APPLICATION

Analysis of backpack microphone recordings using *callsync*

Simeon Q. Smeele [1,2,3,†,*], Stephen A. Tyndel [1,3,†], Barbara C. Klump [1], Mary Brooke McElreath [1,2], Gustavo Alarcon-Nieto [1,3] & Lucy M. Aplin [1,3,4]

[1] *Cognitive & Cultural Ecology Research Group, Max Planck Institute of Animal Behavior, Radolfzell, Germany*

[2] *Department of Human Behavior, Ecology and Culture, Max Planck Institute for Evolutionary Anthropology, Leipzig, Germany*

[3] *Department of Biology, University of Konstanz, Konstanz, Germany*

[4] *Division of Ecology and Evolution, Research School of Biology, The Australian National University, Canberra, Australia*

[†] Co-first author

[*] Correspondence author. E-mail: ssmeele@ab.mpg.de

Summary

1. To better understand how vocalisations are used during interactions of multiple individuals captive studies with microphones on the animal are often performed. The resulting recordings are challenging to analyse, since microphones drift and record the vocalisations of non-focal individuals as well as noise.
2. Here we present audioID, an R package designed to align recordings, detect and assign vocalisations, filter out noise and perform basic analysis on the resulting clips.
3. We present a case study where the pipeline is used for a new dataset of captive cockatiels. We show that xx calls can be detected and assigned across yy hours of recording. We use the resulting calls to show that individuals can be recognised based on xx, xx, xx features.
4. The audioID can be used to go from raw recordings to a cleaned dataset of features. The package is designed to be modular and allow users to replace functions as they wish. We also discuss the challenges that might be faced in each step and how the available literature can provide alternatives for each step.

Keywords: something about sound, other stuff about sound

Introduction

Case study: cockatiel contact calls

We present a case study to show how `callsync` functions can be included in a workflow (see Figure X). The study is based on a captive system with XXX [Stephen can you fill this out].

Installation and set-up

Alignment of raw recordings

Raw recordings consist of 3.5 hour long wav files for six cockatiels for each day. We included four days of data in this case study. The backpack microphones have internal clocks that automatically turn them on and off. However, these clocks drift in time both during the off period, creating start times that differ up to a few minutes, and during the recording period, creating

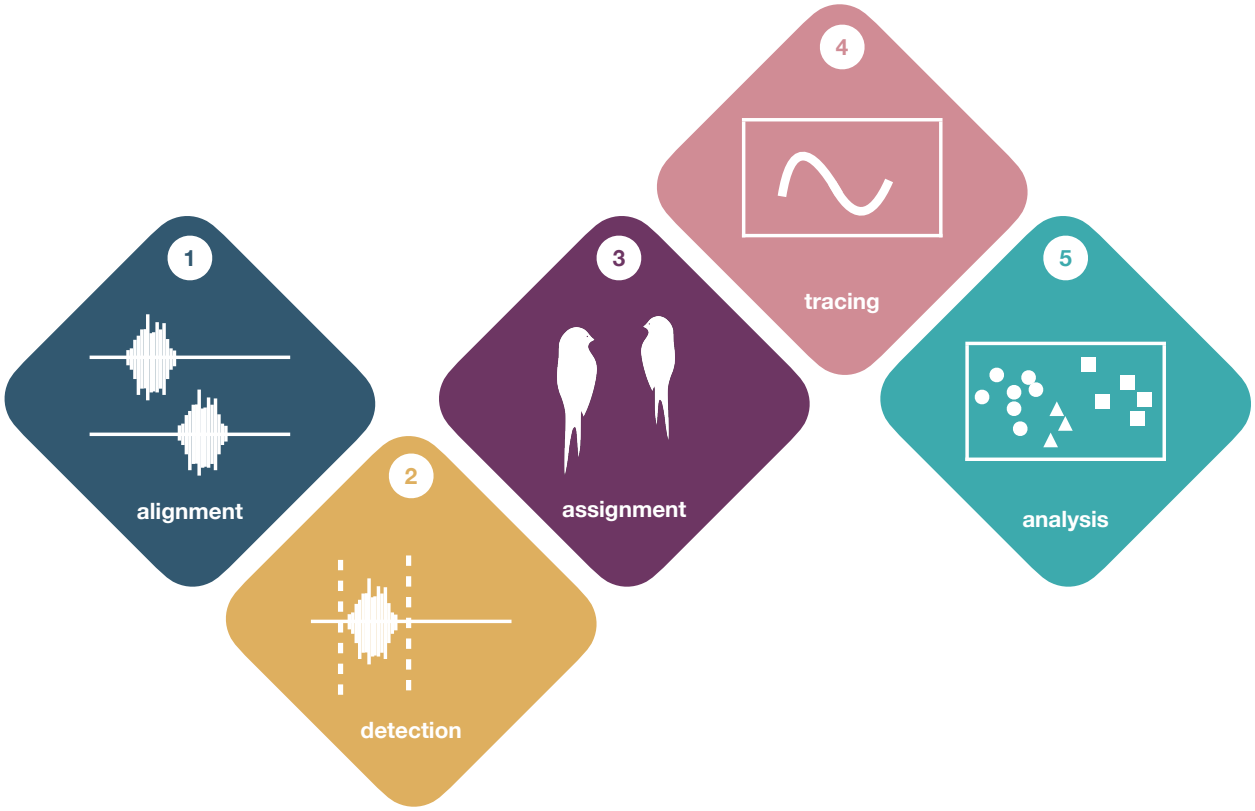

Figure 1: Flowchart from the `callsync` package. The *alignment* module can be used to align multiple microphones that have non-linear temporal drift. The *detection* module can be used to detect vocalisations in each recording. The *assignment* module can be used to assign a vocalisation to the focal individual, making sure that vocalisations from conspecifics are excluded from the focal recording. The *tracing* module can be used to trace and analyse the fundamental frequency for each vocalisation. Filters can be applied to remove false alarms in the detection module. The final *analysis* module can be used to run spectrographic cross correlation and create a feature vector to compare across recordings.

additional variable drift up to a minute between recordings. The first step is therefore to align 15 minute chunks of recording to ensure that drift is reduced to mere seconds.

The function `align` can be used for this. It splits the recordings up into shorter chunks, in our case 15 minutes. It aligns all recordings relative to one of the recordings using cross correlation on the energy content (summed absolute amplitude) per time bin, in our case 0.5 seconds.

```
align(chunk_size = 15, # minutes
      step_size = 0.5, # seconds
      path_folders = 'ANALYSIS/DATA/cockatiel_dataset',
      path_chunks = 'ANALYSIS/RESULTS/cockatiel_dataset/chunks',
      keys_rec = c('_\\(', '\\)_'),
      keys_id = c('bird_', '_tag'),
      blank = 15, # minutes
      wing = 10, # minutes
      save_pdf = TRUE)
```

For cross correlation we load the chunks with additional minutes before and after (option `wing`) to ensure that overlap can be found. The cross correlation is performed using the function `simple.cc`, which takes two vectors (the binned energy content of two recordings) and calculates the absolute difference while sliding the two vectors over each other. It returns the position of minimum summed difference, or in other words the position of maximal overlap. This position is then used to align the recordings relative to the first recording and save chunks that are maximally aligned. Note that due to drift during the recording, the start and end times might still be seconds off; it is the overall alignment of the chunk that is optimised.

The function also allows the user to create a pdf with wave forms per individual and a single page per chunk, to visually verify

if alignment was successful. To illustrate the alignment we ran the function on two minute chunks and plottet the aligned wave forms for two individuals in Figure X.

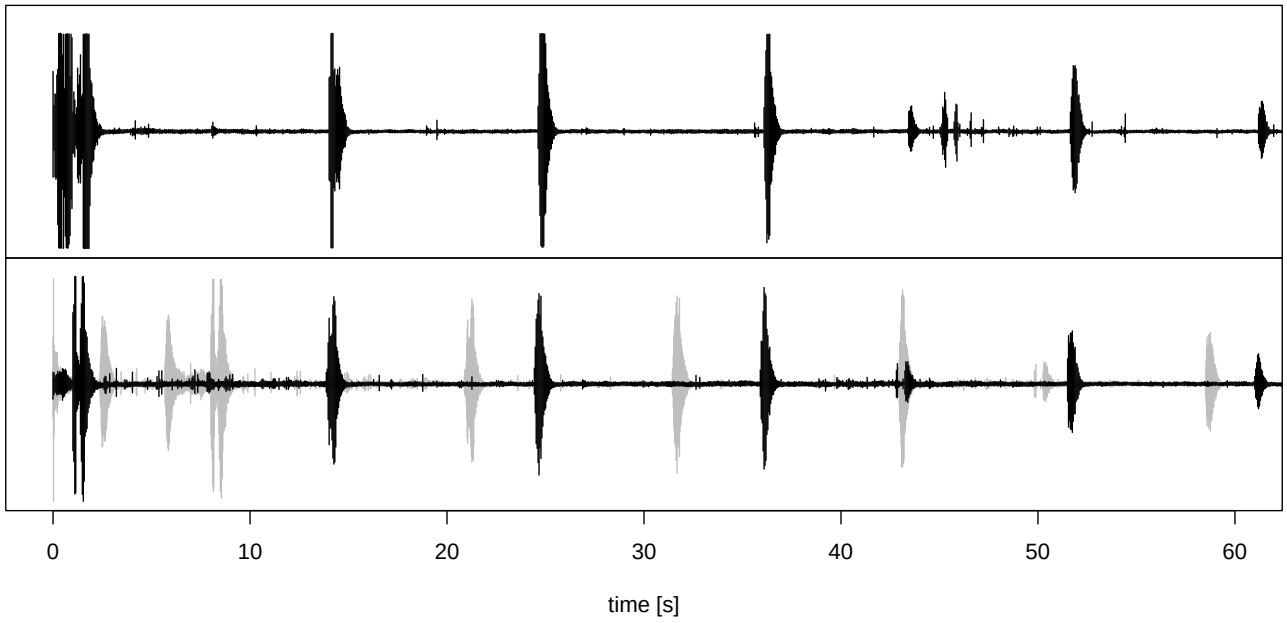

Figure 2: Example of two aligned waveforms (black). The grey waveform in the background is before alignment.

Call detection and assignment

The next step is to detect calls and assign them to the correct individual.

For detection we load the chunks using the wrapper function `load.wave` where we apply a high-pass filter from 1100 Hz. To detect calls we used the `call.detect.multiple` which can detect multiple calls in an R wave object. It first applies the `env` function from the *seewave* package with `msmooth = c(1000, 95)` create a smooth Hilbert amplitude envelope. It then detects all the points on the envelope which are above a certain threshold relative to the maximum of the envelope. After removing detections that are shorter than a set minimum duration it returns all the start and end times as a dataframe.

Because the microphones on non-focal individuals are very likely to record the calls of the vocalising individual as well, we implemented a step that assigns the detected calls to the correct individual. This step runs through all the detections in a given chunk for a given individual and runs the `call.detect` function to more precisely determine the start and end time of the call. It then aligns this call with all the recordings of all other individuals by rerunning the `simple.cc` function to ensure that minor temporal drift is corrected. After alignment it calculates the summed absolute energy content for the time frame when the call was detected on all recordings and compares this to the focal recording. If the focal recording is the loudest, the detection is saved as a separate wav file. If not, the detection is discarded.

Analysis of single calls and call comparison

To analyse the calls, the short wav clips were loaded and the `call.detect` function was rerun to determine the start and end times of the call. The wave objects were then resized to only include the call. To trace the fundamental frequency we applied the `trace.fund` function to the resized wave objects:

```
traces = mclapply(new_waves, function(new_wave)
  trace.fund(new_wave, spar = spar, freq_lim = freq_lim, thr = thr_trace, hop = hop,
             noise_factor = noise_factor), mc.cores = mc.cores)
```

We used the `mclapply` function from *parallel* to run multiple tracings in parallel. We then used `measure.trace.multiple` to take basic measurements on the resulting trace:

```
measurements = measure.trace.multiple(traces, new_waves, waves, snr = snr,
                                       path_pdf = path_pdf_traces)
```

An example of the resulting trace can be seen in Figure X.

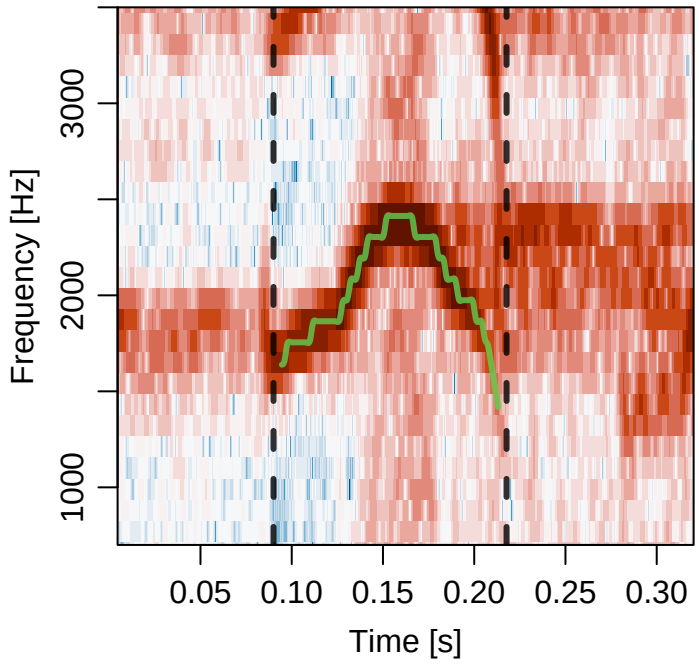

Figure 3: Spectrogram of a cockatiel call with start and end (black dashed lines) and the fundamental frequency trace (green solid line).

The call detection step also picks up on a lot of noise (birds scratching, flying, walking around) as well as calls. We therefore ran a final step to filter the measurements and traces before these were saved:

```
keep = measurements$prop_missing_trace < 0.1 &
  measurements$signal_to_noise > 5 &
  measurements$band_hz > 600 &
  measurements$duration_s > 0.10 & measurements$duration_s < 0.3
measurements = measurements[keep,]
traces = traces[keep]
```

Another way to analyse calls is to measure their similarity directly. A frequently used method is spectrographic cross correlation (SPCC), where two spectrograms are slid over each other and the pixelwise difference is computed for each step. At the point where the signals maximally overlap one will find the minimal difference. This score is than used as a measure of acoustic distance between two calls. The function `xxx` runs SPCC and includes several methods to reduce noise in the spectrogram before running cross correlation (for an example see Figure X).

To visualise the resulting of SPCC on the cockatiel calls we used principle coordinate analysis, and plotted the first two coordinates in Figure X.

Discussion

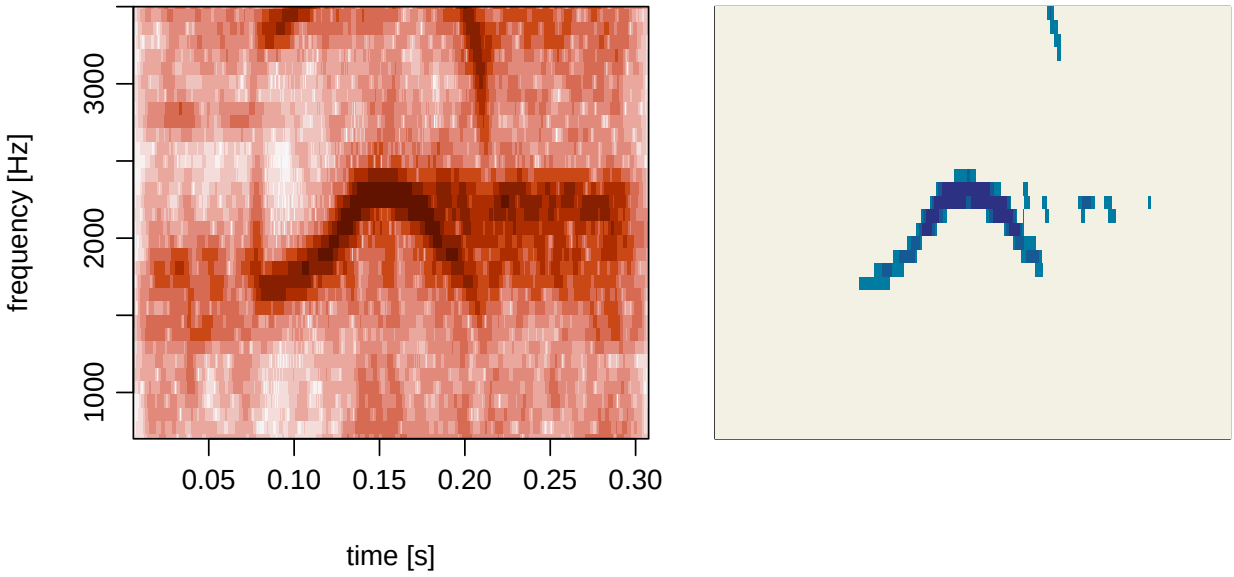

Figure 4: Example spectrogram (left) and noise reduced spectrogram (right) of a cockatiel call. Darker colours indicate higher intensity.

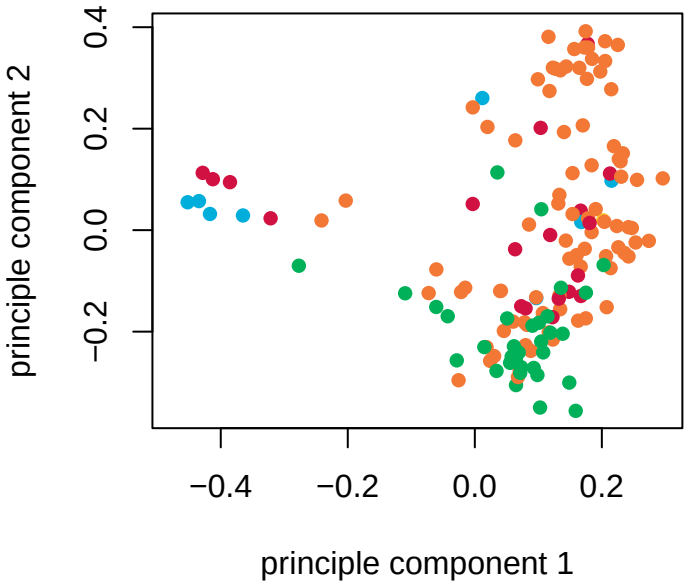

Figure 5: Call distribution in PCO space. Dots represents calls and are coloured by individual.