ambiscaper Documentation

Release 0.1.0

Andres Perez-Lopez

Sep 14, 2018

Contents

Ambiscaper: a tool for automatic dataset generation and annotation of reverberant Ambisonics audio.

CHAPTER 1

Motivation

1.1 Motivation

Due to the recent developments on the field of immersive media and virtual reality, there has been a renewed interest into Ambisonics, specially motivated by its potential to capture the spacial qualities of the sound, and the methodologies to dynamically render it to binaural.

Despite the common approach to Ambisonics recordings as "ambiences", some modern Ambisonics microphones feature dozens of capsules. Therefore, it is possible to use such microphones as beamforming devices, with an accurate spatial resolution.

As a consequence, Ambisonics recordings might be useful in the auditory scene analysis field. More specifically, the intrinsic spatial audio representation can be exploited in the Sound Source Localization and Blind Source Separation fields.

However, there is an important lack of Ambisonics recordings databases, specially in the case of Higher Order Ambisonics. Annotation is also needed to design, train and evaluate the algorithms. The related works presented in last years have used custom databases, which hinder experiment reproducibility. A flexible reverberation configuration is as well needed for the state-of-the-art methods. Manual recording and annotation of sound scenes on that scale would imply an excessive amount of work.

We present AmbiScaper, a python library for procedural creation and annotation of reverberant Ambisonics databases.

1.1.1 Acknowledgements

AmbiScaper is based on Scaper, a related work by Justin Salamon in the context of scene recognition.

The material in the present documentation is as well highly inspired on the original Scaper documentation.

For a more detailed comparison, please refer to the *Differences from Scaper* section.

CHAPTER 2

Getting started

2.1 Installation instructions

2.1.1 Installing AmbiScaper

To install the latest version of ambiscaper from source:

1. Clone or pull the lastest version:

```
>>> git clone https://github.com/andresperezlopez/ambiscaper.git
```

2. Install using pip to handle python dependencies:

```
>>> cd ambiscaper
>>> pip install -e .
```

This procedure will automatically install the required python and non-python dependencies.

Pip online installer comming soon!

2.1.2 Matlab dependency

If you wish to use the simulated reverb feature implemented by SmirGenerator, you will need to have a valid working version of Matlab.

2.1.3 Non-python dependencies (manual installation)

AmbiScaper has two non-python dependencies:

- SoX: http://sox.sourceforge.net/
- FFmpeg: https://ffmpeg.org/

On macOS these can be installed using homebrew:

```
>>> brew install sox
>>> brew install ffmpeg
```

On linux you can use your distribution's package manager, e.g. on Ubuntu (15.04 "Vivid Vervet" or newer):

```
>>> sudo apt-get install sox
>>> sudo apt-get install ffmpeg
```

NOTE: on earlier versions of Ubuntu ffmpeg may point to a Libav binary which is not the correct binary. If you are using anaconda, you can install the correct version by calling:

```
>>> conda install -c conda-forge ffmpeg
```

Otherwise, you can obtain a static binary from the ffmpeg website.

On windows you can use the provided installation binaries:

- SoX: https://sourceforge.net/projects/sox/files/sox/
- FFmpeg: https://ffmpeg.org/download.html#build-windows

2.2 AmbiScaper tutorial

Welcome to the AmbiScaper tutorial!

2.2.1 Introduction

AmbiScaper is a fantastic tool to create Ambisonics sound files (and corresponding annotations) in a procedural way.

Through this tutorial you will learn the basic features, and you will be ready to create amazing Ambisonics datasets.

Ambisonics?

Ambisonics is a spatial sound theory developed by Michael Gerzon in the 1970s, based on the acoustics and psychoacoustics fields.

On its most basic form, Ambisonics can be thought as a way to preserve the spatial characteristics of a soundfield. This is based on the plane-wave decomposition of the sound by means of the Spherical Harmonics Transform (aka the *Spatial Fourier Transform*). Indeed, there are many parallels between the *spectral* Fourier transform and the *spacial* Fourier transform, but for the moment it's enough to imagine the transform as a set of virtual microphones, each one pointing to one spatial direction. This transformation, which is usually referred to as *Ambisonics Encoding*, produces a multichannel audio track which implicitly contains the information about the source position(s). This kind of multichannel file is usually called *B-Format*.

Ambisonics audio cannot be played just as it is, but it neeeds to be *decoded*. The great thing about it is that it does not requiere any specific speaker layout: it will adapt to any given speaker layout, provided a corresponding decoder. Furthermore, it is very easy to render binaural audio from a *B-Format* recording, and this is one of the reasons why it is becoming so popular.

AmbiScaper produces *B-Format* synthetic recordings from a library of mono/stereo tracks. If you want to listen to them, you will need some extra software, as for example the great ambiX plugins suite.

Ambisonics can be very useful as well in some research areas, such as Blind Source Separation and Sound Source Localization. However, it is not easy to create an annotated dataset of Ambisonics audios, and specially with the separated tracks for the BSS evaluation. This is the main motivation behind the development of AmbiScaper. Of course, usages for other purposes, such as generative soundscape creation, are welcomed too.

Source Material

As we already mentioned, the raw material for the Ambisonics soundscape creation are individual audio clips, both mono or stereo. These clips, ideally (semi)anechoic, must be present in your system. You can use any sounds you like but, just in case, AmbiScaper features a selection of clips from the great openAIRlib. You can find them in the AmbiScaper repository, under the *samples* folder.

2.2.2 Creating our first Ambisonics soundscape

AmbiScaper Instanciation

The `AmbiScaper` class is the main object from the library. Let's start by instanciating it:

```
from ambiscaper import *
import numpy as np
import os

# AmbiScaper settings
soundscape_duration = 5.0
ambisonics_order = 1
foreground_folder = os.path.abspath('./samples/Acoustics_Book')

### Create an ambiscaper instance
ambiscaper = AmbiScaper(duration=soundscape_duration,
                        ambisonics_order=ambisonics_order,
                        fg_path=foreground_folder)
```

We are specifying three arguments to the AmbiScaper instance creation.

1. The desired soundscape duration.
2. The Ambisonics order to be used.
3. The path to the sound event folder (in this case, the *Acoustics_Book* samples shipped with the code).

Adding sound events

Once we have an instance of AmbiScaper, we can add audio clips (events) to be rendered.

One of the main features of AmbiScaper is that event parameters might be specified in terms of statistical distributions, and not only as fixed values. For example:

```
### Add an event
ambiscaper.add_event(source_file=('choose', ['adult_female_speech.wav','bagpipe_music.
↪wav']),
                     source_time=('const', 0),
                     event_time=('const', 0),
                     event_duration=('const', soundscape_duration),
                     event_azimuth=('uniform', 0, 2*np.pi),
                     event_elevation=('uniform', -np.pi/2, np.pi/2),
                     event_spread=('const', 0),
                     snr=('const', 0),
```

(continues on next page)

(continued from previous page)

```
                          pitch_shift=('const', 1),
                          time_stretch=('const', 1)
                          )
```

As you can see, every parameter is defined in terms of a *distribution tuple*, i.e., a definition of the possible values that the given parameter might take. The actual values, sampled from the distribution tuples, will be assigned at the soundscape rendering stage (the `generate()` method).

The distribution tuples currently supported are:

- `('const', value)`: a constant, given by `value`.
- `('choose', list)`: uniformly sample from a finite set of values given by `list`.
- `('uniform', min, max)`: sample from a uniform distribution between `min` and `max`.
- `('normal', mean, std)`: sample from a normal distribution with mean `mean` and standard deviation `std`.
- `('truncnorm', mean, std, min, max)`: sample from a truncated normal distribution with mean `mean` and standard deviation `std`, limited to values between `min` and `max`.

Therefore, our `add_event` method is actually specifying the following:

- `source_file` can take one of the two provided values: `adult_female_speech.wav` or `bagpipe_music.wav`.
- **`source_time` (the time in the source file from which to start the event) and `event_time` (the start time of the event in**
  Furthermore, `event_duration` is set equal to the soundscape duration.
- **`event_azimuth` and `event_elevation`, the angles defining the event position, are set to take a random value, unifor**
  Remember that *azimuth* is the angle in the horizontal plane starting from the X axis in counter-clockwise direction, and *elevation* is the angle perpendicular to the horizontal plane, being `0` the horizontal plane, `pi/2` above and `-pi/2` below.
- `event_spread` is set to 0. The spread parameter can be thought as the apparent sound source width, with a value between `0` (no spread) and `1` (fully spread)
- `snr` `pitch_shift` `time_stretch` are set with a constant value.

To summarize up, the `add_event()` method allows to tell AmbiScaper about an *event specification* with statistical distribution values.

Soundscape generation

Once the sound events are described, we can proceed to actually generate the Ambisonics soundscape.

This is provided by the `generate()` method:

```
### Genereate the audio and the annotation
outfolder = '/Volumes/Dinge/ambiscaper/testing/' # watch out! outfolder must exist
destination_path = os.path.join(outfolder,"my_first_ambisonics_soundscape")

ambiscaper.generate(destination_path=destination_path,
                    generate_txt=True)
```

This piece of code will actually sample all values from the event specifications, in a process called *instanciation*, and as a result will provide the rendered audio and the associated annotations.

If everything went good so far, we will have the following output structure:

- **/my_first_ambisonics_soundscape/**
  - *my_first_ambisonics_soundscape.wav*
  - *my_first_ambisonics_soundscape.jams*
  - *my_first_ambisonics_soundscape.txt*
  - **/Source/**
    - *fg0.wav*

Let's explain them briefly:

`my_first_ambisonics_soundscape.wav` is the main audio output: an Ambisonics multichanel audio file (in this case 4 channels, since we specified 1st Order Ambisonics), which contains the spatially encoded representation of the audio source. If you try to open the file with an audio editor (we can recommend Audacity and Ardour, which are open source, multichannel-friendly and cool), you will appreciate that the different channels have the same audio content with different gains - that's hos Ambisonics looks like. The duration of the audio file is 5 seconds, just as we specified.

Note: Ambisonics Audios generated by AmbiScaper follow the conventions:

- Normalization: *SN3D*
- Channel ordering: *ACN*

`my_first_ambisonics_soundscape.jams` is the annotation file, in the JAMS format (a JSON-based specification intended for Music Information Retrieval). It basically contains a bunch of information related to the generated audio file: not only the actual instanciated values, but also the distribution tuples from the event specification. This is great, since we can use this file not only for validation, but also for exact setup reproduction.

Try to open the file with any text editor or python IDE, and inspect a little bit the contents. The instanciated values are under the `data` field, and you can check that they are consistent with the provided AmbiScaper arguments. Another interesting part is located under the field `fg_spec`, containing the given distribution tuples.

`my_first_ambisonics_soundscape.txt` is a small plain text file, which contains some information about the generated soundscape. More precisely, it includes one row for each sound event, and features three columns (separated by tabs): *start time*, *end time* and *event_id*. Please notice that *event_id* is `fg0`, which corresponds to the first foreground event. A very handy usage of this text file is event duration visualization through Audacity (*File/Import/Labels..*).

Note: `event_id` is the unique identifier for each sound event, assigned by AmbiScaperin the order given by the successive calls to `add_event()`.

event_ids are composed of the string `fg` and an index. The correspondence of each *event_id* with the actual source file name is defined in the `data` field of the JAMS file.

Inside the *source* folder, there will be just one file, `fg0.wav`. This is a copy of the original source file, which includes the modifications performed by AmbiScaper (time offset, gain correction, pitch shift, etc). To put it in other words, it contains just the exact audio content before the Ambisonics encoding. This file is very useful if you want to perform Source Separation evaluation tasks.

If you want to explore a litte bit more the capabilities of AmbiScaper, please refer to the *Examples* section.

A reference database produced with AmbiScaper can be found in Zenodo.

2.2.3 Differences from Scaper

As already mentioned, AmbiScaper is (obviously) based on the great Scaper, by Justin Salamon. More precisely, it was forked at 17th October 2017 from commit e0cc1c9.

Scaper is a piece of software intended for automatic generation and annotation of monophonic soundscapes, in the context of Auditory Scene Analysis, sound event recognition, etc. The parallelism with the Blind Source Separation problem is clear: we need big datasets of annotated events, specially when dealing with Deep Neural Network architectures.

Forking such a project is a great idea, since all the nice features (event specification vs instanciation, jams file, etc) are preserver. However, obviously, some changes must be performed in order to adapt the code to the Ambisonics domain.

In that sense, Scaper and AmbiScaper are not mutually compatible. That means that, in general, copying pieces of code from one to the another won't work. The number of arguments to the methods, the default values, the namespaces, the lack of labels, and many other aspects have been changed and adapted to the new situation.

However, don't panic! The code structure is very similar and, if you already know how Scaper works, it will be very fast to catch up with AmbiScaper.

2.3 Examples

2.3.1 Example 1: Foreground and background

In this example, we are instanciating one sound event on the front position (azimuth and elevation are `0`), and adding a background sound.

Background sources are implemented as completely diffused sources: internally, spread is set to `1`. Although this implementation is not accurate in physical terms, and therefore not rigorous (for instance, not valid for Intensity Vector Statistics or parametric spatial audio analysis), it is still a common approach in the scope of spatial audio reproduction and diffuse soundscape creation.

Also notice that most of the event specifications are not available (and relevant) for the background spec. For example, `azimuth` and `elevation` are not relevant on this context, since a fully spread source should come from *all* positions.

Lastly, please notice the `ref_db` and `snr` parameters. Ambiscaper's instance `ref_db` refers to the "noise floor" of the scene. The backgound event's reference level will be set to that value. By default it is set to `-30` dBs. Accordingly, event's `snr` indicates the dBs above `ref_db` of the given event. Loudness computation is performed in a psychoacoustic way through the LKFS scale.

```
# Example 1: Foreground and background
# ------------------------------------

from ambiscaper import *
import os

# AmbiScaper settings
soundscape_duration = 5.0
ambisonics_order = 1

# We want to use the full samples folder as potential events
samples_folder = '../../samples/Handel_Trumpet'

### Create an ambiscaper instance
ambiscaper = AmbiScaper(duration=soundscape_duration,
```

(continues on next page)

(continued from previous page)

```
                        ambisonics_order=ambisonics_order,
                        fg_path=samples_folder,
                        bg_path=samples_folder)

# Configure reference noise floor level
ambiscaper.ref_db = -30

### Add a background event

# Background events, by definition, have maximum spread
# That means that they will only contain energy in the W channel (the first one)
ambiscaper.add_background(source_file=('const', 'tr-1788d-piece4-sl.wav'),
                          source_time=('const', 5.))

### Add an event
ambiscaper.add_event(source_file=('const', 'tr-1888d-high.wav'),
                     source_time=('const', 0),
                     event_time=('const', 0),
                     event_duration=('const', soundscape_duration),
                     event_azimuth=('const', 0),
                     event_elevation=('const', 0),
                     event_spread=('const', 0),
                     snr=('const', 10),
                     pitch_shift=('const', 1),
                     time_stretch=('const', 1)
                     )

### Genereate the audio and the annotation
outfolder = '/Volumes/Dinge/ambiscaper/testing/'  # watch out! outfolder must exist
destination_path = os.path.join(outfolder, "example1")

ambiscaper.generate(destination_path=destination_path,
                    generate_txt=True)
```

2.3.2 Example 2: 15th order, variable spread Ambisonics soundscape

We will create a **15th** (yes, fifteenth!) order ambisonics soundscape (256 channels), consisting of 10 sound events, placed at random positions around the sphere, with a small amount of spread. Furthermore, the parameter `ambisonics_spread_slope` is set to `0.25`, meaning a softer transition in ambisonics order downgrading (please refer to *[Carpentier2017]* for more information).

Importing *example2.txt* into Audacity might be helpful to understand the cacophony and the possibilities of AmbiScaper.

Also, it might be of interest to have a look into *example2.jams*, in order to understand how the file is organized, and the possibilities for evaluation and experiment replication that it offers.

Finally, notice the `allow_repeated_source=True` argument in `generate()`. As its name implies, during event instanciation, it will not take into account that a given source has been already chosen. Setting it to `False` might be useful in many cases, but then there is the risk of not having enough sources in the used database.

```
# Example 2: 15th order, variable spread Ambisonics soundscape
# --------------------------------------------------------------

from ambiscaper import *
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import os

# AmbiScaper settings
soundscape_duration = 10.0
ambisonics_order = 15
ambisonics_spread_slope = 0.25 # soft curve

# We want to use the full samples folder as potential events
samples_folder = os.path.abspath('../../samples/')

### Create an ambiscaper instance
ambiscaper = AmbiScaper(duration=soundscape_duration,
                        ambisonics_order=ambisonics_order,
                        fg_path=samples_folder)

# Make everything a little bit softer to avoid clipping
ambiscaper.ref_db = -40
ambiscaper.ambisonics_spread_slope = ambisonics_spread_slope

# add 10 events!
num_events = 10
for event_idx in range(num_events):
    ### Add an event
    ambiscaper.add_event(source_file=('choose',[]),
                         source_time=('uniform', 0, soundscape_duration),
                         event_time=('uniform', 0, soundscape_duration),
                         event_duration=('const', soundscape_duration),
                         event_azimuth=('uniform', 0, 2 * np.pi),
                         event_elevation=('uniform', -np.pi / 2, np.pi / 2),
                         event_spread=('truncnorm', 0.1, 0.2, 0.0, 0.5),
                         snr=('uniform', 0, 10),
                         pitch_shift=('uniform', -2, 2),
                         time_stretch=('uniform', 0.8, 1.2))

### Genereate the audio and the annotation
outfolder = '/Volumes/Dinge/ambiscaper/testing/'  # watch out! outfolder must exist
destination_path = os.path.join(outfolder, "example2")

ambiscaper.generate(destination_path=destination_path,
                    generate_txt=True,
                    allow_repeated_source=True)
```

2.3.3 Example 3: Reverberant soundscape from recorded SOFA Ambisonics IRs

So far we have been considering the anechoic case, which is great, but unfortunately not very realistic. Reverberation is present in almost all acoustic environments, and most state-of-the-art algorithms for Blind Source Separation and Source Localization consider the reverberant case. Apart from the more scientifical approach, reverberant soundscapes sound very nice!

In this example we will use *sala1.sofa*, which is a nice reverb shipped with AmbiScaper. It corresponds to an acoustic measurement of an exhibition room in the Fundacio Miro, Barcelona. This file is part of the Ambisonics Room Impulse Responses dataset.

Reverberation is captured through Impulse Responses (IRs). In this particular case, we are using *Ambisonics IRs*, wich are IRs recorded with an Ambisonics microphone, thus capturing the spatial cues of the reverberation. It should be noticed that reverberation is variable along a room, in the sense that it depends on both the position of the emitter and the receiver. Since it would be impossible to record every possible pair of emitter/receiver positions, a spatial sampling strategie must be designed. The implication for the soundscape generation is that we can only provide IRs from the actual measured emitter points, and with a limited spatial resolution (the Ambisonics order of the microphone used).

The SOFA conventions conform a file description standard, intended for storing Impulse Responses from different measurement setups. The author has participated in the design of a SOFA convention for Ambisonics IRs, the *AmbisonicsDRIRconvention* (please refer to *[Perez2018]* for more information). AmbiScaper uses SOFA files through the pysofaconventions library, which is automatically installed with pip as a dependence. Therefore, all specific SOFA details remain hidden to the user.

In the following example, we define `ambisonics_order = 2`. However, since we are using a recorded reverb spec of order 1, the system will automatically downgrade the Ambisonics order to match the common minimum.

Furthermore, the source positions will be limited to the ones provided by the `'Foyer'` measurements. How AmbiScaper select the final source positions due to this constrain is selected through the `wrap` argument inside `add_sofa_reverb()` method. There are different options:

- `wrap_azimuth`: source position assigned to the closest speaker position in azimuth
- `wrap_elevation`: source position assigned to the closest speaker position in azimuth
- `wrap_surface`: source position assigned to the closest speaker position around the spherical surface
- `random`: source position assigned randomly to one of the available speaker positions

Please note that the reverb is as well specified in terms of distribution tuples, reusing the logic for the sound events, and allowing for a very flexible dataset creation. Only one reverb type might be specified per soundscape, *i.e.*, per each time the `generate()` method is called.

```
# Example 3: Reverberant soundscape from recorded SOFA Ambisonics IRs
# ------------------------------------------------------------------

from ambiscaper import *
import numpy as np
import os

# AmbiScaper settings
soundscape_duration = 5.0
ambisonics_order = 2
samples_folder = '../../samples/Bicycle_Horn'

### Create an ambiscaper instance
ambiscaper = AmbiScaper(duration=soundscape_duration,
                        ambisonics_order=ambisonics_order,
                        fg_path=samples_folder)

num_events = 2
for event_idx in range(num_events):
    ### Add an event
    ambiscaper.add_event(source_file=('choose', []),
                         source_time=('uniform', 0, soundscape_duration),
                         event_time=('uniform', 0, soundscape_duration),
                         event_duration=('const', soundscape_duration),
                         event_azimuth=('uniform', 0, 2 * np.pi),
                         event_elevation=('uniform', -np.pi / 2, np.pi / 2),
                         event_spread=('uniform', 0, 1),
                         snr=('uniform', 0, 10),
                         pitch_shift=('const', 1),
                         time_stretch=('const', 1))
```

(continues on next page)

(continued from previous page)

```
# Set the path to the SOFA reverbs
ambiscaper.set_sofa_reverb_folder_path('../../SOFA')

# Add a recorded reverb
ambiscaper.add_sofa_reverb(name=('const', 'sala1.sofa'),
                           wrap=('const', 'wrap_azimuth'))

### Genereate the audio and the annotation
outfolder = '/Volumes/Dinge/ambiscaper/testing/'  # watch out! outfolder must exist
destination_path = os.path.join(outfolder, "example3")

ambiscaper.generate(destination_path=destination_path,
                    generate_txt=True,
                    allow_repeated_source=True)
```

Please notice the warning message:

```
AmbiScaperWarning:  User-defined Ambisonics order L=2 is higher
than the maximum order allowed by the reverb spec. Downgrading to 1
AmbiScaperWarning).
```

The last remark is about the IRs. In order to be useful to dereverberation/reverb estimation applications, the actual IRs used are copied into the output */source/* folder, together with the source files. The name of the file corresponds to the `event_id` parameter for each source: for example, source 0, which is named `fg0.wav`, will have a corresponding `h0.wav` file, and so on.

2.3.4 Example 4: Reverberant soundscape from simulated Ambisonics IRs

Note: SIMULATED REVERBS ARE STILL IN DEVELOPMENT!!

AmbiScaper provides the option to use simulated IRs to create synthetic reverberant sound scapes. This option might be useful in the cases in which it is not possible to record IRs, due to limitations on equipment, permissions or whaterver other reason. It is as well indicated for parametric analysis (for example, how is the performance of my BSS algorithm as a function on t60?).

The simulated reverbs are computed through the wonderful SMIR Generator, a Matlab library intended for simulation of IRs on a spherical surface (as for example an Ambisonics Microphone), inside a *shoebox* room model. For a more detailed explanation, please refer to *[Jarrett2012]*.

Note: Please, notice that SMIR Generator is implemented in Matlab, and therefore a valid Matlab installation is needed in order to run the program.

When the simulated reverb is specified, AmbiScaper will internally launch a background Matlab sesssion, which will execute the required computation. When finished, the resulting data will be transferred back to AmbiScaper, and the Matlab session will be automatically closed. All this process remains hidden for the user.

Please, take into account that IR simulation is a computationally expensive process, and the computation time will increase exponentially with the Ambisonics order.

As a result of AmbiScaper's `generate()` method, the computed IRs will be available at the */source/* folder. AmbiScaper provides thus a way to create databases of Ambisonics IRs, defined in statistical terms.

SMIR Generator is a very flexible tool and, consequently, many parameters might be tuned for the IR computation. AmbiScaper exposes a subset of the most relevant ones, described as distribution tuples, in the `add_simulated_reverb()` method:

- `IRlength`: length in samples of the desired IRs
- `room_dimensions`: specified in meters, in the format *[x,y,z]*
- `t60`: reverberation time at 1 kHz, in seconds
- **`source_type`: source directionality, can be choosen between:**
    - 'o': omnidirectional
    - 'c': cardioid
    - 's': subcardioid
    - 'h': hypercardioid
    - 'b': bidirectional
- **`microphone_type`: AmbiScaper features some predefined virtual microphone geometries:**
    - 'soundfield'
    - 'tetramic'
    - 'em32'

Note: It is possible, as well, to define the wall `reflectivity`, specified for each one of the walls. However, it is not possible to define both `t60` and `reflectivity` at the same time, for obvious reasons. In that case, `t60` will be preferent.

By default, the virtual microphone will be placed at the center of the room, and thus the source position is defined with respect to that center.

In Example 4, we will create a file consisting of a trumpet recording in a synthetic reverberant environment, virtually captured with a Soundfield microphone.

```
### EXAMPLE 4

import ambiscaper
import numpy as np
import os

# AmbiScaper settings
soundscape_duration = 5.0
ambisonics_order = 1
samples_folder = os.path.abspath('./samples/Handel_Trumpet')

### Create an ambiscaper instance
ambiscaper = ambiscaper.AmbiScaper(duration=soundscape_duration,
                                   ambisonics_order=ambisonics_order,
                                   fg_path=samples_folder)



### Add an event
ambiscaper.add_event(source_file=('choose',[]),
                     source_time=('uniform', 0, soundscape_duration),
                     event_time=('uniform', 0, soundscape_duration),
```

(continues on next page)

(continued from previous page)

```
                     event_duration=('const', soundscape_duration),
                     event_azimuth=('const',0),
                     event_elevation=('const',0),
                     event_spread=('const',0.5),
                     snr=('uniform', 0, 10),
                     pitch_shift=('const', 1),
                     time_stretch=('const', 1))

# Add Simulated Reverb
ambiscaper.add_simulated_reverb(IRlength=('const', 2048),                  # in␣
↪samples
                                room_dimensions=('const', [3,3,2]),          # [x,y,z]
                                t60=('const', 0.2),                          # in␣
↪seconds
                                source_type=('const', 'o'),                  #␣
↪omnidirectional
                                microphone_type=('const', 'soundfield'))     # order 1


### Genereate the audio and the annotation
outfolder = '/Volumes/Dinge/ambiscaper/testing/'  # watch out! outfolder must exist
destination_path = os.path.join(outfolder, "example4")

ambiscaper.generate(destination_path=destination_path,
                    generate_txt=True,
                    allow_repeated_source=True)
```

CHAPTER 3

API Reference

3.1 API Reference

CHAPTER 4

Contribute

- Issue tracker
- Source code
- Project website

CHAPTER 5

Indices and Tables

- genindex
- modindex
- search

Bibliography

[Carpentier2017] Carpentier, T. (2017, May). Ambisonic spatial blur. In Audio Engineering Society Convention 142. Audio Engineering Society.

[Perez2018] Perez-Lopez, A. and de Muynke, J, (2018) “Ambisonics Directional Room Impulse Response as a New Convention of the Spatially Oriented Format for Acoustics”,http://www.aes.org/e-lib/browse.cfm?elib=19560

[Jarrett2012] D. P. Jarrett, E. A. P. Habets, M. R. P. Thomas and P. A. Naylor, “Rigid sphere room impulse response simulation: algorithm and applications,” Journal of the Acoustical Society of America, Volume 132, Issue 3, pp. 1462-1472, 2012.

Python Module Index

Index