2. Species and Individual Identification

Juergen Niedballa (camtrapr@gmail.com)

2024-02-01

Contents


```
library(camtrapR)
data(camtraps)
```

Methods for species and individual identification - General

Having organised your camera trap images into station (and camera) directories as explained in the vignette "Organising raw camera trap images in camtrapR", you can begin with species (and individual) identification.

Generally, species identification comes before individual identification. camtrapR supports two different methods for identifying species and individuals:

1. by moving renamed images into species/individual directories within station directories (drag & drop) or
2. by assigning metadata tags to images in image management software (metadata tagging).

It is recommended to to work with the renamed images (function `imageRename`), not the original generic file names. Species names are case-sensitive and must be spelled *exactly* the same in each station, including interpunction and white spaces. "Leopard" is not identical to "leopard" and "Pig-tailed Macaque" is different from "Pig tailed Macaque".

Since version 2.0.0, camtrapR supports video files. You can use both of the following methods for identification (drag & drop, metadata) for video files as well.

Drag & drop

For species identification, images can be moved into species directories created within station directories. The species ID is then read from the directory name. Likewise, individuals can be identified by moving them into indivdual directories. Be aware that one first needs to collect all images of the species of interest (using `getSpeciesImages` described below) before individual identification.

Metadata tagging

Metadata tags can be assigned to images in image management software. They are saved in the image metadata automatically, from where they can later be read out by two camtrapR functions (`recordTable` and `recordTableIndividual`). Metadata tagging can be used to assign custom tags to images, e.g. species ID, individual ID for capture-recapture analyses, number of individuals on images, sex or age class of individuals, behaviour etc..

Image management software recommendations for metadata tagging

Here is a list of suitable software for metadata tagging:

- digiKam (https://www.digikam.org)
- Adobe Bridge
- Adobe Lightroom

Among these, we recommend digiKam because it is free and open-source software. Furthermore, if you wish to assign tags to videos and read them in camtrapR, tag the videos in digiKam. Extracting video tags currently only works if videos were tagged with digiKam (requires digiKam 6.0.0 or higher, tested with version 6.1.0).

All of these programs offer customisable metadata structures and are easy to use. We only tested these three programs and it is unclear what other software will work smoothly, if at all.

digiKam setup

digiKam needs to write metadata tags into image metadata, not into .xmp sidecar files. Its behaviour can be configured. Here's how to set up digiKam:

If installing for the first time, it will ask you to make a few settings when you first start the program. When asked "Configure Metadata Storage to Files", set "Add Information to Files" instead of the default "Do Nothing".

If digiKam was installed already, go to “Settings”, then “Configure digiKam”. There, select “Metadata” on the left and, in the tab “Behaviour”, make sure that “Image tags” is checked in the box titled “Write This Information to the Metadata”. Below, in the box “Reading and Writing Metadata”, make sure that both “Read from sidecar files” and “Write to sidecar files” are unchecked. In more recent versions of digiKam, these two options are in a separate “Sidecars” tab, next to the “Behaviour” tab.

Metadata tag structure

Before assigning metadata tags, users need to set up a hierarchical tag structure in image management software. This structure is customisable and can be expanded as needed. In digiKam, this is done in the Tag Manager. Here’s some examples:

- Species
  - SpeciesA
  - SpeciesB
- Individual
  - Female1
  - Female2
  - Male1
  - Male2

Previously, only one level of hierarchy was allowed in tags. Since v.2.1.1, it is possible to specify multiple levels of hierarchy, e.g. the following structure is now accepted:

- Species
  - Cat
    - Leopard Cat
    - Tiger
  - Civet
    - Masked palm civet
    - Large Indian civet

If multiple tags are assigned in the same image (e.g. “Cat” and “Leopard Cat”), the function will assign the “deepest” tag (“Leopard Cat” in this case). Theoretically one can even set up even more tag levels.

The crucial point is to set up tag groups (e.g. “Species” for species identification) which then contain the desired tags (e.g. species tags such as “Leopard Cat”, “Malay Civet” etc.) or further tag group (e.g. “Cat” or “Civet” in the example above). This is what is meant by hierarchical tag structure. For clarification, here is a screenshot of the digiKam tag tree in which an image was identified as showing a Leopard Cat.

Two **important** warnings about metadata tagging: image management software must be configured to write custom image metadata tags into the images themselves, not into .xmp sidecar files. The relevant functions cannot read tags from .xmp sidecar files. Make sure you set the preferences in your image management software accordingly. In digiKam, go to Settings/Configure digiKam/Metadata. There, make sure “Write to sidecar files” is unchecked.

Secondly, do not use spaces or interpunction in the tag group names.

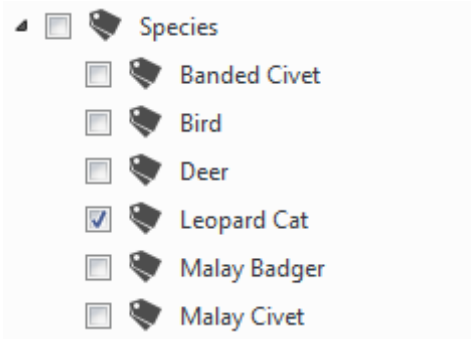

Figure 1: The desired tag structure shown in a digiKam screenshot

How does metadata extraction work?

This if for those who are interested in the technical details of how custom metadata are saved in images. The image management software will save metadata in a metadata tag field called "HierarchicalSubject" which camtrapR can then read out. Let's assume that in the above example we had three images, showing:

1. SpeciesA,
2. Female1 of SpeciesB,
3. both species in the same image.

Here's what the content of the HierarchicalSubject field would look like:

Species|SpeciesA

Species|SpeciesB, Individual|Female1

Species|SpeciesA, Species|SpeciesB

Note that multiple tags are separated by commas and hierarchy levels are by default separated by a vertical bar ("|"). Adobe software also lets you use colon (":"); you can tell camtrapR which to use using argument `metadataHierarchyDelimitor` in a number of functions. camtrapR reads the content of HierarchicalSubject and can automatically tabulate the contents.

And how about metadata extraction from videos?

I'm glad you asked. In digiKam, it is possible to assign tags to videos, just like tagging images. BUT, in contrast to images, digiKam cannot write these tags directly into the video metadata. Instead, video tags are stored in the digiKam database on your hard disk. So camtrapR has to go straight to the digiKam database to read the tags directly from the there. After some internal magic, it will construct and return a HierarchicalSubject field as seen above for JPG images. camtrapR can then extract and tabulate the relevant information of images and videos alike.

That means that video tags are not extracted from the metadata, but from the digiKam database. It also means that the tags are not part of the video files and that sharing data and even copying video files on your hard disk after tagging might cause the loss of metadata, so please check carefully each step of what you are doing.

What if non-hierarchical metadata tags were assigned to images already?

If you have set of tagged images, but the tags are not structured hierarchically as explained above, but rather flat (no tag groups, only tags), they can easily be brought into the format needed by camtrapR in

digiKam. This will, however, change your original metadata structure. So to be on the safe side, take some precautions:

- make a backup of your images,
- first try it on a sub-dataset,
- and maybe even do it on another computer (digiKam's internal tag structure changes and even re-loading the original images will not restore it).

Let's assume you have species tags (e.g. "Leopard Cat") that are not hierarchically ordered. The digiKam workflow to order them hierarchically (in order to make them readable in camtrapR) is as follows:

1. Open the captions/tags window on the right side of your screen.
2. create a new tag (e.g. "Species").
3. You can manipulate the tag structure by dragging tags around. So you could then drag your species tags (e.g. "Leopard Cat") into the "Species" tag group (afterwards it should look similar to the screenshot above). But note: dragging the tags around doesn't update the image metadata yet!
4. To update the image metadata with your new tag structure (and to populate the image metadata tag "HierarchicalSubject" that camtrapR reads), select the images, click the "More" button in the bottom right corner of the screen, then "Write metadata to each file".
5. Repeat the last step for each station directory.

Afterwards, the "HierarchicalSubject" of your image metadata should contain the tag structure. You can check using the `exifTagNames` function as described in vignette 1.

This procedure also works for other tags, not only "Species".

How to import a metadata tag tree into digiKam

Consider this scenario: you created an elaborate tag tree and would like to share it with colleagues for them to tag images independently, but using the same tags that you use. Instead of making them create the tags manually, you can use a dummy image, assign *all* possible metadata tags of interest to it and share that tagged dummy image. Once imported into digiKam on another machine, that machine's digiKam tag tree will be updated automatically and will then contain all tags present in your dummy image.

Double observer identification

Double observer identification is only available with metadata tagging, not with the drag & drop method. It can be used both for species and individual identification (the latter with some limitations detailed below). Both observers need to tag images independently using separate metadata tag groups (e.g. "SpeciesID_Peter" and "SpeciesID_Paul"). but with both observers using identical tags for the same species/individual (e.g., not "Leopard Cat" and "leopard cat"). The function `checkSpeciesIdentification` then returns images that were assigned to different species/individuals by different observers, which can then be harmonised (see below). It also returns images that were tagged by one observer only.

Comparison of identification methods

Functionality	metadata tagging	drag & drop
double observer identification	yes	no
multiple species per image	yes	no[1]

Functionality	metadata tagging	drag & drop
batch ID	yes	yes
filtering by tags	yes	no
crowd-source ID[2]	yes	yes
portability and robustness	high[3]	medium[4]

[1] images need to be copied and moved to different species directories to allow for this functionality

[2] identification performed by independent observers on sub-datasets

[3] species identification tags are written into metadata permanently

[4] Species ID depends on directory structure

Species identification

Checking species names with the ITIS taxonomic database

Generally, you are free to use any species names you wish to - scientific names, common names, local names, abbreviations, codes, etc. But it may often be desirable to be consistent with species names, e.g. to ensure long-term usability of data or to facilitate data exchange between researchers. Therefore, users can check the species names they wish to use in species identification (common or scientific names) against the ITIS taxonomic database using the function `checkSpeciesNames`. The function returns matching common and scientific names to the names provided by the user, taxon ranks and authors, the ITIS taxonomic serial number (TSN) and an ITIS taxon url. Internally, the function `checkSpeciesNames` makes use of the R package `taxize`.

If multiple matches are found, a menu allows users to choose their species of interest (if argument `ask = TRUE`). By default, only accepted species names are returned (controlled by argument `accepted`). Invalid names will return NA.

```
# find data for 2 species with correctly spelled names
checkNames1 <- checkSpeciesNames (speciesNames = c("Bearded Pig", "Malayan Civet"),
                                  searchtype   = "common")
checkNames1
```

```
##      tsn     user_name     scientificName              commonName   authorship rankname
## 1 625012   Bearded Pig       Sus barbatus bearded pig/Bearded Pig Müller, 1838  Species https://www.
## 2 622004 Malayan Civet Viverra tangalunga           Malayan Civet   Gray, 1832  Species https://www.
##   taxon_status
## 1        valid
## 2        valid
```

```
# and with scientific names (note that this is for a subspecies)
checkNames2 <- checkSpeciesNames (speciesNames = "Viverra tangalunga tangalunga",
                                  searchtype   = "scientific")
checkNames2
```

```
##      tsn                      user_name                scientificName commonName authorship   rankname
## 1 726578 Viverra tangalunga tangalunga Viverra tangalunga tangalunga         NA Gray, 1832 Subspecies
##                                                                 itis_url taxon_status
## 1 https://www.itis.gov/servlet/SingleRpt/SingleRpt?search_topic=TSN&search_value=726578        valid
```

```
# an invalid name: the accepted name of the leopard cat is Prionailurus bengalensis
checkNames3 <- checkSpeciesNames (speciesNames = "Felis bengalensis",
                                  searchtype   = "scientific",
                                  accepted     = FALSE)
checkNames3
```

```
##      tsn         user_name    scientificName  commonName authorship rankname
## 1 183793 Felis bengalensis Felis bengalensis leopard cat Kerr, 1792  Species https://www.itis.gov/se
##   taxonUsageRating
## 1          invalid
```

```
# an ambiguous name name: Chevrotain (Tragulus)
checkNames4 <- checkSpeciesNames (speciesNames = "Chevrotain",
                                  searchtype   = "common")
1              # making a choice from the menu
checkNames4
```

```
##      tsn  user_name scientificName  commonName          authorship rankname
## 1 624919 Chevrotain     Tragulidae chevrotains Milne-Edwards, 1864   Family https://www.itis.gov/ser
```

Species identification via drag & drop of images

Species identification by moving images into species directories is simple. It can be done in a generic file manager (provided it shows preview images) or in the file manager that comes with image management software (e.g. FastStone Image Viewer, IrfanView or digiKam). Images must be **moved** into species directories. **Don't copy** images into species directories! All images must be placed in species directories and no image may be left outside species directories after identification.

Creating species directories

The function `createSpeciesFolders` assists with species identification by drag & drop by creating species directories automatically in all station directories. Alternatively, directories can be created manually. `createSpeciesFolders` can also remove empty species directories after identification (by setting `removeFolders = FALSE`). In doing so, it will not touch directories that contain any files.

It makes sense to create directories for all species one expects to find in the study area, and possibly also for images one wants to exclude later on, e.g. unidentifiable images, blank images or team photos.

Here is an example in which we first create station directories and the then species directories within these. Note that creating station directories in the example is for demonstration purposes only. Normally, at this point you would already have a directory structure with station directories (possibly with camera subdirectories), each containing renamed, unsorted images. Running `createStationFolders` here therefore is for demonstration only.

```
# this dummy directory will be used as inDir (containing station directories with species subdirectorie
wd_createSpeciesFoldersTest <- file.path(normalizePath(tempdir(), winslash = "/"), "createSpeciesFolder

# now first create the station directories
# (normally, you'd create species directories in the station directories that
# already contain renamed, unsorted images). Again, this is for demonstation only.
StationFolderCreate1 <- createStationFolders (inDir       = wd_createSpeciesFoldersTest,
                                              stations    = as.character(camtraps$Station),
                                              createinDir = TRUE)
```

```
## created 3 directories
```

```
StationFolderCreate1
```

```
##    station                                                                directory create
## 1 StationA C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationA    TRU
## 2 StationB C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationB    TRU
## 3 StationC C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationC    TRU
```

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

# create species subdirectories
SpeciesFolderCreate1 <- createSpeciesFolders (inDir               = wd_createSpeciesFoldersTest,
                                              species             = species,
                                              hasCameraFolders    = FALSE,
                                              removeFolders       = FALSE)
```

```
## created 6 directories
```

```
SpeciesFolderCreate1
```

```
##                                                                                directory crea
## 1 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationA/Sambar Deer    T
## 2     C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationA/Bay Cat    T
## 3 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationB/Sambar Deer    T
## 4     C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationB/Bay Cat    T
## 5 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationC/Sambar Deer    T
## 6     C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationC/Bay Cat    T
```

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

```
## deleted 6 empty directories
```

```
SpecFolderCreate2
```

```
##                                                                                directory stil
## 1 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationA/Sambar Deer
## 2     C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationA/Bay Cat
## 3 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationB/Sambar Deer
## 4     C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationB/Bay Cat
## 5 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationC/Sambar Deer
## 6     C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/createSpeciesFoldersTest/StationC/Bay Cat
```

Moving images into species directories

Now is the time to move (not copy) images of species into the prepared species directories. If images were misplaced (thereby misidentified) they can be moved to another species directory at any time. Rerun the following function and species identification will be updated in your tables (see function `recordTable` in the Data extraction vignette).

Species identification by metadata tagging

If you wish to use metadata tagging for species identification, first set up the hierarchical metadata tag structure as explained above. Also, make sure your images are in station directories like this:

inDir/Station/...

or

inDir/Station/Camera/...

(... being the renamed JPG images). Note that the directory structure is the one produced by function `imageRename`.

Now you can begin tagging your images. Contrary to species identification by drag an drop, it is not necessary to create species directories and to move images when using metadata tags for identification. Several species tags can be assigned to the same image if images recorded several species. They will be separated into separate records when you tabulate records. Once identification is complete, function `recordTable` will extract and tabulate the metadata for you (see vignette "Extracting Data from Images").

Checking species identification

Misidentified images can act as false positives and can thus have grave effects when analysing detection/non-detection data, e.g. with occupancy models. Therefore it is very important to make sure species identification is correct. Species may be misidentifed by moving images into wrong directories or assigning wrong species tags. This can be (partly) checked using function `checkSpeciesIdentification`. Within each station and for each image, it assesses whether there are images of another species taken within a given time interval. Often, it is unlikely that different species are encountered within very short time intervals at the same location. This can help to find images that were misidentified, particularly if they were part of a sequence of images. 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.

`checkSpeciesIdentification` can check on species identification for both methods of species identification: drag & drop into species directories or metadata tagging. The former requires one of the following directory structures:

inDir/Station/Species
inDir/Station/Camera/Species

The latter requires one of the following directory structures:

inDir/Station
inDir/Station/Camera

Here is an example using images identified with the drag & drop method.

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

# run check with 120 seconds (2 minutes) maximum time differnce
check.folders <- checkSpeciesIdentification(inDir              = wd_images_ID,
                                            IDfrom             = "directory",
                                            hasCameraFolders   = FALSE,
                                            maxDeltaTime       = 120)
```

```
## StationA : checking 8 images in 2 directories

## StationB : checking 23 images in 4 directories

## StationC : checking 37 images in 4 directories
```

```
check.folders
```

```
## $temporalIndependenceCheck
##     station
## 7  StationC C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_imag
## 29 StationC C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_imag
##       DateTimeOriginal
## 7  2009-04-27 00:17:00
## 29 2009-04-27 00:19:00
##
## $IDconflictCheck
## data frame with 0 columns and 0 rows
```

The function returns a list with 2 elements. The first element (`temporalIndependenceCheck`) is the check on temporal proximity of different species (the second element `IDconflictCheck` is describes in the next paragraph). In our example, two images of different species were taken within 2 minutes of one another at StationC (a chevrotain and a moon rat in our example). Now one should check the species identity of these 2 images to make sure they were identified correctly. If not, the species ID in the incorrectly identified image is corrected. The function `checkSpeciesIdentification` can be re-run as often as desired. The behaviour of the function can be further specified by arguments `excludeSpecies` and `stationsToCheck`. The first causes camtrapR to ignore certain species and the latter can be used to check only certain stations.

```
# check only station A and B (will give no results)
checkSpeciesIdentification(inDir               = wd_images_ID,
                           IDfrom              = "directory",
                           hasCameraFolders    = FALSE,
                           maxDeltaTime        = 120,
                           stationsToCheck     = c("StationA", "StationB"))
```

```
## StationA : checking 8 images in 2 directories

## StationB : checking 23 images in 4 directories

## $temporalIndependenceCheck
## data frame with 0 columns and 0 rows
##
## $IDconflictCheck
## data frame with 0 columns and 0 rows
```

```
# Exclude chevrotains (Tragulus spp).  will give no results
checkSpeciesIdentification(inDir               = wd_images_ID,
                           IDfrom              = "directory",
                           hasCameraFolders    = FALSE,
                           maxDeltaTime        = 120,
                           excludeSpecies      = "TRA")
```

```
## StationA : checking 8 images in 2 directories
## StationB : checking 23 images in 4 directories

## StationC : checking 20 images in 3 directories

## $temporalIndependenceCheck
## data frame with 0 columns and 0 rows
##
## $IDconflictCheck
## data frame with 0 columns and 0 rows
```

Comparing double observer identification

The second element returned by `checkSpeciesIdentification` (`IDconflictCheck`) reports on images with conflicting IDs if two observers identified images via metadata tagging. To that end, the arguments `metadataSpeciesTag` and `metadataSpeciesTagToCompare` need to be set. All records that differ in their assigned IDs will be reported (also if one of them is empty). If `metadataSpeciesTagToCompare` is not defined, the data frame will be empty.

Appending species names to image files

Once species are identified correctly, species names can be appended to image file names using `appendSpeciesNames`. The species names are taken from the directory names (of the drag and drop method was used) or from metadata tags and will be appended to the image file names. Set argument `IDfrom = "directory"` or `IDfrom = "metadata"`, respectively. A data frame will show directories, old and new file names.

```
# 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 = normalizePath(tempdir(), winslash = "/") , recursive = TRUE)
```

```
## [1] TRUE
```

```
wd_images_species_copy <- file.path(normalizePath(tempdir(), winslash = "/"), "sample_images_species_di


species_names_append <- appendSpeciesNames(inDir              = wd_images_species_copy,
                                           IDfrom             = "directory",
                                           hasCameraFolders   = FALSE
)
```

```
## renamed 68 out of 68 images in C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species
```

```
head(species_names_append)
```

```
##                                                                    directory
## 1 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 2 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 3 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 4 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
```

```
## 5 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 6 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
##   renamed
## 1    TRUE
## 2    TRUE
## 3    TRUE
## 4    TRUE
## 5    TRUE
## 6    TRUE
```

```
species_names_remove <- appendSpeciesNames(inDir              = wd_images_species_copy,
                                           IDfrom             = "directory",
                                           hasCameraFolders   = FALSE,
                                           removeNames        = TRUE
)
```

```
## renamed 68 out of 68 images in C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species
```

```
head(species_names_remove)
```

```
##                                                                                    directory
## 1 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 2 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 3 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 4 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 5 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
## 6 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sample_images_species_dir/StationA/PBE StationA__
##   renamed
## 1    TRUE
## 2    TRUE
## 3    TRUE
## 4    TRUE
## 5    TRUE
## 6    TRUE
```

Collecting all images of a species

The function `getSpeciesImages` collects all images of a focal species from all stations and saves them in another directory, creating an image report. This should be done after all images were identified. The function serves 2 purposes:

1. it can be used to check species identification by creating a species image report (which can then be sent for expert identification and confirmation), and
2. it is a prerequisite for individual identification.

The images that are to be copied can be specified an a record table (as returned by function `recordTable`) or, alternatively, the function can scan for all images of your target species that were identified using either the drag and drop method or metadata tagging for species identification. Please see the function help file for more detail.

```r
# again, we use a temporary directory for demonstration. Change this in your own code!
wd_images_species_copy <- file.path(normalizePath(tempdir(), winslash = "/"), "sampleSpeciesImages")

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

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

```
## VTA  - copied 5 out of 5 images in C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtra
```

```r
specImagecopy
```

```
##   species
## 1     VTA C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_images
## 2     VTA C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_images
## 3     VTA C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_images
## 4     VTA C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_images
## 5     VTA C:/Users/niedballa/AppData/Local/Programs/R/R-4.3.2/library/camtrapR/pictures/sample_images
##                                                                 DirectoryCopy
## 1 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sampleSpeciesImages/VTA StationA__2009-04-10__05-
## 2 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sampleSpeciesImages/VTA StationA__2009-05-06__19-
## 3 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sampleSpeciesImages/VTA StationB__2009-04-15__19-
## 4 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sampleSpeciesImages/VTA StationB__2009-04-27__05-
## 5 C:/Users/niedballa/AppData/Local/Temp/RtmpYHf97f/sampleSpeciesImages/VTA StationC__2009-04-24__02-
```

Individual identification

Before individual identification, images need to be identified to species level using one of the methods described above. All images of the focal species are then collected using `getSpeciesImages` (you could also do it manually, but it is laborious and cumbersome). Users can choose if they need station directories or not. Note that it is not possible (but also not needed) to have camera subdirectories in station directories. Thus, directory structure is one of these:

speciesImages/SpeciesA/...

or

speciesImages/SpeciesA/Station1/...

where ... stands for the renamed JPG images.

As with species identification, individuals can be identified by drag and drop or by metadata tagging. We recommend identifying individuals using metadata tags in image management software because is more convenient.

Requirements

Identifying individuals from images is similar to species identification, but there are important differences: In order to use the function `recordTableIndividual` which extracts the information from tagged images, images must have been copied into a species directory (e.g. with `getSpeciesImages`). It is further recommended to use `imageRename` for renaming images earlier in the workflow (if images were not renamed, make sure you create station directories when collecting the species images with `getSpeciesImages`.

This will ensure that the function can find all the relevant information:

- species names are read from the directory name
- station ID: read either from the directory name or file name (depending on argument `hasStationFolders`)
- camera ID: read from filename (depending on argument `cameraID`). This is optional.
- individual ID: read from metadata tags or individual directory names (depending on argument `IDfrom`)

Individual identification by drag & drop

Individuals can be identified by moving images into individual directories, either within station directories or not. If no station directories are used, the images must have been renames with `imageRename` because otherwise there will be no way of deriving the station ID.

Individual identification by metadata tagging

Metadata tagging of individual IDs is analogous to species assignment with metadata.

Double observer identification of individuals

Despite its name, the function `checkSpeciesIdentification`can also be used to compare individual IDs assigned via metadata by multiple observers (with some limitations still). The idea is the same as with comparing double-observer species identifcation. In other words, the function treats individual IDs as if they were species IDs. Images of a species need to be stored in a species directory containing only images of that species (see function `getSpeciesImages`), preferably in station subdirectories. If images were not stored in station directories, the function will not be able to tell you which station the images were taken at (but you still have unique file names), do be careful when interpreting the first part of the output (the check for temporal independence).

For lack of an example dataset in the package, here's two theoretical code examples which assumes tagged images of only one species in this directory "C:/Users/Peter/individualID/SpeciesA"", either with or without station subdirectories

```
# all images in one directory, no station subdirectories
# column "station" in outtable will be uninformative
ID_check1 <- checkSpeciesIdentification(inDir    = "C:/Users/Peter/individualID",
                                        stationsToCheck   = "SpeciesA",         # no station subdirecto
                                        IDfrom            = "metadata",
                                        hasCameraFolders  = FALSE,
                                        metadataSpeciesTag          = "Example_Species_ID_Paul",
                                        metadataSpeciesTagToCompare = "Example_Species_ID_Peter",
                                        maxDeltaTime        = 60)

# check results with either of the following two:
# ID_check1[[2]]
```

```r
# ID_check1$IDconflictCheck

# all images in station subdirectories
ID_check2 <- checkSpeciesIdentification(inDir    = "C:/Users/Peter/individualID/SpeciesA",  # with stat
                                        IDfrom   = "metadata",
                                        hasCameraFolders    = FALSE,
                                        metadataSpeciesTag  = "Example_Species_ID_Paul",
                                        metadataSpeciesTagToCompare = "Example_Species_ID_Peter",
                                        maxDeltaTime        = 60)
```