Open signal processing software platform for hearing aid research (openMHA)

Tobias Herzke[1] and **Hendrik Kayser**[2] and **Frasher Loshaj**[1] and **Giso Grimm**[1,2] and **Volker Hohmann**[1,2]
[1] HörTech gGmbH and Cluster of Excellence "Hearing4all",
Marie-Curie-Str. 2, D-26129 Oldenburg, Germany
[2] Medizinische Physik and Cluster of Excellence "Hearing4all"
Universität Oldenburg, D-26111 Oldenburg, Germany
info@openmha.org

Abstract

Hearing aids help hearing impaired users participate in the communication society. Development and improvement of hearing aid signal processing algorithms takes place in the industry and in academic research. With openMHA, we present a development and evaluation platform that is able to execute hearing aid signal processing in real-time on standard computing hardware with a low delay between sound input and output. We lay out the application specific requirements and present how openMHA meets these and will be helpful in future research in the field of signal processing for hearing aids.

Keywords

Hearing aids, audio signal processing, plugin host

1 Introduction

Development of hearing aid signal processing is widely conducted by hearing aid manufacturers on proprietary systems that are not accessible to the research community and that underlie commercial constraints. Providing open tools to the hearing aid research community lowers barriers, accelerates studies with novel acoustic processing algorithms and facilitates translation of these advances into widespread use with hearing aids, cochlear implants, and consumer electronics devices for sub-clinical hearing support. A software platform for the development and evaluation of hearing aid algorithms should

- offer a complete set of hearing aid signal processing reference algorithms that can be combined with newly developed algorithms to form a complete hearing aid signal processing chain,
- enable researchers to perform offline-processing as well as real-time signal processing with a reliable low delay between acoustic input and output of less than 10 milliseconds, even when algorithms need significant processing power,
- provide a library for common signal processing tasks and commonly needed services in hearing aid signal processing, like support for acoustic calibration and filterbanks,
- be able to run on a wide range of hardware, from high-performance PCs to execute bleeding-edge algorithms in real-time, to portable, power-efficient, headless, battery-powered devices for improved testing capabilities in realistic usage scenarios and field tests.

Several open-source tools for audio signal processing, that can also be used in hearing aid research, exist:

Octave. Octave is actively used in hearing aid research for the development of signal processing algorithms for hearing aids. It is a suitable tool to quickly develop, change and evolve isolated algorithms as long as no real-time audio processing is required. However, Octave is unsuitable for executing hearing aid algorithms in real-time with live input and output sound signals with low delay.[Eaton et al., 2015]

NumPy/SciPy. Scientific Computing Tools for Python enable researchers to develop signal processing algorithms. Technically, this software platform is equivalent to octave, but it is to our knowledge currently not actively used in hearing aid research.[Jones et al., 2001]

Pure Data. Pd is a real-time signal processing platform. It features a graphical programming interface. Pd is actively used mainly by artists to perform signal processing of music and other data. Pd can achieve a low delay in real-time processing. In principle it would be possible to develop hearing aid signal processing algorithms on Pd, and have these algorithms process audio signal in real-time. We are not aware of any hearing aid research being performed on the Pd platform and would consider it too la-

borious to implement modern hearing aid algorithms in the graphical programming environment. Pd can be extended with C, therefore, hearing aid algorithms could be implemented for Pd in C or C++.[Puckette, 1996]

Plugin hosts. Various plugin hosts for different plugin architectures (VST, LADSPA, LV2) exist, that can load and combine algorithms in plugins to form complex signal processing chains. Most hosts can achieve a low delay in real-time audio processing. Plugins can be written in C or C++ using the plugin-architecture specific SDK. (Using the VST SDK requires signing a license agreement.) Plugin hosts are mainly used by sound engineers and also by artists to process recorded or live music and other sounds.

Signal processing toolboxes and languages. A signal processing toolbox like the Synthesis ToolKit (STK) [Cook and Scavone, 1999] and domain-specific languages (DSL) like SuperCollider [McCartney, 2002] and Faust [Orlarey et al., 2009] provide useful signal processing primitives to ease development of audio signal processing algorithms. We are not aware of any hearing aid research being performed using these toolboxes and DSLs.

While the dynamic programming languages Octave and Python are suitable to develop algorithms and execute them offline, their runtime environment is not suitable for real-time processing when low delay is required at high processing loads. Octave and Python do not give algorithm implementers the necessary control to prevent heap memory allocation in the signal processing path, which can cause unpredictable interruptions in the real-time processing due to priority inversion situations. Pd and plugin hosts are real-time safe themselves and allow algorithms to be implemented in C or C++. The C and C++ programming languages allow developers sufficient control to implement algorithms in a real-time safe way. However, Pd and plugin hosts do not provide commonly needed services to hearing aid signal processing developers like calibration or an existing set of hearing aid algorithms.

The HörTech Master Hearing Aid (MHA) [Grimm et al., 2006; Grimm et al., 2009a] is an existing software platform for hearing aid algorithm development and evaluation that meets all the requirements and has been used by the hearing aid industry as well as in academic research. Until recently, it was only available as a closed-source commercial product. To enable and facilitate collaborative research efforts and comparative studies in the research community, an open-source version of the MHA software platform for real-time audio signal processing is now being developed and made available: the open Master Hearing Aid (openMHA). In February 2017, a pre-release of the openMHA has been published on GitHub under an open-source license (AGPL3) by [HörTech gGmbH and Universität Oldenburg, 2017]. This pre-release features an initial set of reference algorithms for hearing aid processing, which will be expanded in subsequent releases. Thereby, openMHA provides a growing benchmark for the development and investigation of novel algorithms on this platform in the future. With the openMHA we provide an open-source tool that is tailored to the needs of hearing aid algorithm research which was not available before as a specialized tool in the open-source domain.

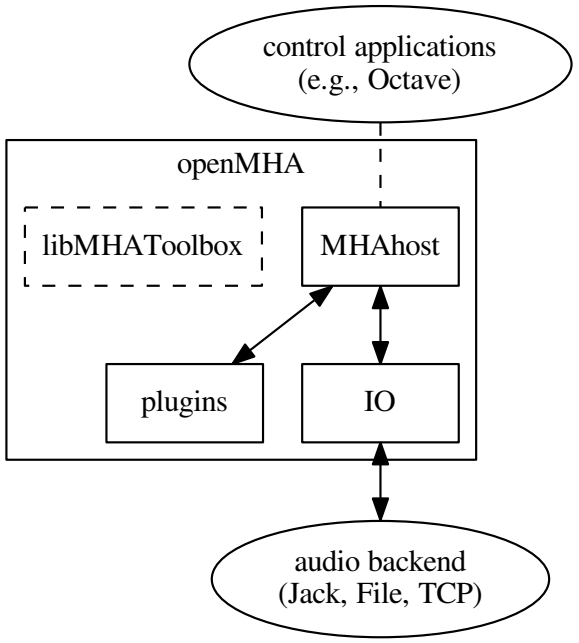

Figure 1: Structure of the openMHA. The openMHA contains a toolbox library "libMHAToolbox", a command line host application, which acts as an openMHA plugin host and provides the configuration interface, and openMHA plugins.

2 Structure

The openMHA can be split into four major components (see Figure 1 for an overview):

1. The openMHA command line application
2. Signal processing plugins
3. Audio input-output (IO) plugins
4. The openMHA toolbox library

The openMHA command line application acts as a plugin host. It can load signal processing plugins as well as audio input-output (IO) plugins. Additionally, it provides the command line configuration interface and a TCP/IP based configuration interface. Several IO plugins exist: For real-time signal processing, commonly the "MHAIOJack" plugin is used, which provides an interface to the Jack Audio Connection Kit (JACK) [Davis, 2003]. Other IO plugins provide audio file access or TCP/IP-based processing.

openMHA plugins provide the audio signal processing capabilities and audio signal handling. Typically, one openMHA plugin implements one specific algorithm. The complete virtual hearing aid signal processing can be achieved by a combination of several openMHA plugins.

The openMHA toolbox library "libMHAToolbox" provides reusable data structures and signal processing classes. Examples are class templates for the implementation of openMHA plugins, and container classes for audio data. Furthermore, several filter classes in temporal or spectral domain, filter banks, and hearing aid specific classes are provided in this library.

3 openMHA Platform Services and Conventions

The openMHA platform offers some services and conventions to algorithms implemented in plugins, that make it especially well suited to develop hearing aid algorithms, while still supporting general-purpose signal processing.

3.1 Audio Signal Domains

As in most other plugin hosts, the audio signal in the openMHA is processed in audio chunks. However, plugins are not restricted to propagate audio signal as blocks of audio samples in the time domain – another option is to propagate the audio signal in the short time Fourier transform (STFT) domain, i.e. as spectra of blocks of audio signal, so that not every plugin has to perform its own STFT analysis and synthesis. Since STFT analysis and re-synthesis of acceptable audio quality always introduces an algorithmic delay, sharing STFT data is a necessity for a hearing aid signal processing platform, because the overall delay of the complete processing has to be as short as possible.

Similar to some other platforms, the openMHA allows also arbitrary data to be exchanged between plugins through a mechanism called "algorithm communication variables" or short "AC vars". This mechanism is commonly used to share data such as filter coefficients or filter states.

3.2 Real-Time Safe Complex Configuration Changes

Hearing aid algorithms in the openMHA can export configuration settings that may be changed by the user at run time. To ensure real-time safe signal processing, the audio processing will normally be done in a signal processing thread with real-time priority, while user interaction with configuration parameters would be performed in a configuration thread with normal priority, so that the audio processing does not get interrupted by configuration tasks. Two types of problems may occur when the user is changing parameters in such a setup:

1. The change of a simple parameter exposed to the user may cause an involved recalculation of internal runtime parameters that the algorithm actually uses in processing. The duration required to perform this recalculation may be a significant portion of (or take even longer than) the time available to process one block of audio signal. In hearing aid usage, it is not acceptable to halt audio processing for the duration that the recalculation may require.

2. If the user needs to change multiple parameters to reach a desired configuration state of an algorithm from the original configuration state, then it may not be acceptable that processing is performed while some of the parameters have already been changed while others still retain their original values. It is also not acceptable to interrupt signal processing until all pending configuration changes have been performed.

The openMHA provides a mechanism in its toolbox library to enable real-time safe configuration changes in openMHA plugins: Basically, existing runtime configurations are used in the processing thread until the work of creating an

updated runtime configuration has been completed in the configuration thread. In hearing aids, it is more acceptable to continue to use an outdated configuration for a few more milliseconds than blocking all processing. The openMHA toolbox library provides an easy-to-use mechanism to integrate real-time safe runtime configuration updates into every plugin.

3.3 Plugins can Themselves Host Other Plugins

An openMHA plugin can itself act as a plugin host. This allows to combine analysis and resynthesis methods in a single plugin. We call plugins that can themselves load other plugins "bridge plugins" in the openMHA. When such a bridge plugin is then called by the openMHA to process one block of signal, it will first perform its analysis, then invoke (as a function call) the signal processing in the loaded plugin to process the block of signal in the analysis domain, wait to receive a processed block of signal in the analysis domain back from the loaded plugin when the signal processing function call to that plugin returns, then perform the re-synthesis transform, and finally return the block of processed signal in the original domain back to the caller of the bridge plugin.

3.4 Central Calibration

The purpose of hearing aid signal processing is to enhance the sound for hearing impaired listeners. Hearing impairment generally means that people suffering from it have increased hearing thresholds, i.e. soft sounds that are audible for normal hearing listeners may be imperceptible for hearing impaired listeners. To provide accurate signal enhancement for hearing impaired people, hearing aid signal processing algorithms have to be able to determine the absolute physical sound pressure level corresponding to a digital signal given to any openMHA plugin for processing. Inside the openMHA, we achieve this with the following convention: The single-precision floating point time-domain sound signal samples, that are processed inside the openMHA plugins in blocks of short durations, have the physical pressure unit Pascal ($1\mathrm{Pa} = 1\mathrm{N/m}^2$). With this convention in place, all plugins can determine the absolute physical sound pressure level from the sound samples that they process. A derived convention is employed in the spectral domain for STFT signals. Due to the dependency of the calibration on the hardware used, it is the responsibility of the user of the openMHA to perform calibration measurements and adapt the openMHA settings to make sure that this calibration convention is met. We provide the plugin *transducers* (cf. section 4.1) which can be configured to perform the necessary signal adjustments in most situations.

4 February 2017 Pre-Release

In February 2017, HörTech and Universität Oldenburg published a pre-release of the openMHA on GitHub under an open-source license (AGPL3). This pre-release contains the openMHA command line application, the toolbox library "libMHAToolbox", an initial set of openMHA plugins and openMHA sound input/output (IO) libraries, and example configurations. The initial set of plugins and sound IO libraries was selected so that a basic research hearing aid configuration can be realized with the contained plugins, and users could process both, live sounds via JACK as well as sound from and to files. The basic hearing aid algorithms present in the pre-release include

- an adaptive differential microphone algorithm that suppresses interfering noise from the rear hemisphere (cf. section 4.3),
- a binaural coherence filter that provides feedback suppression and dereverberation (cf. section 4.5), and
- a multi-band dynamic range compression algorithm that restores audibility of sounds for the hearing impaired user (cf. section 4.7).

Apart from the plugins that implement just these algorithms, additional supporting plugins are contained in the pre-release that are required to form a complete hearing aid implementation. The contained plugins are briefly described in the following subsections.

For real-time hearing aid processing, an input-output delay below 10 ms is required. This ensures that

- the hearing-impaired user is not confused by asynchrony between lip movements of a conversation partner and the perceived sound,
- no echo-effects are audible if the direct sound can also be perceived by the hearing aid user, and

- fewer frequencies are available for possibly annoying acoustic feedback loops [Grimm et al., 2009a].

The example configuration that combines all three example algorithms mentioned here shows an algorithmic delay of 4.4 ms. On top of this algorithmic delay, input and output of the sound through a sound card causes additional delay in the range of two to three block durations depending on the hardware in use. The example configuration uses a block size of 64 samples at 44100 Hz sampling rate. We have found, that e.g. with the RME Multiface II sound card and the snd-hdsp alsa driver used by JACK, this will add 4.4 ms delay between acoustic input and output on a Linux system with a low-latency kernel and real-time priorities set up for JACK and the alsa sound driver.

This results in an overall delay of 8.8 ms of the example configuration containing the plugins described in the following in the order of their processing.

4.1 The *transducers* Plugin

A device-dependent calibration is required for plugins to be able to deduce the physical signal level that is present at the hearing aid input. When connecting a microphone to a sound card and using that sound card to feed sound samples to the openMHA, these sound samples do not automatically follow the openMHA level convention outlined in section 3.4. The same is true when using sound files instead of sound cards for input and output. Different microphones have different sensitivities. Sound cards have adjustable amplification settings. Sound files may have been normalized before they have been saved to disk. To be able to implement the openMHA level convention, i.e., that the numeric value of time-domain sound samples in the openMHA should reflect their sound pressure amplitude in Pascal, we need to be able to adjust for arbitrary physical level to digital level mappings in the openMHA. This is done with the help of the plugin *transducers*, which is the only plugin that must not rely on this convention, because it is the one plugin that has to make sure that all other plugins can rely on this convention. For this reason, *transducers* is usually loaded as the first plugin into the openMHA, and will itself (i.e. as a bridge plugin, cf. section 3.3) load another openMHA plugin into the openMHA process. This other plugin receives the calibrated input signal from *transducers*, and it sends its processed but still calibrated output signal back to the *transducers* plugin to adjust for the physical outputs. *transducers* provides filters and gain adjustments to ensure calibration of inputs and outputs. Typical output calibration values are in the order of 110 dB SPL of a full-scale signal.

4.2 The *mhachain* Plugin

An *mhachain* plugin can itself load several other plugins in a configurable order, where each plugin processes the output signal of the previous plugin.

4.3 The Adaptive Differential Microphone (*adm*) Plugin

Reduced audibility of soft sounds is not the only problem that hearing impaired listeners face when communicating. Another commonly experienced problem is a reduced intelligibility of speech in noisy environments, even if the speech is loud enough to be perceived. Hearing aids therefore regularly employ signal processing algorithms to enhance the signal-to-noise ratio of speech in noisy environments. In this context assumptions about target and noise sources play an important role as well as robustness and generalization capabilities of the method used. Adaptive differential microphones (ADM, [Elko and Pong, 1995]) aim at the preservation of a target signal while suppressing background noise. For this purpose, two general assumptions are made: the target is assumed to be present in the frontal hemisphere of a listener, while noise occurs in the rear hemisphere. ADMs work for pairs of omnidirectional microphones separated by a small distance, and combine a two-channel input to a single-channel output signal by adding up delayed and weighted versions of the input as shown in Figure 2. In a binaural setting two independent, bilateral ADMs are realized, each using a two-microphone pair located in the a hearing aid device on one ear.

4.4 The *overlapadd* Plugin

overlapadd is one of the openMHA plugins that perform conversion between time domain and spectral domain as a service for algorithms that process a series of short time Fourier transform (STFT) signals. Thereby, not every openMHA plugin that processes spectral signal has to perform its own spectral analysis.

overlapadd is a bridge plugin (cf. section 3.3) and performs both, the forward and

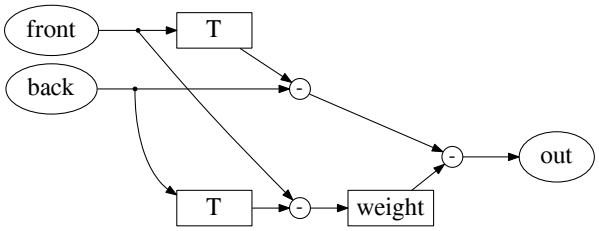

Figure 2: Adaptive differential microphone signal flowchart. The input of the *front* and *back* microphone is combined to a single-channel output after applying a delay T and a weighting.

the backward transform, and can load another openMHA plugin which analyses and modifies the signal while in the spectral domain. The plugin performs the standard process of collecting the input signal, windowing, zero-padding, fast Fourier transform, inverse fast Fourier transform, additional windowing, and overlap-add time signal output. It can be used in standard overlap-add (OLA) and weighted overlap-add (WOLA) contexts.

4.5 The Binaural *coherence* Filter Plugin

An important issue in hearing aid processing is the reduction of feedback that can occur between the hearing aid receivers (outputs) and the closely located inputs (microphones). At high output levels a sound loop can emerge, causing annoying, self-sustaining beep tones.

Binaural coherence filtering, i.e., coherence-based gain control is applied to reduce this effect and enable higher gain levels of the hearing device [Grimm et al., 2009b].

Figure 3 shows that the binaural coherence is measured between the left and the right input signals to the hearing aids and used to derive frequency-dependent gains.

Coherence filtering also contributes to noise and reverberation reduction, as diffuse, incoherent background sounds are also reduced. A combination the binaural coherence filtering with preceding bilateral ADMs was shown to be beneficial, i.e., increased speech intelligibility with a binaural hearing aid setup [Baumgärtel et al., 2015].

4.6 The *fftfilterbank* Plugin

In the hearing impaired, the hearing loss generally varies with frequency. To restore audibility in hearing impaired listeners with amplification and compression in hearing aid signal processing, it is therefore common practice to amplify

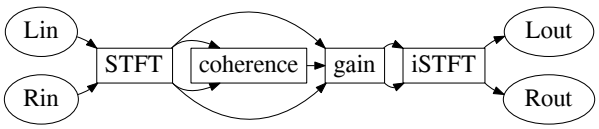

Figure 3: Coherence filter signal flowchart. Binaural coherence-based gain control is applied to the left and the right input channel in different frequency bands in the STFT domain.

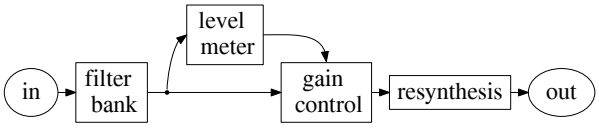

Figure 4: Dynamic compression signal flowchart. The input is split into frequency bands by a *filter-bank*. Before *re-synthesis*, an input-level dependent gain rule is applied.

and compress the signal differently in different frequency bands, and let the time-varying input level in the different frequency bands control the gain selection. The *fftfilterbank* plugin receives broadband spectra for each audio channel and divides the incoming spectra into multiple narrower frequency bands for processing by the following openMHA algorithms. The *fftfilterbank* provides flexibility for filter-bank design. The output frequency bands may overlap or not, with variable degrees of overlap, with customizable filter shapes and different frequency scales to specify the edge or center frequencies of the filters.

4.7 Hearing Loss Compensation (*dc*)

The *dc* plugin applies Multi-band dynamic range compression [Grimm et al., 2015] to the signal. This operation serves two important aspects in a hearing aid: The hearing loss is compensated by defining gain rules between input and output level. Specific gain rules are also used to compensate recruitment effects that often comes along with a hearing loss, i.e., a decreased range between the percept of a soft sound and the loudest sound with a still comfortable level. To compensate for this effect, soft input sounds are usually amplified with higher gains than loud sounds. The *dc* plugin allows to specify a gain-matrix with different gains for different frequencies and input sound levels. Input sound levels in hearing aid frequency bands are commonly measured with attack-release level filters, the time constants of which can be freely configured in the *dc* plugin. Figure 4 shows the

signal flow for dynamic compression with the *dc* plugin. The *dc* plugin also allows to configure binaural and inter-frequency interactions of gain derivation.

4.8 The *combinechannels* Plugin

Because the *fftfilterbank* splits broadband signals into frequency bands for processing by the *dc* plugin, these frequency bands have to be recombined to broadband channels again, after *dc* has processed them. This is done in the *combinechannels* plugin. Of course, the *fftfilterbank* and *combinechannels* plugins could be combined into a single bridge plugin (cf. section 3.3). This would generally be a better implementation choice. It is not done here to showcase the flexibility of the openMHA platform: It is also possible to have analysis and resynthesis of some transform as separate plugins, and to propagate the signal from one plugin to the next inside a single *mhachain* plugin while the domain changes from one plugin to the next (here: few broadband channels vs many narrowband channels).

5 Software

openMHA is a command line application with no graphical user interface (GUI) of its own. openMHA can be configured with command line parameters, configuration files, interactively over a network connection, or by a combination of all three methods. The same text-based configuration language is used in all three methods. Special-purpose GUIs can be produced to control the openMHA over the network connection. Such GUIs can be produced in any programming language or framework that is able to connect to the openMHA over a TCP network connection. Some special-purpose GUIs exist for the closed-source MHA that also work with the openMHA, but are not yet part of the first open-source pre-release. GUIs will be added in later releases of the openMHA.

5.1 Configuration Interface

The openMHA application itself and also its plugins are controlled through a simple, text-based configuration language. The language allows hierarchical configuration similar to the concept of Octave and Matlab structures. The configuration language enables variable assignments, queries, and loading and saving of configuration files. Variables of different types (integers, floating point and complex numbers, strings) and dimensions (scalars, vectors, matrices) are supported. For more details, please refer to [Grimm et al., 2006].

5.2 Plugin Development

New plugins can be developed for the openMHA by implementing a C++ class derived from a generic base class, implementing the methods and compiling it to a shared object. Together with other helper classes provided by the MHA-Toolbox library, out-of-the box support for exporting variables to the configuration interface (cf. section 5.1) and for thread safe configuration updates (cf. section 3.2) is available.

Simple plugins will usually output the signal in the same domain (spectrum or waveform) as the input domain. It is also possible to implement domain transformations (from the time domain to spectrum or vice versa) inside a plugin, as well as change the number of audio channels, and even the number of audio samples per block and the sampling rate (e.g. for resampling).

A detailed manual for plugin development and implementation will be provided with a near-future release.

6 Conclusions

The openMHA provides the means for sustainable research on and development of hearing aid processing algorithms and assistive hearing systems. The software is further developed in the project "Open community platform for hearing aid algorithm research", additionally, updates based on the feedback of the research community will be conducted. Future work will extend the openMHA in several directions: The set of reference algorithms will be expanded and experimental algorithms will be included. Additional hardware and operation systems will be included, i.e., real-time runtime support for Beaglebone Black ARM and similar platforms, as well as support for Windows operations systems. Increased usability on different user levels is achieved by the preparation of a GUI for the pure application of the openMHA, e.g., in the context of audiological measurements, availability of reference manuals for the configuration as well as the implementation of plugins for realization and implementation of own algorithms and methods and their evaluation.

The openMHA is intended to serve as a platform for extensive research and evaluations by the community. A pre-release of the software in

its current version including example configuration files as described here can be downloaded via `http://www.openmha.org`.

7 Acknowledgments

The project "Open community platform for hearing aid algorithm research" is funded by the National Institutes of Health (NIH Grant 1R01DC015429-01).

References

Regina M. Baumgärtel, Martin Krawczyk-Becker, Daniel Marquardt, Christoph Völker, Hongmei Hu, Tobias Herzke, Graham Coleman, Kamil Adiloglu, Stephan M. A. Ernst, Timo Gerkmann, Simon Doclo, Birger Kollmeier, Volker Hohmann, and Mathias Dietz. 2015. Comparing Binaural Preprocessing Strategies I: Instrumental Evaluation. *Trends in Hearing*, 19:article No. 2331216515617916.

Perry R Cook and Gary P Scavone. 1999. The synthesis toolkit (stk). In *ICMC*.

Paul Davis. 2003. Jack audio connection kit. http://jackaudio.org/.

John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. 2015. GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations. `http://www.gnu.org/software/octave/doc/interpreter`.

G. W. Elko and Anh-Tho Nguyen Pong. 1995. A Simple Adaptive First-order Differential Microphone. In *Proceedings of 1995 Workshop on Applications of Signal Processing to Audio and Accoustics*, pages 169–172.

Giso Grimm, Tobias Herzke, Daniel Berg, and Volker Hohmann. 2006. The Master Hearing Aid: a PC-based Platform for Algorithm Development and Evaluation. *Acta acustica united with Acustica*, 92:618–628.

Giso Grimm, Tobias Herzke, and Volker Hohmann. 2009a. Application of Linux Audio in Hearing Aid Research. In *Linux Audio Conference 2009*.

Giso Grimm, Volker Hohmann, and Birger Kollmeier. 2009b. Increase and Subjective Evaluation of Feedback Stability in Hearing Aids by a Binaural Coherence-based Noise Reduction Scheme. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(7):1408–1419.

Giso Grimm, Tobias Herzke, Stephan Ewert, and Volker Hohmann. 2015. Implementation and Evaluation of an Experimental Hearing Aid Dynamic Range Compressor Gain Prescription. In *DAGA 2015*, pages 996–999.

HörTech gGmbH and Universität Oldenburg. 2017. openMHA web site on GitHub. `http://www.openmha.org/`.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`.

James McCartney. 2002. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68.

Yann Orlarey, Dominique Fober, and Stéphane Letz. 2009. Faust: an efficient functional approach to dsp programming. *New Computational Paradigms for Computer Music*, 290.

Miller Puckette. 1996–. Pure data. `https://puredata.info/`.