Package 'camtrapR'

September 28, 2025

Type Package

Title Camera Trap Data Management and Analysis Framework

Version 3.0.0

Date 2025-09-24

Depends R (>= 4.1.0)

Imports data.table, dplyr, DT, generics, ggplot2, leaflet, lubridate, methods, secr, sf, shiny, shinyBS, shinydashboard, shinyjs, terra

Suggests abind, bayesplot, callr, coda, corrplot, elevatr, iNEXT, jsonlite, jsonify, knitr, lattice, magick, mapview, nimble, nimbleEcology, overlap, parallel, patchwork, pbapply, plotly, psych, raster, reshape2, rjags, R.rsp, RSQLite, ritis, rlang, rmarkdown, rstudioapi, scales, shinyWidgets, stringr, tesseract, taxize, testthat, tibble, ubms, units, unmarked, viridisLite, zip

VignetteBuilder R.rsp

SystemRequirements ExifTool (https://exiftool.org/)

Description Management and analysis of camera trap wildlife data through an integrated workflow. Provides functions for image/video organization and metadata extraction, species/individual identification. Creates detection histories for occupancy and spatial capture-recapture analyses, with support for multi-season studies. Includes tools for fitting community occupancy models in JAGS and NIMBLE, and an interactive dashboard for survey data visualization and analysis. Features visualization of species distributions and activity patterns, plus export capabilities for GIS and reports. Emphasizes automation and reproducibility while maintaining flexibility for different study designs.

URL https://github.com/jniedballa/camtrapR, https://jniedballa.github.io/camtrapR/, https://groups.google.com/forum/#!forum/camtrapr

BugReports https://groups.google.com/forum/#!forum/camtrapr

Encoding UTF-8

License GPL (>= 2)

RoxygenNote 7.3.2

NeedsCompilation no

Author Juergen Niedballa [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9187-2116>),
Alexandre Courtiol [aut] (ORCID:
<https://orcid.org/0000-0003-0637-2959>),
Rahel Sollmann [aut] (ORCID: <https://orcid.org/0000-0002-1607-2039>),
John Mathai [ctb],
Seth Timothy Wong [ctb] (ORCID:
<https://orcid.org/0000-0001-8083-9268>),
An The Truong Nguyen [ctb] (ORCID:
<https://orcid.org/0009-0000-2861-2672>),
Azlan bin Mohamed [ctb] (ORCID:
<https://orcid.org/0000-0003-3788-4383>),
Andrew Tilker [ctb] (ORCID: <https://orcid.org/0000-0003-3630-8691>),
Roshan Guharajan [ctb] (ORCID: <https://orcid.org/0000-0001-8124-5461>),
Ioannis Alexiou [ctb] (ORCID: <https://orcid.org/0000-0001-5095-4767>),
Andreas Wilting [ctb, ths] (ORCID:
<https://orcid.org/0000-0001-5073-9186>)

Maintainer Juergen Niedballa <camtrapr@gmail.com>

Repository CRAN

Date/Publication 2025-09-28 12:30:08 UTC

Contents

`camtrapR-package` *Overview of the functions in the camtrapR package*

Description

This package provides a streamlined workflow for processing data generated in camera trap-based wildlife studies and prepares input for further analyses, particularly in occupancy and spatial capture-recapture frameworks. It suggests a simple data structure and provides functions for managing digital camera trap photographs (and videos), generating record tables, maps of species richness and species detections and species activity diagrams. It further helps prepare subsequent analyses by creating detection/non-detection matrices for occupancy analyses, e.g. in the **unmarked** or **ubms** packages, and `capthist` objects for spatial capture-recapture analyses in the **secr** package. In addition, basic survey statistics are computed. The functions build on one another in a logical sequence. The only manual input needed is species (and individual) identification, which is achieved by moving images into species directories or by tagging images in image management software. Besides, a table holding basic information about camera trap station IDs, locations and trapping periods must be created in spreadsheet software.

Details

Image metadata (such as date and time or user-assigned tags) are extracted from the images using Phil Harvey's ExifTool (available from https://exiftool.org/) and the information is stored in a record table. An adjustable criterion for temporal independence of records can be applied. Maps of species presence and species richness can be generated. Several functions are available for plotting single- and two-species activity patterns. Information about the camera-specific trapping periods (and periods of malfunction) are summarized into a matrix about camera trap operability. These, together with the record table, are used to generate species detection histories for occupancy and spatial capture-recapture analyses. The user has considerable freedom in generating the detection histories; sampling occasion length, beginning date and and occasion start times are adjustable. In addition, trapping effort (i.e. active trap nights per station and occasion) can be computed for use as a covariate / offset on detection probability.

User support

The camtrapR Google group is an online support and help forum for camtrapR users. You can find it here: https://groups.google.com/forum/#!forum/camtrapr.

Image organisation and management

The functions in this section set up a directory structure for storing camera trap images and identifying species and individuals from images. They build on one another and can be run in sequential order as needed.

createStationFolders	Create camera trap station directories for raw images
fixDateTimeOriginal	Fix DateTimeOriginal Exif metadata tag in Reconyx Hyperfire cameras
OCRdataFields	Optical character recognition (OCR) from data fields in images
writeDateTimeOriginal	Write values to DateTimeOriginal tag in image metadata
timeShiftImages	Apply time shifts to JPEG images
imageRename	Copy and rename images based on station ID and image creation date
addCopyrightTag	Write a copyright tag into JPEG image metadata
appendSpeciesNames	Add or remove species names from image filenames

Species / individual identification

These functions assist in species identification and prepare individual identification of animals.

checkSpeciesNames	Check species names against the ITIS taxonomic database
createSpeciesFolders	Create directories for species identification
checkSpeciesIdentification	Consistency check on species image identification
getSpeciesImages	Gather all images of a species in a new directory

Image data extraction

These function use the directory structure built above (Section 'Image management workflow') and a table containing basic information about camera traps and/or stations (IDs, location, trapping period).

recordTable	Create a species record table from camera trap images and videos
recordTableIndividual	Create a single-species record table from camera trap images and videos with individual IDs
exifTagNames	Return Exif metadata tags and tag names from JPEG images
addToPath	Add the directory containing exiftool.exe to PATH temporarily (Windows only)
filterRecordTable	Filter existing record table for temporal independence.

Data exploration and visualisation

These plots are generated from the record table and the camera trap table.

detectionMaps	Generate maps of species richness and species presence by station, export shapefiles
activityHistogram	Single-species diel activity histograms
activityDensity	Single-species diel activity kernel density estimation plots
activityRadial	Single-species diel activity radial plot
activityOverlap	Two-species diel activity overlap plots and estimates

Prepare occupancy and spatial capture-recapture analyses, and summarise surveys

createCovariates	Extract covariate values from spatial rasters and prepare rasters for spatial predictions
cameraOperation	Create a camera operation matrix
detectionHistory	Species detection histories for occupancy analyses (single and multi-season)
spatialDetectionHistory	Detection histories of individuals for spatial capture-recapture analyses
surveyReport	Create a report about camera trap surveys and species detections
surveyDashboard	Shiny dashboard for summarizing and analyzing camera trap survey data

Community (multi-species) occupancy models

communityModel	Create a community (multi-species) occupancy model for JAGS or Nimble
commOccu-class	commOccu objects
fit,commOccu-method	Fit a community (multi-species) occupancy model
predict,commOccu-method	Predictions from community occupancy models
summary,commOccu-method	Summarize community occupancy model
plot_coef	Plot effect sizes of covariates in community occupancy model
plot_effects	Plot Marginal Effects of Covariates
PPC.residuals	Calculate residuals from MCMC output of occupancy models
PPC.community	Calculate community-level posterior predictive checks for occupancy models

Sample data

camtraps	Sample camera trap station information table
recordTableSample	Sample species record table
recordTableIndividualSample	Single-species record table with individual IDs
camtrapsMultiSeason	Sample multi season camera trap station information table
recordTableSampleMultiSeason	Sample multi season species record table
recordTableIndividualSampleMultiSeason	Single-species multi season record table with individual IDs
timeShiftTable	Sample camera trap time shift information

Vignettes

1. Organising raw camera trap images
2. Identifying species and individuals
3. Extracting Data from Camera Trapping Images and Videos
4. Data exploration and visualisation
5. Multi-species occupancy models

Author(s)

Juergen Niedballa

Maintainer:Juergen Niedballa <camtrapr@gmail.com>

References

Niedballa, J., Sollmann, R., Courtiol, A., Wilting, A. (2016): camtrapR: an R package for efficient camera trap data management. Methods in Ecology and Evolution, 7(12). https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.12600

camtrapR Google Group https://groups.google.com/forum/#!forum/camtrapr

Phil Harvey's ExifTool https://exiftool.org/

See Also

overlap unmarked ubms secr wiqid

`activityDensity` *Plot kernel density estimation of single-species activity*

Description

The function plots a kernel density estimation of species diel activity using function `densityPlot` from package **overlap**.

Usage

```
activityDensity(
  recordTable,
  species,
  allSpecies = FALSE,
  speciesCol = "Species",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  plotR = TRUE,
  writePNG = FALSE,
  plotDirectory,
  createDir = FALSE,
  pngMaxPix = 1000,
  add.rug = TRUE,
  ...
)
```

Arguments

`recordTable`	data.frame. the record table created by `recordTable`
`species`	Name of the species for which to create an kernel density plot of activity
`allSpecies`	logical. Create plots for all species in `speciesCol` of `recordTable`? Overrides argument `species`
`speciesCol`	character. name of the column specifying species names in `recordTable`
`recordDateTimeCol`	character. name of the column specifying date and time in `recordTable`
`recordDateTimeFormat`	character. format of column `recordDateTimeCol` in `recordTable`
`plotR`	logical. Show plots in R graphics device?
`writePNG`	logical. Create pngs of the plots?
`plotDirectory`	character. Directory in which to create png plots if `writePNG = TRUE`
`createDir`	logical. Create `plotDirectory` if `writePNG = TRUE`?
`pngMaxPix`	integer. image size of png (pixels along x-axis)
`add.rug`	logical. add a rug to the plot?
`...`	additional arguments to be passed to function `densityPlot`

Details

`species` must be in the `speciesCol` of `recordTable`.

`recordDateTimeFormat` defaults to the "YYYY-MM-DD HH:MM:SS" convention, e.g. "2014-09-30 22:59:59". `recordDateTimeFormat` can be interpreted either by base-R via `strptime` or in **lubridate** via `parse_date_time` (argument "orders"). **lubridate** will be used if there are no "%" characters in `recordDateTimeFormat`.

For "YYYY-MM-DD HH:MM:SS", `recordDateTimeFormat` would be either "%Y-%m-%d %H:%M:%S" or "ymd HMS". For details on how to specify date and time formats in R see `strptime` or `parse_date_time`.

Value

Returns invisibly a vector of species record observation times in radians, i.e. scaled to $[0, 2\pi]$. If `allSpecies == TRUE`, all species' vectors are returned in an invisible named list.

Author(s)

Juergen Niedballa

References

Martin Ridout and Matthew Linkie (2009). Estimating overlap of daily activity patterns from camera trap data. Journal of Agricultural, Biological and Environmental Statistics, 14(3), 322-337
Mike Meredith and Martin Ridout (2018). overlap: Estimates of coefficient of overlapping for animal activity patterns. R package version 0.3.2. https://CRAN.R-project.org/package=overlap

See Also

`activityHistogram`, `activityRadial`, `activityOverlap` https://www.kent.ac.uk/smsas/personal/msr/overlap.html

Examples

```
if(requireNamespace("overlap")) {
# load record table
data(recordTableSample)

species4activity <- "VTA"    # = Viverra tangalunga, Malay Civet

activityDensity(recordTable = recordTableSample,
                species     = species4activity)


# all species at once

activityDensity(recordTable = recordTableSample,
                allSpecies  = TRUE,
                writePNG    = FALSE,
                plotR       = TRUE,
```

```
                add.rug      = TRUE)
}
```

`activityHistogram` *Plot histogram of single-species activity*

Description

The function generates a histogram of species diel activity in 1-hour intervals.

Usage

```
activityHistogram(
  recordTable,
  species,
  allSpecies = FALSE,
  speciesCol = "Species",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  plotR = TRUE,
  writePNG = FALSE,
  plotDirectory,
  createDir = FALSE,
  pngMaxPix = 1000,
  ...
)
```

Arguments

`recordTable`	data.frame. the record table created by `recordTable`
`species`	Name of the single species for which to create a histogram of activity
`allSpecies`	logical. Create plots for all species in `speciesCol` of `recordTable`? Overrides argument `species`
`speciesCol`	character. name of the column specifying species names in `recordTable`
`recordDateTimeCol`	character. name of the column specifying date and time in `recordTable`
`recordDateTimeFormat`	character. format of column `recordDateTimeCol` in `recordTable`
`plotR`	logical. Show plots in R graphics device?
`writePNG`	logical. Create pngs of the plots?
`plotDirectory`	character. Directory in which to create png plots if `writePNG = TRUE`
`createDir`	logical. Create `plotDirectory`?
`pngMaxPix`	integer. image size of png (pixels along x-axis)
`...`	additional arguments to be passed to function `hist`

Details

Activity is calculated from the time of day of records. The date is ignored.

`recordDateTimeFormat` defaults to the "YYYY-MM-DD HH:MM:SS" convention, e.g. "2014-09-30 22:59:59". `recordDateTimeFormat` can be interpreted either by base-R via `strptime` or in **lubridate** via `parse_date_time` (argument "orders"). **lubridate** will be used if there are no "%" characters in `recordDateTimeFormat`.

For "YYYY-MM-DD HH:MM:SS", `recordDateTimeFormat` would be either "%Y-%m-%d %H:%M:%S" or "ymd HMS". For details on how to specify date and time formats in R see `strptime` or `parse_date_time`.

Value

It returns invisibly a vector of species record date and time in `POSIXlt` format. If `allSpecies == TRUE`, all species' vectors are returned in an invisible named list.

Note

If you have a sufficiently large number of records you may wish to consider using `activityDensity` instead. Please be aware that this function (like the other activity... function of this package) use clock time. If your survey was long enough to see changes in sunrise and sunset times, this may result in biased representations of species activity.

Author(s)

Juergen Niedballa

See Also

`activityDensity`, `activityRadial`, `activityOverlap`

Examples

```
# load record table
data(recordTableSample)

# generate activity histogram
species4activity <- "VTA"    # = Viverra tangalunga, Malay Civet

activityHistogram (recordTable = recordTableSample,
                   species     = species4activity,
                   allSpecies = FALSE)
```

`activityOverlap` *Plot overlapping kernel densities of two-species activities*

Description

This function plots kernel density estimates of two species' diel activity data by calling the function `overlapPlot` from package **overlap**. It further computes the overlap coefficient Dhat1 by calling `overlapEst`.

Usage

```
activityOverlap(
  recordTable,
  speciesA,
  speciesB,
  speciesCol = "Species",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  plotR = TRUE,
  writePNG = FALSE,
  addLegend = TRUE,
  legendPosition = "topleft",
  plotDirectory,
  createDir = FALSE,
  pngMaxPix = 1000,
  add.rug = TRUE,
  overlapEstimator = c("Dhat1", "Dhat4", "Dhat5"),
  ...
)
```

Arguments

`recordTable`	data.frame. the record table created by `recordTable`
`speciesA`	Name of species 1 (as found in `speciesCol` of recordTable)
`speciesB`	Name of species 2 (as found in `speciesCol` of recordTable)
`speciesCol`	character. name of the column specifying species names in `recordTable`
`recordDateTimeCol`	character. name of the column specifying date and time in `recordTable`
`recordDateTimeFormat`	character. format of column `recordDateTimeCol` in `recordTable`
`plotR`	logical. Show plots in R graphics device?
`writePNG`	logical. Create pngs of the plots?
`addLegend`	logical. Add a legend to the plots?
`legendPosition`	character. Position of the legend (keyword)

plotDirectory	character. Directory in which to create png plots if writePNG = TRUE
createDir	logical. Create plotDirectory?
pngMaxPix	integer. image size of png (pixels along x-axis)
add.rug	logical. add a rug to the plot?
overlapEstimator	character. Which overlap estimator to return (passed on to argument type in overlapEst)
...	additional arguments to be passed to function overlapPlot

Details

... can be graphical parameters passed on to function overlapPlot, e.g. linetype, linewidth, linecol (see example below).

recordDateTimeFormat defaults to the "YYYY-MM-DD HH:MM:SS" convention, e.g. "2014-09-30 22:59:59". recordDateTimeFormat can be interpreted either by base-R via strptime or in **lubridate** via parse_date_time (argument "orders"). **lubridate** will be used if there are no "%" characters in recordDateTimeFormat.

For "YYYY-MM-DD HH:MM:SS", recordDateTimeFormat would be either "%Y-%m-%d %H:%M:%S" or "ymd HMS". For details on how to specify date and time formats in R see strptime or parse_date_time.

Value

Returns invisibly the data.frame with plot coordinates returned by overlapPlot.

Note

Please be aware that the function (like the other activity... function of this package) use clock time, not solar time. If your survey was long enough to see changes in sunrise and sunset times, this may result in biased representations of species activity.

Author(s)

Juergen Niedballa

References

Mike Meredith and Martin Ridout (2018). overlap: Estimates of coefficient of overlapping for animal activity patterns. R package version 0.3.2. https://CRAN.R-project.org/package=overlap

Ridout, M.S. and Linkie, M. (2009) Estimating overlap of daily activity patterns from camera trap data. Journal of Agricultural, Biological and Environmental Statistics, 14, 322-337.

See Also

activityDensity

https://www.kent.ac.uk/smsas/personal/msr/overlap.html

Examples

```
if(requireNamespace("overlap")) {

# load record table
data(recordTableSample)

# define species of interest
speciesA_for_activity <- "VTA"    # = Viverra tangalunga, Malay Civet
speciesB_for_activity <- "PBE"    # = Prionailurus bengalensis, Leopard Cat

# create activity overlap plot (basic)
activityOverlap (recordTable = recordTableSample,
                 speciesA    = "VTA",    # = Viverra tangalunga, Malay Civet
                 speciesB    = "PBE",    # = Prionailurus bengalensis, Leopard Cat
                 writePNG    = FALSE,
                 plotR       = TRUE
)



# create activity overlap plot (prettier and with some overlapPlot arguments set)

activityOverlap (recordTable = recordTableSample,
                 speciesA    = speciesA_for_activity,
                 speciesB    = speciesB_for_activity,
                 writePNG    = FALSE,
                 plotR       = TRUE,
                 createDir   = FALSE,
                 pngMaxPix   = 1000,
                 linecol     = c("black", "blue"),
                 linewidth   = c(5,3),
                 linetype    = c(1, 2),
                 olapcol     = "darkgrey",
                 add.rug     = TRUE,
                 extend      = "lightgrey",
                 ylim        = c(0, 0.25),
                 main        = paste("Activity overlap between ",
                                     speciesA_for_activity, "and",
                                     speciesB_for_activity)
)
}
```

`activityRadial` *Radial plots of single-species activity*

Description

The function generates a radial plot of species diel activity using an adapted version of function `radial.plot` from package **plotrix** (without the need to install the package). Records are aggre-

gated by hour. The number of independent events is used as input, which in turn is based on the argument minDeltaTime in recordTable.

Usage

```
activityRadial(
  recordTable,
  species,
  allSpecies = FALSE,
  speciesCol = "Species",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  byNumber = FALSE,
  plotR = TRUE,
  writePNG = FALSE,
  plotDirectory,
  createDir = FALSE,
  pngMaxPix = 1000,
  ...
)
```

Arguments

recordTable	data.frame. the record table created by recordTable
species	Name of the species for which to create an kernel density plot of activity
allSpecies	logical. Create plots for all species in speciesCol of recordTable? Overrides argument species
speciesCol	character. name of the column specifying species names in recordTable
recordDateTimeCol	character. name of the column specifying date and time in recordTable
recordDateTimeFormat	character. format of column recordDateTimeCol in recordTable
byNumber	logical. If FALSE, plot proportion of records. If TRUE, plot number of records
plotR	logical. Show plots in R graphics device?
writePNG	logical. Create pngs of the plots?
plotDirectory	character. Directory in which to create png plots if writePNG = TRUE
createDir	logical. Create plotDirectory?
pngMaxPix	integer. image size of png (pixels along x-axis)
...	additional arguments to be passed to function radial.plot

Details

radial.plot was adjusted to show a clockwise 24-hour clock face. It is recommended to set argument lwd to a value >= 2. You may also wish to add argument rp.type="p" to show a polygon instead of bars.

recordDateTimeFormat defaults to the "YYYY-MM-DD HH:MM:SS" convention, e.g. "2014-09-30 22:59:59". recordDateTimeFormat can be interpreted either by base-R via strptime or in **lubridate** via parse_date_time (argument "orders"). **lubridate** will be used if there are no "%" characters in recordDateTimeFormat.

For "YYYY-MM-DD HH:MM:SS", recordDateTimeFormat would be either "%Y-%m-%d %H:%M:%S" or "ymd HMS". For details on how to specify date and time formats in R see strptime or parse_date_time.

Value

Returns invisibly a data.frame containing all information needed to create the plot: radial position, lengths, hour (for labels). If allSpecies == TRUE, all species' data frames are returned in an invisible named list.

Author(s)

Juergen Niedballa

References

Lemon, J. (2006) Plotrix: a package in the red light district of R. R-News, 6(4): 8-12.
https://CRAN.R-project.org/package=plotrix

See Also

activityDensity, activityHistogram, activityOverlap

Examples

```
# load record table
data(recordTableSample)

species4activity <- "PBE"    # = Prionailurus bengalensis, Leopard Cat

activityRadial(recordTable       = recordTableSample,
               species           = species4activity,
               allSpecies        = FALSE,
               speciesCol        = "Species",
               recordDateTimeCol = "DateTimeOriginal",
               plotR             = TRUE,
               writePNG          = FALSE,
               lwd               = 5
)

# plot type = polygon

activityRadial(recordTable       = recordTableSample,
               species           = species4activity,
               allSpecies        = FALSE,
               speciesCol        = "Species",
```

```
                recordDateTimeCol = "DateTimeOriginal",
                plotR             = TRUE,
                writePNG          = FALSE,
                lwd               = 5,
                rp.type           = "p"
)
```

---

addCopyrightTag *Write a copyright tag into JPEG image metadata*

Description

This function writes a copyright tag into the copyright field of JPEG image Exif metadata. It does so recursively, so it works both for images that are sorted into subdirectories and unsorted images. Note that all images in subdirectories of inDir will be tagged. It is not required to run this function in the camtrapR workflow, but may be desired for data sharing or publishing.

Usage

```
addCopyrightTag(
  inDir,
  copyrightTag,
  askFirst = TRUE,
  keepJPG_original = TRUE,
  ignoreMinorErrors = FALSE
)
```

Arguments

inDir	character. Name of the directory containing camera trap images.
copyrightTag	character. The tag to be written into the Exif Copyright field
askFirst	logical. Ask user to confirm before execution?
keepJPG_original	logical. Keep original JPG files as .JPG_original files (TRUE) or overwrite JPGs (FALSE)?
ignoreMinorErrors	logical. Ignore minor errors that would cause the function to fail (set TRUE for images with bad MakerNotes, observed in Panthera V4 cameras)

Details

If askFirst = TRUE, the function will show a menu and asks the user to confirm the action before execution. Type "1" to write copyright tags and "2" to abort.

By default Exiftool creates a copy of each JPG image and preserves the original images (without the copyright tag) as .JPG_original files. Note that this behaviour will instantly double the number of images in inDir and the disk space required. If this is not desired, set keepJPG_original = FALSE.

`ignoreMinorErrors` is useful if copyright tags can't be updated correctly. This can be caused by bad MakerNotes and so far was only observed in Panthera V4 cameras. In that case, set `ignoreMinorErrors` to `TRUE`. This will add the "-m" option to the Exiftool call, thereby ignoring minor errors and warnings and assigning the copyright tag regardless.

Value

An invisible list of Exiftool output.

More importantly, the specified copyright tag is written into the Copyright field of the Exif metadata of all images in inDir.

Author(s)

Juergen Niedballa

Examples

```
## Not run:

if (Sys.which("exiftool") != ""){       # only run this example if ExifTool is available

# copy sample images to temporary directory (so we don't mess around in the package directory)
wd_images_ID <- system.file(file.path("pictures", "sample_images_species_dir"),
                            package = "camtrapR")
file.copy(from = wd_images_ID, to = tempdir(), recursive = TRUE)
wd_images_ID_copy <- file.path(tempdir(), "sample_images_species_dir")

# define a sample tag
copyrightTagToAdd <- "Your Name (Your Organisation)"

# add the tag to the images
addCopyrightTag(inDir        = wd_images_ID_copy,
                copyrightTag = copyrightTagToAdd)
1     # we choose "YES", i.e., we want to add a copyright tag

# you can check the outcome with function exifTagNames

metadat <- exifTagNames(wd_images_ID_copy)
metadat [metadat$tag_name == "Copyright",]
}



## End(Not run)
```

addToPath *Add a directory to PATH temporarily*

Description

Temporarily adds a directory to the environmental variable PATH for system calls from within R. This allows Windows users to store exiftool.exe anywhere on their hard drive and is useful if they cannot store the file in system directories. It is not needed on Linux or MacOS machines.

Usage

```
addToPath(directory)
```

Arguments

directory	character. the directory in the file system to add to PATH (e.g. the directory containing exiftool.exe).

Details

Several functions within this package depend on ExifTool. Under Windows, exiftool.exe cannot be used if it is not in a directory path specified in PATH. This can be solved by adding the directory containing exiftool.exe for temporary use within the running R process. It can also be useful in other contexts besides Exiftool.

Value

invisible logical indicating whether `directory` was added to PATH successfully (in the running R process).

Note

The directories in PATH can be queried by `Sys.getenv("PATH")`.

Author(s)

Juergen Niedballa

Examples

```
exiftool_dir <- "C:/Path/To/Exiftool"
addToPath(directory = exiftool_dir)

# check if it has been added to PATH
grepl(exiftool_dir,  Sys.getenv("PATH"))
```

`aggregateStations` *Aggregate Camera Trap Table to Station Level*

Description

Aggregates a camera trap table from a station-camera (location-deployment) level to a station (location) level. This function is useful when modeling or analysis is conducted at the station level, when multiple cameras were deployed at a single station or a single camera had multiple deployments.

Usage

```
aggregateStations(
  CTtable,
  stationCol,
  cameraCol = NULL,
  setupCol = NULL,
  retrievalCol = NULL,
  dateFormat = NULL
)
```

Arguments

`CTtable`	A data frame or 'sf' object representing the camera trap table. Each row typically represents a unique camera deployment.
`stationCol`	A character string specifying the name of the column in `CTtable` that contains the unique station identifiers.
`cameraCol`	A character string specifying the name of the column in `CTtable` that contains the unique camera (deployment) identifiers.
`setupCol`	character. name of the column containing camera setup dates in `CTtable`
`retrievalCol`	character. name of the column containing camera retrieval dates in `CTtable`
`dateFormat`	character. The format of columns `setupCol` and `retrievalCol` (and potential problem columns) in `CTtable`. Must be interpretable by either `as.Date` or the "orders" argument `parse_date_time` in **lubridate**. Can be a date or a date-time.

Details

The aggregation logic handles different column types as follows:

- **Station ID (specified by 'stationCol'):** Remains unique for each row in the output.
- **Camera ID (if present):** A character vector (comma-separated) of unique camera IDs deployed at that station.
- **Setup/Retrieval Dates:** The earliest setup date and the latest retrieval date for all deployments within a station will be retained.
- **Numeric Columns:** The 'mean' of all values for that station will be calculated.

- **Logical Columns:** The ‘mean‘ (which effectively calculates the proportion of ‘TRUE‘s) of all values for that station will be calculated.
- **Character Columns (other than Camera ID):** A character vector of unique values (comma-separated) for that station will be created.
- **Factor Columns:** Similar to character columns, unique levels will be combined.
- **Geometry Column (‘sf‘ objects):** Centroid (midpoint) of all unique points at a station is calculated.

It is recommended to inspect the aggregated output carefully, especially for columns with mixed data types or specific aggregation requirements not covered by the defaults.

Value

A data frame with one row per unique station, containing aggregated information. Spatial information (for ‘sf‘ objects) is preserved.

Note

The function is mainly intended for aggregating covariates to station level. It does currently not handle camera malfunction (via columns ‘ProblemX_from‘ / ‘ProblemX_to‘) and does not provide proper handling of problem columns. Use `cameraOperation` to aggregate camera trap tables to station level for analyses while accounting for camera malfunction / problem periods.

---

appendSpeciesNames *Add or remove species names from JPEG image filenames*

Description

Add or remove species names from JPEG image filenames. It makes it easier to find images of a species.

Usage

```
appendSpeciesNames(
  inDir,
  IDfrom,
  hasCameraFolders,
  metadataSpeciesTag,
  metadataHierarchyDelimitor = "|",
  removeNames = FALSE,
  writecsv = FALSE
)
```

Arguments

inDir	character. Directory containing camera trap images sorted into station subdirectories (e.g. inDir/StationA/)
IDfrom	character. Read species ID from image metadata ("metadata") of from species directory names ("directory")?
hasCameraFolders	logical. Do the station subdirectories of inDir have camera-subdirectories (e.g. inDir/StationA/CameraA1; inDir/StationA/CameraA2)?
metadataSpeciesTag	character. The species ID tag name in image metadata (if IDfrom = "metadata").
metadataHierarchyDelimitor	character. The character delimiting hierarchy levels in image metadata tags in field "HierarchicalSubject". Either "|" or ":".
removeNames	logical. remove appended species names?
writecsv	logical. write csv table containing old and new file names into inDir?

Details

Species names can be appended or removed from image filenames. Before running the function, you may want to run checkSpeciesIdentification to detect possible misidentifications. As an example, the function would change an image file name from "StationA__2015-05-41__20-59-59(1).JPG" to "StationA__2015-05-41__20-59-59(1)__Species Name.JPG". If species names were appended several times by accident, they can all be removed by running the function with removeNames = TRUE

Value

A data.frame containing the old and new file names and directories.

Author(s)

Juergen Niedballa

Examples

```
## Not run:

# copy sample images to another location (so we don't mess around in the package directory)
wd_images_ID <- system.file("pictures/sample_images_species_dir", package = "camtrapR")
file.copy(from = wd_images_ID, to = getwd(), recursive = TRUE)
wd_images_ID_copy <- file.path(getwd(), "sample_images_species_dir")

# append species names
SpecNameAppend1 <- appendSpeciesNames(inDir           = wd_images_ID_copy,
                                      IDfrom          = "directory",
                                      hasCameraFolders = FALSE,
                                      removeNames     = FALSE,
                                      writecsv        = FALSE)
```

```
SpecNameAppend1

# remove species names
SpecNameRemove1 <- appendSpeciesNames(inDir            = wd_images_ID_copy,
                                      IDfrom           = "directory",
                                      hasCameraFolders = FALSE,
                                      removeNames      = TRUE,
                                      writecsv         = FALSE)

SpecNameRemove1

## End(Not run)
```

`cameraOperation` *Create a camera trap station operation matrix*

Description

Construct a matrix of daily camera trap station operation for use in `detectionHistory` and `spatialDetectionHistory`, where it is needed for calculating trapping effort per occasion. It is also used in `surveyReport` to calculate the number of trap nights durig a survey. If several cameras were deployed per station, the matrix can contain camera- or station-specific trap operation information, or information about sessions during repeated surveys.

Usage

```
cameraOperation(
  CTtable,
  stationCol = "Station",
  cameraCol,
  sessionCol,
  setupCol,
  retrievalCol,
  hasProblems = FALSE,
  byCamera,
  allCamsOn,
  camerasIndependent,
  dateFormat = "ymd",
  occasionStartTime = 0,
  writecsv = FALSE,
  outDir
)
```

Arguments

`CTtable`	data.frame containing information about location and trapping period of camera trap stations

stationCol	character. name of the column specifying Station ID in CTtable
cameraCol	character. name of the column specifying Camera ID in CTtable (optional). If empty, 1 camera per station is assumed.
sessionCol	character. name of the column specifying session ID in CTtable (optional). Use it for creating multi-session / multi-season detection histories (unmarked: unmarkedMultFrame; secr: capthist)
setupCol	character. name of the column containing camera setup dates in CTtable
retrievalCol	character. name of the column containing camera retrieval dates in CTtable
hasProblems	logical. If TRUE, function will look for columns specifying malfunction periods in CTtable (naming convention: ProblemX_from and ProblemX_to, where X is a number)
byCamera	logical. If TRUE, camera operation matrix is computed by camera, not by station (requires cameraCol)
allCamsOn	logical. Takes effect only if cameraCol is defined and if byCamera is FALSE. If allCamsOn = TRUE, all cameras at a station need to be operational for the station to be operational (e.g. 1 camera out of 2 malfunctioning renders the station inoperational). Output values can be 1/0/NA only (all cameras at a station operational/ at least 1 camera not operational/ no camera set up). If allCamsOn = FALSE, at least 1 active camera makes a station operational.
camerasIndependent	logical. Return number of active camera traps by station? Only if byCamera is FALSE and allCamsOn is FALSE. If camerasIndependent is TRUE, output values will be the number of operational cameras at a station. If camerasIndependent is FALSE, the value is 1 if at least 1 camera was operational, otherwise 0. In both cases, values are NA if no camera was set up.
dateFormat	character. The format of columns setupCol and retrievalCol (and potential problem columns) in CTtable. Must be interpretable by either as.Date or the "orders" argument parse_date_time in **lubridate**. Can be a date or (since version 2.1) a date-time.
occasionStartTime	integer. time of day (the full hour) at which to begin occasions. Replaces occasionStartTime from detectionHistory and spatialDetectionHistory.
writecsv	logical. Should the camera operation matrix be saved as a .csv?
outDir	character. Directory into which csv is saved

Details

cameraCol is NULL by default, meaning the function assumes there was 1 camera per station in CTtable. If more than 1 camera was deployed per station, cameraCol needs to be specified to identify individual cameras within a station. Likewise, sessionCol can be used to if camera trap stations were operated during multiple sessions / trapping seasons.

dateFormat defaults to "YYYY-MM-DD", e.g. "2014-10-31", but can be any other date format or date-time also. It can be specified either in the format required by strptime or the 'orders' argument in parse_date_time in **lubridate**. In the example above, "YYYY-MM-DD" would be specified as "%Y-%m-%d" in base R or "ymd" in **lubridate**.

Since version 2.1, dateFormat can be a date-time. That makes it possible to specify the exact time cameras were set up / retrieved / malfunctioned / worked again. This information is used to calculate the daily trapping effort more precisely on days with incomplete effort.

Previously, setup and retrival day were counted as 1, indicating a whole day of effort on those days. Since version 2.1, setup and retrieval are assumed to have happened at 12 noon (resulting in daily effort of 0.5 instead of 1). Users can also specify the exact time cameras were set up (by providing a date-time in the setup / retrieval / problem columns). See vignette 3 for more details.

If `hasProblems` is TRUE, the function tries to find columns `ProblemX_from` and `ProblemX_to` in `CTtable`. `X` is a consecutive number from 1 to n, specifying periods in which a camera or station was not operational. If `hasProblems` is FALSE, cameras are assumed to have been operational uninterruptedly from setup to retrieval (see `camtraps` for details).

`allCamsOn` only has an effect if there was more than 1 camera at a station. If TRUE, for the station to be considered operational, all cameras at a station need to be operational. If FALSE, at least 1 active camera renders the station operational. Argument `camerasIndependent` defines if cameras record animals independently (it thus only has an effect if there was more than 1 camera at a station). This is the case if an observation at one camera does not increase the probability for detection at another camera (cameras face different trails at a distance of one another). Non-independence occurs if an animal is likely to trigger both camers (as would be the case with 2 cameras facing each other).

If `camerasIndependent` is TRUE, 2 active cameras at a station will result in a station operation value of 2 in the resulting matrix, i.e., 2 independent trap days at 1 station and day. If `camerasIndependent` is FALSE, 2 active cameras will return value 1, i.e., 1 trap night at 1 station per day.

Row names depend on the input arguments and contain the station name and potentially session and camera names (if `sessionCol` and/or `cameraCol` are defined).

Naming convention is (since version 1.2) **Bold** information are from the columns `stationCol`, `sessionCol` and `cameraCol` in `CTtable`:

Station
Station__SESS_**SessionID**
Station__CAM_**CameraID**
Station__SESS_**SessionID**__CAM_**CameraID**

Session are designated with prefix "__SESS_", cameras with prefix "__CAM_". Therefore, these are reserved words and may not be part of station, session or camera names. Here's what it may look like in real life:

Station1
Station1__SESS_**2019**
Station1__CAM_**1024152**
Station1__SESS_**2019**__CAM_**1024152**

Functions detectionHistory and spatialDetectionHistory recognize these and use the information accordingly.

Value

A matrix. Row names always indicate Station IDs. If `sessionCol` and/or `cameraCol` are defined, they are contained in the row names also (camera ID only if `byCamera = TRUE`). Column names are dates.

Legend: NA: camera(s) not set up, 0: camera(s) not operational, 1 (or higher): number of operational camera(s) or an indicator for whether the station was operational (depending on `camerasIndependent` and `allCamsOn`)

Note

Setting `camerasIndependent` according to the sampling situation is important for the functions `detectionHistory` and `spatialDetectionHistory`, if sampling effort (the number of active trap nights in a occasion) is to be computed and returned.

Author(s)

Juergen Niedballa

Examples

```
data(camtraps)

# no problems/malfunction
camop_no_problem <- cameraOperation(CTtable      = camtraps,
                                    stationCol   = "Station",
                                    setupCol     = "Setup_date",
                                    retrievalCol = "Retrieval_date",
                                    writecsv     = FALSE,
                                    hasProblems  = FALSE,
                                    dateFormat   = "dmy"
)

# with problems/malfunction
camop_problem <- cameraOperation(CTtable      = camtraps,
                                 stationCol   = "Station",
                                 setupCol     = "Setup_date",
                                 retrievalCol = "Retrieval_date",
                                 writecsv     = FALSE,
                                 hasProblems  = TRUE,
                                 dateFormat   = "dmy"
)

# with problems/malfunction / dateFormat in strptime format
camop_problem_lubridate <- cameraOperation(CTtable      = camtraps,
                                           stationCol   = "Station",
                                           setupCol     = "Setup_date",
                                           retrievalCol = "Retrieval_date",
                                           writecsv     = FALSE,
                                           hasProblems  = TRUE,
                                           dateFormat   = "%d/%m/%Y"
)

camop_no_problem
camop_problem
camop_problem_lubridate
```

camtraps *Sample camera trap station information*

Description

Example camera trap station information table

Usage

```
data(camtraps)
```

Format

A data frame with 3 rows and 7 variables

Details

This is a general example of how information about camera trap stations are arranged in camtrapR. It contains setup and retrieval dates and coordinates. If more than 1 camera was set up at a station (e.g. 2 cameras facing each other), a camera ID column must be added, with camera-specific information instead of station-specific information. If cameras malfunctioned repeatedly, additional pairs of problem columns can be added, e.g. "Problem2_from" and "Problem2_to" etc..

The variables are as follows:

`Station`	Camera trap station ID
`utm_y`	y coordinate of station (northing)
`utm_x`	x coordinate of station (easting)
`Setup_date`	camera trap setup date
`Retrieval_date`	camera trap retrieval date
`Problem1_from`	first day of camera malfunction
`Problem1_to`	last day of camera malfunction

Note

The coordinates can be in the units of any coordinate system. UTM was chosen as an example, but it could be latlong or anything else, too. `capthist` objects (as created by `spatialDetectionHistory` for spatial capture-recapture analyses) expect the unit to be meters.

`camtrapsMultiSeason` *Sample multi-season camera trap station information*

Description

Example multi-season camera trap station information table

Usage

```
data(camtrapsMultiSeason)
```

Format

A data frame with 7 rows and 8 variables

Details

This is a general example of how information about camera trap stations from multiple seasons are arranged in camtrapR. It contains setup and retrieval dates, coordinates and a season identifier. If more than 1 camera was set up at a station (e.g. 2 cameras facing each other), a camera ID column must be added, with camera-specific information instead of station-specific information. If cameras malfunctioned repeatedly, additional pairs of problem columns can be added, e.g. "Problem2_from" and "Problem2_to" etc..

Note that season 2010 has an additional station (StationD). This is to simulate a situation where a station was not set up during an entire season.

The variables are as follows:

`Station`	Camera trap station ID
`utm_y`	y coordinate of station (northing)
`utm_x`	x coordinate of station (easting)
`Setup_date`	camera trap setup date
`Retrieval_date`	camera trap retrieval date
`Problem1_from`	first day of camera malfunction
`Problem1_to`	last day of camera malfunction
`session`	Identified for trapping session / season

Note

The coordinates can be in the units of any coordinate system. UTM was chosen as an example, but it could be latlong or anything else, too. `capthist` objects (as created by `spatialDetectionHistory` for spatial capture-recapture analyses) expect the unit to be meters. `capthist` alse require session information as integer numbers starting with 1.

"Season" and "session" are used synonymously here. **secr** nomenclature is "session", in **unmarked** it is "season".

Examples

```
# data were created with the following code:
data(camtraps)

camtraps_season2 <- camtraps

# change 2009 to 2010
camtraps_season2[, "Setup_date"] <- gsub("2009", "2010", camtraps_season2[,
    "Setup_date"])
camtraps_season2[, "Retrieval_date"] <- gsub("2009", "2010", camtraps_season2[,
    "Retrieval_date"])
camtraps_season2[, "Problem1_from"] <- gsub("2009", "2010", camtraps_season2[,
    "Problem1_from"])
camtraps_season2[, "Problem1_to"] <- gsub("2009", "2010", camtraps_season2[,
    "Problem1_to"])

# add an extra station with different dates in session 2010
camtraps_season2 <- rbind(camtraps_season2, NA)
camtraps_season2$Station[4] <- "StationD"
camtraps_season2$utm_y[4] <- 607050
camtraps_season2$utm_x[4] <- 525000
camtraps_season2$Setup_date[4] <- "04/04/2010"
camtraps_season2$Retrieval_date[4] <- "17/06/2010"
camtraps_season2$Problem1_from[4] <- "20/05/2010"
camtraps_season2$Problem1_to[4] <- "30/05/2010"

# add season column
camtraps$session <- 2009
camtraps_season2$session <- 2010

# combine the tables for 2 seasons
camtrapsMultiSeason <- rbind(camtraps, camtraps_season2)
```

---

checkSpeciesIdentification

Consistency check on species image identification

Description

This function serves 2 purposes: 1) it assesses possible misidentification of species and 2) compares double observer species identification (only if metadata tagging was used for species identification).

Usage

```
checkSpeciesIdentification(
  inDir,
  IDfrom,
  hasCameraFolders,
  metadataSpeciesTag,
```

```
  metadataSpeciesTagToCompare,
  metadataHierarchyDelimitor = "|",
  maxDeltaTime,
  excludeSpecies,
  stationsToCheck,
  writecsv = FALSE
)
```

Arguments

inDir	character. Directory containing identified camera trap images sorted into station subdirectories (e.g. inDir/StationA/)
IDfrom	character. Read species ID from image metadata ("metadata") of from species directory names ("directory")?
hasCameraFolders	logical. Do the station directories in inDir have camera subdirectories (e.g. "inDir/StationA/Camera1" or "inDir/StationA/Camera1/Species1")?
metadataSpeciesTag	character. The species ID tag name in image metadata (if IDfrom = "metadata").
metadataSpeciesTagToCompare	character. A second species ID tag name in image metadata (if IDfrom = "metadata"). For comparing double observer species identification.
metadataHierarchyDelimitor	character. The character delimiting hierarchy levels in image metadata tags in field "HierarchicalSubject". Either "|" or ":"
maxDeltaTime	numeric. Maximum time interval between images to be returned (in seconds)
excludeSpecies	character. vector of species to exclude from checks
stationsToCheck	character. vector of stations to be checked (optionally)
writecsv	logical. Should the resulting data.frame be saved as a .csv?

Details

Within each station, it assesses whether there are images of a species taken within a given time interval of another species. Often, it is unlikely that different species are encountered within a very short time intervals at the same location. This type of misidentification can arise easily if some images belonging to a sequence of images were accidentally moved into different species directories or tagged incorrectly.

Double observer identification may be desirable to increase reliability of species identification. The function returns conflicts in species identification between 2 observers. These conflicts can then be corrected.

Images may accidentally be misidentified by assigning wrong species tags or by moving them into wrong species directories. Imagine your cameras take sequences of images each time they are triggered and one image of the sequence is misidentified. The time difference between these images (that have different species assigned to them) will be very small, usually a few seconds. This function will return all these images for you to check if they were identified correctly.

If multiple observers identify images independently using metadata tagging, their identifications can be compared by setting `metadataSpeciesTagToCompare`. Conflicting or missing identifications will be reported. This feature is only available if images were identified by metadata tagging.

Species like "blank" or "team" can be ignored using `excludeSpecies`. If only specific stations are to be checked, `stationsToCheck` can be set.

Value

A `list` containing 2 data frames. The first contains a data frame with images file names, directories, time stamp and species ID that were taken within `maxDeltaTime` seconds of another species image at a particular station. The second data frame contains images with conflicting species IDs (if `IDfrom = "metadata"` and `metadataSpeciesTagToCompare` is defined)

Note

The function will not be able to find "isolated" images, i.e. images that were misidentified, but were not part of a sequence of images. Likewise, if all images of a sequence were misidentified, they cannot be found either. From version 0.99.0, the function can also handle images identied with metadata tags.

Author(s)

Juergen Niedballa

Examples

```
wd_images_ID <- system.file("pictures/sample_images_species_dir", package = "camtrapR")

if (Sys.which("exiftool") != ""){       # only run this example if ExifTool is available
check.folders <- checkSpeciesIdentification(inDir             = wd_images_ID,
                                            IDfrom            = "directory",
                                            hasCameraFolders  = FALSE,
                                            maxDeltaTime      = 120,
                                            writecsv          = FALSE)

check.folders   # In the example, 2 different species were photographed within 2 minutes.
}


## Not run:
# now exclude one of these 2 species
check.folders2 <- checkSpeciesIdentification(inDir             = wd_images_ID,
                                             IDfrom            = "directory",
                                             hasCameraFolders  = FALSE,
                                             maxDeltaTime      = 120,
                                             excludeSpecies    = "EGY",
                                             writecsv          = FALSE)

check.folders2   # the data frame is empty
```

```
# now we check only one station
check.folders3 <- checkSpeciesIdentification(inDir             = wd_images_ID,
                                             IDfrom            = "directory",
                                             hasCameraFolders  = FALSE,
                                             maxDeltaTime      = 120,
                                             stationsToCheck   = "StationB",
                                             writecsv          = FALSE)
check.folders3   # the data frame is empty

## End(Not run)
```

---

checkSpeciesNames *Check species names against the ITIS taxonomic database*

Description

The function checks species names (common or scientific names) provided by the user with the ITIS taxonomic database (https://www.itis.gov/) via functions from the package **taxize**. It returns both common and scientific names, the taxon authors, taxon rank name and status, the TSN (taxonomic serial numbers) and ITIS urls.

Usage

```
checkSpeciesNames(speciesNames, searchtype, accepted = TRUE, ask = TRUE)
```

Arguments

speciesNames	character. Vector of species names to check. Either common names or scientific names.
searchtype	character. Type of names specified in speciesNames. One of 'scientific' or 'common'.
accepted	logical. Return only accepted valid names? If TRUE, invalid names are returned as NA. Set to FALSE to return both accepted and unaccepted names.
ask	logical. Should the function be run in interactive mode? If TRUE and more than one TSN is found for a species, the user is asked to choose one. If FALSE, NA is returned for multiple matches.

Details

Arguments searchtype, accepted and ask are passed on to get_tsn.

Value

A data.frame with the names supplied by the user, matching common and scientific names, taxon author and year, taxonomic rank, status, TSNs (taxonomic serial numbers) and ITIS urls.

Author(s)

Juergen Niedballa

References

https://www.itis.gov/

Examples

```
## Not run:


species_common <- c("Leopard Cat", "moonrat")

# ask = TRUE. Multiple matches for leopard cat will cause menu to pop up asking user input.

species.names.check1 <- checkSpeciesNames(speciesNames = species_common,
                                          searchtype   = "common",
                                          accepted     = TRUE,
                                          ask          = TRUE)
2   # we choose entry 2
species.names.check1


# ask = FALSE. Multiple matches for leopard cat will cause NA.

species.names.check2 <- checkSpeciesNames(speciesNames = species_common,
                                          searchtype   = "common",
                                          accepted     = TRUE,
                                          ask          = FALSE)
species.names.check2



# search for scientific names

species_scientific <- c("Tragulus", "Prionailurus bengalensis")

species.names.check3 <- checkSpeciesNames(speciesNames = species_scientific,
                                          searchtype   = "scientific",
                                          accepted     = TRUE,
                                          ask          = TRUE)
species.names.check3

## End(Not run)
```

commOccu-class *commOccu objects*

Description

commOccu objects

Value

commOccu object

Slots

modelText JAGS model code as a character vector (made up of code chunks, use cat() to print)

params Parameters to monitor in the model runs

inits_fun Function to create start values for the MCMC chains. It being a function ensures different values in each chain

data List with data needed to run the model (detection & effort matrices, site covariates, number of species / stations / occasions)

input Input data_list (unchanged)

nimble logical indicator for whether it is a Nimble model

modelFile Path of the text file containing the model code

covariate_info Data frame containing information about covariates. Only used internally in plot_* and predict methods

model character indicating whether it is a standard "Occupancy" model or Royle-Nichols ("RN") occupancy model

Note

The data slot is a list of model input data. While the exact content depends on function input, it can be summarized as:

y	array of detection histories. Dimensions are: y[species, station, occasion]
effort_binary	matrix of binary (1/0) survey effort. Only used to ensure p = 0 when effort = 0. Dimensions ar
site-occasion covariates	The required content of data_list$obsCovs as named matrices with dimensions [station, occasi
site covariates	The required columns of data_list$siteCovs as named vectors (length = number of stations)
M	Number of species
J	Number of stations
maxocc	Number of occasions

For categorical site-occasion covariates, an addition matrix containing an integer representation of the character matrix with suffix "_integer" is stored in the data slot.

communityModel *Create a community (multi-species) occupancy model for JAGS or Nimble*

Description

Flexibly creates complete code and input data for community occupancy models for JAGS amd Nimble (both standard occupancy models and Royle-Nichols occupancy models), and automatically sets initial values and parameters to monitor. Supports fixed and random effects of covariates on detection and occupancy probabilities, using both continuous and categorical covariates (both site and site-occasion covariates).

Optionally includes data augmentation (fully open community, or up to known maximum number of species, or no data augmentation). Allows combination of all these parameters for fast and flexible customization of community occupancy models.

Incidentally, the function can also be used to create model code and input for single-species single-season occupancy models (it is the special case of the community model with only one species). Such a model will run slower than proper single-species model JAGS code due to the additional species loop, but it is possible.

The function returns several derived quantities, e.g. species richness, Bayesian p-values (overall and by species), Freeman-Tukey residuals for actual and simulated data (by station and total). If doing data augmentation, metacommunity size and number of unseen species are returned also.

Usage

```
communityModel(
  data_list,
  model = c("Occupancy", "RN"),
  occuCovs = list(fixed = NULL, independent = NULL, ranef = NULL),
  detCovs = list(fixed = NULL, ranef = NULL),
  detCovsObservation = list(fixed = NULL, ranef = NULL),
  speciesSiteRandomEffect = list(det = FALSE, occu = FALSE),
  intercepts = list(det = "ranef", occu = "ranef"),
  effortCov = "effort",
  richnessCategories = NULL,
  augmentation = NULL,
  modelFile = NULL,
  nimble = FALSE,
  keyword_quadratic = "_squared"
)
```

Arguments

data_list	list. Contains 3 slots: ylist, siteCovs, obsCovs. ylist is a list of detection histories (can be named), e.g. from detectionHistory. siteCovs is a data.frame with site covariates (optional). obsCovs is a list of site-occasion level covariates (e.g. site-occasion-specific effort, which is also returned by detectionHistory.

`model` character. "Occupancy" for standard occupancy model, or "RN" for the occupancy model of Royle and Nichols (2003), which relates probability of detection of the species to the number of individuals available for detection at each station

`occuCovs` list. Up to 3 items named "fixed", "independent", and/or "ranef". Specifies fixed, independent or random effects of covariates on occupancy probability (continuous or categorical covariates). Independent effects are only supported for continuous covariates.

`detCovs` list. Up to 3 items named "fixed", "independent", and/or "ranef". Specifies fixed, independent or random effects of covariates on detection probability (continuous or categorical covariates). Independent effects are only supported for continuous covariates.

`detCovsObservation` list. Up to 2 items named "fixed" and/or "ranef". Specifies fixed or random effects of observation-level covariates on detection probability (continuous or categorical covariates - categorical must be coded as character matrix)

`speciesSiteRandomEffect` list. Two items named "det" and "occu". If TRUE, adds a random effect of species and station. Only implemented for detection probability.

`intercepts` list. Two items named "det" and "occu" for detection and occupancy probability intercepts. Values can be "fixed" (= constant across species), "independent" (= independent estimates for each species), or "ranef" (= random effect of species on intercept).

`effortCov` character. Name of list item in `data_list$obsCovs` which contains effort. This does not include effort as a covariate on detection probability, but only uses NA / not NA information to create binary effort and ensure detection probabilities p are 0 when there was no effort (p will be 0 whereever `effortCov` is NA).

`richnessCategories` character. Name of categorical covariate in `data_list$siteCovs` for which to calculate separate richness estimates (optional). Can be useful to obtain separate richness estimates for different areas.

`augmentation` If NULL, no data augmentation (only use species in `data_list$ylist`), otherwise named list or vector with total number of (potential) species. Names: "maxknown" or "full". Example: `augmentation = c(maxknown = 30)` or `augmentation = c(full = 30)`

`modelFile` character. Text file name to save model to

`nimble` logical. If TRUE, model code will be for Nimble (incompatible with JAGS). If FALSE, model code is for JAGS.

`keyword_quadratic` character. A suffix in covariate names in the model that indicates a covariate is a quadratic effect of another covariate which does not carry the suffix in its name (e.g. if the covariate is "elevation", the quadratic covariate would be "elevation_squared").

Details

For examples of implementation, see Vignette 5: Multi-species occupancy models.

Fixed effects of covariates are constant across species, whereas random effect covariates differ between species. Independent effect differ between species and are independent (there is no underlying hyperdistribution). Fixed, independent and random effects are allowed for station-level detection and occupancy covariates (a.k.a. site covariates). Fixed and random effects are also allowed for station-occasion level covariates (a.k.a. observation covariates). Currently independent effects are only supported for continuous site covariates, not categorical site covariates or observation-level covariates.

By default, random effects will be by species. It is however possible to use categorical site covariates for grouping (continuous|categorical). Furthermore, is is possible to use use nested random effects of species and another categorical site covariate (so that there is a random effect of species and an additional random effect of a categorical covariate within each species).

Derived quantities returned by the model are:

`Bpvalue`	Bayesian p-value (overall)
`Bpvalue_species`	Bayesian p-value (by species)
`Nspecies`	Species richness (only in JAGS models)
`Nspecies_station`	Species richness at each sampling locations (only in JAGS models)
`Nspecies_Covariate`	Species richness by categorical covariate (when using `richnessCategories`, only in JAGS models)
`R2`	sum of Freeman-Tukey residuals of observed data within each species
`new.R2`	sum of Freeman-Tukey residuals of simulated data within each species
`R3`	Total sum of Freeman-Tukey residuals of observed data
`new.R3`	Total sum of Freeman-Tukey residuals of simulated data
`Ntotal`	Total metacommunity size (= observed species + n0)
`n0`	Number of unseen species in metacommunity
`omega`	Data augmentation parameter
`w`	Metacommunity membership indicator for each species

Quantities in *italic* at the bottom are only returned in full data augmentation. `Nspecies` and `Nspecies_Covariate` are only returned in JAGS models (because Nimble models don't explicitly return latent occupancy status z).

Value

`commOccu` object. It is an S4 class containing all information required to run the models. See `commOccu-class` for details.

Parameter naming convention

The parameter names are assembled from building blocks. The nomenclature is as follows:

Name	Refers to	Description
`alpha`	Submodel	detection submodel
`beta`	Submodel	occupancy submode
`0`	Intercept	denotes the intercepts (alpha0, beta0)
`fixed`	Effect type	fixed effects (constant across species)
`indep`	Effect type	independent effects (separate for each species)
`ranef`	Effect type	random effects (of species and/or other categorical covariates)
`cont`	Covariate type	continuous covariates

categ	Covariate type	categorical covariates
mean	Hyperparameter	mean of random effect
sigma	Hyperparameter	standard deviation of random effect
tau	Hyperparameter	precision of random effect (used internally, not returned)

For example, a fixed intercept of occupancy (constant across species) is beta0, and a fixed intercept of detection probability is alpha0.

An occupancy probability intercept with a random effect of species is:

beta0.mean community mean of the occupancy probability intercept

beta0.sigma standard deviation of the community occupancy probability intercept.

beta0[1] occupancy probability intercept of species 1 (likewise for other species).

For effects of site covariates, the pattern is:

submodel.effectType.covariateType.CovariateName.hyperparameter

For example:

beta.ranef.cont.habitat.mean is the mean community effect of the continuous site covariate 'habitat' on occupancy probability.

beta.ranef.cont.habitat[1] is the effect of continuous site covariate 'habitat' on occupancy probability of species 1.

Site-occasion covariates are denoted by ".obs" after the submodel, e.g.:

alpha.obs.fixed.cont.effort is the fixed effect of the continuous observation-level covariate 'effort' on detection probability

Author(s)

Juergen Niedballa

References

Kéry, M., and J. A. Royle. "Applied hierarchical modelling in ecology - Modeling distribution, abundance and species richness using R and BUGS." Volume 1: Prelude and Static Models. Elsevier/Academic Press, 2016.

Examples

```
## Not run:
# the example below fits community occupancy models to the sample data in camtrapR
# models are fit both in JAGS and Nimble
# The data set only contains 5 species and 3 stations, so the results will be nonsense.
# It is only a technical demonstration with the camtrapR workflow
# for more complete examples, see vignette 5

data("camtraps")

# create camera operation matrix
camop_no_problem <- cameraOperation(CTtable      = camtraps,
                                    stationCol   = "Station",
```

```
                                    setupCol     = "Setup_date",
                                    retrievalCol = "Retrieval_date",
                                    hasProblems  = FALSE,
                                    dateFormat   = "dmy"
)

data("recordTableSample")

# make list of detection histories
species_to_include <- unique(recordTableSample$Species)

  DetHist_list <- detectionHistory(
    recordTable         = recordTableSample,
    camOp               = camop_no_problem,
    stationCol          = "Station",
    speciesCol          = "Species",
    recordDateTimeCol   = "DateTimeOriginal",
    species             = species_to_include,
    occasionLength      = 7,
    day1                = "station",
    datesAsOccasionNames = FALSE,
    includeEffort       = TRUE,
    scaleEffort         = TRUE,
    timeZone            = "Asia/Kuala_Lumpur"
)


# create some fake covariates for demonstration
sitecovs <- camtraps[, c(1:3)]
sitecovs$elevation <- c(300, 500, 600)

# scale numeric covariates
sitecovs[, c(2:4)] <- scale(sitecovs[,-1])


# bundle input data for communityModel
data_list <- list(ylist = DetHist_list$detection_history,
                  siteCovs = sitecovs,
                  obsCovs = list(effort = DetHist_list$effort))


# create community model for JAGS
modelfile1 <- tempfile(fileext = ".txt")
mod.jags <- communityModel(data_list,
                           occuCovs = list(fixed = "utm_y", ranef = "elevation"),
                           detCovsObservation = list(fixed = "effort"),
                           intercepts = list(det = "ranef", occu = "ranef"),
                           modelFile = modelfile1)

summary(mod.jags)

# fit in JAGS
fit.jags <- fit(mod.jags,
```

```
                n.iter = 1000,
                n.burnin = 500,
                chains = 3)
summary(fit.jags)

# response curves (= marginal effect plots)
plot_effects(mod.jags,
             fit.jags,
             submodel = "state")
plot_effects(mod.jags,
             fit.jags,
             submodel = "det")

# effect sizes plot
plot_coef(mod.jags,
          fit.jags,
          submodel = "state")
plot_coef(mod.jags,
          fit.jags,
          submodel = "det")

# create community model for Nimble
modelfile2 <- tempfile(fileext = ".txt")
mod.nimble <- communityModel(data_list,
                             occuCovs = list(fixed = "utm_x", ranef = "utm_y"),
                             detCovsObservation = list(fixed = "effort"),
                             intercepts = list(det = "ranef", occu = "ranef"),
                             modelFile = modelfile2,
                             nimble = TRUE)      # set nimble = TRUE

# load nimbleEcology package
# currently necessary to do explicitly, to avoid additional package dependencies
require(nimbleEcology)

# fit uncompiled model in Nimble
fit.nimble.uncomp <- fit(mod.nimble,
                         n.iter = 10,
                         chains = 1)

# fit compiled model in Nimble
fit.nimble.comp <- fit(mod.nimble,
                       n.iter = 5000,
                       n.burnin = 2500,
                       chains = 3,
                       compile = TRUE)

# parameter summary statistics
summary(fit.nimble.comp)


# response curves (= marginal effect plots)
plot_effects(mod.nimble,
             fit.nimble.comp,
```

```
              submodel = "state")
plot_effects(mod.nimble,
              fit.nimble.comp,
              submodel = "det")

# effect sizes plot
plot_coef(mod.nimble,
          fit.nimble.comp,
          submodel = "state")
plot_coef(mod.nimble,
          fit.nimble.comp,
          submodel = "det")

# traceplots
plot(fit.nimble.comp)



## End(Not run)
```

`createCovariates` *Extract covariate values from spatial rasters and prepare rasters for spatial predictions*

Description

This function extracts covariate values from spatial raster data (e.g. for use in modelling) and prepares these covariates for use in spatial predictions.

It accepts a camera trap table containing spatial information, along with either a directory containing covariate raster files, a character vector specifying the file paths of these covariate rasters, or direct SpatRaster objects from the terra package.

Additionally, users can provide parameters to control how covariates are extracted, and how they are aggregated to prediction rasters.

The function can also download elevation data and calculate terrain metrics if requested.

The function generates prediction rasters based on a provided template or creates one automatically if no template is provided.

The function returns a list containing the camera trap dataset with extracted covariate values (e.g for use in occupancy modelling), and prediction rasters ready for spatial modeling.

Usage

```
createCovariates(
  CTtable,
  directory,
  filenames,
  rasters,
```

```
  buffer_ct = 0,
  bilinear = FALSE,
  buffer_aoi = 1000,
  raster_template = NULL,
  resolution = NULL,
  append = TRUE,
  formats = ".tif",
  recursive = FALSE,
  download_elevation = FALSE,
  elevation_zoom = 10,
  terrain_measures = NULL,
  standardize_na = FALSE,
  scale_covariates = FALSE
)
```

Arguments

CTtable	sf object as defined by the sf package. Essentially a camera trap data frame with spatial information.
directory	character. The directory containing the covariate rasters.
filenames	character (optionally named). A vector of file paths of covariate rasters. If it is named the covariates will be named according to the names. If unnamed the file names will be used as covariate names.
rasters	SpatRaster object or list of SpatRaster objects. Direct input of rasters from the terra package instead of reading from disk.
buffer_ct	numeric. A value (in meters) by which to buffer the point locations in CTtable for extraction of covariate values.
bilinear	logical. If TRUE, extract covariate values with bilinear interpolation (nearest 4 raster cells). If FALSE, extract value at the cell the point falls in. Only relevant if buffer_ct is 0.
buffer_aoi	numeric. A value (in meters) by which to buffer the overall camera trapping grid to ensure that prediction rasters are larger than the camera trapping grid.
raster_template	SpatRaster. A SpatRaster (as defined in the terra package) to use as template for the creation of prediction rasters.
resolution	numeric. Spatial resolution of prediction rasters in the units of the coordinate reference system. Ignored if raster_template is provided.
append	logical. If TRUE, add the extracted covariates to the existing CTtable. If FALSE, return only the extracted covariate values without the existing CTtable.
formats	character. Possible file formats for raster data (must include the dot). Defaults to .tif files.
recursive	logical. If TRUE, search for raster files recursively in subdirectories when a directory is provided. Defaults to FALSE.
download_elevation	logical. If TRUE, download elevation data from AWS. Defaults to FALSE.

`elevation_zoom` numeric. Zoom level for elevation data download (6-12). Higher values provide more detail but longer download times. Zoom 12 corresponds to ~20m pixel resolution, 11 to ~40m, 10 to ~80m, and so on (resolution halves with each decrease in zoom level). Defaults to 9.

`terrain_measures`
character. Vector of terrain metrics to calculate from elevation data. Options include "slope" (slope in degrees), "aspect" (compass direction in degrees), "TRI" (Terrain Ruggedness Index, measuring elevation difference between adjacent cells), "TPI" (Topographic Position Index, comparing cell elevation to mean of surrounding cells), and "roughness" (difference between max and min of surrounding cells). Defaults to NULL (no terrain metrics).

`standardize_na` logical. Logical. If `TRUE`, ensures all layers in the prediction raster have identical NA patterns by setting a cell to NA in all bands if it's NA in any band. This creates consistency for spatial predictions across covariates but may lose data in covariates.

`scale_covariates`
logical. If `TRUE`, scale numeric covariates and return both original and scaled versions of data and prediction rasters. Scaling is performed using R's `scale` function. Defaults to `FALSE`.

Details

The input camera trap table must be an `sf` object (a data frame with a geometry column specifying the spatial information). For details on how to convert an exising camera trap table to sf, see the examples below. The input rasters can be in different coordinate systems. During covariate extraction the CTtable is projected to each raster's coordinate system individually. For the prediction raster all input rasters are either resampled or reprojected to a consistent coordinate system.

When `recursive = TRUE` and a directory is provided, the function will search for raster files in all subdirectories. In this case, the subdirectory names are used as covariate names, and only one raster file per subdirectory is allowed.

When `download_elevation = TRUE`, the function will download elevation data from AWS using the `elevatr` package. The `elevation_zoom` parameter controls the level of detail, with values between 6 and 12. Higher zoom levels provide finer resolution but require longer download times and may consume significant memory. Approximate resolutions: zoom 12 = ~20m, 11 = ~40m, 10 = ~80m, etc.

If `terrain_measures` is specified, the function calculates the requested terrain metrics from the elevation data using `terra::terrain()` with the default 3x3 neighborhood. Available terrain metrics include "slope", "aspect", "TRI" (Terrain Ruggedness Index), "TPI" (Topographic Position Index), and "roughness".

When using `scale_covariates = TRUE`, the function returns both original and scaled versions of the data and prediction rasters. #' The function uses R's `scale()` function to perform centering and scaling, and includes the scaling parameters in the returned metadata.

Warning about Categorical Covariates: This function does not explicitly handle categorical rasters. All raster values are treated as numeric, which can be problematic when scaling is applied. The function attempts to identify "likely categorical" variables (numeric variables with few unique integer values) and will provide warnings, but it cannot automatically handle them correctly for scaling.

When using scaled covariates with categorical variables in models:

- Use `CTtable_scaled` for numeric predictors
- Use `CTtable` (original) for categorical predictors
- Similarly, use `predictionRaster_scaled` for numeric predictors in spatial predictions
- Use `predictionRaster` for categorical predictors in spatial predictions

Future versions may implement proper categorical raster handling with RAT (Raster Attribute Table) support.

Value

When `scale_covariates = FALSE`, a list containing three elements:

An `sf` object representing the camera trap data frame with extracted covariate values.

CTtable predictionRaster A `SpatRaster` object containing covariate raster layers

originalRaster A list of the original input rasters

When `scale_covariates = TRUE`, a list containing six elements:

CTtable The original `sf` object with unscaled covariate values

CTtable_scaled The `sf` object with scaled numeric covariate values

predictionRaster The original unscaled prediction raster

predictionRaster_scaled The prediction raster with scaled numeric layers

originalRaster A list of the original input rasters

scaling_params A list containing center and scale information of numeric covariates

Examples

```
## Not run:
# load camera trap table
data(camtraps)

# create sf object
camtraps_sf <- st_as_sf(camtraps,
                        coords = c("utm_x", "utm_y"),
                        crs = 32650)

# extract covariates (with 100m buffer around cameras)
# doesn't run because 'directory' is only a placeholder

covariates <- createCovariates(camtraps_sf,
"path/to/covariate_rasters",
buffer_ct = 100,
buffer_aoi = 1000,
resolution = 100)


# extract covariates with elevation data (this code runs)

covariates_elev <- createCovariates(camtraps_sf,
buffer_ct = 100,
```

```
buffer_aoi = 1000,
resolution = 100,
download_elevation = TRUE,
elevation_zoom = 11,
terrain_measures = c("slope", "aspect", "TRI"))

# Note that if local rasters are available they can be extracted alongside
# elevation data in a single function call

# camera trap table with extracted covariate values
camtraps_sf_cov <- covariates_elev$CTtable

# covariate raster layer
r_cov <- covariates_elev$predictionRaster
plot(r_cov)

# Use SpatRaster objects directly as input
r1 <- rast("elevation.tif")
r2 <- rast("landcover.tif")
raster_list <- list(elevation = r1, landcover = r2)

covariates_direct <- createCovariates(camtraps_sf,
                                      rasters = raster_list,
                                      buffer_ct = 100,
                                      resolution = 100)

# Scale numeric covariates for modeling
covariates_scaled <- createCovariates(camtraps_sf,
                                      rasters = raster_list,
                                      buffer_ct = 100,
                                      resolution = 100,
                                      scale_covariates = TRUE)

# Use scaled data with categorical variables
# Mix and match from original and scaled outputs for tabular data
model_data <- covariates_scaled$CTtable_scaled  # Use scaled numeric covariates
model_data$landcover <- covariates_direct$CTtable$landcover  # Use original categorical covariate

# Mix and match for prediction rasters
# Create combined prediction raster (scaled numeric variables & original categorical variables)
# Extract scaled elevation layer
elev_scaled <- covariates_scaled$predictionRaster_scaled$elevation

# Extract original landcover layer (categorical)
landcover_orig <- covariates_direct$predictionRaster$landcover

# Combine into a new SpatRaster for predictions
prediction_raster <- c(elev_scaled, landcover_orig)
names(prediction_raster) <- c("elevation", "landcover")

# Use this combined raster for spatial predictions
plot(prediction_raster)
```

```
## End(Not run)
```

---

createSpeciesFolders *Create species directories for species identification*

Description

This function creates species subdirectories within station directories. They can be used for species identification by manually moving images into the respective species directories. The function can also delete empty species directories (if species were not detected at sites). It is not necessary to run this function if animals will be identified by metadata tagging.

Usage

```
createSpeciesFolders(inDir, hasCameraFolders, species, removeFolders = FALSE)
```

Arguments

inDir	character. Directory containing camera trap images sorted into station subdirectories (e.g. inDir/StationA/)
hasCameraFolders	logical. Do the station directories in inDir have camera-subdirectories (e.g. inDir/StationA/CameraA1; inDir/StationA/CameraA2)?
species	character. names of species directories to be created in every station (or station/camera) subdirectory of inDir
removeFolders	logical. Indicating whether to create (TRUE) or remove (FALSE) species directories .

Details

This function should be run after imageRename. Empty directories can be created as containers for species identification if images are identified with the drag & drop method. After species identification is complete, empty species directories can be deleted using removeFolders = TRUE. The function will delete only directories which are specified in species. If hasCameraFolders was set to TRUE in function imageRename, hasCameraFolders must be set to TRUE here too. Species directories will then be created within each camera subdirectory of each station directory. if the user wishes to identify species by metadata tagging, running this function is not needed.

Value

A data.frame with directory names and an indicator for whether directories were created or deleted.

Author(s)

Juergen Niedballa

Examples

```
## Not run:

# create dummy directories for tests
# (normally, you'd use directory containing renamed, unsorted images)

# this will be used as inDir
wd_createDirTest <- file.path(getwd(), "createSpeciesFoldersTest")

# now we create 2 station subdirectories
dirs_to_create <- file.path(wd_createDirTest, c("StationA", "StationB"))
sapply(dirs_to_create, FUN = dir.create, recursive = TRUE)

# species names for which we want to create subdirectories
species <- c("Sambar Deer", "Bay Cat")

# create species subdirectories
SpecFolderCreate1 <- createSpeciesFolders (inDir               = wd_createDirTest,
                                           species             = species,
                                           hasCameraFolders = FALSE,
                                           removeFolders       = FALSE)

SpecFolderCreate1

# check if directories were created
list.dirs(wd_createDirTest)

# delete empty species directories
SpecFolderCreate2 <- createSpeciesFolders (inDir               = wd_createDirTest,
                                           species             = species,
                                           hasCameraFolders = FALSE,
                                           removeFolders       = TRUE)

SpecFolderCreate2

# check if species directories were deleted
list.dirs(wd_createDirTest)


## End(Not run)
```

---

createStationFolders *Create camera trap station directories for raw camera trap images*

Description

This function creates camera trap station directories, if needed with camera subdirectories. They can be used as an initial directory structure for storing raw camera trap images.

Usage

```
createStationFolders(inDir, stations, cameras, createinDir)
```

Arguments

inDir	character. Directory in which station directories are to be created
stations	character. Station IDs to be used as directory names within inDir
cameras	character. Camera trap IDs to be used as subdirectory names in each station directory (optionally)
createinDir	logical. If inDir does not exist, create it?

Details

The empty directories serve as containers for saving raw camera trap images. If more than 1 camera was set up at a station, specifying cameras is required in order to keep images from different cameras separate. Otherwise, generic filenames (e.g., IMG0001.JPG) from different cameras may lead to accidental overwriting of images if images from these cameras are saved in one station directory.

Value

A data.frame with station (and possibly camera) directory names and an indicator for whether they were created successfully.

Author(s)

Juergen Niedballa

Examples

```
## Not run:

# create dummy directory for tests (this will be used as inDir)
# (normally, you'd set up an empty directory, e.g. .../myStudy/rawImages)
wd_createStationDir <- file.path(tempdir(), "createStationFoldersTest")

# now we load the sample camera trap station data frame
data(camtraps)

# create station directories in wd_createStationDir
StationFolderCreate1 <- createStationFolders (inDir       = wd_createStationDir,
                                              stations    = as.character(camtraps$Station),
                                                 createinDir = TRUE)

StationFolderCreate1

# check if directories were created
list.dirs(wd_createStationDir)
```

```
## End(Not run)
```

`detectionHistory` *Species detection histories for occupancy analyses*

Description

This function generates species detection histories that can be used in single-species occupancy analyses with packages unmarked and **ubms**, as well as multi-species/community occupancy models via `communityModel`. It generates detection histories in different formats, with adjustable occasion length and occasion start time.

Usage

```
detectionHistory(
  recordTable,
  species,
  camOp,
  output = c("binary", "count"),
  stationCol = "Station",
  speciesCol = "Species",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  occasionLength,
  minActiveDaysPerOccasion,
  maxNumberDays,
  day1 = "survey",
  buffer,
  includeEffort = TRUE,
  scaleEffort = FALSE,
  occasionStartTime = "deprecated",
  datesAsOccasionNames = FALSE,
  timeZone,
  writecsv = FALSE,
  outDir,
  unmarkedMultFrameInput
)
```

Arguments

`recordTable`	data.frame. the record table created by `recordTable`
`species`	character. species name(s) for which to compute detection histories. Can be either a single species name (for use with **unmarked/ubms**) or a vector of multiple species names (for input to `communityModel`)
`camOp`	The camera operation matrix as created by `cameraOperation`

output character. Return binary detections ("binary") or counts of detections ("count")

stationCol character. name of the column specifying Station ID in recordTable

speciesCol character. name of the column specifying species in recordTable

recordDateTimeCol
: character. name of the column specifying date and time in recordTable

recordDateTimeFormat
: character. Format of column recordDateTimeCol in recordTable

occasionLength integer. occasion length in days

minActiveDaysPerOccasion
: integer. minimum number of active trap days for occasions to be included (optional)

maxNumberDays integer. maximum number of trap days per station (optional)

day1 character. When should occasions begin: station setup date ("station"), first day of survey ("survey"), a specific date (e.g. "2015-12-31")?

buffer integer. Makes the first occasion begin a number of days after station setup. (optional)

includeEffort logical. Compute trapping effort (number of active camera trap days per station and occasion)?

scaleEffort logical. scale and center effort matrix to mean = 0 and sd = 1?

occasionStartTime
: (DEPRECATED) integer. time of day (the full hour) at which to begin occasions. Please use argument occasionStartTime in cameraOperation instead.

datesAsOccasionNames
: If day1 = "survey", occasion names in the detection history will be composed of first and last day of that occasion.

timeZone character. Must be a value returned by OlsonNames

writecsv logical. Should the detection history be saved as a .csv?

outDir character. Directory into which detection history .csv file is saved

unmarkedMultFrameInput
: logical. Return input for multi-season occupancy models in unmarked (argument "y" in unmarkedMultFrame?

Details

The function creates species detection matrices in two possible formats: detection-by-date or detection-by-occasion. The start of detection histories is controlled by day1:

- "station": Each station's history begins on its setup day
- "survey": All stations begin on the first day of the survey
- A specific date (e.g., "2015-12-31"): All stations begin on this date

Dates must be in "YYYY-MM-DD" format if specified directly.

Two output formats are available via the output parameter:

- "binary": Records detection (1) or non-detection (0)
- "count": Records the number of detections per occasion

The includeEffort parameter determines how camera operation affects the output:

- If FALSE: Periods when cameras were not operational or only partly operational appear as NA in the detection history. This may lose species record from incomplete occasions.
- If TRUE: Incomplete occasions are retained. Outputs contain a separate effort matrix that can be used as an observation covariate in occupancy models.

It is generally advisable to include effort as a covariate to account for uneven sampling effort.

occasionLength controls how many days are aggregated into each sampling occasion. Note that occasionStartTime has moved to cameraOperation to ensure proper calculation of daily effort.

The values of stationCol in recordTable must be matched by the row names of camOp (case-insensitive), otherwise an error is raised.

For date/time formatting, recordDateTimeFormat accepts two syntax styles:

- Base R style (using %): e.g., "%Y-%m-%d %H:%M:%S"
- lubridate style: e.g., "ymd HMS"

lubridate will be used if there are no "%" characters in recordDateTimeFormat. The default and recommended format is "YYYY-MM-DD HH:MM:SS" (e.g., "2014-09-30 22:59:59").

For multi-season studies where sessionCol was used in cameraOperation, the function automatically detects this structure. Set unmarkedMultFrameInput = TRUE to format output for unmarkedMultFrame, with rows representing sites and columns ordered by season-major, occasion-minor (e.g., season1-occasion1, season1-occasion2, etc.).

Value

If a single species is provided (typical for **unmarked/ubms** analyses), returns a list with either 1, 2 or 3 elements depending on the value of includeEffort and scaleEffort:

detection_history
: A species detection matrix

effort
: A matrix giving the number of active camera trap days per station and occasion (= camera trapping effort). Only returned if includeEffort = TRUE

effort_scaling_parameters
: Scaling parameters of the effort matrix. Only returned if includeEffort and scaleEffort are TRUE

If multiple species are provided (for use with communityModel), returns a similar list structure but with detection_history containing a named list of detection matrices, one for each species. The effort matrix is identical for all species and thus returned only once.

Warning

Setting output = "count" returns a count of detections, not individuals. These counts are not suitable for abundance modeling (e.g., N-mixture models) as they do not represent individual animals.

For important information about the timeZone parameter, please refer to the "Data Extraction" vignette (vignette("DataExtraction") or online at https://cran.r-project.org/package=camtrapR/vignettes/camtrapr3.pdf).

Author(s)

Juergen Niedballa

Examples

```
# define image directory
wd_images_ID <- system.file("pictures/sample_images_species_dir", package = "camtrapR")

# load station information
data(camtraps)

# create camera operation matrix
camop_no_problem <- cameraOperation(CTtable      = camtraps,
                                    stationCol   = "Station",
                                    setupCol     = "Setup_date",
                                    retrievalCol = "Retrieval_date",
                                    hasProblems  = FALSE,
                                    dateFormat   = "dmy"
)

## Not run:
if (Sys.which("exiftool") != ""){       # only run this function if ExifTool is available
recordTableSample <- recordTable(inDir               = wd_images_ID,
                                 IDfrom              = "directory",
                                 minDeltaTime        = 60,
                                 deltaTimeComparedTo = "lastRecord",
                                 exclude             = "UNID",
                                 timeZone            = "Asia/Kuala_Lumpur"
)
}

## End(Not run)
data(recordTableSample)    # load the record history, as created above


# compute detection history for a species

# without trapping effort
DetHist1 <- detectionHistory(recordTable         = recordTableSample,
                             camOp               = camop_no_problem,
                             stationCol          = "Station",
                             speciesCol          = "Species",
                             recordDateTimeCol   = "DateTimeOriginal",
                             species             = "VTA",
                             occasionLength      = 7,
                             day1                = "station",
                             datesAsOccasionNames = FALSE,
                             includeEffort       = FALSE,
                             timeZone            = "Asia/Kuala_Lumpur"
)
```

```
DetHist1                     # this is a list with 1 element
DetHist1$detection_history   # this is the contained detection/non-detection matrix


# with effort / using base R to define recordDateTimeFormat
DetHist2 <- detectionHistory(recordTable          = recordTableSample,
                             camOp                = camop_no_problem,
                             stationCol           = "Station",
                             speciesCol           = "Species",
                             recordDateTimeCol    = "DateTimeOriginal",
                             species              = "VTA",
                             occasionLength       = 7,
                             day1                 = "station",
                             datesAsOccasionNames = FALSE,
                             includeEffort        = TRUE,
                             scaleEffort          = FALSE,
                             timeZone             = "Asia/Kuala_Lumpur"
)

DetHist2$detection_history  # detection history  (alternatively, use: DetHist2[[1]])
DetHist2$effort             # effort (alternatively, use: DetHist2[[2]])

# with effort / using lubridate package to define recordDateTimeFormat
DetHist2_lub <- detectionHistory(recordTable          = recordTableSample,
                                 camOp                = camop_no_problem,
                                 stationCol           = "Station",
                                 speciesCol           = "Species",
                                 recordDateTimeCol    = "DateTimeOriginal",
                                 recordDateTimeFormat = "ymd HMS",
                                 species              = "VTA",
                                 occasionLength       = 7,
                                 day1                 = "station",
                                 datesAsOccasionNames = FALSE,
                                 includeEffort        = TRUE,
                                 scaleEffort          = FALSE,
                                 timeZone             = "Asia/Kuala_Lumpur"
)

DetHist2_lub$detection_history  # detection history  (alternatively, use: DetHist2_lub[[1]])
DetHist2_lub$effort             # effort (alternatively, use: DetHist2_lub[[2]])


# multi-season detection history

# load multi-season data
data(camtrapsMultiSeason)
data(recordTableSampleMultiSeason)

# multi-season camera operation matrix
camop_season <- cameraOperation(CTtable          = camtrapsMultiSeason,
                                stationCol   = "Station",
                                setupCol     = "Setup_date",
                                sessionCol   = "session",
```

```
                                    retrievalCol = "Retrieval_date",
                                    hasProblems  = TRUE,
                                    dateFormat   = "dmy"
)

# multi-season detection history
DetHist_multi_season <- detectionHistory(recordTable     = recordTableSampleMultiSeason,
                            camOp                 = camop_season,
                            stationCol            = "Station",
                            speciesCol            = "Species",
                            species               = "VTA",
                            occasionLength        = 10,
                            day1                  = "station",
                            recordDateTimeCol     = "DateTimeOriginal",
                            includeEffort         = TRUE,
                            scaleEffort           = FALSE,
                            timeZone              = "UTC",
                            unmarkedMultFrameInput = TRUE
)

DetHist_multi_season


# Multi-species example for community occupancy analysis with communityModel()
DetHist_multi_species <- detectionHistory(recordTable = recordTableSample,
                                species = c("VTA", "PBE", "EGY"),   # multiple species
                                camOp = camop_no_problem,
                                stationCol = "Station",
                                speciesCol = "Species",
                                recordDateTimeCol = "DateTimeOriginal",
                                occasionLength = 7,
                                day1 = "station",
                                includeEffort = TRUE,
                                scaleEffort = FALSE,
                                timeZone = "Asia/Kuala_Lumpur"
)


# bundle input data for communityModel
data_list <- list(ylist = DetHist_multi_species$detection_history,
                  siteCovs = camtraps,
                  obsCovs = list(effort = DetHist_multi_species$effort))


## Not run:

# create community model
mod.jags <- communityModel(data_list,
                           ...) # model specification

## End(Not run)
```

`detectionMaps` *Generate maps of observed species richness and species presences by station*

Description

Generates maps of observed species richness and species presence by species and station. Output can be R graphics, PNG graphics or a shapefile for use in GIS software.

Usage

```
detectionMaps(
  CTtable,
  recordTable,
  Xcol,
  Ycol,
  backgroundPolygon,
  stationCol = "Station",
  speciesCol = "Species",
  speciesToShow,
  richnessPlot = TRUE,
  speciesPlots = TRUE,
  addLegend = TRUE,
  printLabels = FALSE,
  smallPoints,
  plotR = TRUE,
  writePNG = FALSE,
  plotDirectory,
  createPlotDir = FALSE,
  pngMaxPix = 1000,
  writeShapefile = FALSE,
  shapefileName,
  shapefileDirectory,
  shapefileProjection
)
```

Arguments

`CTtable`	data.frame. contains station IDs and coordinates
`recordTable`	data.frame. the record table created by `recordTable`
`Xcol`	character. name of the column specifying x coordinates in `CTtable`
`Ycol`	character. name of the column specifying y coordinates in `CTtable`
`backgroundPolygon`	SpatialPolygons or SpatialPolygonsDataFrame. Polygon to be plotted in the background of the map (e.g. project area boundary)
`stationCol`	character. name of the column specifying station ID in `CTtable` and `recordTable`

speciesCol	character. name of the column specifying species in recordTable
speciesToShow	character. Species to include in the maps. If missing, all species in recordTable will be included.
richnessPlot	logical. Generate a species richness plot?
speciesPlots	logical. Generate plots of all species number of independent events?
addLegend	logical. Add legends to the plots?
printLabels	logical. Add station labels to the plots?
smallPoints	numeric. Number by which to decrease point sizes in plots (optional).
plotR	logical. Create plots in R graphics device?
writePNG	logical. Create PNGs of the plots?
plotDirectory	character. Directory in which to save the PNGs
createPlotDir	logical. Create plotDirectory?
pngMaxPix	integer. number of pixels in pngs on the longer side
writeShapefile	logical. Create a shapefile from the output?
shapefileName	character. Name of the shapefile to be saved. If empty, a name will be generated automatically.
shapefileDirectory	character. Directory in which to save the shapefile.
shapefileProjection	character. A character string of projection arguments to use in the shapefile.

Details

The column name stationCol must be identical in CTtable and recordTable and station IDs must match.

Shapefile creation depends on the packages **sf**. Argument shapefileProjection must be a valid argument of st_crs (one of (i) character: a string accepted by GDAL, (ii) integer, a valid EPSG value (numeric), or (iii) an object of class crs. If shapefileProjection is undefined, the resulting shapefile will lack a coordinate reference system.

Value

An invisible data.frame with station coordinates, numbers of events by species at each station and total species number by station. In addition and optionally, R graphics or png image files.

Author(s)

Juergen Niedballa

References

A great resource for coordinate system information is https://spatialreference.org/. Use the Proj4 string as shapefileProjection argument.

Examples

```
# load station information
data(camtraps)

# load record table
data(recordTableSample)


# create maps
Mapstest <- detectionMaps(CTtable           = camtraps,
                          recordTable       = recordTableSample,
                          Xcol              = "utm_x",
                          Ycol              = "utm_y",
                          stationCol        = "Station",
                          speciesCol        = "Species",
                          writePNG          = FALSE,
                          plotR             = TRUE,
                          printLabels       = TRUE,
                          richnessPlot      = TRUE,
                          addLegend         = TRUE
)



# with a polygon in the background, and for one species only

# make a dummy polygon for the background
library(sf)

Sr1 = st_polygon(list(cbind(c(521500,526500,527000, 521500, 521500),
                            c(607500, 608000, 603500, 603500, 607500))))
aoi <- data.frame(name = "My AOI")
st_geometry(aoi) <- st_geometry(Sr1)
st_crs(aoi) <- 32650  # assign CRS: UTM50N

Mapstest2 <- detectionMaps(CTtable           = camtraps,
                           recordTable       = recordTableSample,
                           Xcol              = "utm_x",
                           Ycol              = "utm_y",
                           backgroundPolygon = aoi,               # this was added
                           speciesToShow     = c("PBE", "VTA"),   # this was added
                           stationCol        = "Station",
                           speciesCol        = "Species",
                           writePNG          = FALSE,
                           plotR             = TRUE,
                           printLabels       = TRUE,
                           richnessPlot      = TRUE,
                           addLegend         = TRUE
)
```

exifTagNames *Show Exif metadata of JPEG images or other image or video formats*

Description

The function will return metadata values, metadata tag names and group names of Exif metadata of JPEG images or other formats.

Usage

```
exifTagNames(
  inDir,
  whichSubDir = 1,
  fileName,
  returnMetadata = "DEPRECATED",
  returnTagGroup = "DEPRECATED"
)
```

Arguments

inDir	character. Directory containing camera trap images sorted into station subdirectories (e.g. inDir/StationA/)
whichSubDir	integer or character. Either number or name of subdirectory of inDir in which to look for an image
fileName	character. A filename, either the file name of an image in inDir or a full path with file name (in which case inDir is not needed)
returnMetadata	deprecated and ignored
returnTagGroup	deprecated and ignored

Details

Many digital cameras record information such as ambient temperature or moon phase under maker-specific tag names in Exif metadata of JPEG images. In addition, many technical information are stored in Exif metadata. In order to extract those information from images and add them to the record tables created by the functions recordTable and recordTableIndividual, the tag names must be known so they can be passed to these functions via the additionalMetadataTags argument.

By default the function returns both metadata tag names and the metadata group they belong to (via argument returnTagGroup). This is helpful to unambiguously address specific metadata tags, because different groups can contain tags of identical names, which may cause problems executing the functions recordTable and recordTableIndividual. The format is "GROUP:tag", e.g. "EXIF:Flash".

Value

A data frame containing three columns: metadata tag group, tag name, and values.

Author(s)

Juergen Niedballa

References

Phil Harvey's ExifTool https://exiftool.org/

See Also

recordTable

Examples

```
## Not run:

wd_images_ID <- system.file("pictures/sample_images_species_dir", package = "camtrapR")

# specify directory, camtrapR will automatically take first image from first subdirectory
exifTagNames(inDir          = wd_images_ID)

# specify subdirectory by name, camtrapR will use first image
exifTagNames(inDir          = wd_images_ID,
             whichSubDir    = "StationA")

# specifying fileName only (line break due to R package policy)
exifTagNames(fileName       = file.path(wd_images_ID, "StationC", "TRA",
                                        "StationC__2009-05-02__00-10-00(1).JPG"))

# specify inDir and fileName
exifTagNames(inDir          = wd_images_ID,
         fileName     = file.path("StationC", "TRA", "StationC__2009-05-02__00-10-00(1).JPG"))

# it also works this way
exifTagNames(inDir          = file.path(wd_images_ID, "StationC", "TRA"),
             fileName       = "StationC__2009-05-02__00-10-00(1).JPG")


# with tagged sample images
wd_images_ID_tagged <- system.file("pictures/sample_images_indiv_tag", package = "camtrapR")
exifTagNames(inDir          = wd_images_ID_tagged)


## End(Not run)
```

`filterRecordTable` *Filter species record table for temporal independence*

Description

Filter species record table for temporal independence

Usage

```
filterRecordTable(
  recordTable,
  minDeltaTime = 0,
  deltaTimeComparedTo,
  speciesCol = "Species",
  stationCol,
  cameraCol,
  camerasIndependent,
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  removeDuplicateRecords = TRUE,
  exclude,
  timeZone,
  writecsv = FALSE,
  outDir,
  eventSummaryColumn,
  eventSummaryFunction,
  quiet = FALSE
)
```

Arguments

`recordTable`	data frame as created by `recordTable`.
`minDeltaTime`	integer. Time difference between records of the same species at the same station to be considered independent (in minutes)
`deltaTimeComparedTo`	character. For two records to be considered independent, must the second one be at least `minDeltaTime` minutes after the last independent record of the same species (`"lastIndependentRecord"`), or `minDeltaTime` minutes after the last record (`"lastRecord"`)?
`speciesCol`	character. name of the column specifying species in `recordTable`
`stationCol`	character. Name of the camera trap station column. Assuming "Station" if undefined.
`cameraCol`	character. Name of the column specifying cameras in `recordTable` (optional).
`camerasIndependent`	logical. If `TRUE`, species records are considered to be independent between cameras at a station.

recordDateTimeCol
: character. Name of the column specifying date and time in recordTable.

recordDateTimeFormat
: character. Format of column recordDateTimeCol in recordTable

removeDuplicateRecords
: logical. If there are several records of the same species at the same station (also same camera if cameraID is defined) at exactly the same time, show only one?

exclude
: character. Vector of species names to be excluded from the record table

timeZone
: character. Must be a value returned by OlsonNames

writecsv
: logical. Should the record table be saved as a .csv?

outDir
: character. Directory to save csv to. If NULL and writecsv = TRUE, recordTable will be written to inDir.

eventSummaryColumn
: character. A column in the record table (e.g. from a metadata tag) by to summarise non-independent records (those within minDeltaTime of a given record) with a user-defined function (eventSummaryFunction)

eventSummaryFunction
: character. The function by which to summarise eventSummaryColumn of non-independent records, e.g. "sum", "max" (optional)

quiet
: logical. If TRUE, suppress printing of progress.

Value

A data frame containing species records and additional information about stations, date, time, filtered for temporal independence.

Author(s)

Juergen Niedballa

Examples

```
if (Sys.which("exiftool") != ""){        # only run example if ExifTool is available

# set directory with camera trap images in station directories
wd_images_ID_species <- system.file("pictures/sample_images_species_dir",
                                    package = "camtrapR")

# create record table without temporal filtering
rec_table <- recordTable(inDir                 = wd_images_ID_species,
                         IDfrom                = "directory",
                         minDeltaTime          = 0,
                         exclude               = "UNID",
                         timeZone              = "Asia/Kuala_Lumpur",
                         removeDuplicateRecords = TRUE
                         )

# filter for 60 minutes temporal independence
rec_table_filt <- filterRecordTable(recordTable         = rec_table,
```

```
                                    minDeltaTime        = 60,
                                    stationCol          = "Station",
                                    deltaTimeComparedTo = "lastIndependentRecord")
 nrow(rec_table)
 nrow(rec_table_filt)

 }
```

`fit,commOccu-method` *Fit a community (multi-species) occupancy model*

Description

Convenience function for fitting community occupancy models (defined in a commOccu object) in JAGS or Nimble.

Usage

```
## S4 method for signature 'commOccu'
fit(
  object,
  n.iter = 100,
  thin = 1,
  n.burnin = n.iter/2,
  n.adapt = 0,
  chains = 3,
  inits = NULL,
  compile = TRUE,
  WAIC = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

`object`	commOccu object
`n.iter`	number of iterations to monitor
`thin`	thinning interval for monitors
`n.burnin`	burnin length. Defaults to half of n.iter.
`n.adapt`	Length of adaptive phase
`chains`	number of MCMC chains to run
`inits`	named list. Initial values to use. If NULL (default), the values from the inits function in `object` are used.
`compile`	logical. If Nimble model, compile model with `compileNimble` before running model?

WAIC	logical. Return WAIC (only Nimble models)
quiet	if TRUE messages and progress bar will be suppressed
...	additional arguments to pass to runMCMC (only relevant for Nimble)

Details

Models will be fit either in JAGS or Nimble, depending on the decision made in the nimble argument in communityModel.

For Nimble, compilation is strongly recommended for long model runs. Uncompiled models can run extremely slow. Compilation itself can take a while also, and requires that Rtools is available on the system.

This is a convenience function only which hides some of the configuration options. If you require more control over model fitting, you can run all steps individually. See vignette 5 for details.

Value

A coda::mcmc.list

---

fixDateTimeOriginal *Fix DateTimeOriginal Exif metadata tag in Reconyx Hyperfire cameras*

Description

Some camera models don't store the date/time information in the standard Exif metadata tag. Consequently, camtrapR cannot find that information. This function uses Exiftool to update the DateTimeOriginal metadata tag in all images within a directory to make them readable with camtrapR (and other software).

Usage

```
fixDateTimeOriginal(inDir, recursive = TRUE)
```

Arguments

inDir	character. Name of the directory containing images to be fixed
recursive	logical. Recursively find images in subdirectories of inDir?

Details

Some Reconyx Hyperfire cameras (e.g. HC500) are known to show this problem.

Value

Returns invisibly the messages returned by the Exiftool call (warnings etc.).

Warning

Please make a backup of your images before running this function.

Author(s)

Juergen Niedballa

References

This function uses the code from:
Tobler, Mathias (2015). Camera Base Version 1.7 User Guide https://www.atrium-biodiversity.org/tools/camerabase/files/CameraBaseDoc1.7.pdf

Examples

```
## Not run:
# a hypothetical example

wd_images_hyperfire <- "C:/Some/Directory"

fixDateTimeOriginal(inDir     = wd_images_hyperfire,
                    recursive = TRUE)

## End(Not run)
```

---

getSpeciesImages *Collect all images of a species*

Description

This function will fetch all images of a particular species from all camera trap stations and copies these images to a new location. The images which are to be copied are found in one of 2 possible ways, 1) by providing an existing record table (created with recordTable) or 2) by reading species IDs from species directories or from metadata (calling ExifTool). Earlier in the workflow, i.e., before running this function, images should have been renamed (with imageRename) to give images unique file names based on station ID and date/time.

Usage

```
getSpeciesImages(
  species,
  recordTable,
  speciesCol = "Species",
  stationCol = "Station",
  inDir,
  outDir,
  createStationSubfolders = FALSE,
```

```
  IDfrom,
  metadataSpeciesTag,
  metadataHierarchyDelimitor = "|"
)
```

Arguments

species	character. Species whose images are to be fetched
recordTable	data frame. A data frame as returned by function recordTable. If you specify this argument, do not specify inDir
speciesCol	character. Name of the column specifying species ID in recordTable. Only required if recordTable is defined
stationCol	character. Name of the column specifying station ID in recordTable. Only required if recordTable is defined
inDir	character. Directory containing identified (species level) camera trap images sorted into station subdirectories (e.g. inDir/StationA/). If you specify this argument, do not specify recordTable.
outDir	character. Directory in which to save species images. A species subdirectory will be created in outDir automatically.
createStationSubfolders	logical. Save images in station directories within the newly created species directory in outDir?
IDfrom	character. Read species ID from image metadata ("metadata") of from species directory names ("directory")? Only required if inDir is defined.
metadataSpeciesTag	character. The species ID tag name in image metadata (if IDfrom = "metadata"). Only required if inDir is defined.
metadataHierarchyDelimitor	character. The character delimiting hierarchy levels in image metadata tags in field "HierarchicalSubject". Either "|" or ":" (if IDfrom = "metadata"). Only required if inDir is defined and IDfrom = "metadata".

Details

The function finds the images to be copied by either consulting a record table created with recordTable or by reading species IDs from images. The former is considerable faster because ExifTool is not called, but requires images to be in precisely the location given by the columns Directory and FileName in recordTable. To use this feature, provide the function with a record table in argument recordTable.

If you'd rather read species IDs from images within the function (to make sure all file paths are correct), images need to be in the directory structure required by the package, e.g.

> inDir/Station/Species

or

> inDir/Station/Camera/Species

if using species directories for species IDs, and

> inDir/Station

or

> inDir/Station/Camera

if reading IDs from species metadata tags. In the latter case, only station directories are needed. In any case, the argument `species` must match species IDs (either the `speciesCol` in `recordTable`, species directory names or species metadata tags).

Before running the function, first rename the images using function `imageRename` to provide unique file names and prevent several images from having the same name (if generic names like "IMGP0001.jpg" are used). The function will not copy images if there are duplicate filenames to prevent overwriting images unintentionally.

Value

A `data.frame` with old and new directories and file names and the copy status (`copy_ok`; TRUE if copying was successful, FALSE if not).

Author(s)

Juergen Niedballa

Examples

```
## Not run:
# define image directory
wd_images_ID <- system.file("pictures/sample_images_species_dir", package = "camtrapR")
wd_images_ID_copy <- file.path(tempdir(), "sample_images_species_dir")

species_to_copy <- "VTA"    # = Viverra tangalunga, Malay Civet

specImagecopy <- getSpeciesImages(species                = species_to_copy,
                                  inDir                  = wd_images_ID,
                                  outDir                 = wd_images_ID_copy,
                                  createStationSubfolders = FALSE,
                                  IDfrom                 = "directory"
                                  )

## End(Not run)
```

`imageRename` *Copy and rename images based on camera trap station ID and creation date*

Description

The function renames and copies raw camera trap images into a new location where they can be identified. Images are renamed with camera trap station ID, camera ID (optional), creation date and a numeric identifier for images taken within one minute of each other at a given station. Station ID and camera ID are derived from the raw image directory structure. The creation date is extracted from image metadata using ExifTool.

Usage

```
imageRename(
  inDir,
  outDir,
  hasCameraFolders,
  keepCameraSubfolders,
  createEmptyDirectories = FALSE,
  copyImages = FALSE,
  writecsv = FALSE,
  video
)
```

Arguments

`inDir`	character. Directory containing camera trap images sorted into station subdirectories (e.g. inDir/StationA/)
`outDir`	character. Directory into which the renamed images will be copied
`hasCameraFolders`	logical. Do the station directories in `inDir` have camera subdirectories (e.g. "inDir/StationA/Camera1")?
`keepCameraSubfolders`	logical. Should camera directories be preserved as subdirectories of `outDir` (e.g. "outDir/StationA/CameraA1")?
`createEmptyDirectories`	logical. If station or camera directories are empty, should they be copied nevertheless (causing empty directories in `outDir`, but preserving the whole directory structure)?
`copyImages`	logical. Copy images to `outDir`?
`writecsv`	logical. Save a data frame with a summary as a .csv? The csv will be saved in `outDir`.
`video`	list. Contains information on how to handle video data (optional). See details.

Details

Setting up the correct raw image directory structure is necessary for running the function successfully. `inDir` is the main directory that contains camera trap station subdirectories (e.g. inDir/StationA). If one camera was deployed per station and no camera subdirectories are used within station directories, `hasCameraFolders` can be set to `FALSE`. If more than one camera was deployed at stations, there must be subdirectories for the individual camera traps within the station directories

(e.g. "inDir/StationA/CameraA1" and "inDir/StationA/CameraA2"). Even if only some stations had multiple cameras, all station will need camera subdirectories. The argument `hasCameraFolders` must be `TRUE`. Within the camera subdirectories, the directory structure is irrelevant.

Renaming of images follows the following pattern: If `hasCameraFolders` is TRUE, it is: "StationID__CameraID__Date__Time(Number).JPG", e.g. "StationA__CameraA1__2015-01-31__18-59-59(1).JPG". If `hasCameraFolders` is FALSE, it is: "StationID__Date__Time(Number).JPG", e.g. "StationA__2015-01-31__18-59-59(1).JPG".

The purpose of the number in parentheses is to prevent assigning identical file names to images taken at the same station (and camera) in the same second, as can happen if cameras take sequences of images. It is a consecutive number given to all images taken at the same station by the same camera within one minute. The double underscore "__" in the image file names is for splitting and extracting information from file names in other functions (e.g. for retrieving camera IDs in `recordTable` if camera subdirectories are not preserved (`keepCameraSubfolders = FALSE`)).

The function finds all JPEG images (optionally, also videos) and extracts the image timestamp from the image metadata using ExifTool (digiKam database for videos) and copies the images with new file names into `outDir`, where it will set up a directory structure based on the station IDs and, if required by `keepCameraSubfolders = TRUE`, camera IDs (e.g. outDir/StationA/ or outDir/StationA/CameraA1).

`copyImages` can be set to FALSE to simulate the renaming and check the file names of the renamed images without copying. If you are handling large number of images (>e.g., 100,000), the function may take some time to run.

Argument `video` is a named list 4 items (`file_formats`, `dateTimeTag`, (`db_directory`, `db_filename`). Video date/time is read from video metadata stored in the digiKam database. Hence, `inDir` must be in your digiKam database.

The items of argument `video` are:

`file_formats`	The video formats to extract (include "jpg" if you want .JPG image metadata)
`dateTimeTag`	the metadata tag to extract date/time from (use `exifTagNames` to find out which tag is suitable)
`db_directory`	The directory containing digiKam database
`db_filename`	The digiKam database file in `db_directory`

See the examples in `recordTable` for for how to specify the argument `video`.

Value

A `data.frame` with original directory and file names, new directory and file names and an indicator for whether images were copied successfully.

Author(s)

Juergen Niedballa

References

Phil Harvey's ExifTool https://exiftool.org/

Examples

```
  ## Not run:

### "trial" run. create a table with file names after renaming, but don't copy images.

# first, find sample image directory in package directory:
wd_images_raw <- system.file("pictures/raw_images", package = "camtrapR")

# because copyImages = FALSE, outDir does not need to be defined
renaming.table <- imageRename(inDir              = wd_images_raw,
                              hasCameraFolders = FALSE,
                              copyImages         = FALSE,
                              writecsv           = FALSE
  )



### a real example in which images are copied and renamed

  # define raw image location
wd_images_raw <- system.file("pictures/raw_images", package = "camtrapR")

  # define destination for renamed images
wd_images_raw_renamed <- file.path(tempdir(), "raw_images_renamed")


  # now we have to define outDir because copyImages = TRUE
renaming.table2 <- imageRename(inDir               = wd_images_raw,
                               outDir              = wd_images_raw_renamed,
                               hasCameraFolders    = FALSE,
                               copyImages          = TRUE,
                               writecsv            = FALSE
  )

  # show output files
  list.files(wd_images_raw_renamed, recursive = TRUE)

  # output table
  renaming.table2


## End(Not run)
```

OCRdataFields *Optical character recognition (OCR) from data fields in digital images*

Description

Extracts information from the data fields in camera trap images (not the metadata). Many camera traps include data fields in camera trap images, often including date and time of images, and sometimes other information. This function extracts the information from these fields using optical character recognition provided by the package **tesseract** after reading images using the package **magick**.

Usage

```
OCRdataFields(inDir, geometries, invert = FALSE)
```

Arguments

`inDir`	character. Directory containing camera trap images (or subdirectories containing images)
`geometries`	list. A (possibly named) list of geometry strings defining the image area(s) to extract.
`invert`	logical. Invert colors in the image? Set to TRUE if text in data field is white on black background. Leave if FALSE if text is black in white background.

Details

Normally all these information should be in the image metadata. This function is meant as a last resort if image metadata are unreadable or were removed from images. OCR is not perfect and may misidentify characters, so check the output carefully.

The output of this function can be used in `writeDateTimeOriginal` to write date/time into the DateTimeOriginal tag in image metadata, making these images available for automatic processing with `recordTable` and other functions that extract image metadata.

This function reads all images in inDir (including subdirectories), crops them to the geometries in the "geometries" list, and performs optical character recognition (OCR) on each of these fields (leveraging the magick and tesseract packages).

Geometries are defined with `geometry_area` from **magick**. See `geometry` for details on how to specify geometries with `geometry_area`. The format is: "widthxheight+x_off+y_off", where:

width width of the area of interest

height height of the area of interest

x_off offset from the left side of the image

y_off offset from the top of the image

Units are pixels for all fields. digiKam can help in identifying the correct specification for geometries. Open the Image Editor, left-click and draw a box around the data field of interest. Ensure the entire text field is included inside the box, but nothing else. Now note two pairs of numbers at the bottom of the window, showing the offsets and box size as e.g.:

"(400, 1800) (300 x 60)"

This corresponds to the geometry values as follows:

"(x_off, y_off) (width x height)"

Using these values, you'd run:

`geometry_area(x_off = 400, y_off = 1800, width = 300, height = 60)`

and receive

"300x60+400+1800"

as your geometry.

OCR in tesseract has problems with white font on black background. If that is the case in your images, set `invert` to `TRUE` to invert the image and ensure OCR uses black text on white background.

Even then, output will not be perfect. Error rates in OCR depend on multiple factors, including the text size and font type used. We don't have control over these, so check the output carefully and edit as required.

Value

A `data.frame` with original directory and file names, and additional columns for the OCR data of each extracted geometry.

Author(s)

Juergen Niedballa

See Also

writeDateTimeOriginal

Examples

```
## Not run:
# dontrun is to avoid forcing users to install additional dependencies

wd_images_OCR <- system.file("pictures/full_size_for_ocr", package = "camtrapR")

library(magick)

# define geometries
geometry1 <- geometry_area(x_off = 0, y_off = 0, width = 183, height = 37)
geometry2 <- geometry_area(x_off = 196, y_off = 0, width = 200, height = 17)
geometry3 <- geometry_area(x_off = 447, y_off = 0, width = 63, height = 17)
geometry4 <- geometry_area(x_off = 984, y_off = 0, width = 47, height = 17)
geometry5 <- geometry_area(x_off = 0, y_off = 793, width = 320, height = 17)

# combine geometries into list
geometries <- list(date = geometry1,
                   time = geometry2,
                   sequence_id = geometry3,
                   temperature = geometry4,
                   camera_model = geometry5)

df_image_data  <- OCRdataFields(inDir = wd_images_OCR,
```

```
                                  geometries = geometries,
                                  invert = TRUE)
df_image_data

# note the mistake in "camera_model"
# it should be "PC850", not "PC8S0O"
# date and time are correct though


## End(Not run)
```

`plot_coef,commOccu-method`

Plot effect sizes of covariates in community occupancy model

Description

Plot effect sizes for all species in a community (multi-species) occupancy model. Currently only supports continuous covariates, not categorical covariates.

Usage

```
## S4 method for signature 'commOccu'
plot_coef(
  object,
  mcmc.list,
  submodel = "state",
  speciesSubset,
  ordered = TRUE,
  combine = FALSE,
  outdir,
  level = c(outer = 0.95, inner = 0.75),
  colorby = "significance",
  scales = "free_y",
  community_lines = FALSE,
  ...
)
```

Arguments

`object`	commOccu object
`mcmc.list`	mcmc.list. Output of `fit` called on a `commOccu` object
`submodel`	character. Submodel to get plots for. Can be `"det"` or `"state"`
`speciesSubset`	character. Species to include in coefficient plots.

ordered	logical. Order species in plot by median effect (TRUE) or by species name (FALSE)
combine	logical. Combine multiple plots into one plot (via facets)?
outdir	character. Directory to save plots to (optional)
level	numeric. Probability mass to include in the uncertainty interval (two values named "outer" and "inner", in that order). Second value (= inner interval) will be plotted thicker.
colorby	character. Whether to color estimates by "significance" (of the effect estimates), or "Bayesian p-value" (of the species). Currently allows only "significance".
scales	character. Passed to facet_grid. Can be "free" to scale x axes of effect estimates independently, or "free_y" to scale all x axes identically.
community_lines	logical. Add faint vertical lines to the plot indicating median community effect (solid line) and its confidence interval (first value from level).
...	additional arguments for ggsave - only relevant if outdir is defined.

Details

Users who wish to create their own visualizations can use the data stored in the ggplot output. It is accessed via e.g. output$covariate_name$data

Value

A list of ggplot objects (one list item per covariate).

---

plot_effects,commOccu-method

Plot Marginal Effects of Covariates

Description

Plot marginal effect plots (= response curves if covariates are continuous) for all species in a community (multi-species) occupancy model. Takes into account species-specific intercepts (if any). Currently only supports continuous covariates, not categorical covariates.

Usage

```
## S4 method for signature 'commOccu'
plot_effects(
  object,
  mcmc.list,
  submodel = "state",
  response = "occupancy",
  speciesSubset,
  draws = 1000,
```

```
  outdir,
  level = 0.95,
  keyword_quadratic = "_squared",
  ...
)
```

Arguments

object	commOccu object
mcmc.list	mcmc.list. Output of fit called on a commOccu object
submodel	character. Submodel to get plots for. Can be "det" (detection submodel) or "state" (occupancy submodel)
response	character. response type on y axis. Only relevant for submodel = "state". Default is "occupancy", can be set to "abundance" for Royle-Nichols models
speciesSubset	character. Species to include in effect plots.
draws	integer. Number of draws from the posterior to use when generating the plots. If fewer posterior samples than specified in draws are available, all posterior samples are used.
outdir	character. Directory to save plots to (optional)
level	numeric. Probability mass to include in the uncertainty interval.
keyword_quadratic	character. A suffix in covariate names in the model that indicates a covariate is a quadratic effect of another covariate which does not carry the suffix in its name (e.g. if the covariate is "elevation", the quadratic covariate would be "elevation_squared").
...	additional arguments for ggsave - only relevant if outdir is defined

Details

Users who wish to create their own visualizations can use the data stored in the ggplot output. It is accessed via e.g. output$covariate_name$data

Value

A list of ggplot objects (one list item per covariate).

PPC.community *Calculate Community-Level Posterior Predictive Checks for Occupancy Models*

Description

A wrapper function that applies Posterior Predictive Checks (PPC) across multiple species in a community occupancy model (from communityModel. It calculates species-specific and community-level Bayesian p-values to assess model fit. The function accepts both predict output format and list format for model parameters.

Usage

```
PPC.community(
  p,
  psi,
  y,
  input_format = c("predict", "lists"),
  K = NULL,
  model = c("Occupancy", "RN"),
  zhat = NULL,
  z.cond = TRUE,
  type = c("FT", "PearChi2", "Deviance"),
  return.residuals = TRUE,
  show_progress = TRUE,
  ...
)
```

Arguments

p	Either a 4D array [stations, species, iterations, occasions] from predict or a list of 3D arrays [iterations, stations, occasions], one per species
psi	Either a 3D array [stations, species, iterations] from predict or a list of 2D arrays [iterations, stations], one per species
y	List of detection histories, one matrix/vector per species
input_format	Character indicating format of p and psi ("predict" or "lists")
K	Number of occasions as either a scalar or site vector. Calculated automatically if y is a list of matrices.
model	Character indicating model type ("Occupancy" or "RN")
zhat	List of z estimate matrices, one per species (optional). Each matrix should follow the format specified in PPC.residuals.
z.cond	Logical. If TRUE, new data is conditioned on estimated z (testing only detection model fit). If FALSE, generates new z for each posterior sample (testing complete model).
type	Character indicating residual type ("FT", "PearChi2", or "Deviance")
return.residuals	Logical. If TRUE, returns species-specific residuals along with Bayesian p-values. If FALSE, returns only the p-values.
show_progress	Logical; whether to show a progress bar during computation (if package pbapply is available)
...	Additional arguments passed to PPC.residuals.

Details

This function extends the single-species PPC analysis to the community level by:

- Applying residual calculations to each species in the community

- Aggregating results to assess community-level model fit
- Providing both species-specific and community-level diagnostics

This function provides flexibility in input formats to accommodate different workflows:

- Direct output from camtrapR's predict() function (4D/3D arrays)
- Lists of species-specific arrays (for custom workflows)

Value

If return.residuals=TRUE, returns a list containing:

- bp_values - Data frame with species-specific and community-level Bayesian p-values
- species_results - List containing the complete PPC.residuals output for each species

If return.residuals=FALSE, returns only the data frame of Bayesian p-values.

Author(s)

Rahel Sollmann

See Also

PPC.residuals for details on the underlying single-species calculations

Examples

```
## Not run:

# Create and fit model
data("camtraps")

# create camera operation matrix
camop_no_problem <- cameraOperation(CTtable      = camtraps,
                                    stationCol   = "Station",
                                    setupCol     = "Setup_date",
                                    retrievalCol = "Retrieval_date",
                                    hasProblems  = FALSE,
                                    dateFormat   = "dmy"
)

data("recordTableSample")

# make list of detection histories
species_to_include <- unique(recordTableSample$Species)

  DetHist_list <- detectionHistory(
    recordTable         = recordTableSample,
    camOp                = camop_no_problem,
    stationCol           = "Station",
    speciesCol           = "Species",
    recordDateTimeCol    = "DateTimeOriginal",
```

```
    species              = species_to_include,
    occasionLength       = 7,
    day1                 = "station",
    datesAsOccasionNames = FALSE,
    includeEffort        = TRUE,
    scaleEffort          = TRUE,
    timeZone             = "Asia/Kuala_Lumpur"
)


# create some fake covariates for demonstration
sitecovs <- camtraps[, c(1:3)]
sitecovs$elevation <- c(300, 500, 600)

# scale numeric covariates
sitecovs[, c(2:4)] <- scale(sitecovs[,-1])


# bundle input data for communityModel
data_list <- list(ylist = DetHist_list$detection_history,
                  siteCovs = sitecovs,
                  obsCovs = list(effort = DetHist_list$effort))


# create community model for JAGS
modelfile1 <- tempfile(fileext = ".txt")
mod.jags <- communityModel(data_list,
                           occuCovs = list(fixed = "utm_y", ranef = "elevation"),
                           detCovsObservation = list(fixed = "effort"),
                           intercepts = list(det = "ranef", occu = "ranef"),
                           modelFile = modelfile1)

summary(mod.jags)

# fit in JAGS
fit.jags <- fit(mod.jags,
                n.iter = 1000,
                n.burnin = 500,
                chains = 3)
summary(fit.jags)



# create predictions for p and psi
draws <- 100

p <- predict(object = mod.jags,
             mcmc.list = fit.jags,
             type = "p_array",
             draws = draws)
# output is in order [station, species, draw, occasion]

 psi <- predict(object = mod.jags,
```

```
                mcmc.list = fit.jags,
                type = "psi_array",
                draws = draws)
 # output is in order [station, species, draw]



ppc_comm <- PPC.community(
  p = p,
  psi = psi,
  y = mod.jags@input$ylist,
  model = "Occupancy",
  type = "FT")

# Bayesian p-values
 ppc_comm$BP


 str(ppc_comm$residuals)

 # get individual species PPC results
 ppc_species <- ppc_comm$residuals[[1]] # first species

 plot(apply(ppc_species$res.obs, 2, mean), apply(ppc_species$res.new, 2, mean),
 xlab = "Observed residuals",
 ylab = "Predicted residuals"
 )

 abline(0,1)  # diagonal line is not visible due to tiny data set


## End(Not run)
```

PPC.residuals *Calculate Residuals from MCMC Output of Occupancy Models*

Description

Posterior Predictive Check (PPC) function that calculates Freeman-Tukey (FT) residuals, Pearson"s Chi-squared residuals, or deviance from MCMC output of occupancy models. This function compares observed data to simulated data from the posterior distribution to assess model fit.

Usage

```
PPC.residuals(
  y,
  p,
  psi,
  model = c("Occupancy", "RN"),
  type = c("FT", "PearChi2", "Deviance"),
```

```
  K = NULL,
  z.cond = TRUE,
  zhat = NULL,
  nmax = 20,
  return.residuals = TRUE,
  return.z = TRUE
)
```

Arguments

y	Observations as either a site vector or site by occasion matrix. For matrix format, use NA for unsampled occasions.
p	Array of posterior samples for detection probability (p). Dimensions should be iterations by sites (by occasion optionally). For RN models, p should represent individual-level detection (not conditional on local abundance).
psi	Array of posterior samples for occupancy probability (psi). Dimensions should be iterations by sites. For RN models, psi should represent expected abundance
model	Character indicating model type: either "Occupancy" or "RN" (Royle-Nichols).
type	Type of residual to calculate: "FT" (Freeman Tukey), "PearChi2" (Pearson Chi-squared), or "Deviance" (not technically a residual).
K	Number of occasions as either a scalar or site vector. Calculated automatically if y is a matrix.
z.cond	Logical. If TRUE, new data is conditioned on estimated z (testing only detection model fit). If FALSE, generates new z for each posterior sample (testing complete model).
zhat	Optional matrix with same dimensions as psi containing estimates of z from the same model. If not provided, will be generated internally.
nmax	Maximum site-level abundance (default = 20). Only used if model="RN". Higher values increase computation time. Warning given if set too low.
return.residuals	Logical. If TRUE (default), returns residuals along with Bayesian p-value.
return.z	Logical. If TRUE, returns z values conditional on y, and unconditional z's if z.cond = FALSE. Note: if zhat is provided, the returned conditional-on-y z values will be identical to those provided.

Details

This function helps assess model fit for occupancy models using various types of residuals:

Types of Residuals:

- Freeman-Tukey (FT):

$$R_j = (\sqrt{y_j} - \sqrt{E(y_j)})^2$$

  Measures the squared difference between the square root of observed detections and the square root of expected detections at each site.

- Pearson Chi-squared:

$$R_j = \left(\frac{y_j - E(y_j)}{\sqrt{Var(y_j)}} \right)^2$$

  Measures the squared difference between observed and expected detections, standardized by the theoretical variance calculated from the model parameters.

- Deviance:

$$R_j = -2 \log[y_j | \theta_j, K_j]$$

  Measures the contribution of each site to the overall model likelihood, quantifying the discrepancy between observed data and model predictions based on likelihood ratios

Where:

- y_j is the number of detections of the species at site j, out of K_j repeated surveys
- $E(y_j) = K_j p_j z_j$, with p_j = species detection probability and z_j = occupancy state (1 if occupied, 0 otherwise)
- $Var(y_j) = p_j z_j (1 - p_j z_j) K_j$
- For Royle-Nichols occupancy models, the term $p_j z_j$ is replaced with $1 - (1 - r_j)^{N_j}$, where r_j = individual detection probability and N_j = local abundance
- For Deviance, θ_j is either occupancy and species detection probability at site j (ψ_j, p_j) for regular occupancy models, or expected abundance and individual detection probability (λ_j, r_j) for Royle-Nichols occupancy models

Bayesian P-values: The function calculates Bayesian p-values as a measure of model fit. These values:

- Range from 0 to 1
- Values close to 0.5 suggest good model fit
- Values close to 0 or 1 suggest poor fit
- Are calculated by comparing observed residuals to residuals from simulated data

Conditional vs Unconditional Assessment: The z.cond parameter allows for two types of model assessment:

- z.cond = TRUE: Tests only the detection component of the model, fixing occupancy/abundance to estimates from the model, rather than generating them anew
- z.cond = FALSE: Tests the complete model, including both occupancy and detection components

Value

If return.residuals=TRUE (default), returns a list containing:

- res.obs - residuals for observed data
- res.new - residuals for newly generated data
- BP - Bayesian p-value

If return.residuals=FALSE, returns only the Bayesian p-value.

Warning

This is a beta version of the function. While it has been tested extensively, not all possible data configurations may have been captured in testing. This is particularly true for:

- Deviance calculations (type = "Deviance")
- Royle-Nichols models (model = "RN")

If you encounter issues with the function, please contact the package developers.

Note

FT and Chi-squared residuals have been extensively tested. Deviance calculations have undergone less testing and are only available for scenarios with constant detection probability across occasions. FT and Chi-squared residuals can handle varying detection probabilities.

Author(s)

Rahel Sollmann

References

Sollmann, Rahel. Occupancy models and the "good fit, bad prediction" dilemma. Ecology (submitted)

---

predict,commOccu-method

Predictions from community occupancy models

Description

Create (spatial) predictions of species occupancy and species richness from community occupancy models and spatial rasters or covariate data frames.

Usage

```
## S4 method for signature 'commOccu'
predict(
  object,
  mcmc.list,
  type,
  draws = 1000,
  level = 0.95,
  interval = c("none", "confidence"),
  x = NULL,
  aoi = NULL,
  speciesSubset,
  batch = FALSE,
  seed = NULL
)
```

Arguments

object	commOccu object
mcmc.list	mcmc.list. Output of fit called on a commOccu object
type	character. "psi" for species occupancy estimates, "richness" for species richness estimates, "pao" for percentage of area occupied (by species), "psi_array" for raw occupancy probabilities in an array. For Royle-Nichols models, "abundance" for species abundance, or "lambda_array" for raw species abundance estimates in an array. "p_array" for raw detection probabilities in an array.
draws	Number of draws from the posterior to use when generating the plots. If fewer than draws are available, they are all used
level	Probability mass to include in the uncertainty interval
interval	Type of interval calculation. Can be "none" or "confidence" (can be abbreviated). Calculation can be slow for type = "psi" with many cells and posterior samples.
x	SpatRaster, data.frame or NULL. Must be scaled with same parameters as site covariates used in model, and have same names. If NULL, use site covariate data frame from model input (commOccu object in parameter object)
aoi	SpatRaster with same dimensions as x (if x is a SpatRaster), indicating the area of interest (all cells with values are AOI, all NA cells are ignored). If NULL, predictions are made for all cells.
speciesSubset	species to include in richness estimates. Can be index number or species names.
batch	logical or numeric. If FALSE, all raster cells / data frame rows will be processed at once (can be memory intensive). If TRUE, computation is conducted in batches of 1000. If numeric, it is the desired batch size.
seed	numeric. Seed to use in set.seed for reproducible results (ensures that draws are identical).

Details

Processing can be very memory-intensive. If memory is insufficient, use the batch parameter. This can enable processing for higher numbers of draws or very large rasters / data frames.

Value

A SpatRaster or data.frame, depending on x (for type = "psi", "abundance", "richness". If type = "pao", a list. If type = "psi_array" or "lambda_array", a 3D-array [site, species, draw]. If type = "p_array", a 4D-array [site, species, draw, occasion].

`readcamtrapDP` *Convert Camtrap DP format data to camtrapR format*

Description

This function converts camera trap data from the Camtrap DP standard format to the format used by camtrapR. Camtrap DP is an open standard for the FAIR exchange and archiving of camera trap data, structured in a simple yet flexible data model consisting of three tables: Deployments, Media, and Observations.

Usage

```
readcamtrapDP(
  deployments_file = "deployments.csv",
  media_file = "media.csv",
  observations_file = "observations.csv",
  datapackage_file = "datapackage.json",
  min_gap_hours = 24,
  removeNA = FALSE,
  removeEmpty = FALSE,
  remove_bbox = TRUE,
  add_file_path = FALSE,
  filter_observations = NULL
)
```

Arguments

`deployments_file`	character. Path to deployments.csv file
`media_file`	character. Path to media.csv file
`observations_file`	character. Path to observations.csv file
`datapackage_file`	character. Path to datapackage.json file
`min_gap_hours`	numeric. Minimum gap in hours to consider as a camera interruption (default: 24)
`removeNA`	logical. Whether to remove columns with only NA values
`removeEmpty`	logical. Whether to remove columns with only empty values
`remove_bbox`	logical. Whether to remove bounding box columns in the observation table
`add_file_path`	logical. Whether to add file path from media table to the observation table
`filter_observations`	Controls which observation types to include. `NULL` or `FALSE` keeps all observations (default), `TRUE` keeps only animal observations, or provide a character vector of specific observation types to include.

Details

Camtrap DP is a standardized format developed under the umbrella of the Biodiversity Information Standards (TDWG) that facilitates data exchange between different camera trap platforms and systems. It supports a wide range of camera deployment designs, classification techniques, and analytical use cases.

While the 'camtrapdp' package (available on CRAN) provides general functionality for reading, manipulating, and transforming Camtrap DP data, this function specifically converts Camtrap DP data directly into the format required by camtrapR, producing the CTtable and recordTable objects that camtrapR functions expect.

This function reads the three primary tables from a Camtrap DP dataset:

- Deployments: Information about camera trap placements
- Observations: Classifications derived from the media files
- Media: Information about the media files (images/videos) recorded

The Media table is only read if the `add_file_path` parameter is set to `TRUE`.

The function converts these into two primary camtrapR data structures:

- CTtable: Contains deployment information including station ID, setup/retrieval dates, camera operation problems, camera setup, and more
- recordTable: Contains observation records with taxonomic and temporal information

Additional features include:

- Parsing of deploymentTags and deploymentGroups (in deployments) and observationTags (in observations)
- Extraction of taxonomic information from metadata
- Handling of deployment intervals and gaps

Value

List containing three elements:

- CTtable: Data frame with camera trap deployment information in camtrapR format
- recordTable: Data frame with species records in camtrapR format
- metadata: List containing project metadata extracted from datapackage.json

Note

Camtrap DP was developed as a consensus of a long, in-depth consultation process with camera trap data management platforms and major players in the field of camera trapping. It builds upon the earlier Camera Trap Metadata Standard (CTMS) but addresses its limitations.

The Camtrap DP standard structures data in a way that supports both media-based observations (using a single media file as source) and event-based observations (considering an event with a specified duration as source). For the purpose of this function, both are treated to be equivalent. Event-based observations are converted to a single timestamp in the recordTable (using the event start time).

Consider using the 'camtrapdp' package (Desmet et al., 2024) for more general operations on Camtrap DP data before converting to camtrapR format with this function.

References

Bubnicki, J.W., Norton, B., Baskauf, S.J., et al. (2023). Camtrap DP: an open standard for the FAIR exchange and archiving of camera trap data. Remote Sensing in Ecology and Conservation, 10(3), 283-295.

Desmet, P., Govaert, S., Huybrechts, P., Oldoni, D. (2024). camtrapdp: Read and Manipulate Camera Trap Data Packages. R package version 0.3.1. https://CRAN.R-project.org/package=camtrapdp

See Also

read_camtrapdp in the 'camtrapdp' package for reading Camtrap DP data check_camtrapdp in the 'camtrapdp' package for validating Camtrap DP data

Examples

```
## Not run:
# Read a Camtrap DP dataset
camtrap_data <- readcamtrapDP(
  deployments_file = "path/to/deployments.csv",
  media_file = "path/to/media.csv",
  observations_file = "path/to/observations.csv",
  datapackage_file = "path/to/datapackage.json"
)

# alternatively, set the working directory only
setwd("path/to/camtrapdp_data")
camtrap_data <- readcamtrapDP()   # uses default file names


# Extract components
ct_table <- camtrap_data$CTtable
record_table <- camtrap_data$recordTable
metadata <- camtrap_data$metadata


## End(Not run)
```

---

readWildlifeInsights *Import Wildlife Insights data to camtrapR*

Description

This function imports camera trap data from Wildlife Insights into a format compatible with camtrapR. It can read data from a directory containing CSV files, from a ZIP file, or from individual CSV files.

Usage

```
readWildlifeInsights(
  directory = NULL,
  zipfile = NULL,
  deployment_file = NULL,
  image_file = NULL
)
```

Arguments

directory	character. Path to folder containing CSV files exported from Wildlife Insights.
zipfile	character. Path to a ZIP file exported from Wildlife Insights.
deployment_file	character. Path to the deployments CSV file.
image_file	character. Path to the images CSV file.

Value

A list containing three elements:

CTtable The full camera trap table, based on deployments.csv

CTtable_aggregated An aggregated version of the camera trap table, with one row per station

recordTable The record table, based on images.csv with additional columns from deployments.csv

Examples

```
## Not run:
# Reading from a directory
wi_data <- readWildlifeInsights(directory = "path/to/csv/files")

# Reading from a ZIP file
wi_data <- readWildlifeInsights(zipfile = "path/to/wildlife_insights_export.zip")

# Reading from separate CSV files
wi_data <- readWildlifeInsights(deployment_file = "path/to/deployments.csv",
                                image_file = "path/to/images.csv")

## End(Not run)
```

---

recordTable *Generate a species record table from camera trap images and videos*

Description

Generates a record table from camera trap images or videos. Images/videos must be sorted into station directories at least. The function can read species identification from a directory structure (Station/Species or Station/Camera/Species) or from image metadata tags.

Usage

```
recordTable(
  inDir,
  IDfrom,
  cameraID,
  camerasIndependent,
  exclude,
  minDeltaTime = 0,
  deltaTimeComparedTo,
  timeZone,
  stationCol,
  writecsv = FALSE,
  outDir,
  metadataHierarchyDelimitor = "|",
  metadataSpeciesTag,
  additionalMetadataTags,
  removeDuplicateRecords = TRUE,
  returnFileNamesMissingTags = FALSE,
  eventSummaryColumn,
  eventSummaryFunction,
  video
)
```

Arguments

inDir	character. Directory containing station directories. It must either contain images in species subdirectories (e.g. inDir/StationA/SpeciesA) or images with species metadata tags (without species directories, e.g. inDir/StationA).
IDfrom	character. Read species ID from image metadata ("metadata") of from species directory names ("directory")?
cameraID	character. Where should the function look for camera IDs: 'filename', 'directory'. 'filename' requires images renamed with imageRename. 'directory' requires a camera subdirectory within station directories (station/camera/species). Can be missing.
camerasIndependent	logical. If TRUE, species records are considered to be independent between cameras at a station.
exclude	character. Vector of species names to be excluded from the record table
minDeltaTime	integer. Time difference between records of the same species at the same station to be considered independent (in minutes)
deltaTimeComparedTo	character. For two records to be considered independent, must the second one be at least minDeltaTime minutes after the last independent record of the same species ("lastIndependentRecord"), or minDeltaTime minutes after the last record ("lastRecord")?
timeZone	character. Must be a value returned by OlsonNames

stationCol — character. Name of the camera trap station column. Assuming "Station" if undefined.

writecsv — logical. Should the record table be saved as a .csv?

outDir — character. Directory to save csv to. If NULL and writecsv = TRUE, recordTable will be written to inDir.

metadataHierarchyDelimitor — character. The character delimiting hierarchy levels in image metadata tags in field "HierarchicalSubject". Either "|" or ":".

metadataSpeciesTag — character. In custom image metadata, the species ID tag name.

additionalMetadataTags — character. Additional camera model-specific metadata tags to be extracted. (If possible specify tag groups as returned by exifTagNames)

removeDuplicateRecords — logical. If there are several records of the same species at the same station (also same camera if cameraID is defined) at exactly the same time, show only one?

returnFileNamesMissingTags — logical. If species are assigned with metadata and images are not tagged, return a few file names of these images as a message?

eventSummaryColumn — character. A column in the record table (e.g. from a metadata tag) by to summarise non-independent records (those within minDeltaTime of a given record) with a user-defined function (eventSummaryFunction)

eventSummaryFunction — character. The function by which to summarise eventSummaryColumn of non-independent records, e.g. "sum", "max" (optional)

video — list. Contains information on how to handle video data (optional). See details.

Details

The function can handle a number of different ways of storing images, and supports species identification by moving images into species directories as well as metadata tagging. In every case, images need to be stored into station directories. If images are identified by moving them into species directories, a camera directory is optional: "Station/Species/XY.JPG" or "Station/Camera/Species/XY.JPG". Likewise, if images are identified using metadata tagging, a camera directory can be used optionally: "Station/XY.JPG" or "Station/Camera/XY.JPG".

If images are identified by metadata tagging, metadataSpeciesTag specifies the metadata tag group name that contains species identification tags. metadataHierarchyDelimitor is "|" for images tagged in DigiKam and images tagged in Adobe Bridge / Lightroom with the default settings. It is only necessary to change it if the default was changed in these programs.

minDeltaTime is a criterion for temporal independence of species recorded at the same station. Setting it to 0 will make the function return all records. Setting it to a higher value will remove records that were taken less than minDeltaTime minutes after the last record (deltaTimeComparedTo = "lastRecord") or the last independent record (deltaTimeComparedTo = "lastIndependentRecord").

removeDuplicateRecords determines whether duplicate records (identical station, species, date/time, (and camera if applicable)) are all returned (FALSE) or collapsed into a single unique record (TRUE).

camerasIndependent defines if the cameras at a station are to be considered independent. If TRUE, records of the same species taken by different cameras are considered independent (e.g. if they face different trails). Use FALSE if both cameras face each other and possibly TRUE).

exclude can be used to exclude "species" directories containing irrelevant images (e.g. "team", "blank", "unidentified"). stationCol can be set to match the station column name in the camera trap station table (see camtraps).

Many digital images contain Exif metadata tags such as "AmbientTemperature" or "MoonPhase" that can be extracted if specified in metadataTags. Because these are manufacturer-specific and not standardized, function exifTagNames provides a vector of all available tag names. Multiple names can be specified as a character vector as: c(Tag1,Tag2, ...). The metadata tags thus extracted may be used as covariates in modelling species distributions.

eventSummaryColumn and eventSummaryFunction can be used to extract summary statistics for independent sampling events. For example, you assigned a "count" tag to your images, indicating the number of individuals in a picture. In a sequence of pictures taken within 1 minute, most pictures show one individual, but one image shows two individuals. You tagged the images accordingly (count = 1 or count = 2) and run recordTable. Set eventSummaryColumn = "count" and eventSummaryFunction = "max" to obtain the maximum number of count in all images within minDeltaTime minutes of a given record. The results is in a new column, in this example count_max. You can also calculate several statistics at the same time, by supplying vectors of values, e.g. eventSummaryColumn = c("count", "count", "camera") and eventSummaryFunction = c("min", "max", "unique") to get minimum and maximum count and all unique camera IDs for that event. Note that eventSummaryColumn and eventSummaryFunction must be of same length.

Argument video is a named list with 2 or 4 items. 2 items (file_formats, dateTimeTag) are always required, and are sufficent if IDfrom = "directory". In that case, no digiKam tags will be returned. To return digiKam tags, two additional items are required (db_directory, db_filename). This is essential when using IDfrom = "metadata". When using IDfrom = "directory", it is optional, but allows to extract metadata tags assigned to videos in digiKam. This workaround is necessary because digiKam tags are not written into video metadata, but are only saved in the digiKam database. So in contrast to JPG images, they can not be extracted with ExifTool. It also requires that inDir is in your digiKam database.

The items of argument video are:

file_formats	The video formats to extract (include "jpg" if you want .JPG image metadata)
dateTimeTag	the metadata tag to extract date/time from (use exifTagNames to find out which tag is suitable)
db_directory	The directory containing digiKam database (optional if IDfrom = "directory")
db_filename	The digiKam database file in db_directory (optional if IDfrom = "directory")

See the examples below for for how to specify the argument video.

Value

A data frame containing species records and additional information about stations, date, time and (optionally) further metadata.

Warning

Custom image metadata must be organised hierarchically (tag group - tag; e.g. "Species" - "Leopard Cat"). Detailed information on how to set up and use metadata tags can be found in vignette 2: Species and Individual Identification.

Custom image metadata tags must be written to the images. The function cannot read tags from .xmp sidecar files. Make sure you set the preferences accordingly. In DigiKam, go to Settings/Configure digiKam/Metadata. There, make sure "Write to sidecar files" is unchecked.

Please note the section about defining argument `timeZone` in the vignette on data extraction (accessible via `vignette("DataExtraction")` or online (https://cran.r-project.org/package=camtrapR/vignettes/camtrapr3.pdf)).

Note

The results of a number of other function will depend on the output of this function (namely on the arguments `exclude` for excluding species and `minDeltaTime`/ `deltaTimeComparedTo` for temporal independence):

`detectionMaps`
`detectionHistory`
`activityHistogram`
`activityDensity`
`activityRadial`
`activityOverlap`
`activityHistogram`
`surveyReport`

Author(s)

Juergen Niedballa

References

Phil Harvey's ExifTool https://exiftool.org/

Examples

```
## Not run:    # the examples take too long to pass CRAN tests


# set directory with camera trap images in station directories
wd_images_ID_species <- system.file("pictures/sample_images_species_dir",
                                    package = "camtrapR")

if (Sys.which("exiftool") != ""){       # only run these examples if ExifTool is available
```

```
rec_table1 <- recordTable(inDir               = wd_images_ID_species,
                          IDfrom              = "directory",
                          minDeltaTime        = 60,
                          deltaTimeComparedTo = "lastRecord",
                          writecsv            = FALSE,
                          additionalMetadataTags = c("EXIF:Model", "EXIF:Make")
)
# note argument additionalMetadataTags: it contains tag names as returned by function exifTagNames

rec_table2 <- recordTable(inDir               = wd_images_ID_species,
                          IDfrom              = "directory",
                          minDeltaTime        = 60,
                          deltaTimeComparedTo = "lastRecord",
                          exclude             = "UNID",
                          writecsv            = FALSE,
                          timeZone            = "Asia/Kuala_Lumpur",
                    additionalMetadataTags = c("EXIF:Model", "EXIF:Make", "NonExistingTag"),
                          eventSummaryColumn  = "EXIF:Make",
                          eventSummaryFunction = "unique"
                          )

# note the warning that the last tag in "additionalMetadataTags" ("NonExistingTag") was not found


any(rec_table1$Species == "UNID")    # TRUE
any(rec_table2$Species == "UNID")    # FALSE


# here's how the removeDuplicateRecords argument works

rec_table3a <- recordTable(inDir              = wd_images_ID_species,
                          IDfrom              = "directory",
                          minDeltaTime        = 0,
                          exclude             = "UNID",
                          timeZone            = "Asia/Kuala_Lumpur",
                          removeDuplicateRecords = FALSE
)

rec_table3b <- recordTable(inDir              = wd_images_ID_species,
                          IDfrom              = "directory",
                          minDeltaTime        = 0,
                          exclude             = "UNID",
                          timeZone            = "Asia/Kuala_Lumpur",
                          removeDuplicateRecords = TRUE
)


anyDuplicated(rec_table3a[, c("Station", "Species", "DateTimeOriginal")])  # got duplicates
anyDuplicated(rec_table3b[, c("Station", "Species", "DateTimeOriginal")])  # no duplicates

# after removing duplicates, both are identical:
whichAreDuplicated <- which(duplicated(rec_table3a[,c("Station", "Species", "DateTimeOriginal")]))
all(rec_table3a[-whichAreDuplicated,] == rec_table3b)
```

```
### extracting species IDs from metadata

wd_images_ID_species_tagged <- system.file("pictures/sample_images_species_tag",
                                           package = "camtrapR")

rec_table4 <- recordTable(inDir              = wd_images_ID_species_tagged,
                      IDfrom                 = "metadata",
                      metadataSpeciesTag     = "Species",
                      exclude                = "unidentified")


###  Including videos
# sample videos are not included in package

# with videos, IDfrom = "directory", not extracting digiKam metadata

rec_table4 <- recordTable(inDir  = wd_images_ID_species,
                          IDfrom = "directory",
                          video  = list(file_formats = c("jpg", "mp4"),
                                        dateTimeTag  = "QuickTime:CreateDate")
)

# with videos, IDfrom = "metadata", extracting digiKam metadata

rec_table5 <- recordTable(inDir  = wd_images_ID_species,
                          IDfrom = "metadata",
                          metadataSpeciesTag = "Species",
                          video  = list(file_formats = c("jpg", "mp4", "avi", "mov"),
                                        dateTimeTag  = "QuickTime:CreateDate",
                                        db_directory = "C:/Users/YourName/Pictures",
                                        db_filename = "digikam4.db")
)

} else {
# show function output if ExifTool is not available
message("ExifTool is not available. Cannot test function. Loading recordTableSample instead")
data(recordTableSample)
}


## End(Not run)
```

---

recordTableIndividual *Generate a single-species record table with individual identification from camera trap images or videos*

Description

The function generates a single-species record table containing individual IDs, e.g. for (spatial) capture-recapture analyses. It prepares input for the function spatialDetectionHistory.

Usage

```
recordTableIndividual(
  inDir,
  hasStationFolders,
  IDfrom,
  cameraID,
  camerasIndependent,
  minDeltaTime = 0,
  deltaTimeComparedTo,
  timeZone,
  stationCol,
  writecsv = FALSE,
  outDir,
  metadataHierarchyDelimitor = "|",
  metadataIDTag,
  additionalMetadataTags,
  removeDuplicateRecords = TRUE,
  returnFileNamesMissingTags = FALSE,
  eventSummaryColumn,
  eventSummaryFunction,
  video
)
```

Arguments

inDir	character. Directory containing images of individuals. Must end with species name (e.g. ".../speciesImages/Clouded Leopard")
hasStationFolders	logical. Does inDir have station subdirectories? If TRUE, station IDs will be taken from directory names. If FALSE, they will be taken from image filenames (requires images renamed with imageRename).
IDfrom	character. Read individual ID from image metadata ("metadata") of from directory names ("directory")?
cameraID	character. Should the function look for camera IDs in the image file names? If so, set to 'filename'. Requires images renamed with imageRename. If missing, no camera ID will be assigned and it will be assumed there was 1 camera only per station.
camerasIndependent	logical. If TRUE, cameras at a station are assumed to record individuals independently. If FALSE, cameras are assumed to be non-independent (e.g. in pairs). Takes effect only if there was more than 1 camera per station and cameraID = "filename".

`minDeltaTime` numeric. time difference between observation of the same individual at the same station/camera to be considered independent (in minutes)

`deltaTimeComparedTo`
character. For two records to be considered independent, must the second one be at least `minDeltaTime` minutes after the last independent record of the same individual (`"lastIndependentRecord"`), or `minDeltaTime` minutes after the last record (`"lastRecord"`)?

`timeZone` character. Must be a value returned by `OlsonNames`

`stationCol` character. Name of the camera trap station column in the output table.

`writecsv` logical. Should the individual record table be saved as a .csv file?

`outDir` character. Directory to save csv file to. If NULL and `writecsv = TRUE`, the output csv will be written to `inDir`.

`metadataHierarchyDelimitor`
character. The character delimiting hierarchy levels in image metadata tags in field "HierarchicalSubject". Either "|" or ":".

`metadataIDTag` character. In custom image metadata, the individual ID tag name.

`additionalMetadataTags`
character. additional camera model-specific metadata tags to be extracted. (If possible specify tag groups as returned by `exifTagNames`)

`removeDuplicateRecords`
logical. If there are several records of the same individual at the same station (also same camera if cameraID is defined) at exactly the same time, show only one?

`returnFileNamesMissingTags`
logical. If species are assigned with metadata and images are not tagged, return a few file names of these images as a message?

`eventSummaryColumn`
character. A column in the record table (e.g. from a metadata tag) by to summarise non-independent records (those within `minDeltaTime` of a given record) with a user-defined function (`eventSummaryFunction`)

`eventSummaryFunction`
character. The function by which to summarise `eventSummaryColumn` of non-independent records, e.g. "sum", "max" (optional)

`video` list. Contains information on how to handle video data (optional). See details.

Details

The function can handle a number of different ways of storing images and videos. In every case, images need to be stored in a species directory first (e.g. using function `getSpeciesImages`). Station subdirectories are optional. Camera subdirectories are not supported. This directory structure can be created easily with function `getSpeciesImages`.

As with species identification, individuals can be identified in 2 different ways: by moving images into individual directories ("Species/Station/Individual/XY.JPG" or "Species/Individual/XY.JPG") or by metadata tagging (without the need for individual directories: "Species/XY.JPG" or "Species/Station/XY.JPG").

`minDeltaTime` is a criterion for temporal independence of records of an individual at the same station/location. Setting it to 0 will make the function return all records. `camerasIndependent`

defines if the cameras at a station are to be considered independent (e.g. FALSE if both cameras face each other and possibly TRUE if they face different trails). stationCol is the station column name to be used in the resulting table. Station IDs are read from the station directory names if hasStationFolders = TRUE. Otherwise, the function will try to extract station IDs from the image filenames (requires images renamed with imageRename.

If individual IDs were assigned with image metadata tags, metadataIDTag must be set to the name of the metadata tag group used for individual identification. metadataHierarchyDelimitor is "|" for images tagged in DigiKam and images tagged in Adobe Bridge/ Lightroom with the default settings. Manufacturer-specific Exif metadata tags such as "AmbientTemperature" or "MoonPhase" can be extracted if specified in additionalMetadataTags. Multiple names can be specified as a character vector as: c(Tag1, Tag2, ...). Because they are not standardized, function exifTagNames provides a vector of all available tag names. The metadata tags thus extracted may be used as individual covariates in spatial capture-recapture models.

eventSummaryColumn and eventSummaryFunction can be used to extract summary statistics for independent sampling events. For example, you assigned a "count" tag to your images, indicating the number of individuals in a picture. In a sequence of pictures taken within 1 minute, most pictures show one individual, but one image shows two individuals. You tagged the images accordingly (count = 1 or count = 2) and run recordTable. Set eventSummaryColumn = "count" and eventSummaryFunction = "max" to obtain the maximum number of count in all images within minDeltaTime minutes of a given record. The results is in a new column, in this example count_max. You can also calculate several statistics at the same time, by supplying vectors of values, e.g. eventSummaryColumn = c("count", "count", "camera") and eventSummaryFunction = c("min", "max", "unique") to get minimum and maximum count and all unique camera IDs for that event. Note that eventSummaryColumn and eventSummaryFunction must be of same length.

Argument video is analogous to recordTable, a named list with 2 or 4 items. 2 items (file_formats, dateTimeTag) are always required, and are sufficent if IDfrom = "directory". In that case, no digiKam tags will be returned. To return digiKam tags, two additional items are required (db_directory, db_filename). This is essential when using IDfrom = "metadata". When using IDfrom = "directory", it is optional, but allows to extract metadata tags assigned to videos in digiKam. This workaround is necessary because digiKam tags are not written into video metadata, but are only saved in the digiKam database. So in contrast to JPG images, they can not be extracted with ExifTool. It also requires that inDir is in your digiKam database.

The items of argument video are:

file_formats	The video formats to extract (include "jpg" if you want .JPG image metadata)
dateTimeTag	the metadata tag to extract date/time from (use exifTagNames to find out which tag is suitable)
db_directory	The directory containing digiKam database (optional if IDfrom = "directory")
db_filename	The digiKam database file in db_directory (optional if IDfrom = "directory")

See the example below for for how to specify the argument video.

Value

A data frame containing species records with individual IDs and additional information about stations, date, time and (optionally) further metadata.

Warning

Be sure to read the section on individual identification in the package vignette (https://CRAN.R-project.org/package=camtrapR/vignettes/camtrapr2.pdf).

Af you use image metadata tags for identification, the tags must be written to the image metadata. The function cannot read tags from .xmp sidecar files. Make sure you set the preferences of your image management software accordingly. In DigiKam, go to Settings/Configure digiKam/Metadata. There, make sure "Write to sidecar files" is unchecked.

Please note the section about defining argument `timeZone` in the vignette on data extraction (accessible via `vignette("DataExtraction")` or online (https://cran.r-project.org/package=camtrapR/vignettes/camtrapr3.pdf)).

Author(s)

Juergen Niedballa

References

Phil Harvey's ExifTool https://exiftool.org/

Examples

```
## Not run:    # the examples run too long to pass CRAN tests

 wd_images_ID_individual <- system.file("pictures/sample_images_indiv_tag/LeopardCat",
                                        package = "camtrapR")
 # missing space in species = "LeopardCat" is because of CRAN package policies
 # note argument additionalMetadataTags: contains tag names as returned by function exifTagNames

 if (Sys.which("exiftool") != ""){      # only run these examples if ExifTool is available

 rec_table_pbe <- recordTableIndividual(inDir                = wd_images_ID_individual,
                                        minDeltaTime          = 60,
                                        deltaTimeComparedTo   = "lastRecord",
                                        hasStationFolders     = FALSE,
                                        IDfrom                = "metadata",
                                        camerasIndependent    = FALSE,
                                        writecsv              = FALSE,
                                        metadataIDTag         = "individual",
                                    additionalMetadataTags = c("EXIF:Model", "EXIF:Make"),
                                        timeZone              = "Asia/Kuala_Lumpur"
 )


 # extracting some example summary stats too
 # a nonsensical example, get all unique cameras with which the event was photographed

  rec_table_pbe2 <- recordTableIndividual(inDir               = wd_images_ID_individual,
                                        minDeltaTime          = 60,
                                        deltaTimeComparedTo   = "lastRecord",
```

```
                                        hasStationFolders      = FALSE,
                                        IDfrom                 = "metadata",
                                        camerasIndependent     = FALSE,
                                        writecsv               = FALSE,
                                        metadataIDTag          = "individual",
                                    additionalMetadataTags = c("EXIF:Model", "EXIF:Make"),
                                        timeZone               = "Asia/Kuala_Lumpur",
                                        eventSummaryColumn     = "EXIF:Make",
                                        eventSummaryFunction   = "unique"
 )

 ### Video example (the sample data don't contain a video, this is hypothetical)
 # with JPG, video mp4, avi, mov, ID = metadata

rec_table_ind_video <- recordTableIndividual(inDir = wd_images_ID_individual,
                           hasStationFolder  = FALSE,
                           IDfrom            = "metadata",
                           metadataIDTag     = "individual",
                           video = list(file_formats = c("jpg", "mp4", "avi", "mov"),
                                        dateTimeTag = "QuickTime:CreateDate",
                                        db_directory = "C:/Users/YourName/Pictures",
                                        db_filename = "digikam4.db")
)


} else {
# show function output if ExifTool is not available
message("ExifTool is not available. Cannot test function. Loading recordTableSample instead")
data(recordTableSample)
}

## End(Not run)
```

`recordTableIndividualSample` *Sample single-species record table with custom metadata from camera trap images*

Description

Sample single-species record table with individual IDs from the tagged sample images in the package. Generated with function `recordTableIndividual`.

Usage

```
data(recordTableIndividualSample)
```

Format

A data frame with 21 rows and 17 variables

Details

The variables are as follows:

`Station`	Camera trap station ID
`Species`	Species ID
`Individual`	Individual ID
`DateTimeOriginal`	Date and time as extracted from image
`Date`	record date
`Time`	record time of day
`delta.time.secs`	time difference to first species record at a station (seconds)
`delta.time.mins`	time difference to first species record at a station (minutes)
`delta.time.hours`	time difference to first species record at a station (hours)
`delta.time.days`	time difference to first species record at a station (days)
`Directory`	Image directory
`FileName`	image filename
`HierarchicalSubject`	content of the HierarchicalSubject image metadata tag
`Model`	camera model extracted from image metadata
`Make`	camera make extracted from image metadata
`metadata_Species`	content of custom image metadata tag "Species" (see HierarchicalSubject)
`metadata_individual`	content of custom image metadata tag "individual" (see HierarchicalSubject)

---

recordTableIndividualSampleMultiSeason

Sample single-species multi-season record table with custom metadata from camera trap images

---

Description

Sample single-species multi-season record table with individual IDs from the tagged sample images in the package. Generated with function `recordTableIndividual`, then duplicated to simulate a second year.

Usage

```
data(recordTableIndividualSampleMultiSeason)
```

Format

A data frame with 31 rows and 17 variables

Details

The variables are as follows:

Station	Camera trap station ID
Species	Species ID
Individual	Individual ID
DateTimeOriginal	Date and time as extracted from image
Date	record date
Time	record time of day
delta.time.secs	time difference to first species record at a station (seconds)
delta.time.mins	time difference to first species record at a station (minutes)
delta.time.hours	time difference to first species record at a station (hours)
delta.time.days	time difference to first species record at a station (days)
Directory	Image directory
FileName	image filename
HierarchicalSubject	content of the HierarchicalSubject image metadata tag
Model	camera model extracted from image metadata
Make	camera make extracted from image metadata
metadata_Species	content of custom image metadata tag "Species" (see HierarchicalSubject)
metadata_individual	content of custom image metadata tag "individual" (see HierarchicalSubject)

Examples

```
# example data were created as follows:
data(recordTableIndividualSample)

recordTableIndividualSample_season2 <- recordTableIndividualSample[1:10, ]
recordTableIndividualSample_season2$DateTimeOriginal <- gsub("2009", "2010",
    recordTableIndividualSample_season2$DateTimeOriginal)
recordTableIndividualSampleMultiSeason <- rbind(recordTableIndividualSample,
    recordTableIndividualSample_season2)
```

---

recordTableSample *Sample species record table from camera trap images*

Description

Sample species record table from camera trap images generated from the sample images in the package with the function recordTable .

Usage

```
data(recordTableSample)
```

Format

A data frame with 39 rows and 11 variables

Details

The variables are as follows:

`Station`	Camera trap station ID
`Species`	Species ID
`DateTimeOriginal`	Date and time as extracted from image
`Date`	record date
`Time`	record time of day
`delta.time.secs`	time difference to first species record at a station (seconds)
`delta.time.mins`	time difference to first species record at a station (minutes)
`delta.time.hours`	time difference to first species record at a station (hours)
`delta.time.days`	time difference to first species record at a station (days)
`Directory`	Image directory
`FileName`	image filename

---

recordTableSampleMultiSeason

Sample multi-season species record table from camera trap images

---

Description

Sample multi-season species record table from camera trap images generated from the sample images in the package with the function `recordTable`. Season 2009 is the same as `recordTableSample`, season 2010 was simulated by adding 1 year to these records.

Usage

```
data(recordTableSampleMultiSeason)
```

Format

A data frame with 78 rows and 11 variables

Details

The variables are as follows:

`Station`	Camera trap station ID
`Species`	Species ID
`DateTimeOriginal`	Date and time as extracted from image
`Date`	record date
`Time`	record time of day
`delta.time.secs`	time difference to first species record at a station (seconds)
`delta.time.mins`	time difference to first species record at a station (minutes)
`delta.time.hours`	time difference to first species record at a station (hours)

delta.time.days	time difference to first species record at a station (days)
Directory	Image directory
FileName	image filename

Examples

```
# data were created with the following code:

data(recordTableSample)
recordTableSample_season2 <- recordTableSample

# substitute 2009 with 2010
recordTableSample_season2$DateTimeOriginal <- gsub("2009", "2010",
    recordTableSample_season2$DateTimeOriginal)
# combine with season 2009
recordTableSampleMultiSeason <- rbind(recordTableSample, recordTableSample_season2)
```

---

spatialDetectionHistory

Generate a capthist *object for spatial capture-recapture analyses from camera-trapping data*

Description

This function generates spatial detection histories of individuals of a species for spatial capture-recapture analyses with package secr. Data are stored in a capthist object. The capthist object contains detection histories, camera-trap station location and possibly individual and station-level covariates. Detection histories can have adjustable occasion length and occasion start time (as in the function detectionHistory).

Usage

```
spatialDetectionHistory(
  recordTableIndividual,
  species,
  camOp,
  CTtable,
  output = c("binary", "count"),
  stationCol = "Station",
  speciesCol = "Species",
  sessionCol,
  Xcol,
  Ycol,
  stationCovariateCols,
  individualCol,
  individualCovariateCols,
  recordDateTimeCol = "DateTimeOriginal",
```

```
  recordDateTimeFormat = "ymd HMS",
  occasionLength,
  minActiveDaysPerOccasion,
  occasionStartTime = "deprecated",
  maxNumberDays,
  day1,
  buffer,
  includeEffort = TRUE,
  scaleEffort = FALSE,
  binaryEffort = FALSE,
  timeZone,
  makeRMarkInput
)
```

Arguments

recordTableIndividual	data.frame. the record table with individual IDs created by recordTableIndividual
species	character. the species for which to compute the detection history
camOp	The camera operability matrix as created by cameraOperation
CTtable	data.frame. contains station IDs and coordinates. Same as used in cameraOperation.
output	character. Return individual counts ("count") or binary observations ("binary")?
stationCol	character. name of the column specifying Station ID in recordTableIndividual and CTtable
speciesCol	character. name of the column specifying species in recordTableIndividual
sessionCol	character. name of the column specifying session IDs, either in recordTableIndividual or in CTtable. See 'Details' for more information. Session ID values must be a sequence of integer numbers beginning with 1 (i.e., 1,2,3,...).
Xcol	character. name of the column specifying x coordinates in CTtable
Ycol	character. name of the column specifying y coordinates in CTtable
stationCovariateCols	character. name of the column(s) specifying station-level covariates in CTtable
individualCol	character. name of the column specifying individual IDs in recordTableIndividual
individualCovariateCols	character. name of the column(s) specifying individual covariates in recordTableIndividual
recordDateTimeCol	character. name of the column specifying date and time in recordTableIndividual
recordDateTimeFormat	format of column recordDateTimeCol in recordTableIndividual
occasionLength	integer. occasion length in days
minActiveDaysPerOccasion	integer. minimum number of active trap days for occasions to be included (optional)

occasionStartTime
: (DEPRECATED) integer. time of day (the full hour) at which to begin occasions. Please use argument occasionStartTime in cameraOperation instead.

maxNumberDays
: integer. maximum number of trap days per station (optional)

day1
: character. When should occasions begin: station setup date ("station"), first day of survey ("survey"), a specific date (e.g. "2015-12-31")?

buffer
: integer. Makes the first occasion begin a number of days after station setup. (optional)

includeEffort
: logical. Include trapping effort (number of active camera trap days per station and occasion) as usage in capthist object?

scaleEffort
: logical. scale and center effort matrix to mean = 0 and sd = 1? Currently not used. Must be FALSE.

binaryEffort
: logical. Should effort be binary (1 if >1 active day per occasion, 0 otherwise)?

timeZone
: character. Must be a value returned by OlsonNames

makeRMarkInput
: logical. If FALSE, output will be a data frame for RMark. If FALSE or not specified, a secr capthist object

Details

The function creates a capthist object by combining three different objects: 1) a record table of identified individuals of a species, 2) a camera trap station table with station coordinates and 3) a camera operation matrix computed with cameraOperation. The record table must contain a column with individual IDs and optionally individual covariates. The camera trap station table must contain station coordinates and optionally station-level covariates. The camera operation matrix provides the dates stations were active or not and the number of active stations.

day1 defines if each stations detection history will begin on that station's setup day (day1 = "station") or if all station's detection histories have a common origin (the day the first station was set up if day1 = "survey" or a fixed date if, e.g. day1 = "2015-12-31").

includeEffort controls whether an effort matrix is computed or not. If TRUE, effort will be used for object usage information in a traps. binaryEffort makes the effort information binary. scaleEffort is currently not used and must be set to FALSE. The reason is that usage can only be either binary, or nonnegative real values, whereas scaling effort would return negative values.

The number of days that are aggregated is controlled by occasionLength. occasionStartTime will be removed from the function. It has moved to cameraOperation, to ensure daily effort is computed correctly and takes the occasion start time into account. another hour than midnight (the default). This may be relevant for nocturnal animals, in which 1 whole night would be considered an occasion.

Output can be returned as individual counts per occasion (output = "count") or as binary observation (output = "binary").

Argument sessionCol can be used to a create multi-session capthist object. There are two different ways in which the argument is interpreted. It depends on whether a column with the name you specify in argument sessionCol exists in recordTableIndividual or in CTtable. If sessionCol is found in recordTableIndividual, the records will be assigned to the specified sessions, and it will be assumed that all camera trap station were used in all sessions. Alternatively, if sessionCol is found inCTtable, it will be assumed that only a subset of stations was used in each session, and

the records will be assigned automatically (using the station IDs to identify which session they belong into). In both cases, session information must be provided as a sequence of integer numbers beginnign with 1, i.e., you provide the session number directly in `sessionCol`. See `session` for more information about sessions in **secr**.

`capthist` objects (as created by `spatialDetectionHistory` for spatial capture-recapture analyses) expect the units of coordinates (`Xcol` and `col` in `CTtable`) to be meters. Therefore, please use a suitable coordinate system (e.g. UTM).

`recordDateTimeFormat` defaults to the "YYYY-MM-DD HH:MM:SS" convention, e.g. "2014-09-30 22:59:59". `recordDateTimeFormat` can be interpreted either by base-R via `strptime` or in **lubridate** via `parse_date_time` (argument "orders"). **lubridate** will be used if there are no "%" characters in `recordDateTimeFormat`.

For "YYYY-MM-DD HH:MM:SS", `recordDateTimeFormat` would be either "%Y-%m-%d %H:%M:%S" or "ymd HMS". For details on how to specify date and time formats in R see `strptime` or `parse_date_time`.

Value

Output depends on argument `makeRMarkInput`:

`list("makeRMarkInput = FALSE")`
: A `capthist` object

`list("makeRMarkInput = TRUE")`
: A data frame for use in RMark

Warning

Please note the section about defining argument `timeZone` in the vignette on data extraction (accessible via `vignette("DataExtraction")` or online (https://cran.r-project.org/package=camtrapR/vignettes/camtrapr3.pdf)).

Author(s)

Juergen Niedballa

See Also

secr RMark

Examples

```
data(recordTableIndividualSample)
data(camtraps)

# create camera operation matrix (with problems/malfunction)
camop_problem <- cameraOperation(CTtable      = camtraps,
                                 stationCol   = "Station",
                                 setupCol     = "Setup_date",
                                 retrievalCol = "Retrieval_date",
                                 writecsv     = FALSE,
```

```
                                  hasProblems  = TRUE,
                                  dateFormat   = "dmy"
)

sdh <- spatialDetectionHistory(recordTableIndividual = recordTableIndividualSample,
                               species               = "LeopardCat",
                               camOp                 = camop_problem,
                               CTtable               = camtraps,
                               output                = "binary",
                               stationCol            = "Station",
                               speciesCol            = "Species",
                               Xcol                  = "utm_x",
                               Ycol                  = "utm_y",
                               individualCol         = "Individual",
                               recordDateTimeCol     = "DateTimeOriginal",
                               recordDateTimeFormat  = "ymd HMS",
                               occasionLength        = 10,
                               day1                  = "survey",
                               includeEffort         = TRUE,
                               timeZone              = "Asia/Kuala_Lumpur"
  )

# missing space in species = "LeopardCat" was introduced by recordTableIndividual
# (because of CRAN package policies.
# In your data you can have spaces in your directory names)

  summary(sdh)
  plot(sdh, tracks = TRUE)

  ## multi-season capthist object
 # see vignette "3. Extracting Data from Camera Trapping Images, creating occupancy & secr input"

  data(camtrapsMultiSeason)
  camtrapsMultiSeason$session[camtrapsMultiSeason$session == 2009] <- 1
  camtrapsMultiSeason$session[camtrapsMultiSeason$session == 2010] <- 2

  data(recordTableIndividualSampleMultiSeason)

  # create camera operation matrix (with problems/malfunction)
  camop_session <- cameraOperation(CTtable         = camtrapsMultiSeason,
                                   stationCol   = "Station",
                                   setupCol     = "Setup_date",
                                   sessionCol   = "session",
                                   retrievalCol = "Retrieval_date",
                                   hasProblems  = TRUE,
                                   dateFormat   = "dmy"
  )

sdh_multi <- spatialDetectionHistory(recordTableIndividual = recordTableIndividualSampleMultiSeason,
                               species               = "LeopardCat",
                               output                = "binary",
                               camOp                 = camop_session,
                               CTtable               = camtrapsMultiSeason,
```

```
                              stationCol            = "Station",
                              speciesCol            = "Species",
                              sessionCol            = "session",
                              Xcol                  = "utm_x",
                              Ycol                  = "utm_y",
                              individualCol         = "Individual",
                              recordDateTimeCol     = "DateTimeOriginal",
                              recordDateTimeFormat  = "ymd HMS",
                              occasionLength        = 10,
                              day1                  = "survey",
                              includeEffort         = TRUE,
                              timeZone              = "Asia/Kuala_Lumpur",
                              stationCovariateCols  = "utm_y",         # example
                              individualCovariateCols = "Individual"   # example
)

summary(sdh_multi)
plot(sdh_multi, tracks = TRUE)
```

`speciesAccum` *Species Accumulation Curves for Camera Trap Data*

Description

Generates species accumulation, rarefaction and extrapolation curves from camera trap data using the **iNEXT** package (Chao et al. 2014). The function creates sampling effort-based accumulation curves with sampling units being either camera trap stations or days.

Note that these curves are based on observed detections only and do not account for imperfect detection. Species may be present but not detected, leading to underestimation of true species richness. For analyses that explicitly account for imperfect detection, consider using occupancy-based approaches (see `communityModel`).

Usage

```
speciesAccum(
  CTtable,
  recordTable,
  speciesCol,
  recordDateTimeCol,
  recordDateTimeFormat = "ymd HMS",
  setupCol,
  dateFormat = "ymd",
  stationCol,
  assemblageCol = NULL,
  q = 0,
  x_unit = c("station", "survey_day", "station_day"),
  knots = 40,
```

```
  conf = 0.95,
  nboot = 50,
  endpoint = NULL
)
```

Arguments

CTtable	data.frame containing the camera trap deployment information.
recordTable	data.frame containing the camera trap records.
speciesCol	character. Name of the column specifying species names in recordTable
recordDateTimeCol	character. Name of the column containing date and time information in recordTable
recordDateTimeFormat	character. Format of column recordDateTimeCol in recordTable
setupCol	character. Name of the column containing camera setup dates in CTtable
dateFormat	character. Format of column setupCol in CTtable
stationCol	character. Name of the column containing station IDs in both tables
assemblageCol	character. Optional. Name of column in recordTable for grouping data into separate assemblages
q	numeric. The order of diversity measure. Default is 0 (species richness)
x_unit	character. Whether to use "station" or "day" as sampling unit. Default is "station"
knots	numeric. number of values along x axis for which values are computed
conf	numeric. confidence interval
nboot	numeric. number of replications
endpoint	integer. Sample size used as endpoint for rarefaction/extrapolation (in iNEXT)

Details

The function provides three types of curves:

- Sample-size-based rarefaction/extrapolation curve
- Sample completeness curve
- Coverage-based rarefaction/extrapolation curve

While these curves provide useful insights into sampling completeness and species richness patterns, they should be interpreted with caution in camera trap studies. Unlike occupancy models, they do not account for:

- Imperfect detection (species present but not detected)
- Variation in detection probability among species
- Spatial variation in species occurrence
- Temporal variation in species activity

Value

An object of class "iNEXT" containing:

- DataInfo - data information
- iNextEst - diversity estimates for rarefied and extrapolated samples
- AsyEst - asymptotic diversity estimates

Note

Requires package **iNEXT**.

Author(s)

Juergen Niedballa

References

Chao, A., Gotelli, N. J., Hsieh, T. C., Sander, E. L., Ma, K. H., Colwell, R. K., & Ellison, A. M. (2014). Rarefaction and extrapolation with Hill numbers: A framework for sampling and estimation in species diversity studies. Ecological Monographs, 84(1), 45-67.

See Also

`surveyDashboard` for interactive species accumulation analysis

Examples

```
## Not run:
# Basic usage with stations as sampling units
result <- speciesAccum(
  CTtable = cams,
  recordTable = recs,
  speciesCol = "Species",
  recordDateTimeCol = "DateTime",
  setupCol = "Setup_date",
  stationCol = "Station",
  q = 0,
  x_unit = "station"
)

# Plot results
ggiNEXT(result, type = 1)  # Sample-size-based R/E curve
ggiNEXT(result, type = 2)  # Sample completeness curve
ggiNEXT(result, type = 3)  # Coverage-based R/E curve

# With assemblage grouping and days as sampling units
result_by_assemblage <- speciesAccum(
  CTtable = cams,
  recordTable = recs,
  speciesCol = "Species",
  recordDateTimeCol = "DateTime",
```

```
  setupCol = "Setup_date",
  stationCol = "Station",
  assemblageCol = "Season",
  q = 0,
  x_unit = "day"
)

## End(Not run)
```

---

summary,commOccu-method

Summarize community occupancy model

Description

Gives an overview of the number of species, stations and occasions in a commOccu object. Also returns covariates.

Usage

```
## S4 method for signature 'commOccu'
summary(object, ...)
```

Arguments

object	commOccu object
...	currently ignored

Details

The summary method is very basic and still work in progress.

Value

Model summary printed to console

surveyDashboard *Survey Dashboard for Camera Trap Data Analysis*

Description

A comprehensive Shiny dashboard for analyzing camera trap survey data. The dashboard provides interactive visualization, data exploration, and analysis tools including:

- Data import from CSV files, Wildlife Insights exports, or camtrapDP format
- Interactive maps for camera locations and species detections
- Species activity pattern analysis
- Covariate extraction and analysis tools
- Single-species and community occupancy modeling
- Spatial prediction capabilities

Usage

```
surveyDashboard(
  CTtable = NULL,
  recordTable = NULL,
  stationCol = NULL,
  cameraCol = NULL,
  xcol = NULL,
  ycol = NULL,
  crs = NULL,
  setupCol = NULL,
  retrievalCol = NULL,
  hasProblems = FALSE,
  CTdateFormat = "ymd",
  camerasIndependent = NULL,
  speciesCol = "Species",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  timeZone = "UTC",
  exclude = NULL
)
```

Arguments

CTtable	A data.frame containing the camera trap deployment information.
recordTable	A data.frame containing the camera trap records.
stationCol	The column name containing the camera trap station ID
cameraCol	The column name containing the camera trap IDs (optional, only if 2 or more cameras per station)

xcol	The column name containing the X coordinate of the camera trap station.
ycol	The column name containing the Y coordinate of the camera trap station.
crs	The coordinate reference system (CRS) of the camera trap data. Must be a valid argument to st_crs
setupCol	The column name containing the camera trap deployment date (and time).
retrievalCol	The column name containing the camera trap retrieval date (optionally date-time).
hasProblems	A logical indicating whether there are periods of cameras malfunctioning
CTdateFormat	The date format of the camera trap deployment and retrieval date and time (default: "ymd").
camerasIndependent	logical. If multiple camera per station, are they independent?
speciesCol	The column name containing the species names
recordDateTimeCol	The column name containing the record date and time
recordDateTimeFormat	The date/time format of recordDateTimeCol
timeZone	Time zone of records in recordTable
exclude	Species to be excluded from the data set

Details

The dashboard includes several major components:

Data Import & Management:

- CSV file import with column mapping
- Wildlife Insights data import (zip, CSV, or directory)
- camtrap DP data import
- Study area import from shapefile
- Save/restore functionality for app state
- Export functionality to save data from dashboard

Data Processing:

- Flexible station filtering with multiple criteria
- Temporal record filtering with independence criteria
- Filtering species records by species name
- Automated covariate extraction from local rasters or online elevation data
- Covariate correlation analysis with visualization
- Species accumulation curves

Basic Analysis:

- Basic summary statistics

- Interactive overview and species detection maps
- Activity pattern analysis (single species and two-species overlap)
- Camera operation visualization

Occupancy Modeling:

- Basic workflow for simple model specification (linear effects)
- Support for both unmarked and ubms packages
- Automated detection history creation
- Model comparison and selection
- Response curves and spatial predictions

Community Occupancy Modeling:

- Flexible species selection with filtering
- Support for fixed, random, and independent effects
- Species-site random effects
- Effort handling on detection
- MCMC diagnostics and convergence assessment
- Species occupancy, richness and PAO predictions

Value

A Shiny dashboard application for camera trap survey data analysis

Note

- Interactive maps with multiple basemap options
- Covariate scaling is performed automatically if requested (includes automatic scaling of prediction rasters)
- The app state can be saved and restored

Current limitations include:

- supports only single-season data - no support for spatial capture-recapture models (or anything related to individual IDs)

Author(s)

Juergen Niedballa

Examples

```
## Not run:

# Start the dashboard without parameters
# This opens the application with a welcome screen where data can be imported
surveyDashboard()

# Basic usage with minimal parameters

data("camtraps")
data("recordTableSample")

  surveyDashboard(
    CTtable = camtraps,
    recordTable = recordTableSample,
    xcol = "utm_x",
    ycol = "utm_y",
    crs = "epsg:32650",      # = UTM50N
    stationCol = "Station",
    setupCol = "Setup_date",
    retrievalCol = "Retrieval_date",
    CTdateFormat = "dmy"
  )

## End(Not run)
```

`surveyReport` *Create a report about a camera trapping survey and species detections*

Description

This function creates a report about a camera trapping survey and species records. It uses a camera trap station information table and a record table (generated with `recordTable`) as input. Output tables can be saved and a zip file for simple data sharing can be created easily.

Usage

```
surveyReport(
  recordTable,
  CTtable,
  camOp,
  speciesCol = "Species",
  stationCol = "Station",
  cameraCol,
  setupCol,
  retrievalCol,
```

```
  CTDateFormat = "ymd",
  CTHasProblems = "deprecated",
  recordDateTimeCol = "DateTimeOriginal",
  recordDateTimeFormat = "ymd HMS",
  Xcol,
  Ycol,
  sinkpath,
  makezip
)
```

Arguments

recordTable	data.frame containing a species record table as given by recordTable
CTtable	data.frame containing information about location and trapping period of camera trap stations (equivalent to camtraps
camOp	camera operation matrix created with cameraOperation
speciesCol	character. name of the column specifying Species ID in recordTable
stationCol	character. name of the column specifying Station ID in CTtable and recordTable
cameraCol	character. name of the column specifying Camera ID in CTtable and recordTable
setupCol	character. name of the column containing camera setup dates in CTtable
retrievalCol	character. name of the column containing camera retrieval dates in CTtable
CTDateFormat	character. The format of columns setupCol and retrievalCol (and potential problem columns) in CTtable. Must be interpretable by either as.Date or the "orders" argument parse_date_time in **lubridate**.
CTHasProblems	deprecated (since version 2.1)
recordDateTimeCol	character. The name of the column containing date and time of records in recordTable
recordDateTimeFormat	character. The date/time format of column recordDateTimeCol in recordTable.
Xcol	character. name of the column specifying x coordinates in CTtable. Used to create detection maps if makezip is TRUE. (optional)
Ycol	character. name of the column specifying y coordinates in CTtable. Used to create detection maps if makezip is TRUE. (optional)
sinkpath	character. The directory into which the survey report is saved (optional)
makezip	logical. Create a zip file containing tables, plots and maps in sinkpath?

Details

dateFormat defaults to "YYYY-MM-DD", e.g. "2014-10-31". It can be specified either in the format required by strptime or the 'orders' argument in parse_date_time in **lubridate**. In the example above, "YYYY-MM-DD" would be specified as "%Y-%m-%d" or "ymd".

recordDateTimeFormat defaults to the "YYYY-MM-DD HH:MM:SS" convention, e.g. "2014-09-30 22:59:59". recordDateTimeFormat can be interpreted either by base-R via strptime or in

lubridate via parse_date_time (argument "orders"). **lubridate** will be used if there are no "%" characters in recordDateTimeFormat.

For "YYYY-MM-DD HH:MM:SS", recordDateTimeFormat would be either "%Y-%m-%d %H:%M:%S" or "ymd HMS". For details on how to specify date and time formats in R see strptime or parse_date_time.

Note: as of version 2.1, argument CTHasProblems is deprecated and defunct. Please use camOp instead to provide information about periods of camera activity and malfunction. If camOp is not provided the legacy version of surveyReport (from camtrapR 2.0.3) will be run with a warning.

Value

An invisible list containing 5 data.frames.

survey_dates
: station and image date ranges, number of total and active trap days (calendar days and taking into account independent effort of multiple cameras, if applicable), number of cameras per station

species_by_station
: species numbers by station

events_by_species
: number of events and stations by species

events_by_station
: number of events for every species by station (only species that were recorded)

events_by_station2
: number of events for all species at all stations (including species that were not recorded)

The output will be saved to a .txt file if sinkpath is defined.

If makezip is TRUE, a zip file will be created in sinkpath. It contains single-species activity plots, detection maps (if Xcol and Ycol are defined), the survey report tables, the record table and the camera trap station table, and an example R script.

Author(s)

Juergen Niedballa

See Also

recordTable

Examples

```
data(camtraps)
data(recordTableSample)

# since version 2.1, camera operation matrix is required as input

camop_no_problem <- cameraOperation(CTtable      = camtraps,
                                    stationCol   = "Station",
                                    setupCol     = "Setup_date",
```

```
                                    retrievalCol = "Retrieval_date",
                                    writecsv     = FALSE,
                                    hasProblems  = FALSE,
                                    dateFormat   = "dmy"
)

reportTest <- surveyReport (recordTable          = recordTableSample,
                            CTtable              = camtraps,
                            camOp                = camop_no_problem,
                            speciesCol           = "Species",
                            stationCol           = "Station",
                            setupCol             = "Setup_date",
                            retrievalCol         = "Retrieval_date",
                            CTDateFormat         = "dmy",
                            recordDateTimeCol    = "DateTimeOriginal",
                            recordDateTimeFormat = "ymd HMS")

class(reportTest)  # a list with
length(reportTest) # 5 elements

reportTest[[1]]    # camera trap operation times and image date ranges
reportTest[[2]]    # number of species by station
reportTest[[3]]    # number of events and number of stations by species
reportTest[[4]]    # number of species events by station
reportTest[[5]]   # number of species events by station including 0s (non-observed species)

# with camera problems

camop_problem <- cameraOperation(CTtable      = camtraps,
                                 stationCol   = "Station",
                                 setupCol     = "Setup_date",
                                 retrievalCol = "Retrieval_date",
                                 writecsv     = FALSE,
                                 hasProblems  = TRUE,
                                 dateFormat   = "dmy"
)

reportTest_problem <- surveyReport (recordTable          = recordTableSample,
                                    CTtable              = camtraps,
                                    camOp                = camop_problem,
                                    speciesCol           = "Species",
                                    stationCol           = "Station",
                                    setupCol             = "Setup_date",
                                    retrievalCol         = "Retrieval_date",
                                    CTDateFormat         = "dmy",
                                    recordDateTimeCol    = "DateTimeOriginal",
                                    recordDateTimeFormat = "ymd HMS")

reportTest_problem$survey_dates



## if camOp is missing, the legacy version (from 2.0.3) will be used:
```

```
reportTest_problem_old <- surveyReport (recordTable          = recordTableSample,
                                        CTtable              = camtraps,
                                       # camOp                = camop_problem,
                                        speciesCol           = "Species",
                                        stationCol           = "Station",
                                        setupCol             = "Setup_date",
                                        retrievalCol         = "Retrieval_date",
                                        CTDateFormat         = "dmy",
                                        recordDateTimeCol    = "DateTimeOriginal",
                                        recordDateTimeFormat = "ymd HMS")

## Not run:
# run again with sinkpath defined
reportTest <- surveyReport (recordTable          = recordTableSample,
                            CTtable              = camtraps,
                            camOp                = camop_no_problem,
                            speciesCol           = "Species",
                            stationCol           = "Station",
                            setupCol             = "Setup_date",
                            retrievalCol         = "Retrieval_date",
                            CTDateFormat         = "dmy",,
                            recordDateTimeCol    = "DateTimeOriginal",
                            recordDateTimeFormat = "ymd HMS",
                            sinkpath             = getwd())

# have a look at the text file
readLines(list.files(getwd(), pattern = paste("survey_report_", Sys.Date(), ".txt", sep = ""),
 full.names = TRUE))

## End(Not run)
```

`timeShiftImages` *Apply time shifts to JPEG image metadata*

Description

Change the values of digital timestamps in image metadata using ExifTool. If date/time of images were set incorrectly, they can be corrected easily in batch mode for further analyses. Please, always make a backup of your data before using this function to avoid data loss or damage. This is because ExifTool will make a copy of your images and applies the time shifts to the copies. The file extension of the original images (.JPG) will be renamed to ".JPG_original".

Usage

```
timeShiftImages(
  inDir,
  hasCameraFolders,
  timeShiftTable,
```

```
  stationCol,
  cameraCol,
  timeShiftColumn,
  timeShiftSignColumn,
  undo = FALSE,
  ignoreMinorErrors = FALSE
)
```

Arguments

inDir	character. Name of directory containing station directories with images
hasCameraFolders	logical. Do the station directories in inDir have camera subdirectories (e.g. "inDir/StationA/Camera1")?
timeShiftTable	data.frame containing information about station-/camera-specific time shifts.
stationCol	character. name of the column specifying Station ID in timeShiftTable
cameraCol	character. name of the column specifying Camera ID in timeShiftTable (optional)
timeShiftColumn	character. The name of the column containing time shift values in timeShiftTable
timeShiftSignColumn	character. The name of the column with the direction of time shifts in timeShiftTable. Can only be "-" or "+".
undo	logical. Undo changes and restore the original images? Please be careful, this deletes any edited images if TRUE
ignoreMinorErrors	logical. Ignore minor errors that would cause the function to fail (set TRUE for images with bad MakerNotes, observed in Panthera V4 cameras)

Details

timeShiftTable is a data frame with columns for station ID, camera ID (optional), time shift value and direction of time shift (for an example see timeShiftTable). Images in inDir must be sorted into station directories. If hasCameraFolders = TRUE, the function expects camera subdirectories in the station directories and will only apply time shifts to the camera subdirectories specified by CameraCol in timeShiftTable. If hasCameraFolders = FALSE, shifts will be applied to the whole station directory (including potential subdirectories).

The values of timeShiftColumn must adhere to the following pattern: "YYYY:mm:dd HH:MM:SS" ("year:month:day hour:minute:second"). Examples: "1:0:0 0:0:0" is a shift of exactly 1 year and "0:0:0 12:10:01" 12 hours and 10 minutes and 1 second. Note that stating "00" may cause problems, so use "0" instead if an entry is zero.

timeShiftSignColumn signifies the direction of the time shift. "+" moves image dates into the future (i.e. the image date lagged behind the actual date) and "-" moves image dates back (if the image dates were ahead of actual time).

ExifTool stores the original images as .JPG_original files in the original file location. By setting undo = TRUE, any JPG files in the directories specified by timeShiftTable will be deleted and the

original JPEGs will be restored from the JPG_original files. Please make a backup before using undo.

Years can have 365 or 366 days, and months 28 to 31 days. Here is how the function handles these (from the exiftool help page): "The ability to shift dates by Y years, M months, etc, conflicts with the design goal of maintaining a constant shift for all time values when applying a batch shift. This is because shifting by 1 month can be equivalent to anything from 28 to 31 days, and 1 year can be 365 or 366 days, depending on the starting date. The inconsistency is handled by shifting the first tag found with the actual specified shift, then calculating the equivalent time difference in seconds for this shift and applying this difference to subsequent tags in a batch conversion."

ignoreMinorErrors is useful if image timestamps are not updated correctly (entries in column "n_images" of the output are "... files weren't updated due to errors"). This can be caused by bad MakerNotes and so far was only observed in Panthera V4 and V6 cameras. In that case, set ignoreMinorErrors to TRUE. This will add the "-m" option to the Exiftool call, thereby ignoring minor errors and warnings and applying the time shift nevertheless.

Value

A data.frame containing the information about the processed directories and the number of images.

Author(s)

Juergen Niedballa

References

https://exiftool.org/#shift

Examples

```
## Not run:

# copy sample images to temporary directory (so we don't mess around in the package directory)
wd_images_ID <- system.file("pictures/sample_images_species_dir", package = "camtrapR")
file.copy(from = wd_images_ID, to = tempdir(), recursive = TRUE)
wd_images_ID_copy <- file.path(tempdir(), "sample_images_species_dir")

data(timeShiftTable)


timeshift_run <- timeShiftImages(inDir               = wd_images_ID_copy,
                                 timeShiftTable      = timeShiftTable,
                                 stationCol          = "Station",
                                 hasCameraFolders    = FALSE,
                                 timeShiftColumn     = "timeshift",
                                 timeShiftSignColumn = "sign",
                                 undo                = FALSE
)
```

```
timeshift_undo <- timeShiftImages(inDir               = wd_images_ID_copy,
                                  timeShiftTable      = timeShiftTable,
                                  stationCol          = "Station",
                                  hasCameraFolders    = FALSE,
                                  timeShiftColumn     = "timeshift",
                                  timeShiftSignColumn = "sign",
                                  undo                = TRUE
)

## End(Not run)
```

`timeShiftTable` *Sample camera trap time shift table*

Description

Sample camera trap time shift table

Usage

```
data(timeShiftTable)
```

Format

A data frame with 2 rows and 4 variables

Details

If image Exif metadata timestamps are wrong systematically (e.g. because camera system time was not set after changing batteries), it can be corrected using a `data.frame` in the following format using function `timeShiftImages`. For details on data format, please see `timeShiftImages`.

The variables are as follows:

`Station`	Camera trap station ID
`camera`	Camera trap ID (optional)
`timeshift`	time shift amount to be applied
`sign`	direction of time shift

`writeDateTimeOriginal` *Write values to DateTimeOriginal tag in image metadata*

Description

This function assigns values to the DateTimeOriginal tag in image's EXIF metadata using Exiftool. It can be used when the original DateTimeOriginal values in the metadata were lost for whatever reason. In order to first read the Date/Time values from the data fields in the images, see the function `OCRdataFields`. After running `OCRdataFields` and checking its output you can run `writeDateTimeOriginal`.

Usage

```
writeDateTimeOriginal(DateTimeOriginal, fileNames, parallel, overwrite = FALSE)
```

Arguments

`DateTimeOriginal`	character. DateTimeOriginal values to write into EXIF:DateTimeOriginal tag of the files in fileNames.
`fileNames`	character. Full file names (including directories) of images to process.
`parallel`	A parallel cluster object. Specify if you wish to run this function in parallel. Provide a cluster object (output of makeCluster()) - optional.
`overwrite`	logical. Overwrite existing files (TRUE) or create new files while saving the original data as jpg_original files as a backup (FALSE)?

Details

The first value in `DateTimeOriginal` will be assigned to the first image in `fileNames`, and so on. Both `DateTimeOriginal` and `fileNames` can be obtained from the output of `OCRdataFields`. `DateTimeOriginal` uses the standard "YYYY-MM-SS HH:MM:SS"" notation. If the values extracted via `OCRdataFields` are in a different format you'll need to reformat them first. Please provide them as character. Also, before using this function, make sure that the date/time values read by `OCRdataFields` are correct (sometimes OCR misreads values, so check carefully).

Parallel processing is advised since the function is rather slow (due to calling Exiftool separately on every, so about 1 second per image). If you know how to batch-assign DateTimeOriginal values in one Exiftool call, please let me know.

The function only works on JPG images, not video files.

Value

Invisible NULL. The actual output is the JPG images which now have a DateTimeOriginal tag.

Author(s)

Juergen Niedballa

See Also

OCRdataFields

Examples

```
## Not run:
# dontrun is to avoid forcing users to install additional dependencies

wd_images_OCR <- system.file("pictures/full_size_for_ocr", package = "camtrapR")

library(magick)

# define geometries
geometry1 <- geometry_area(x_off = 0, y_off = 0, width = 183, height = 37)
geometry2 <- geometry_area(x_off = 196, y_off = 0, width = 200, height = 17)
geometry3 <- geometry_area(x_off = 447, y_off = 0, width = 63, height = 17)
geometry4 <- geometry_area(x_off = 984, y_off = 0, width = 47, height = 17)
geometry5 <- geometry_area(x_off = 0, y_off = 793, width = 320, height = 17)

# combine geometries into list
geometries <- list(date = geometry1,
                   time = geometry2,
                   sequence_id = geometry3,
                   temperature = geometry4,
                   camera_model = geometry5)

df_image_data  <- OCRdataFields(inDir = wd_images_OCR,
                                geometries = geometries,
                                invert = TRUE)
df_image_data



library(parallel)
library(lubridate)

# prepare DateTimeOriginal column (ymd_hms() automatically respects the PM indicator)
df_image_data$DateTimeOriginal <- paste(df_image_data$date, df_image_data$time)
df_image_data$DateTimeOriginal <- as.character(ymd_hms(df_image_data$DateTimeOriginal))

# create cluster (3 cores)
cl <- makeCluster(3)

# assign new DateTimeOriginal
writeDateTimeOriginal(DateTimeOriginal = df_image_data$DateTimeOriginal,
                      fileNames = df_image_data$filename_full,
                      parallel = cl)

stopCluster(cl)
```

```
## End(Not run)
```

Index