miro :: measured impulse response object

data type description

Fachhochschule Köln
Cologne University of Applied Sciences

Document Information

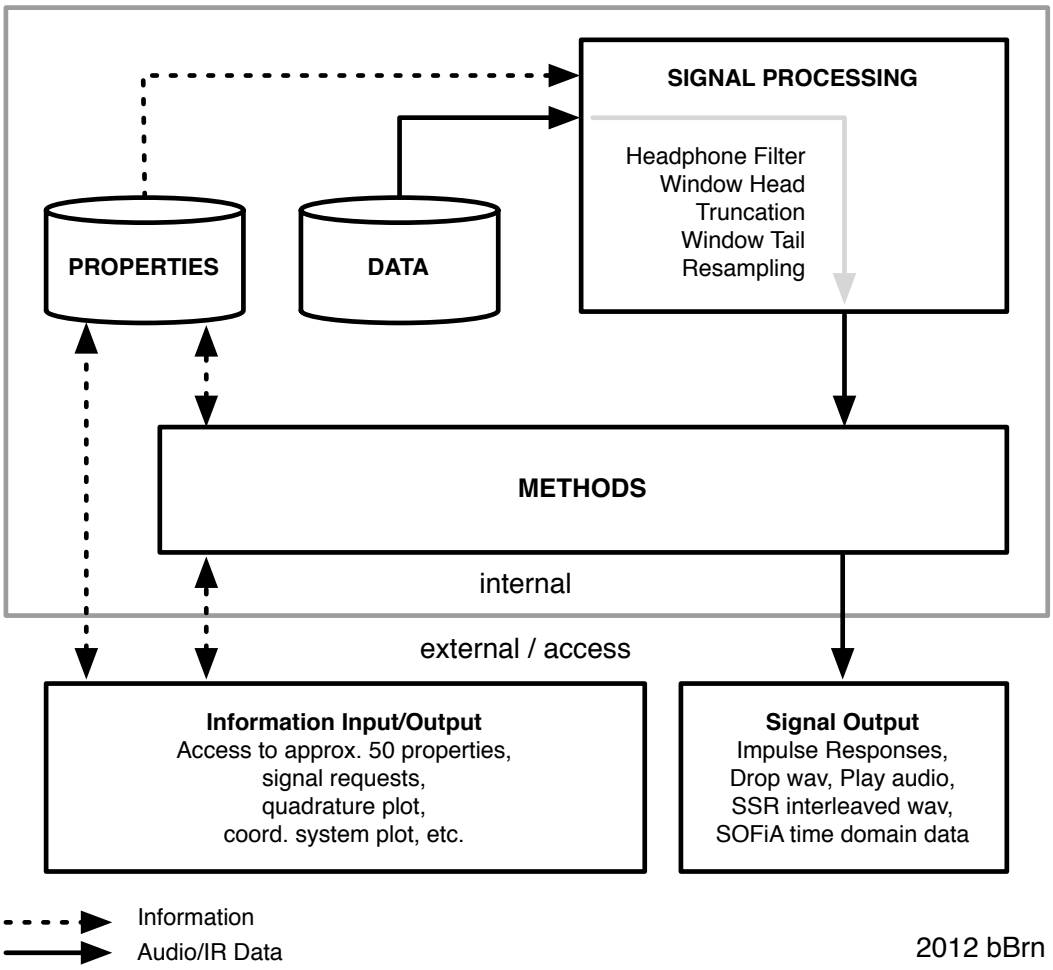
Author	Benjamin Bernschütz
Source	http://www.audiogroup.web.fh-koeln.de
miro Version	1.04
Document Revision	1.2
Date	07/February/2014
Institution	Cologne University of Applied Sciences, Institute of Communication Systems, Betzdorfer Str. 2, 50679 Cologne, Germany
Mail	benjamin.bernschuetz@fh-koeln.de
GSM	+49 171 4176069
Phone	+49 221 8275 2496

Introduction

miro is a simple object oriented data type for the storage and handling of measured audio impulse responses, especially designed e.g. for complex microphone array or rotated dummy head datasets. *miro* works under MATLAB© and has been developed for the storage and distribution of the WDR Spatial Audio Impulse Response Collection and the Neumann KU100 spherical HRIR datasets that were captured by the audio group of Cologne University of Applied Sciences during the summer of 2012. But the *miro* datatype will also be used for future work and can naturally be used and modified/optimized by the audio community. Each measurement position or session (e.g. microphone array or rotated dummy head measurement) is stored in a separate *miro* instance. The *miro* class combines three basic elements:

1. **PROPERTIES:** Description and detail information on the content and the measurement session that are stored in the properties.
2. **METHODS:** Basic methods for the extraction and treatment of the measured data.
3. **DATA:** Measured Data (Raw Impulse Responses).

miro datatype overview



Important general information:

- Besides to the object file itself, MATLAB© needs to have access to the *miro* class definition that is stored in the file *miro.m*. This file must be put in the object's directory or to an accessible MATLAB© searchpath.
- Miro has a basic signal-processing core (inside the *obj.getIR* method). All impulse responses run through this core before they are returned, written to a file or played back. Thus the processing settings are globally valid and affect all impulse responses

automatically. The processing blocks (headphone filters, windowing/truncation, resampling) can be bypassed or adapted via the properties.

- Every impulse response has a specific ID (*irID*). A stereo impulse response (e.g. from a dummy head) has two channels but one single *irID* only. The numbering/indexing of the logical contiguous impulse response sets starts at $irID = 1$.
- Every miro instance contains a corresponding center impulse response that is captured using an omnidirectional microphone (*obj.centerMicrophone*) at the physical center (origin) of e.g. the microphone array or dummy head. The center impulse response can is adressed via $irID = 0$. Be careful to distinguish this particular ID.
- For the ease of use, all main methods and properties are intentionally set *public*[1].
- Miro is a *Value Class* by default (recommended), but can be changed to a *Handle Class* if needed. For more information on that topic refer to the *Mathworks* website.
- The default angles are in RAD but the object can be set to operate in DEG.

[1]Thus the risk of misuse and damaging an object is high. But this is not an important problem in the present case as the original objects can always be reloaded if the current instance is broken for some reason. If the properties and methods were set private to offer an increased protection, the objects would become much more complex to use and would require e.g. specific getter and setter methods for every property. This is not very convenient in the present case. Easy handling was rated to be more valuable than a high protection level here.

Properties

Properties	Description	Type
name	name/description of the IR set	string
context	indentifier of a larger measurement session	string
location	recording location	string
date	recording date	string
engineer	recording engineer	string
contact	email/telephone for requests	string
comments	comment(s)	string
miroVersion	version of the miro class definintion	1x1 single
type	*{HRIR, BRIR, MICARRAY,SINGULAR}*	string
fs	audio sampling rate in Hz	1x1 single
taps	number of taps	1x1 single
nIr	number of impulse responses in dataset	1x1 single
excitatationSignal	excitation signal	string
gapTime	gap time in s	1x1 single
microphone	manufacturer/type microphone or dummy head	string
source	manufacturer/type source loudspeaker	string
audioInterface	manufacturer/type audio interface	string
micPreamp	manufacturer/type mic preamp	string
capturingSystem	manufacturer/type of the capturing system	string
systemLoopLatency	electrical system I/O loop latency	1x1 single
latencyCompensated	*{true,false}* system loop latency compensation	1x1 bool
headCut	number of empty leading head samples that were cut	1x1 single
sourcePosition	may contain coordinates or verbal description e.g. 0AZ, 90EL or Left, Center	string
sourceDistance	distance between microphone and source in m	1x1 single

avgAirTemp	average air temperature in °C	1x1 single
avgRelHumidity	average relative air humidity in %	1x1 single
positionReference	*{Virtual Source, Head Rotation, Microphone}*	string
postProcessing	description of the post processing	string
nomalization	normalization factor (1=no normalization)	1x1 single
ctIRnomalization	normalization factor (1=no normalization)	1x1 single
quadGrid	name/description of the quadrature grid	string
scatterer	*{true, false}* (for a dummy head set true)	1x1 bool
radius	microphone radius/radii in m	1x[1,2] single
azimuth	position azimuth	[1xnIR]
elevation	position elevation	[1xnIR]
quadWeight	quadrature weighting	[1xnIR]
chOne	content description e.g. *Left Ear*	string
chTwo	content description e.g. *Right Ear* or []	string
irChOne	impulse responses	[tapsxnIR]
irChTwo	impulse responses or []	[tapsxnIR]
centerMicrophone	manufacturer/type of the omni center microphone	string
irCenter	omni center impulse response at the coordinate origin	[tapsx1]
returnTaps	number of taps to be returned ($\leq$ taps)	1x1 single
resampleToFS	target sampling frequency, [] = no resampling	1x1 single
headWin	head window length	1x1 single
tailWin	tail window length	1x1 single
headPhone	manufacturer/type of the headphone	string
hpcfKernel	headphone compensation filter kernel	string
headPhoneComp	true, false enables/disables HP compensation	1x1 bool
shutUp	true, false if set true no console messages are printed	1x1 bool
angles	RAD, DEG RAD is default	string

Methods

[ir, azimuth, elevation, quadWeight] = getIR(obj, irID)

Returns the impulse response, angles and the quadrature weight for a specific ID number *irID*. The returned impulse response can have one or two channels depending on the respective object. This method involves the signal processing core.

[irID, azimuth, elevation] = closestIr(obj, az_approx, el_approx)

Returns the ID number *irID* of the closest angle to *az_approx, el_approx* that is available within the object. *azimuth* and *elevation* return the corresponding closest fitting angles. The angles can be defined either in *RAD* or *DEG*, depending on the *obj.angles* status.

quadrature = getQuadrature(obj)

Returns the quadrature including azimuth and elevation angles and weights.

[] = plotQuadrature(obj)

Shows a sphere plot of the quadrature.

[] = miroCoordinates(obj)

lllustrates the coordinate system that is used by *miro*.

obj = setDEG(obj)

Changes the object's angle reference to *DEG*. All angle handling is then in *DEG*. (The default angle reference is *RAD*.)

obj = setRAD(obj)

Changes the object's angle reference to *RAD*. All angle handling is then in *RAD*. (The default angle reference is *RAD*.)

obj = setReturnTaps(obj, returnTaps, [tailWin])

Allows for changing the number of returned impulse response taps. By default, all available taps are returned (*obj.returnTaps* = taps). Setting the *returnedTaps* property to a different value will cause that all returned impulse responses are cut off at the value defined by *obj.returnTaps* and are by default windowed using a half-sided Hann window of the size (*obj.returnedTaps*/8). The size of the window can be defined by *tailWin*. Using *tailWin* = 0 turns off windowing.

obj = setResampling(obj, targetFS)

Sets the resampleToFS property and can be used to extract the IRs at a different audio sampling rate. *targetFS* defines the target sampling rate, e.g. 44100Hz or 96000Hz. If called without the *targteFS* argument, the *obj.resampleToFS* property is set to [] (default) and disabled. The resampling process is done within the getIR method after truncation and windowing. The returned IRs then do NOT have the amount of samples set in returnTaps property as these refer to the original FS. The resampling is based on the native MATLAB© *resample()* method which is included in the Signal Processing Library. **ADVICE**: If not urgently necessary try to avoid resampling.

dropWaveFile(obj, irID, [nBits], [filename])

Drops a wave audio file containing the impulse response for a specific ID number *irID*. The arguments *nBits* and *filename* are optional. Defaults: *nBits* = 16, *filename* = obj.name,'_IR',num2str(irID),'-AZ',az, 'EL',num2str(el), *obj.angles*, [*obj.headphone*]. No dithering/noiseshaping is applied.

[] = playAudio(obj, irID, audioSignal)

Plays an audio signal convolved with the preprocessed impulse response(s) given by *irID*. The method serves for a quick pre-listening of the datasets. If the signal *audioSignal* has

more than one channel, the first channel is taken.

obj = setHeadPhones(obj, hpFilter, [linearPhase])

Sets headphone compensation filters for HRIR and BRIR datasets. Once the kernel is loaded the filters are applied to all outgoing impulse responses as long as obj.headphoneComp is set *true*. The *linearPhase* flag {*true, false*} can be set *true* to enable linear phase compensation (Careful: High Latency). By default the flag status is *false* and a minimum phase filter is applied.

[] = miroToSSR(obj, [mirror], [nBits], [filename], [normalize])

Writes a 720-channel interleaved wave file for the Sound Scape Renderer (SSR). The SSR can e.g. be used for dynamic binaural synthesis.

SSR Website: http://www.tu-berlin.de/?id=ssr

The object must be a circular HRIR or BRIR set to be exported to an SSR wave file. Defaults: *mirror {true, false}* = false (This option can be useful for symmetrical venues to mirror a source from left to right or vice versa.), *nBits* = 16, *filename* = 'SSR_', obj.name, [obj.headphone], *normalize {true, false}* = true. No dithering/noiseshaping is applied.

[timeData1,timeData2] = miroToSOFiA(obj)

Returns a struct that is readable by the F/D/T function of the SOFiA sound field analysis toolbox. The following S/T/C spatial transform core transforms the object's data into the spherical harmonics domain.

SOFiA Website: http://code.google.com/p/sofia-toolbox/

Examples/Tutorial

Application examples are available at http://www.audiogroup.web.fh-koeln.de.