Call detection, assignment and analysis from backpack mics using audioID

Simeon Q. Smeele [1,2,3,*] Stephen A. Tyndel [1,3,*]

[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

*Co-first author

Keywords: something about sound, other stuff about sound

Author for correspondence: Simeon Q. Smeele, Max Planck Institute of Animal Behavior, Am Obstberg 1, 78315 Radolfzell am Bodensee, ssmeele@ab.mpg.de

Abstract

1.

2.

3.

4.

Introduction

Case study: cockatiel contact calls

We present a case study to show how audioID functions can be included in a workflow. 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. 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, creating additional 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 `coarse.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.

```
coarse.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 all start at the same time.

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.

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 mininum 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 timeframe 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 seperate wav file. If not, the detection is discarded.

Analysis of single calls and call comparison

Discussion