing the column name containing the last_download date. Defaults to "last_download."

Details

Data are loaded using fread package and timestamps are coerced to POSIXct using the fastPOSIXct. All times must be in UTC timezone per GLATOS standard.

Column names are changed to match GLATOS standard columns when possible. Otherwise, OTN columns and column names are retained.

Value

A data.frame of class glatos_receivers that includes OTN columns that do not map directly to GLATOS columns.

Author(s)

A. Nunes, <anunes@dal.ca>

Examples

```
#get path to example deployments file
deployment_file <- system.file("extdata", "hfx_deployments.csv",
                          package = "glatos")
dep <- read_otn_deployments(deployment_file)
```

---

read_otn_detections *Read data from a OTN detection file*

Description

Read data from a standard OTN detection (csv) file and return a data.frame of class glatos_detections.

Usage

```
read_otn_detections(det_file)
```

Arguments

det_file	A character string with path and name of detection file in OTN detection extract format (*.csv). If only file name is given, then the file must be located in the working directory.

Details

Data are loaded using fread package and timestamps are coerced to POSIXct using the fastPOSIXct. All times must be in UTC timezone per GLATOS standard.

Column names are changed to match GLATOS standard columns when possible. Otherwise, OTN columns and column names are retained.

Value

A data.frame of class `glatos_detections` that includes OTN columns that do not map directly to GLATOS columns.

Author(s)

A. Nunes, <anunes@dal.ca>

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "blue_shark_detections.csv",
                        package = "glatos")
det <- read_otn_detections(det_file)
```

---

read_vemco_tag_specs *Read telemetry transmitter (tag) specification data from a Vemco file*

Description

Read telemetry transmitter (tag) specification data from a file and return a list with tag specifications and tag operating schedule.

Usage

```
read_vemco_tag_specs(tag_file, file_format = NULL)
```

Arguments

`tag_file`	A character string with path and name of file in a supported standard format in quotes. If only file name is given, then the file must be located in the working directory.
`file_format`	A character string with the tag spec file format in quotes. If NULL (default value) then version will be determined by evaluating file structure. The only allowed values are `NULL` and `"vemco_xls"`. Any other values will trigger an error.

Details

The file format `vemco_xls` is a MS Excel file provided to tag purchasers by Vemco.

This function is not endorsed or supported by any transmitter manufacturer.

Value

A list containing two data frames with tag specifications and tag operating schedule.

A list element called `specs` is a data frame contains tag specifications data in 17 columns:

serial_number

manufacturer

model

id_count

code_space

id_code

n_steps

sensor_type

sensor_range

sensor_units

sensor_slope

sensor_intercept

accel_algorithm

accel_sample_rate

sensor_transmit_ratio

est_battery_life_days

battery_life_stat

A list element called `schedule` is a data frame containing tag operating shedule data in 11 columns:

serial_number

code_space

id_code

step

next_step

status

duration_days

power

min_delay_secs

max_delay_secs

accel_on_time_secs

Author(s)

C. Holbrook, <cholbrook@usgs.gov>

Examples

```
#get path to example Vemco tag spec file
spec_file <- system.file("extdata",
  "lamprey_tag_specs.xls", package = "glatos")
my_tags <- read_vemco_tag_specs(spec_file, file_format = "vemco_xls")
```

`real_sensor_values` *Add 'real'-scale sensor values to glatos detetections*

Description

Get transmitter sensor (e.g., depth, temperature) conversion parameters (e.g., intercept, slope) from a Vemco transmitter specification object (e.g., from read_vemco_tag_specs, calculate 'real'-scale values (e.g., depth in meters), and add real values to detection data in a new column.

Usage

```
real_sensor_values(det, tag_specs)
```

Arguments

`det` A `glatos_detections` object (e.g., produced by read_glatos_detections).

OR A data frame containing detection data with the following columns:

transmitter_codespace A character string with transmitter code space (e.g., "A69-1061" for Vemco PPM coding").

transmitter_id A character string with transmitter ID code (e.g., "1363" for Vemco PPM coding").

sensor_value A numeric sensor measurement (e.g., an integer for 'raw' Vemco sensor tags).

sensor_unit A character string with `sensor_value` units (e.g., "ADC" for 'raw' Vemco sensor tag detections).

`tag_specs` An object produced by read_vemco_tag_specs.

OR A data frame containing transmitter specification data with the following columns:

code_space A character string with transmitter code space (e.g., "A69-1061" for Vemco PPM coding").

id_code A character string with transmitter ID code (e.g., "1363" for Vemco PPM coding").

sensor_type A numeric sensor measurement (e.g., an integer for 'raw' Vemco sensor tags).

sensor_range A numeric with max. range of the sensor in 'real' units (e.g., "Meters" for Vemco depth tags).

sensor_units A character string with 'real'-scale units (e.g., "Meters" for 'raw' Vemco pressure tags).

The following columns are also required for **depth** and **temperature** sensors:

sensor_slope Slope parameter, for converting 'raw' (ADC) to 'real' measurements.

sensor_intercept Intercept parameter, for converting 'raw' (ADC) to 'real' measurements.

The following columns are also required for **acceleration** sensors:

accel_algorithm The algorithm used, accelerometers only.

accel_sample_rate Sample rate used, accelerometers only.

sensor_transmit_ratio Sensor transmit rate used, accelerometers only.

Details

Tag spec data are joined to detection data and then raw-scale sensor measurements are converted to real-scale using $sensor_value_real = sensor_intercept + (sensor_value * sensor_slope)$, where $sensor_value$ is in raw scale.

It is possible that `transmitter_codespace` and `transmitter_id` are not unique among transmitters, so users must ensure that the each combination of those columns occurs only once in `tag_specs` and is the correct record for the corresponding tags in `det`.

Value

The input data frame, data.table, or tibble with the following columns added (see column descriptions above):

- sensor_range
- sensor_units
- sensor_slope
- sensor_intercept
- accel_algorithm
- accel_sample_rate
- sensor_transmit_ratio
- sensor_value_real

Author(s)

Chris Holbrook, <cholbrook@usgs.gov>

Examples

```
#get path to example detection file
det_file <- system.file("extdata",
  "lamprey_detections.csv", package="glatos")

lamprey_detections <- read_glatos_detections(det_file)

#get path to example Vemco tag spec file
```

```
spec_file <- system.file("extdata",
  "lamprey_tag_specs.xls", package="glatos")

lamprey_tags <- read_vemco_tag_specs(spec_file, file_format = "vemco_xls")

#note use of '$specs' in tag_specs argument
dtc <- real_sensor_values(lamprey_detections, lamprey_tags$specs)

#now view records with sensor measurements
dtc[!is.na(dtc$sensor_value_real),]
```

`receiver_line_det_sim` *Simulate detection of acoustic-tagged fish crossing a receiver line*

Description

Estimate, by simulation, the probability of detecting an acoustic-tagged fish on a receiver line, given constant fish velocity (ground speed), receiver spacing, number of receivers, and detection range curve.

Usage

```
receiver_line_det_sim(
  vel = 1,
  delayRng = c(120, 360),
  burstDur = 5,
  recSpc = 1000,
  maxDist = 2000,
  rngFun,
  outerLim = c(0, 0),
  nsim = 1000,
  showPlot = FALSE
)
```

Arguments

`vel`	A numeric scalar with fish velocity in meters per second.
`delayRng`	A 2-element numeric vector with minimum and maximum delay (time in seconds from end of one coded burst to beginning of next)
`burstDur`	A numeric scalar with duration (in seconds) of each coded burst (i.e., pulse train).
`recSpc`	A numeric vector with distances (in meters) between receivers. The length of vector is N-1, where N is number of receivers. One receiver is simulated when `recSpc = NA` (default).
`maxDist`	A numeric scalar with maximum distance between tagged fish and any receiver during simulation (i.e., sets spatial boundaries)

rngFun	A function that defines detection range curve; must accept a numeric vector of distances and return a numeric vector of detection probabilities at each distance.
outerLim	A two-element numeric vector with space (in meters) in which simulated fish are allowed to pass to left (first element) and right (second element) of the receiver line.
nsim	Integer scalar with the number of crossings (fish) to simulate
showPlot	A logical scalar. Should a plot be drawn showing receivers and fish paths?

Details

Virtual tagged fish (N=nsim) are "swum" through a virtual receiver line. The first element of recSpc determines spacing between first two receivers in the line, and each subsequent element of recSpc determine spacing of subsequent receivers along the line, such that the number of receivers is equal to length(recSpc) + 1. Each fish moves at constant velocity (vel) along a line perpendicular to the receiver line. The location of each fish path along the receiver line is random (drawn from uniform distribution), and fish can pass outside the receiver line (to the left of the first receiver or right of last receiver) if outerLim[1] or outerLim[2] are greater than 0 meters. Each fish starts and ends about maxDist meters from the receiver line.

A simulated tag signal is transmitted every delayRng[1] to delayRng[2] seconds. At time of each transmission, the distance is calculated between the tag and each receiver, and rngFun is used to calculate the probability (p) that the signal was detected on each receiver. Detection or non-detection on each receiver is determined by a draw from a Bernoulli distribution with probability p.

Value

A data frame with one column:

detProb	The proportion of simulated fish that were detected more than once on any single receiver.

Author(s)

C. Holbrook <cholbrook@usgs.gov>

References

For application example, see:

Hayden, T.A., Holbrook, C.M., Binder, T.R., Dettmers, J.M., Cooke, S.J., Vandergoot, C.S. and Krueger, C.C., 2016. Probability of acoustic transmitter detections by receiver lines in Lake Huron: results of multi-year field tests and simulations. Animal Biotelemetry, 4(1), p.19.
https://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-016-0112-9

Examples

```
#EXAMPLE 1 - simulate detection on line of ten receivers

 #Define detection range function (to pass as rngFun)
```

```
 # that returns detection probability for given distance
  # assume logistic form of detection range curve where
  #   dm = distance in meters
  #   b = intercept and slope
 pdrf <- function(dm, b=c(5.5, -1/120)){
    p <- 1/(1+exp(-(b[1]+b[2]*dm)))
    return(p)
 }

 #preview detection range curve
 plot(pdrf(0:2000),type="l",ylab="Probability of detecting each coded burst",
xlab="Distance between receiver and transmitter")

 #Simulate detection using pdrf; default values otherwise
 dp <- receiver_line_det_sim(rngFun=pdrf)
 dp

 #Again with only 10 virtual fish and optional plot to see simulated data
 dp <- receiver_line_det_sim(rngFun=pdrf, nsim=10, showPlot=T) #w/ optional plot
 dp

 #Again but six receivers and allow fish to pass to left and right of line
 dp <- receiver_line_det_sim(rngFun=pdrf, recSpc=rep(1000,5),
 outerLim=c(1000, 1000), nsim=10, showPlot=T)
 dp

 #Again but four receivers with irregular spacing
 dp <- receiver_line_det_sim(rngFun=pdrf, recSpc=c(2000,4000,2000),
  outerLim=c(1000, 1000), nsim=10, showPlot=T)
 dp


#EXAMPLE 2 - summarize detection probability vs. receiver spacing

 #two receivers only, spaced 'spc' m apart
 #define scenarios where two receiver are spaced
 spc <- seq(100,5000, 100) #two receivers spaced 100, 200, ... 5000 m
 #loop through scenarios, estimate detection probability for each
 for(i in 1:length(spc)){
   if(i==1) dp <- numeric(length(spc)) #pre-allocate
   dp[i] <- receiver_line_det_sim(recSpc=spc[i], rngFun=pdrf)
 }
 cbind(spc,dp) #view results
 #plot results
 plot(spc, dp, type="o",ylim=c(0,1),
   xlab="distance between receivers in meters",
  ylab="proportion of virtual fish detected")
 # e.g., >95% virtual fish detected up to 1400 m spacing in this example


#EXAMPLE 3 - summarize detection probability vs. fish swim speed

 #define scenarios of fish movement rate
```

```
swim <- seq(0.1, 5.0, 0.1) #constant velocity
for(i in 1:length(swim)){
  if(i==1) dp <- numeric(length(swim)) #pre-allocate
  dp[i] <- receiver_line_det_sim(vel=swim[i], rngFun=pdrf)
}
cbind(swim,dp) #view results
#plot results
plot(swim, dp, type="o", ylim=c(0,1), xlab="fish movement rate, m/s",
 ylab="proportion of virtual fish detected")
# e.g., >95% virtual fish detected up to 1.7 m/s rate in this example
# e.g., declines linearly above 1.7 m/s


#EXAMPLE 4 - empirical detection range curve instead of logistic

#create data frame with observed det. efficiency (p) at each distance (x)
edr <- data.frame(
  x=c(0,363,444,530,636,714,794,889,920), #tag-receiver distance
  p=c(1,1,0.96,0.71,0.67,0.75,0.88,0.21,0)) # detection prob

#now create a function to return the detection probability
# based on distance and linear interpolation within edr
# i.e., estimate p at given x by "connecting the dots"
edrf <- function(dm, my.edr=edr) {
  p <- approx(x=my.edr$x,y=my.edr$p,xout=dm, rule=2)$y
  return(p)
}

#preview empirical detection range curve
plot(edrf(0:2000),type="l",
  ylab="probability of detecting each coded burst",
 xlab="distance between receiver and transmitter, meters")

#use empirical curve (edrf) in simulation
dp <- receiver_line_det_sim(rngFun=edrf, nsim=10, showPlot=T) #w/ optional plot
dp
```

REI *Calculates a returns a list of each station and the REI (defined here)*

Description

The receiver efficiency index is number between 0 and 1 indicating the amount of relative activity at each receiver compared to the entire set of receivers, regardless of positioning. The function takes a set detections and a deployment history of the receivers to create a context for the detections. Both the amount of unique tags and number of species are taken into consideration in the calculation.

(Ellis, R., Flaherty-Walia, K., Collins, A., Bickford, J., Walters Burnsed, Lowerre-Barbieri S. 2018. Acoustic telemetry array evolution: from species- and project-specific designs to large-scale, multispecies, cooperative networks, https://doi.org/10.1016/j.fishres.2018.09.015*)*

REI() takes two arguments. The first is a dataframe of detections the detection timstamp, the station identifier, the species, and the tag identifier. The next is a dataframe of deployments for each station. The station name should match the stations in the detections. The deployments need to include a deployment date and recovery date.

$$REI = (Tr/Ta)x(Sr/Sa)x(DDr/DDa)x(Da/Dr)$$

- Tr = The number of tags detected on the receievr
- Ta = The number of tags detected across all receivers
- Sr = The number of species detected on the receiver
- Sa = The number of species detected across all receivers
- DDa = The number of unique days with detections across all receivers
- DDr = The number of unique days with detections on the receiver
- Da = The number of days the array was active
- Dr = The number of days the receiver was active

Usage

```
REI(detections, deployments)
```

Arguments

detections	a glatos detections class data table
deployments	a glatos receivers class data table

Value

a list of receivers with lat and long and the receiver efficiency index

Author(s)

Alex Nunes <anunes@dal.ca>

Examples

```
det_file <- system.file("extdata", "hfx_detections.csv",
package = "glatos")

dep_file <- system.file("extdata", "hfx_deployments.csv",
                        package = "glatos")

hfx_deployments <- glatos::read_otn_deployments(dep_file)
dets <- glatos::read_otn_detections(det_file)

hfx_receiver_efficiency_index <- glatos::REI(dets,hfx_deployments)
```

`residence_index` *Generate the residence index from a set of detections*

Description

This residence index tool will take condensed detection event data (from `detection_events` and caculate the residence index for each location. The information passed to the function is what is used to calculate the residence index, make sure you are only passing the data you want taken into consideration for the residence index (i.e. species, stations, tags, etc.).

Usage

```
residence_index(
  detections,
  calculation_method = "kessel",
  locations = NULL,
  group_col = "animal_id",
  time_interval_size = "1 day",
  groupwise_total = TRUE
)
```

Arguments

`detections` A data.frame from the `detection_events` function.

`calculation_method`
A character string with the calculation method using one of the following: `kessel`, `time_interval`, `timedelta`, `aggregate_with_overlap`, or `aggregate_no_overlap`.

`locations` An optional data frame that identifies all unique locations where RI will be calculated. Three columns required:

location Character string with unique location identifier.

mean_longitude Location longitude (for mapping).

mean_latitude Location latitude (for mapping).

If `locations = NULL` (default value) then RI will only be calculated at locations present in `detections$location`.

`group_col` Optional character string (can be multiple) that identifies additional grouping variables for RI calculations. The default value (`group_col = "animal_id"`) will calculate and return RI for each animal at each location (i.e., for each unique combination of `location` and `animal_id`. If `group_col = NULL` then RI will be calculated by location only (will not account for animal or any other variable).

`time_interval_size`
Character string with size of the time interval used when `calculation_method = "time_interval"`. This is passed to seq.Date's by argument, so must meet the requirements of that argument for that function (e.g., "1 day", "4 hours", etc.). Default is `"1 day"`.

`groupwise_total`
: Logical that determines how the denominator is calculated in RI. If FALSE (default) then the denominator represents the total number of time intervals or time (depending on calculation method) among all records. Otherwise (if FALSE), the denominator represents the total number of time intervals or time within each group level (e.g., for each animal if `group_col = "animal_id"`.

Details

The **kessel** method converts both the `first_detection` and `last_detection` columns into a date with no hours, minutes, or seconds. Next it creates a list of the unique days where a detection was seen. The size of the list is returned as the total number of days as an integer. This calculation is used to determine the total number of distinct days (T) and the total number of distinct days per location (S). Possible rounding error may occur as a detection on 2016-01-01 23:59:59 and a detection on 2016-01-02 00:00:01 would be counted as two days when it is really 2-3 seconds.

$$RI = S/T$$

$$RI = ResidenceIndex$$

$$S = Distinctnumberofdaysdetectedatthelocation$$

$$T = Distinctnumberofdaysdetectedatanylocation$$

The **time_interval** calculation method determines the number of time intervals (size determined by `time_interval_size` argument) in which detections occurred at each `location` and as a fraction of the number of time intervals in which detections occurred among all sites. For each location, residency index (RI) is calculated:

$$RI = L/T$$

$$RI = ResidenceIndex$$

$$L = Distinctnumberoftimeintervalsinwhichdetectionobservedatthislocation$$

$$T = Distinctnumberoftimeintervalsinwhichdetectionobservedatanylocation$$

For consistency with other `calculation_methods`, the L and T are not reported, but are converted cumulative time covered in days and reported in columns `days_detected` and `total_days`.

The **timedelta** calculation method determines the first detection and the last detection of all detections. The time difference is then taken as the values to be used in calculating the residence index. The timedelta for each station is divided by the timedelta of the array to determine the residence index.

$$RI = DeltaS/DeltaT$$

$$RI = ResidenceIndex$$

$$DeltaS = Lastdetectiontimeatthelocation - Firstdetectiontimeatthelocation$$

$$DeltaT = Lastdetectiontimeatanylocation - Firstdetectiontimeatanylocation$$

The **aggregate_with_overlap** calculation method takes the length of time of each detection and sums them together. A total is returned. The sum for each location is then divided by the sum among all locations to determine the residence index.

$$RI = AwOS/AwOT$$

$$RI = ResidenceIndex$$

$$AwOS = Sumoflengthoftimeofeachdetectionatthelocation$$

$$AwOT = Sumoflengthoftimeofeachdetectionamongalllocations$$

The **aggregate_no_overlap** calculation method takes the length of time of each detection and sums them together. However, any overlap in time between one or more detections is excluded from the sum. For example, if the first detection is from `2016-01-01 01:02:43` to `2016-01-01 01:10:12` and the second detection is from `2016-01-01 01:09:01` to `2016-01-01 01:12:43`, then the sum of those two detections would be 10 minutes. A total is returned once all detections of been added without overlap. The sum for each location is then divided by the sum among all locations to determine the residence index.

$$RI = AnOS/AnOT$$

$$RI = ResidenceIndex$$

$$AnOS = Sumoflengthoftimeofeachdetectionatthelocation, excludinganyoverlap$$

$$AnOT = Sumoflengthoftimeofeachdetectionamongalllocations, excludinganyoverlap$$

Value

A data.frame of days_detected, residency_index, location, mean_latitude, mean_longitude

Author(s)

A. Nunes, <anunes@dal.ca>

References

Kessel, S.T., Hussey, N.E., Crawford, R.E., Yurkowski, D.J., O'Neill, C.V. and Fisk, A.T., 2016. Distinct patterns of Arctic cod (*Boreogadus saida*) presence and absence in a shallow high Arctic embayment, revealed across open-water and ice-covered periods through acoustic telemetry. Polar Biology, 39(6), pp.1057-1068. https://www.researchgate.net/publication/279269147

Examples

```
#get path to example detection file
det_file <- system.file("extdata", "walleye_detections.csv",
                        package = "glatos")
det <- read_glatos_detections(det_file)
detection_events <- glatos::detection_events(det)
rik_data <- glatos::residence_index(detection_events,
                                    calculation_method = 'kessel')
rit_data <- glatos::residence_index(detection_events,
                                    calculation_method = 'time_interval')
rit_data <- glatos::residence_index(detection_events,
                                    calculation_method = 'timedelta')
riawo_data <- glatos::residence_index(detection_events,
                              calculation_method = 'aggregate_with_overlap')
riano_data <- glatos::residence_index(detection_events,
                              calculation_method = 'aggregate_no_overlap')
```

---

rotate_points *Rotate points in a 2-d plane*

Description

Rotate points around a point in a 2-d plane

Usage

```
rotate_points(x, y, theta, focus)
```

Arguments

x	A numeric vector of x coordinates; minimum of 2.
y	A numeric vector of y coordinates; minimum of 2.
theta	A numeric scalar with the angle of rotation in degrees; positive is clockwise.
focus	A numeric vector of x (first element) and y (second element) coordinates for the point around which x and y will rotate.

Details

Points are shifted to be centered at the focus, then rotated using a rotation matrix, then shifted back to original focus.

Value

A two-column data frame containing:

x	x coordinates
y	y coordinates

Note

This function is called from crw_in_polygon

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
x <- runif(10,0,10)
y <- runif(10,0,10)
plot(x,y,type="b",pch=20)
foo <- rotate_points(x, y, 20, c(5, 5))
points(foo$x,foo$y,type="b",pch=20,col="red")
```

---

shoreline *zipped polygon shapefile of Great Lakes*

Description

Polygon coastline of Great Lakes in WGS84 projection. Includes outlines of Tittabawassee River (Lake Huron), Maumee River (Lake Erie), and Sandusky River (Lake Erie)

Usage

```
system.file("extdata", "shoreline.zip", package="glatos")
```

Format

shapefile

Filename

shoreline.zip

Author(s)

Todd Hayden

Source

http://glatos.glos.us/home

summarize_detections *Summarize detections by animal, location, or both*

Description

Calculate number of fish detected, number of detections, first and last detection timestamps, and/or mean location of receivers or groups, depending on specific type of summary requested.

Usage

```
summarize_detections(
  det,
  location_col = "glatos_array",
  receiver_locs = NULL,
  animals = NULL,
  summ_type = "animal"
)
```

Arguments

det	A glatos_detections object (e.g., produced by read_glatos_detections). *OR* a data frame containing detection data with four columns described below and one column containing a location grouping variable, whose name is specified by location_col (see below). The following four columns must appear in det, except deploy_lat and deploy_lon are not needed if receiver_locs is specified: animal_id Individual animal identifier; character. detection_timestamp_utc Timestamps for the detections (MUST be of class 'POSIXct'). deploy_lat Latitude of receiver deployment in decimal degrees, NAD83. deploy_long Longitude of receiver deployment in decimal degrees, NAD83.
location_col	A character string indicating the column name in det (and receiver_locs if specified) that will be used as the location grouping variable (e.g. "glatos_array"), in quotes.
receiver_locs	An optional data frame containing receiver data with the two columns ('deploy_lat', 'deploy_long') described below and one column containing a location grouping variable, whose name is specified by location_col (see above). The following two columns must appear in receiver_locs: • deploy_lat Latitude of receiver deployment in decimal degrees, NAD83. • deploy_long Longitude of receiver deployment in decimal degrees, NAD83.
animals	A character vector with values of 'animal_id' that will be included in summary. This allows (1) animals *not* detected (i.e., not present in det) to be included in the summary and/or (2) unwanted animals in det to be excluded from the summary.
summ_type	A character string indicating the primary focus of the summary. Possible values are "animal" (default), "location", and "both". See Details below.

Details

Input argument summ_type determines which of three possible summaries is conducted. If summ_type = "animal" (default), the output summary includes the following for each unique value of animal_id: number of unique locations (defined by unique values of location_col), total number of detections across all locations, timestamp of first and last detection across all locations, and a space-delimited string showing all locations where each animal was detected. If summ_type = "location", the output summary includes the following for each unique value of location_col: number of animals (defined by unique values of animal_id), total number of detections across all animals, timestamp of first and last detection across all animals, mean latitude and longitude of each location group, and a space-delimited string of each unique animal that was detected. If summ_type = "both", the output summary includes the following for each unique combination of location_col and animal_id: total number of detections, timestamp of first and last detection, and mean latitude and longitude.

If receiver_locs = NULL (default), then mean latitude and longitude of each location (mean_lat and mean_lon in output data frame) will be calculated from data in det. Therefore, mean locations in the output summary may not represent the mean among all receiver stations in a particular group if detections did not occur on all receivers in each group. However, when actual receiver locations are specified by receiver_locs, then mean_lat and mean_lon will be calculated from receiver_locs. Also, if mean location is not desired or suitable, then receiver_locs can be used to pass a single user-specified deploy_lat and deploy_long for each unique value of location_col, whose values would then represent mean_lat and mean_lon in the output summary.

Value

If summ_type = "animal" (default): A data frame, data.table, or tibble containing six columns:

- animal_id: described above.
- num_locs: number of locations.
- num_dets: number of detections.
- first_det: first detection timestamp.
- last_det: last detections timestamp.
- locations: character string with locations detected, separated by spaces.

If summ_type = "location": A data frame, data.table, or tibble containing eight columns:

- LOCATION_COL: defined by location_col.
- num_fish: number of unique animals detected.
- num_dets: number of detections.
- first_det: first detection timestamp.
- last_det: last detections timestamp.
- mean_lat: mean latitude of receivers at this location.
- mean_lon: mean longitude of receivers at this location.
- animals: character string with animal_ids detected, separated by spaces.

If summ_type = "both": A data frame, data.table, or tibble containing seven columns:

- animal_id: described above.

- LOCATION_COL: defined by location_col.
- num_dets: number of detections.
- first_det: first detection timestamp.
- last_det: last detections timestamp.
- mean_lat: mean latitude of receivers at this location.
- mean_lon: mean longitude of receivers at this location.

Author(s)

T. R. Binder and C. Holbrook

Examples

```
#get path to example detection file
 det_file <- system.file("extdata", "walleye_detections.csv",
   package = "glatos")
 det <- read_glatos_detections(det_file)

 #Basic summaries

 # by animal
 ds <- summarize_detections(det)

 # by location
 ds <- summarize_detections(det, summ_type = "location")

 # by animal and location
 ds <- summarize_detections(det, summ_type = "both")


 #Include user-defined location_col

 # by animal
 det$some_place <- ifelse(grepl("^S", det$glatos_array), "s", "not_s")

 ds <- summarize_detections(det, location_col = "some_place")

 # by location
 ds <- summarize_detections(det, location_col = "some_place",
                            summ_type = "location")

 # by animal and location
 ds <- summarize_detections(det, location_col = "some_place",
                            summ_type = "both")


 #Include locations where no animals detected

 #get example receiver data
 rec_file <- system.file("extdata", "sample_receivers.csv",
   package = "glatos")
```

```
rec <- read_glatos_receivers(rec_file)

ds <- summarize_detections(det, receiver_locs = rec, summ_type = "location")


#Include animals that were not detected
#get example animal data from walleye workbook
wb_file <- system.file("extdata", "walleye_workbook.xlsm",
  package = "glatos")
wb <- read_glatos_workbook(wb_file)

ds <- summarize_detections(det, animals = wb$animals, summ_type = "animal")

#Include by animals and locations that were not detected
ds <- summarize_detections(det, receiver_locs = rec, animals = wb$animals,
  summ_type = "both")
```

`total_diff_days` *The function below determines the total days difference.*

Description

The difference is determined by the minimal first_detection of every detection and the maximum last_detection of every detection. Both are converted into a datetime then subtracted to get a timedelta. The timedelta is converted to seconds and divided by the number of seconds in a day (86400). The function returns a floating point number of days (i.e. 503.76834).

Usage

```
total_diff_days(detections)
```

Arguments

`detections`	• data frame pulled from the compressed detections CSV

`transmit_along_path` *Simulate telemetry transmitter signals along a path*

Description

Simulate tag signal transmission along a pre-defined path (x, y coords) based on constant movement velocity, transmitter delay range, and duration of signal.

Usage

```
transmit_along_path(
  path = NA,
  vel = 0.5,
  delayRng = c(60, 180),
  burstDur = 5,
  colNames = list(x = "x", y = "y"),
  pathCRS = NA,
  sp_out = TRUE
)
```

Arguments

path	A data frame or matrix with at least two rows and named columns with coordinates that define path. *OR* A object of class sf or sfc containing POINT features with a geometry column. (SpatialPointsDataFrame is also allowed.)
vel	A numeric scalar with movement velocity along track; assumed constant; in meters per second.
delayRng	A 2-element numeric vector with minimum and maximum delay (time in seconds from end of one coded burst to beginning of next).
burstDur	A numeric scalar with duration (in seconds) of each coded burst (i.e., pulse train).
colNames	A named list containing the names of columns with coordinates (defaults are x and y) in path. Ignored if trnsLoc is a spatial object with a geometry column.
pathCRS	Defines the coordinate reference system (object of class crs or a numeric EPSG code) of coordinates in path, if missing; ignored otherwise. If no valid crs is specified in path or via pathCRS = NA (default value), then path coordinates are assumed to be in an arbitrary Cartesian coordinate system with base unit of 1 meter. See Note.
sp_out	Logical. If TRUE (default) then output is an sf object. If FALSE, then output is a data.frame.

Details

Delays are drawn from uniform distribution defined by delay range. First, elapsed time in seconds at each vertex in path is calculated based on path length and velocity. Next, delays are simulated and burst durations are added to each delay to determine the time of each signal transmission. Location of each signal transmission along the path is linearly interpolated.

Computation time is fastest if coordinates in path are in a Cartesian (projected) coordinate system and slowest if coordinates are in a geographic coordinate system (e.g., longitude, latitude) because different methods are used to calculate step lengths in each case. When path CRS is Cartesian (e.g., UTM), step lengths are calculated as simple Euclidean distance. When CRS is geographic, step lengths are calculated as Haversine distances using geodist (with measure = "haversine").

Value

When sp_out = TRUE, an sf object containing one POINT feature for each simulated transmission and a column named time (defined below).

When sp_out = FALSE, a data.frame with the following columns:

x	x coordinates for start of each transmission.
y	y coordinates for start of each transmission.
time	Elapsed time, in seconds, from the start of input path to the start of each transmission.

Note

This function was written to be called after crw_in_polygon and before detect_transmissions, which was designed to accept the result as input (trnsLoc).

Author(s)

C. Holbrook <cholbrook@usgs.gov>

Examples

```
#Example 1 - data.frame input (default column names)

mypath <- data.frame(x = seq(0, 1000, 100),
                     y = seq(0, 1000, 100))

mytrns <- transmit_along_path(mypath, vel = 0.5,
                              delayRng = c(60, 180),
                              burstDur = 5.0,
                              sp_out = FALSE)
plot(mypath, type = "o")
points(mytrns, pch = 20, col = "red")


#Example 2 - data.frame input (non-default column names)

mypath <- data.frame(Easting = seq(0, 1000, 100),
                     Northing = seq(0, 1000, 100))

mytrns <- transmit_along_path(mypath, vel = 0.5, delayRng = c(60, 180),
                              burstDur = 5.0,
                              colNames = list(x = "Easting",
                                              y = "Northing"),
                              sp_out = FALSE)
plot(mypath, type = "o")
points(mytrns, pch = 20, col = "red")


#Example 3 - data.frame input using pathCRS arg
```

```
mypath <- data.frame(deploy_long = c(-87, -87.1, -87),
                     deploy_lat = c(44, 44.1, 44.2))

mytrns <- transmit_along_path(mypath, vel = 0.5, delayRng = c(600, 1800),
                              burstDur = 5.0,
                              colNames = list(x = "deploy_long",
                                              y = "deploy_lat"),
                              pathCRS = 4326,
                              sp_out = FALSE)
plot(mypath, type = "o")
points(mytrns, pch = 20, col = "red")


#Example 4 - sf POINT input

#simulate in great lakes polygon
data(great_lakes_polygon)

mypath_sf <- crw_in_polygon(great_lakes_polygon,
                            theta = c(0, 25),
                            stepLen = 100,
                            initHeading = 0,
                            nsteps = 10,
                            cartesianCRS = 3175)

mytrns_sf <- transmit_along_path(mypath_sf,
                                 vel = 0.5,
                                 delayRng = c(60, 180),
                                 burstDur = 5.0)
plot(mypath_sf, type = "o")
points(sf::st_coordinates(mytrns_sf), pch = 20, col = "red")


#Example 5 - SpatialPointsDataFrame input

#simulate in great lakes polygon
data(greatLakesPoly)

mypath_sp <- crw_in_polygon(greatLakesPoly,
                            theta = c(0, 25),
                            stepLen = 100,
                            initHeading = 0,
                            nsteps = 10,
                            cartesianCRS = 3175)

mytrns_sp <- transmit_along_path(mypath_sp,
                                 vel = 0.5,
                                 delayRng = c(60, 180),
                                 burstDur = 5.0)

plot(sf::st_coordinates(sf::st_as_sf(mypath_sp)), type = "o")
points(sf::st_coordinates(mytrns_sp), pch = 20, col = "red")
```

utm_to_lonlat *Convert UTM positions to lonlat*

Description

Convert UTM positions to lonlat

Usage

```
utm_to_lonlat(utm, hemisphere)
```

vector_heading *Calculate direction (heading) of a vector (in degrees)*

Description

Calculate direction (heading) of each link of a vector (in degrees)

Usage

```
vector_heading(x, y = NULL, coord_sys = NA)
```

Arguments

x	A numeric vector of x coordinates; minimum of 2. OR A two-column matrix or data frame with x coordinates in column 1 and y coordinates in column 2.
y	A numeric vector of y coordinates; minimum of 2.
coord_sys	The type of geographical coordinate system used. Possible values are NA (for any cartesian grid; e.g., UTM) or longlat (for WGS84 in decimal degrees).

Details

Calculates direction (in degrees) for each of k-1 vectors, where k = length(x) - 1. Lengths of x and y must be equal.

Value

A numeric scalar with heading in degrees or a numeric vector of headings if length(x) > 2.

If units are decimal degrees (i.e., coord_sys = "longlat") then the angles returned will represent the heading at the start of each vector.

Note

This function is called from within crw_in_polygon

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
#example using generic cartesian (regular grid) coordinates
x=c(2,4)
y=c(2,4)
vector_heading(x, y)

x2=c(2,4,2)
y2=c(2,4,2)
vector_heading(x2, y2)


#example using WGS84 lat-lon
#e.g., from Duluth to Toronto to Detroit

path1 <- data.frame(city = c("Duluth", "Toronoto", "Detroit"),
                    longitude = c(-92.1005, -79.3832, -83.0458),
                    latitude = c(46.7867, 43.6532, 42.3314))

#example using the x, y input method way
vector_heading(x = c(-92.1005, -79.3832, -83.0458),
               y = c(46.7867, 43.6532, 42.3314),
               coord_sys = "longlat")

#example using the x-only input method
vector_heading(x = path1[ , c("longitude", "latitude")],
               coord_sys = "longlat")
```

---

video images *Video frames of walleye movements in Lake Huron*

Description

Sequential images of walleye movements in Lake Huron for testing functionality of ffmpeg function.

Usage

```
system.file("extdata", "frames", package="glatos")
```

Format

Folder contains 30 sequentially labeled .png image files

Filename

frames

Author(s)

Todd Hayden

Source

http://glatos.glos.us/home/project/HECWL

`vrl2csv` *Convert Vemco VRL file(s) to CSV format (detection data only)*

Description

Convert detection data from a VEMCO VRL file(s) to comma-separated-values (CSV) format by invoking a system command in VUE (> 2.06; courtesy of Tim Stone, Vemco).

Usage

```
vrl2csv(vrl, outDir = NA, overwrite = TRUE, vueExePath = NA)
```

Arguments

`vrl`	A character string or vector with names of VRL file(s) or a single directory containing VRL files.
`outDir`	A character string directory where CSV files will be written. If `NA` (default) then file(s) will be written to the current working directory (e.g., `getwd()`).
`overwrite`	Logical. If TRUE (default), output CSV file(s) will overwrite existing CSV file(s) with same name in `outDir`. When FALSE, '_n' (i.e., _1, _2, etc.) will be appended to names of output files that already exist in `outDir`.
`vueExePath`	An optional character string with directory containing VUE.exe. If NA (default) then the path to VUE.exe must be added to the PATH environment variable of your system. See Note below.

Details

If `vrl` is a directory, then all VRL files in that directory will be converted to CSV. Otherwise, only those files specified in `vrl` will be converted. Each output CSV file will have same name as its source VRL file.

Value

A character vector with output directory and file name(s).

Note

Receiver event data are not exported because that functionality was not supported by the VUE system command at time of writing.

The path to VUE.exe must either be specified by `vueExePath` or added to the PATH environment variable of your system. To get the path to VUE.exe in Windows, right click on the icon, select "Properties", and then copy text in "Target" box.

To create a CSV for time-corrected VRL files, first time-correct each file using the VRL editor in VUE (under Tools menu). To speed up that process, uncheck the "Import" checkbox next to each filename, then run `vrl2csv` to create a CSV for each edited (e.g. time-corrected) VRL.

When using versions of VUE before 2.3, VUE can return an error code or warning message even if conversion was successful.

Author(s)

C. Holbrook (cholbrook@usgs.gov)

Examples

```
## Not run:

#get path to example VRL in this package
myVRL <- system.file("extdata", "VR2W_109924_20110718_1.vrl",
 package="glatos")
vrl2csv(dirname(myVRL)) #directory input
vrl2csv(myVRL) #file name input

#setting 'overwrite=FALSE' will make new file with '_n'added to name
vrl2csv(myVRL, overwrite=F)

## End(Not run)
```

Index