Package 'callsync'

May 24, 2024

Type Package

Title Recording Synchronisation, Call Detection and Assignment, Audio Analysis

Version 0.2.3

Description Intended to analyse recordings from multiple microphones (e.g., backpack microphones in captive setting). It allows users to align recordings even if there is non-linear drift of several minutes between them. A call detection and assignment pipeline can be used to find vocalisations and assign them to the vocalising individuals (even if the vocalisation is picked up on multiple microphones). The tracing and measurement functions allow for detailed analysis of the vocalisations and filtering of noise. Finally, the package includes a function to run spectrographic cross correlation, which can be used to compare vocalisations. It also includes multiple other functions related to analysis of vocal behaviour.

License GPL-3

URL https://github.com/simeonqs/callsync

BugReports https://github.com/simeonqs/callsync/issues

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.10), oce (>= 1.7), seewave (>= 2.2.0), signal (>= 0.7), stringr (>= 1.4.1), tuneR (>= 1.4.0), scales (>= 1.2.1)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.3.1

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Simeon Q. Smeele [cre, aut], Stephen A. Tyndel [ctb]

Maintainer Simeon Q. Smeele <simeonqs@hotmail.com>

Repository CRAN

Date/Publication 2024-05-24 10:20:03 UTC

R topics documented:

`align` *align*

Description

Aligns multiple recordings (.wav files). It assumes all microphones are within recording range of each other most of the time.

Usage

```
align(
  chunk_size = 15,
  step_size = 0.5,
  all_files = NULL,
  path_recordings = NULL,
  path_chunks = NULL,
  chunk_seq = NULL,
  keys_id = NULL,
  keys_rec = NULL,
  blank = 15,
```

```
  wing = 10,
  ffilter_from = NULL,
  down_sample = NULL,
  save_pdf = FALSE,
  save_log = FALSE,
  quiet = FALSE
)
```

Arguments

chunk_size	numeric, duration in minutes of the chunks to output. Default is '15'.
step_size	numeric, duration in seconds of the bins for signal compression before cross correlation. Default is '0.5'.
all_files	character vector, paths to all raw recordings to consider. If 'NULL' files are listed based on the argument 'path_recordings'.
path_recordings	character, the path where the raw recordings are stored. Can be nested in folders, in this case provide the top-level folder.
path_chunks	character, the path where aligned chunks should be stored.
chunk_seq	numeric vector or 'NULL'. If supplied only these chunks are rerun.
keys_id	character vector of length 2. The characters before and after the unique ID of the individual or microphone. This can be in the file name or in the folder structure. E.g., if the path to the recording is '../data/week_1/recording_mic1.wav' the keys would be 'c('recording_', '.wav')' and the function would retrieve 'mic1' as individual id.
keys_rec	character vector of length 2. The characters before and after the unique ID of the recording. This can be in the file name or in the folder structure. E.g., if the path to the recording is '../data/week_1/recording_mic1.wav' the keys would be 'c('data/', '/recording')' and the function would retrieve 'week_1' as recording id.
blank	numeric, the duration in minutes to be discarded at the beginning and end of the recording.
wing	numeric, the duration in minutes to load before and after each chunk to improve alignment. This is not saved with the aligned chunk.
ffilter_from	numeric, frequency in Hz for the high-pass filter.
down_sample	numeric, the sample rate for down-sampling. If 'NULL' no down-sampling is done.
save_pdf	logical, if 'TRUE' a pdf is saved with a page per chunk that shows all the aligned recordings.
save_log	logical, if 'TRUE' a csv file with all alignment times is saved in path_chunks.
quiet	logical, if 'TRUE' no messages are printed.

Details

There are two ways to tell the function where the files are. You can either compile a character vector of pathnames yourself and enter this under ‘all_files‘ or you can give a single character path to ‘path_recordings‘. You need to make sure that there is an identifier by which to group the recordings and an identifier for each individual or microphone in the path. This can either be a in the folder structure or in the file names themselves. The align function will align all individuals per recording id (e.g., date). These identifiers are found using regexp, so mostly you can use the first few characters before and after them (see examples in the argument descriptions). The function loads chunks of the recordings, sums the absolute amplitude per bin and runs cross correlation to find alignment relative to the first recording. The aligned chunks are then saved.

Value

saves all the aligned chunks in the location specific by ‘path_chunks‘.

Examples

```
## Not run:
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/chunk@1@1@1@1.wav'
file_2 = '/chunk@2@1@1@1.wav'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
all_files = c(local_file_1, local_file_2)
a = align(chunk_size = 2,
          step_size = 0.1,
          all_files = all_files,
          keys_id = c('c', '@'),
          keys_rec = c('c', '@'),
          blank = 0,
          wing = 0,
          quiet = TRUE)

## End(Not run)
```

---

better.spectro *better.spectro*

Description

Creates a spectrogram and plots it to the current window.

Usage

```
better.spectro(
  wave,
  main = "",
  wl = 512,
  ovl = wl/2,
  xlim = NULL,
  ylim = NULL,
  mar = rep(3, 4),
  cex.main = 1,
  cex.axis = 0.75,
  cex.lab = 0.5
)
```

Arguments

wave	wave object, e.g., from ‘load.wave‘ or ‘readWave‘.
main	character, title for the spectrogram. Default is no title.
wl	numeric, window length in samples. Default is ‘512‘.
ovl	numeric, overlap in samples. Default is ‘wl/2‘.
xlim	numeric vector of length 2, limits for the x-axis. Default is no limits.
ylim	numeric vector of length 2, limits for the y-axis. Default is no limits.
mar	numeric vector of length 4, the margins of the plot for the ‘impagep‘ function. Default is ‘rep(3, 4)‘.
cex.main	numeric the relative size of the title
cex.axis	numeric the relative size of the axis labels.
cex.lab	numeric the relative size of the axis titles

Value

Plots the spectrogram to current window.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
```

```
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
better.spectro(wave)
```

---

calc.am *calc.am*

Description

Calculates the amplitude modulation for a wave object and returns several measurements in a dataframe.

Usage

```
calc.am(wave, msmooth = c(1000, 90))
```

Arguments

wave	wave object, e.g., from 'load.wave' or 'readWave'.
msmooth	used as argument for the 'seewave::env' function. *A vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %).* Default is 'c(500, 95)'.

Value

Returns a data frame with nr_notes = total number of amplitude modulations in the signal, amp_mod_med = median difference between highest and lowest amplitude from the normalised envelope, internote_med = median internote distance in seconds.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
result = calc.am(wave)
```

calc.fm *calc.fm*

Description

Calculates the frequency modulation for a wave object and returns several measurements in a data frame.

Usage

```
calc.fm(trace, min_height = 8, plot_it = FALSE)
```

Arguments

trace	numeric vector, e.g., the fundamental frequency from 'trace.fund', NOTE this would be 'trace$fund' and not the whole 'trace' object.
min_height	the minimum difference between a bottom and a peak for an infliction point to be accepted.
plot_it	logical, if 'TRUE' plot the trace and peaks to current window. Default is 'FALSE'.

Value

Returns a data frame with fm = median difference between peaks and bottoms and ipi = inter peak, np = number of peaks. interval (s).

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
trace = trace.fund(wave)
result = calc.fm(trace$fund)
```

calc.perf *calc.perf*

Description

Calculates the performance of the detections. Detections are true positive if they overlap to any extend with a ground truth selection.

Usage

```
calc.perf(d, gt)
```

Arguments

d	data frame, detection selection table with start = start time in seconds, end = end time in seconds and file = file name
gt	data frame, ground truth selection table with start = start time in seconds, end = end time in seconds and file = file name

Value

Returns a named list with tp = the row numbers (in d) for the true positives, fp = the row numbers (in d) for the false positives, fn = the row numbers (in gt) for the false negatives, fp_rate = ‘length(fp)/nrow(d)‘, tp_rate = ‘length(tp)/nrow(gt)‘, fn_rate = ‘length(fn)/nrow(gt)‘.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/audacity/chunk_15_ground_truth.txt'
url_1 = paste0(path_git, path_repo, file_1)
local_dir = paste(tempdir(), 'audacity', sep = '/')
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!dir.exists(local_dir)) dir.create(local_dir)
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
d = load.selection.tables.audacity(path_selection_tables = local_dir)
result = calc.perf(d, d)
```

call.assign *call.assign*

Description

Assigns calls from a detection table. Or rather removes calls that are not the loudest and returns the cleaned detection table. Uses fine alignment and energy content.

Usage

```
call.assign(
  all_files = NULL,
  detections = NULL,
  save_files = TRUE,
  path_calls = NULL,
  ffilter_from = 1100,
  wing = 5,
  step_size = 0.01,
  assign_fraq = 0.05,
  save_wing = 0.1,
  quiet = FALSE
)
```

Arguments

all_files	character vector, should contain all the paths to the raw recordings that should be considered. If 'NULL' files are loaded from 'path_chunks'.
detections	data frame with start = start time in samples and end = end time in samples for each detection.
save_files	logical, if 'TRUE' the files are stored in the 'path_chunks' location. Results are also returned.
path_calls	character, path to where to store the results.
ffilter_from	numeric, frequency in Hz for the high-pass filter.
wing	numeric, the duration in seconds to load before and after each detection to improve alignment. This is not saved with the aligned call.
step_size	numeric, duration in seconds of the bins for signal compression before cross correlation. Default is '0.01'.
assign_fraq	numeric between 0 and 1, how much louder does the focal needs to be than the second loudest track to be accepted. Default is '0.05' and accepts if the focal is just 0.05 louder.
save_wing	numeric, how much extra to export before and after a detection to make sure the whole call is included in seconds. Default is '0.1'.
quiet	logical, if 'TRUE' no messages are printed.

Value

Returns a data frame with file = file name, start = start time in samples and end = end time in samples for each detection.

Examples

```
## Not run:
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/chunk@1@1@1@1.wav'
file_2 = '/chunk@2@1@1@1.wav'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
all_files = c(local_file_1, local_file_2)
detections = lapply(all_files, function(file){
  wave = load.wave(file, ffilter_from = 1100)
  detections = call.detect.multiple(wave, plot_it = FALSE)
  return(detections)
})
names(detections) = basename(all_files)
ca = call.assign(all_files = all_files,
                 detections = detections,
                 quiet = TRUE,
                 save_files = FALSE)

## End(Not run)
```

`call.detect` *call.detect*

Description

Detects single call in a wave object using an amplitude envelope.

Usage

```
call.detect(wave, threshold = 0.3, msmooth = c(500, 95), plot_it = FALSE)
```

Arguments

wave	wave object, e.g., from ‘load.wave‘ or ‘readWave‘.
threshold	rector of length 1 or 2. The fraction of the maximum of the normalised envelope to use as threshold to detect start and end. If a vector of length 2 is supplied, the first is used to detect the start and the second to detect the end (in case of echo).
msmooth	used as argument for the ‘seewave::env‘ function. *A vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %).* Default is ‘c(500, 95)‘.
plot_it	if ‘TRUE‘, returns three-panel plot of wave form, envelope and spectrogram to current plotting window. Default is ‘FALSE‘.

Value

Returns a dataframe with start = start time in samples and end = end time in samples for each detection. Optionally also plots the wave form and detections to current window.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
cd = call.detect(wave)
```

---

call.detect.multiple *call.detect.multiple*

Description

Detects multiple calls in a wave object using an amplitude envelope.

Usage

```
call.detect.multiple(
  wave,
  threshold = 0.3,
  msmooth = c(500, 95),
  plot_it = FALSE,
```

```
  min_dur = 0.1,
  max_dur = 0.3,
  save_extra = 0,
  env_type = "Hilbert",
  bin_depth = 512,
  merge_overlap = FALSE
)
```

Arguments

wave	wave object, e.g., from 'load.wave' or 'readWave'.
threshold	rector of length 1 or 2. The fraction of the maximum of the normalised envelope to use as threshold to detect start and end. If a vector of length 2 is supplied, the first is used to detect the start and the second to detect the end (in case of echo).
msmooth	used as argument for the 'seewave::env' function. *A vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %).* Default is 'c(500, 95)'.
plot_it	logical, if 'TRUE', returns three-panel plot of wave form, envelope and spectrogram to current plotting window. Default is 'FALSE'.
min_dur	numeric, the minimal duration in seconds for a detection to be saved. Default is '0.1'.
max_dur	numeric, the maximal duration in seconds for a detection to be saved. Default is '0.3'.
save_extra	numeric, how much to add to start and end time in seconds. Can be used to make sure the whole vocalisation is included.
env_type	character, what type of envelope to calculate. If 'Hilbert' returns the modulus (Mod) of the analytical signal of wave obtained through the Hilbert transform (hilbert) using seewave::env. If 'summed' returns the summed absolute amplitude. Default is 'Hilbert'.
bin_depth	numeric, how many samples to sum if env_type is 'summed'. Default is '512'.
merge_overlap	logical, if 'TRUE' overlapping detections (due to 'save_extra') are merged. Default is 'FALSE'.

Value

Returns a data frame with start = start time in samples and end = end time in samples for each detection. Optionally also plots the wave form and detections to current window.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
```

```
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
cd = call.detect.multiple(wave)
```

`callsync` *callsync: sychronous analysis of multiple microphones*

Description

Intended to analyse recordings from multiple microphones (e.g., backpack microphones in captive setting). It allows users to align recordings even if there is non-linear drift of several minutes between them. A call detection and assignment pipeline can be used to find vocalisations and assign them to the vocalising individuals (even if the vocalisation is picked up on multiple microphones). The tracing and measurement functions allow for detailed analysis of the vocalisations and filtering of noise. Finally, the package includes a function to run spectrographic cross correlation, which can be used to compare vocalisations. It also includes multiple other functions related to analysis of vocal behaviour.

Details

The main features of the package are:

- alignment and partitioning of drifting microphones using signal compression and cross correlation
- call detection using an amplitude envelope
- fine-scale alignment and call assignment across recordings using cross correlation and energy content
- fundamental frequency tracing
- analysis of the resulting traces and wav clips

The package offers functions to:

- create flexible spectrograms
- run spectrographic cross correlation
- analyse amplitude and frequency modulation
- load selection tables from Raven and Audacity
- calculate performance of the detection and assignment functions

License: GNU (>= 2)

Authors: Simeon Q. Smeele and Stephen A. Tyndel

Maintainer: Simeon Q. Smeele, <simeonqs@hotmail.com>

`create.spec.object` *create.spec.object*

Description

Creates a tailored spectrogram (matrix) for spectrographic cross correlation.

Usage

```
create.spec.object(
  wave,
  wl = 512,
  ovl = 450,
  freq_range = c(0, 20000),
  plot_it = TRUE,
  thr_low = 1.5,
  thr_high = 3,
  sum_one = FALSE,
  method = "sd"
)
```

Arguments

`wave`	wave object, e.g., from 'load.wave' or 'readWave'.
`wl`	numeric, window length in samples. Default is '512'.
`ovl`	numeric, overlap in samples. Default is '450'.
`freq_range`	numeric vector of length 2, the frequency range in Hz to return.
`plot_it`	logical, if 'TRUE', returns three-panel plot of wave form, envelope and spectrogram to current plotting window. Default is 'FALSE'.
`thr_low`	numeric, the lower range (see 'method'). Pixels with lower values are set to 0 for noise reduction.
`thr_high`	numeric, the upper range (see 'method'). Pixels with higher values are set to 'thr_high'.
`sum_one`	logical, if 'TRUE' pixels are divided by the sum of all pixels, such that they sum to one.
`method`	character, either 'sd' or 'max'. If 'sd', pixels are standardised. If 'max', pixels are normalised.

Value

Returns a numeric matrix with the spectrogram values.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
result = create.spec.object(wave, plot_it = FALSE)
```

`detect.and.assign` *detect.and.assign*

Description

Traces the fundamental frequency from a wave object. Also applies smoothening to trace.

Usage

```
detect.and.assign(
  all_files = NULL,
  path_chunks = NULL,
  path_calls = NULL,
  ffilter_from = 1100,
  threshold = 0.4,
  msmooth = c(1000, 95),
  min_dur = 0.1,
  max_dur = 0.3,
  step_size = 0.01,
  wing = 6,
  save_files = TRUE,
  quiet = FALSE,
  save_extra = 0
)
```

Arguments

`all_files`	character vector or 'NULL'. Character vector should contain all the paths to the raw recordings that should be considered. If 'NULL' files are loaded from 'path_chunks'.
`path_chunks`	character, path to where the chunks are stored.
`path_calls`	character, path to where to store the results.

ffilter_from	numeric, frequency in Hz for the high-pass filter.
threshold	numeric, threshold (fraction of the maximum) for amplitude envelope when detecting call.
msmooth	used as argument for the 'seewave::env' function. *A vector of length 2 to smooth the amplitude envelope with a mean sliding window. The first component is the window length (in number of points). The second component is the overlap between successive windows (in %).* Default is 'c(500, 95)'.
min_dur	numeric, the minimal duration in seconds for a detection to be saved. Default is '0.1'.
max_dur	numeric, the maximal duration in seconds for a detection to be saved. Default is '0.3'.
step_size	numeric, duration in seconds of the bins for signal compression before cross correlation. Default is '0.01'.
wing	numeric, the duration in seconds to load before and after each detection to improve alignment. This is not saved with the aligned call.
save_files	logical, if 'TRUE' the files are stored in the 'path_chunks' location. Results are also returned.
quiet	logical, if 'TRUE' no messages are printed.
save_extra	numeric, how much to add to start and end time in seconds. Can be used to make sure the whole vocalisation is included.

Value

Returns a data frame with start = start time in samples and end = end time in samples for each detection.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/chunk@1@1@1@1.wav'
file_2 = '/chunk@2@1@1@1.wav'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
all_files = c(local_file_1, local_file_2)
## Not run:
result = detect.and.assign(all_files = all_files,
                           quiet = TRUE,
                           save_files = FALSE)
```

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

`export.detections` *export.detections*

Description

Exports detection table from ‘call.detect‘ into a txt file that can be read by Raven Lite. All columns other than those containing start and end times are filled with 0 or ”.

Usage

```
export.detections(detections, sr = 1, path_out = "out.txt")
```

Arguments

`detections`	data.frame, the object generated by ‘call.detect‘.
`sr`	numeric, the sampling rate of the wave on which detections were run. Default to ‘1‘, which allows users to transform the start and end times before feeding the data.frame to this function.
`path_out`	character, the path including file name where to store the txt file. Default is ‘’out.txt’‘.

Value

Stores a Raven Lite readable selection table.

`load.selection.table` *load.selection.table*

Description

Loads single Raven selection table into a dataframe.

Usage

```
load.selection.table(path_selection_table)
```

Arguments

`path_selection_table`	the path to the file containing the selection table.

Value

Returns data frame with all selections.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/2020_10_27_091634.Table.1.selections.txt'
file_2 = '/2020_10_27_132148.Table.1.selections.txt'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
st = load.selection.tables(path_selection_tables = tempdir())
```

---

load.selection.tables *load.selection.tables*

Description

Loads multiple Raven selection tables into one dataframe. Also adds a column with file-selection

Usage

```
load.selection.tables(path_selection_tables, recursive = FALSE)
```

Arguments

path_selection_tables	the path to the folder containing selection tables. Folder should not contain any other files.
recursive	if 'TRUE' lists files recursively before loading, default is 'FALSE'.

Value

Returns data frame with all selection tables.

load.selection.tables.audacity *load.selection.tables.audacity*

Description

Loads multiple Audacity selection tables into one data frame.

Usage

```
load.selection.tables.audacity(path_selection_tables)
```

Arguments

path_selection_tables
: character, the path to the folder containing selection tables. Folder should not contain any other txt files.

Value

Returns data frame with all selection tables.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/audacity/chunk_15_ground_truth.txt'
url_1 = paste0(path_git, path_repo, file_1)
local_dir = paste(tempdir(), 'audacity', sep = '/')
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!dir.exists(local_dir)) dir.create(local_dir)
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
st = load.selection.tables.audacity(path_selection_tables = local_dir)
```

load.wave *load.wave*

Description

Wrapper function for ‘readWave‘ from *tuneR*. Also optionally applies ‘ffilter‘ from *seewave*.

Usage

```
load.wave(path_audio_file, from = 0, to = Inf, ffilter_from = NULL)
```

Arguments

path_audio_file	the path to the .wav file
from	time in seconds from where to start the loading of the audio file. Default is '0' which loads the whole file.
to	time in seconds until where to load the audio file. Default is 'Inf' which loads the whole file.
ffilter_from	numeric, frequency in Hz for the high-pass filter. Default is 'NULL', which does not apply a filter.

Value

Returns an R wave object.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = load.wave(local_file_1)
```

---

measure.trace *measure.trace*

Description

Takes several measurements on a fundamental frequency trace.

Usage

```
measure.trace(trace, sr = 44100, hop = 5)
```

Arguments

trace	data frame, e.g., the output of the 'trace.fund' function. Should contain columns with time = time in seconds, fund = fundamental frequency in Hz and missing = logical indicating if the fundamental was detected ('TRUE') or interpolated ('FALSE').
sr	sample rate of the wave object used for 'trace.fund'.
hop	the 'hop' parameter used to generate the trace.

Value

Returns a dataframe with all measurements.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
trace = trace.fund(wave)
result = measure.trace(trace)
```

---

measure.trace.multiple

measure.trace.multiple

Description

Takes several measurements on multiple fundamental frequency traces.

Usage

```
measure.trace.multiple(
  traces,
  new_waves = NULL,
  waves = NULL,
  detections = NULL,
  sr = NULL,
  path_pdf = NULL
)
```

Arguments

traces	a list of data frames, e.g., the output of the 'trace.fund' function. Should contain columns with time = time in seconds, fund = fundamental frequency in Hz and missing = logical indicating if the fundamental was detected ('TRUE') or interpolated ('FALSE'). If the list is named the names will be used as file names in the output.
new_waves	a list of wave objects, should only contain the call.
waves	a list of wave objects, should not be resized.
detections	the detections.
sr	numeric, sample rate of the waves objects used for the traces. Only needed if 'waves' is 'NULL'.
path_pdf	numeric or 'NULL', where to store the pdf. If 'NULL' no pdf is stored.

Value

Returns a data frame with all measurements.

Examples

```
## Not run:
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
file_2 = '/wave_2.wav'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
all_files = c(local_file_1, local_file_2)
waves = lapply(all_files, load.wave)
new_waves = waves
detections = lapply(waves, call.detect)
traces = lapply(waves, trace.fund)
mt = measure.trace.multiple(traces = traces, waves = waves,
                            new_waves = new_waves, detections = detections)

## End(Not run)
```

`o.to.m` *o.to.m*

Description

Transforms a vector into a matrix where it assumes that the vector values are the lower triangular of the matrix: 'm[lower.tri(m)] = o'. It includes 0 on the diagonal.

Usage

```
o.to.m(o, n = seq(sqrt(length(o) + 1) + 1))
```

Arguments

`o`	the vector containing the values for the lower triangular (required)
`n`	the names for the rows and columns of the matrix (optional)

Value

Returns a matrix where it assumes that 'm[lower.tri(m)] = o'.

Examples

```
m = matrix(1:9, nrow = 3, ncol = 3)
o = m[lower.tri(m)]
m_new = o.to.m(o)
```

`run.spcc` *run.spcc*

Description

Runs spectrographic cross correlation on multiple wave objects.

Usage

```
run.spcc(
  waves,
  freq_range = c(700, 3500),
  thr_low = 0.45,
  thr_high = 0.6,
  wl = 256,
  ovl = 250,
  method = "sd",
  sum_one = TRUE,
  mc.cores = 1,
  step_size = 10
)
```

Arguments

waves	a list of wave objects, e.g., from ‘lapply‘ in combination with ‘load.wave‘ or ‘readWave‘.
freq_range	numeric vector of length 2, the frequency range in Hz to return.
thr_low	numeric, the lower range (see ‘method‘). Pixels with lower values are set to 0 for noise reduction.
thr_high	numeric, the upper range (see ‘method‘). Pixels with higher values are set to ‘thr_high‘.
wl	numeric, window length in samples. Default is ‘512‘.
ovl	numeric, overlap in samples. Default is ‘450‘.
method	character, either ‘sd‘ or ‘max‘. If ‘sd‘, pixels are standardised. If ‘max‘, pixels are normalised.
sum_one	logical, if ‘TRUE‘ pixels are divided by the sum of all pixels, such that they sum to one.
mc.cores	numeric, how many threads to run in parallel. For Windows only one can be used.
step_size	numeric, argument for ‘sliding.pixel.comparison‘ how many pixels should be moved for each step. Default is ‘10‘.

Value

Matrix with row and columns names equal to the names of the wave list. Diagonal is zeroes. Other values are the normalised pairwise distances from ‘sliding.pixel.comparison‘.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
file_2 = '/wave_2.wav'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
all_files = c(local_file_1, local_file_2)
waves = lapply(all_files, load.wave)
spcc_out = run.spcc(waves)
```

simple.cc *simple.cc*

Description

Simple cross correlation of two vectors. Uses zero embedding to find optimal overlap. Also has an option to normalise by the longest vector (divides final difference by length). This version returns the time difference for best overlap.

Usage

```
simple.cc(s1, s2, norm = FALSE)
```

Arguments

s1	the first numeric vector (required)
s2	the second numeric vector (required)
norm	if 'TRUE' the final difference is divided by the length of the longest vector

Value

Returns an integer, which is the start of s1 relative to s2. E.g., -1 means that s1 has to be moved one step back to be aligned with s2.

Examples

```
s1 = c(0, 0, 0, 1, 1, 2, 0)
s2 = c(0, 0, 2, 2, 3, 0, 0, 0, 0)
offset = simple.cc(s1, s2) # -1
index_s1 = seq(1, length(s1)) + offset # align
plot(s2, type = 'b')
points(index_s1, s1, col = 2, type = 'b')
```

sliding.pixel.comparison *sliding.pixel.comparison*

Description

Can be used to run spectrographic cross correlation. Both spectrograms are zero-padded and slid over each other. For each step the difference is computed. The function returns the absolute difference at the point at the minimum (maximal signal overlap).

Usage

```
sliding.pixel.comparison(s1, s2, step_size = 1)
```

Arguments

s1	numeric matrix, the first spectrogram.
s2	numeric matrix, the second spectrogram.
step_size	numeric, how many pixels should be moved for each step. Default is '1'.

Value

Returns the distance at the point of maximal signal overlap.

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
file_2 = '/wave_2.wav'
url_1 = paste0(path_git, path_repo, file_1)
url_2 = paste0(path_git, path_repo, file_2)
local_file_1 = paste(tempdir(), file_1, sep = '/')
local_file_2 = paste(tempdir(), file_2, sep = '/')
if(!file.exists(local_file_1))
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
if(!file.exists(local_file_2))
  download.file(url_2, destfile = local_file_2, mode = 'wb')
wave_1 = readWave(local_file_1)
wave_2 = readWave(local_file_2)
so_1 = create.spec.object(wave = wave_1, plot_it = FALSE)
so_2 = create.spec.object(wave = wave_2, plot_it = FALSE)
out = sliding.pixel.comparison(so_1, so_2)
```

---

trace.fund *trace.fund*

Description

Traces the fundamental frequency from a wave object. Also applies smoothening to trace.

Usage

```
trace.fund(
  wave,
  hop = 5,
  wl = 200,
  freq_lim = c(1.1, 4),
  spar = 0.4,
  noise_factor = 3.5,
  thr = 0.3
)
```

Arguments

wave	wave object, e.g., from 'load.wave' or 'readWave'.
hop	integer, how many samples to skip for each trace point.
wl	integer, window length for the spectrum
freq_lim	numeric vector of length 2, frequency in kHz between which to find the fundamental
spar	numeric between 0-1, for the 'smooth.spline' function
noise_factor	numeric, how much louder the fundamental has to be than the noise to be accepted
thr	numeric between 0-1, the fraction of the maximum of the spectrum used to detect the fundamental

Details

Tracing step is based on a sliding window for which the spectrum is calculated. A threshold is based on the maximum y value and the first frequency to cross the threshold is considered the fundamental frequency. If the average hight before the fundamental is higher than 'noise_factor', the detection is discarded and NA is returned for that window. Smoothing step is based on 'smooth.spline'. Finally, all points outside 'freq_lim' are reset to these limits.

Value

Data frame with time = time in seconds, fund = fundamental frequency in Hz and missing = logical indicating if the fundamental was detected ('TRUE') or interpolated ('FALSE').

Examples

```
require(callsync)
require(seewave)
require(tuneR)
path_git = 'https://raw.githubusercontent.com'
path_repo = '/simeonqs/callsync/master/tests/testthat/files'
file_1 = '/wave_1.wav'
url_1 = paste0(path_git, path_repo, file_1)
local_file_1 = paste(tempdir(), file_1, sep = '/')
if(!file.exists(local_file_1))
```

```
  download.file(url_1, destfile = local_file_1, mode = 'wb',)
wave = readWave(local_file_1)
trace = trace.fund(wave)
```

Index