

The Open Master Hearing Aid (openMHA)

4.13.0

Getting Started



HörTech

Kompetenzzentrum für
Hörgeräte-Systemtechnik

The Open Master Hearing Aid (openMHA) – Getting Started
HörTech gGmbH
Marie-Curie-Str. 2
D-26129 Oldenburg

LICENSE AGREEMENT

This file is part of the HörTech Open Master Hearing Aid (openMHA)

Copyright © 2005 2006 2007 2008 2009 2010 2012 2013 2014 2015 2016 HörTech gGmbH.

Copyright © 2017 2018 2019 2020 HörTech gGmbH.

openMHA is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, version 3 of the License.

openMHA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License, version 3 for more details.

You should have received a copy of the GNU Affero General Public License, version 3 along with openMHA. If not, see <<http://www.gnu.org/licenses/>>.

Contents

1	Introduction	1
1.1	About this Manual	1
1.2	Structure	1
1.3	Platform Services and Conventions	2
2	Requirements	2
2.1	Required Programs	2
2.2	Update to Latest Version	3
2.3	System-Specific Settings	3
3	Getting Started	3
3.1	Starting openMHA	3
4	Step-by-Step Exercise: Gain Application	5
4.1	File to File	7
4.2	Starting openMHA with JACK Input/Output	8
5	Control Frequency Shifter using Octave/Matlab GUI	11
6	Control Dynamic Compression using Octave/Matlab GUI	13
7	Using AC Variables	15
8	Writing your own Configuration Script	18

1 Introduction

The HörTech *open Master Hearing Aid* (openMHA), is a development and evaluation software 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.

1.1 About this Manual

This manual provides instructions for first steps to be taken when starting to work with openMHA. After outlining the purpose and basic structure of openMHA the user is guided through the installation and invocation of the openMHA command line application. Then, some basic configurations and step-by-step instructions on how to run them are presented in order to give a first insight how openMHA is controlled. Furthermore, tools are introduced that are helpful to operate openMHA such as using the Jack Audio Connection Kit (JACK) with the openMHA, invoking openMHA inside Matlab/Octave, and basic instructions on writing your own configuration are given.

1.2 Structure

The openMHA can be split into four major components:

- The openMHA command line application (MHA)
- Signal processing plugins (plugins)
- Audio input-output modules (IO)
- The openMHA toolbox library (libopenmha)

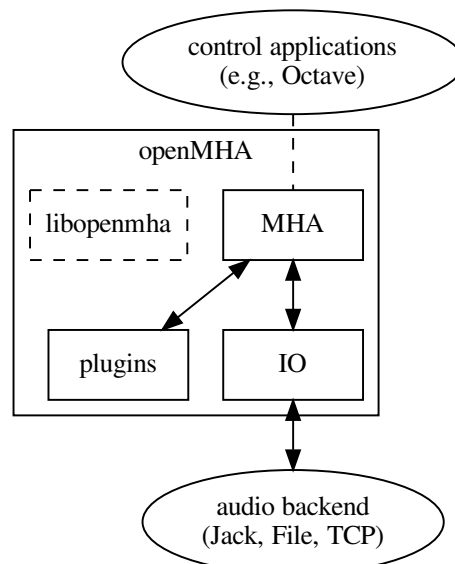


Figure 1 Layered structure of the open Master Hearing Aid

The MHA command line application acts as a plugin host. It can load signal processing plugins as well as audio input-output modules (IO). Additionally, it provides the command line configuration interface and a TCP/IP based configuration interface. Different IO modules exist: For real-time signal processing, commonly the openMHA *MHAIOJack* module is used, which

provides an interface to the Jack Audio Connection Kit (JACK), the module *MHAIOFile* provide audio file access and *MHAIOTCP* TCP/IP-based signal exchange.

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

1.3 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.

As in most other plugin hosts, the audio signal in the openMHA is processed in fragments, i.e., in chunks of the input signal stream with a defined length. However, plugins are not restricted to propagate audio signal as fragments 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 fragments 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 in order to achieve a sufficiently low delay for the whole processing chain. Sharing non-audio information between the plugins is achieved by using algorithm communication (AC) variables. They will be discussed further in section 7 of this manual.

2 Requirements

2.1 Required Programs

Please install the following software to work with this guide:

- Operating System
 - **Linux:** Ubuntu 18.04 or 20.04, 64 bits
 - **Windows:** Windows 10, 64 bits
 - **macOS:** High Sierra or later
- openMHA
 - <https://github.com/HoerTech-gGmbH/openMHA/blob/master/INSTALLATION.md>
- either Octave or Matlab
 - Octave:
 - * **Linux:**
 - `sudo apt install octave-signal`
 - * **Windows, macOS:**
 - <https://www.gnu.org/software/octave/download.html>
 - Matlab:
 - <https://www.mathworks.com/downloads/>
- JACK Audio Connection Kit
 - **Linux:**
 - `sudo apt install jackd2 qjackctl`
 - **Windows, macOS:**
 - <http://jackaudio.org/downloads/>

2.2 Update to Latest Version

This guide was released with openMHA version 4.13.0. If you have already installed openMHA on your system, make sure that you are using the latest version.

- **Windows, macOS**

Repeat the installation process using the latest **Windows installer** or the latest **macOS installer** respectively. The installation instructions can be found at <https://github.com/HoerTech-gGmbH/openMHA/blob/master/INSTALLATION.md>

- **Linux**

In Linux, all installed openMHA packages need to be updated. For updating openMHA when a new release is available, execute:

```
sudo apt-get update
sudo apt-get install openmha
```

This will upgrade all installed openmha packages to their latest version.

2.3 System-Specific Settings

- **Linux**

- Add your user to the `audio` group (replace `YourUserName` with your actual user name on the Linux system):

```
→ sudo adduser YourUserName audio
```

- Install a low-latency Linux kernel:

```
→ sudo apt install linux-image-lowlatency
```

- Reboot the computer to use the new kernel and to activate the group membership.

- **Windows, macOS**

Ensure that your Octave/Matlab installation can make use of Java. Test by executing in the Octave/Matlab command window:

```
→ javaclasspath
```

If this responds with "STATIC JAVA PATH ... DYNAMIC JAVA PATH ..." then Java is set up correctly (even if there is also a warning).

If Octave/Matlab responds with an error then you need to install a suitable Java Runtime Environment on your computer, restart Octave/Matlab and test again. Refer to Octave/-Matlab documentation for details.

3 Getting Started

3.1 Starting openMHA

After openMHA and its dependencies have been installed (see section 2.1) you can start openMHA by:

Linux

In order to start openMHA open your **terminal** and type:

```
→ mha --interactive
```

Windows

Open your **terminal** by:

→ "**Windows + R**"

→ type in **cmd** and press **Enter**

Now type **mha --interactive** into the terminal window.

macOS

Open your **terminal** by pressing **Command + Space** in order to open spotlight search, type "terminal" and press **Enter**. Type:

→ **mha --interactive**

in your **terminal**.

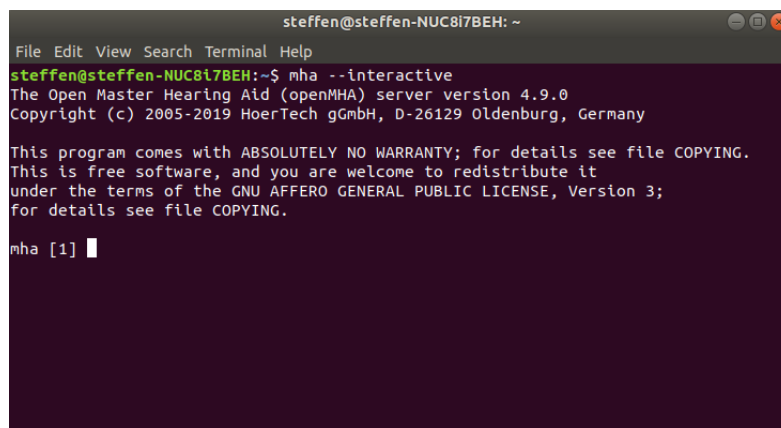
A screenshot of a Linux terminal window. The title bar shows 'steffen@steffen-NUC8i7BEH: ~'. The terminal has a menu bar with 'File Edit View Search Terminal Help'. The prompt is 'steffen@steffen-NUC8i7BEH:~\$'. The command 'mha --interactive' has been entered. The output shows: 'The Open Master Hearing Aid (openMHA) server version 4.9.0', 'Copyright (c) 2005-2019 HoerTech gGmbH, D-26129 Oldenburg, Germany', a disclaimer about warranty and redistribution, and finally 'mha [1]' with a cursor.

Figure 2 - Linux Terminal: Type `mha --interactive`

.....

Note: The current directory of the terminal becomes the current working directory (CWD) of the openMHA process. openMHA resolves relative file names relative to the CWD. If in some of the following examples in this guide openMHA raises an error because it cannot find some file, check that the file name can be resolved from the CWD. To fix file lookup problems, either change the CWD in the terminal before restarting the openMHA, or adapt any file names to include correct absolute or relative paths.

You have managed to start openMHA. In the next section there will be a step-by-step tutorial on how to use a simple configuration.

In order to terminate the openMHA started in this chapter, you can type

→ **cmd=quit**

followed by **Enter** into the terminal.

4 Step-by-Step Exercise: Gain Application

First Steps

A simple openMHA use case is the application of a gain to an audio signal. We will start by applying a gain factor to an audio file named *1speaker_diffNoise_2ch.wav*. The corresponding parameters (e.g. gain factor, input channels, fragment size and sample size) can be set manually, however for this example there is already an openMHA configuration file available at:

- **Linux:** */usr/share/openmha/examples/00-gain*
- **Windows:** *C:\Program Files\openMHA\examples\00-gain*
- **macOS:** */usr/local/share/openmha/examples/00-gain*

A shortened version of the gain.cfg file is shown below. A short description precedes each command in a line starting with # which is used for comments. The actual file on disk contains more verbose comments.

gain_getting_started.cfg:

```
1 #The number of channels we want to process
2 nchannels_in = 2
3 #Number of frames to be processed in each block.
4 fragsize = 64
5 #Sampling rate. Has to be the same as the input file
6 srate = 44100
7 #We want to use the plugin "mhachain"
8 mhalib = mhachain
9 #Now we need to define input-output backend "iolib"
10 #Here we decide if the audio should come from a static audio
11 #file or from e.g. a live input source such as a microphone
12 #input
13 #In this case we will use simple static audio files
14 iolib = MHAIOFile
15 #The plugin "mhachain" can load multiple plugins and
16 #will connect them in series which is denoted by "[...]"
17 #Here we will only use one plugin "gain"
18 mha.algos=[ gain ]
19 #Set max and min gain factors in dB
20 mha.gain.min=-20
21 mha.gain.max=20
22 #nchannels_in was set to 2 (see line 2), so we have to define
23 #two gain factors (left and right)
24 mha.gain.gains=[ -10 10 ]
25 #Define the name of the input and output file
26 #The input file needs to be in the same directory
27 #as the .cfg file itself
28 io.in = 1speaker_diffNoise_2ch.wav
29 io.out = 1speaker_diffNoise_2ch_OUT.wav
```

In this guide we will use the example files from the openMHA installer. Since these are installed in a read-only directory, we need to copy the examples to a writable location before using them. Follow these steps:

1. **Close all** running openMHA processes
2. **Copy** the examples folder from the installation directory (e.g. `/usr/share/openmha/examples/`) (**see the list on page 5 for your specific operating system**)

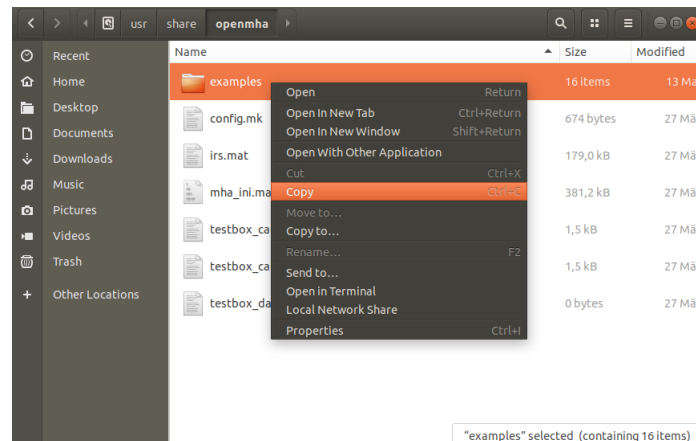


Figure 3 - Copy the examples folder from the protected directory (e.g. `/usr/share/openmha/`)

3. **Paste** the examples into folder within a writable directory, e.g. your home directory:

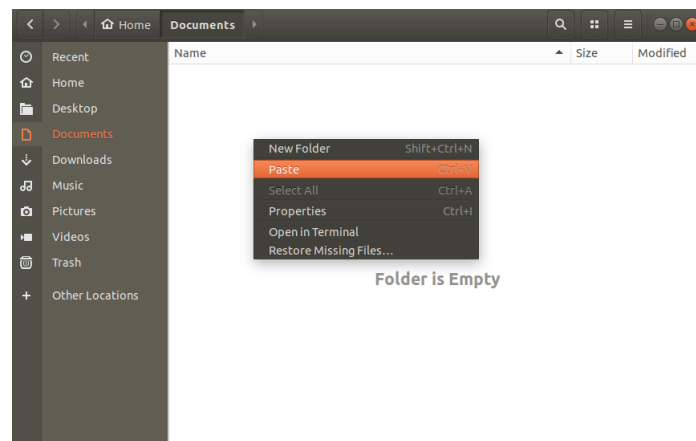


Figure 4 - Paste examples into folder inside a non-protected directory (e.g. `/home/YourUserName/Documents`)

4. Open your **terminal** (see [3.1](#))
5. **Navigate inside the examples folder into the subdirectory of the first example** → **00-gain**
(e.g. `/home/YourUserName/Documents/examples/00-gain`)

→ you can use **cd ..** to navigate one folder level up

- and **cd foldername** to descend into the subfolder **foldername**
- if the **macOS** or **Linux** terminal does not show the current directory, type **pwd**

6. Type `mha --interactive` and press **Enter**

4.1 File to File

You have started the openMHA in interactive mode and can now type in openMHA commands. The current working directory of the openMHA should be the copy of the 00-gain example directory in the writable location. In order to read in the configuration file **gain.cfg** (which lies directly in *00-gain*), type:

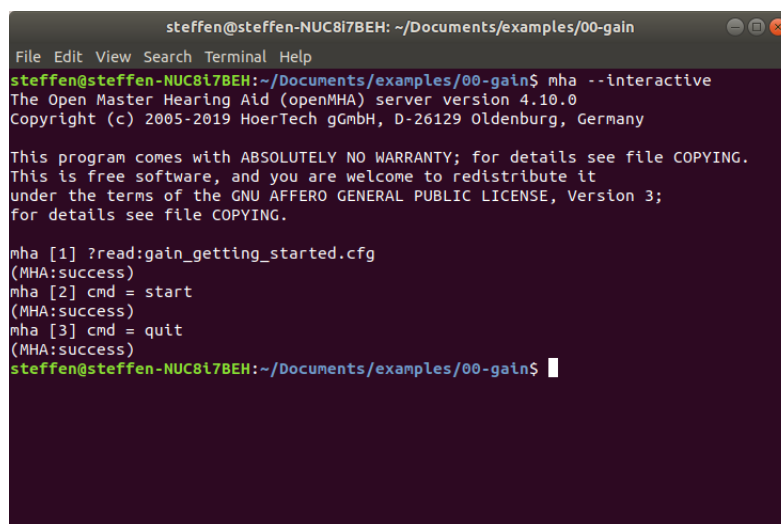
→ `?read:gain_getting_started.cfg`

Start the openMHA signal processing by:

→ `cmd=start`

and then exit openMHA by typing:

→ `cmd=quit`



```

steffen@steffen-NUC8i7BEH: ~/Documents/examples/00-gain
File Edit View Search Terminal Help
steffen@steffen-NUC8i7BEH:~/Documents/examples/00-gain$ mha --interactive
The Open Master Hearing Aid (openMHA) server version 4.10.0
Copyright (c) 2005-2019 HoerTech gGmbH, D-26129 Oldenburg, Germany

This program comes with ABSOLUTELY NO WARRANTY; for details see file COPYING.
This is free software, and you are welcome to redistribute it
under the terms of the GNU AFFERO GENERAL PUBLIC LICENSE, Version 3;
for details see file COPYING.

mha [1] ?read:gain_getting_started.cfg
(MHA:success)
mha [2] cmd = start
(MHA:success)
mha [3] cmd = quit
(MHA:success)
steffen@steffen-NUC8i7BEH:~/Documents/examples/00-gain$

```

Figure 5 Interactive mode: Applying gain to a static audio signal

openMHA has created a second .wav file "1speaker_diffNoise_2ch_OUT.wav" in the current 00-gain folder. (e.g. /home/YourUserName/Documents/examples/00-gain). You can listen to it and compare it to "1speaker_diffNoise_2ch.wav".

4.2 Starting openMHA with JACK Input/Output

In this section we perform the same signal processing as before, but replace the sound files with the JACK server as audio backend. This means that we can apply a gain to e.g. our microphone input in real time. The configuration file *gain_live.cfg* will be used for this.

Note: This and all following live processing examples configure the sound card with small buffer sizes. Your combination of computer, sound card, and operating system may be unable to process the sound with these settings without dropouts. If you experience problems, try a faster computer, optimize the operating system for real-time performance, or use a different sound card or operating system.

gain_live_getting_started.cfg:

```

1 #The number of channels we want to process
2 nchannels_in = 2
3 #Number of frames to be processed in each block.
4 fragsize = 64
5 #Sampling rate. Has to be the same as the input signal of JACK
6 srate = 44100
7 #Again, we want to use the plugin "mhachain"
8 mhalib = mhachain
9 #Here we will only use one plugin "gain"
10 mha.algos=[ gain ]
11 #Set max and min gain factors in dB
12 mha.gain.min=-20
13 mha.gain.max=20
14 #two gain factors (left and right)
15 mha.gain.gains=[ -10 10 ]
16 #In this example, we load the IO library that connects
17 #the MHA to the Jack audio server.
18 iolib = MHAIOJackdb
19 # The following variable is used to select the input sound
20 # channel(s), following the usual Jack nomenclature
21 io.con_in = [system:capture_1 system:capture_2]
22 # con_out sets the output channels
23 io.con_out = [system:playback_1 system:playback_2]
```

In order to set up and connect a JACK server you can follow the steps below:

1. Start Jack Audio Connection Kit

- **Linux:**
Type `qjackctl` into your **terminal**.
- **Windows:**
Use the **JACK Control** start menu entry.
- **macOS:**
Start the Jack Audio Connection Kit GUI by starting the *qjackctl* application found in */Applications/Jack/*.

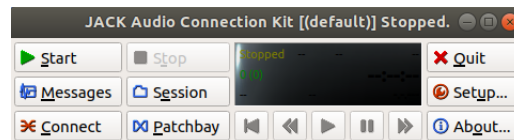


Figure 6 JACK Audio Connection Kit: GUI

2. **Setup** → **Settings** → select proper Driver, Interface, Sample Rate=44100, Frames/Period=64 → **OK**

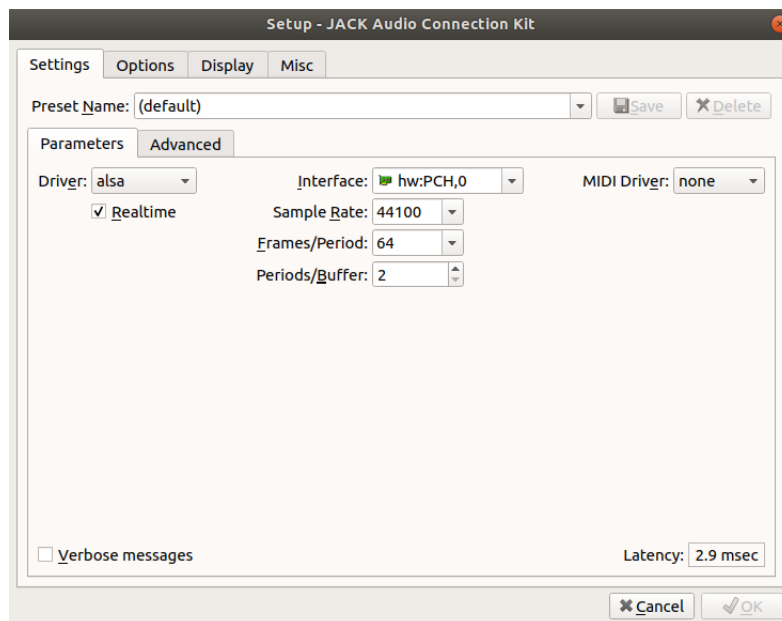


Figure 7 Jack GUI: Setup

3. Click **Start** for starting a JACK server → Check *Messages* for any errors (sometimes it can be difficult to find proper driver settings, try out different settings)

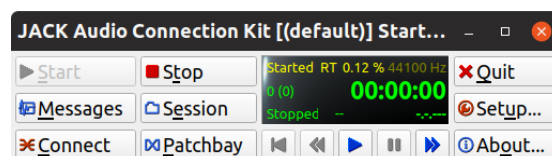


Figure 8 Jack GUI with running server

4. In order to test your Jack server, you can go to the **Connect** section and connect the inputs of your (internal) microphones to the output channels of the jack server (see Figure 9).

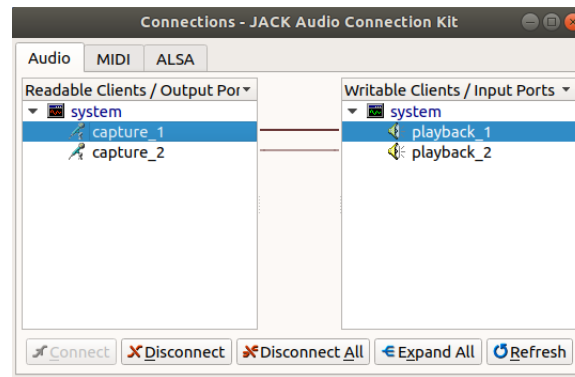


Figure 9 Jack GUI: Setup

Disconnect these connections again before proceeding. You can now use the JACK server as audio backend. To do this, start openMHA in the same directory as before:

5. Open your **terminal** (see [3.1](#))
6. **Navigate inside the examples folder into the subdirectory of the first example → 00-gain**
7. Type `mha --interactive` and press **Enter**
8. → `?read:gain_live_getting_started.cfg`
9. → `cmd=start`
 openMHA is now applying a gain to your own voice input. In order to close openMHA type:
10. → `cmd=quit`

You have now managed to start some simple configurations for a static audio file as well as a live input using Jack. In the next session Matlab or Octave will be used as user interface for openMHA.

5 Control Frequency Shifter using Octave/Matlab GUI

1. **End** all running **mha** processes
2. **Open Matlab or Octave**
3. **Linux:** Set LD_LIBRARY_PATH to empty by typing
 → `setenv('LD_LIBRARY_PATH', '')`
 into the **Command Window**
MacOS only:
 Type:
 → `setenv('PATH', [getenv('PATH') ':/usr/local/bin']);`
 into the **Command Window**
4. Use the Matlab/Octave "**Current Folder**" control to navigate to:
 - **Linux:**
 /usr/share/openmha/examples/05-frequency-shifting
 - **Windows:**
 C:\Program Files\openMHA\examples\05-frequency-shifting
 - **macOS:**
 /usr/local/share/openmha/examples/05-frequency-shifting
5. In order to use the Matlab functions of openMHA type the following using the **Command Window**:
 - **Linux:**
 → `addpath('/usr/lib/openmha/mfiles')`
 - **Windows:**
 → `addpath('C:\Program Files\openMHA\mfiles')`
 - **macOS:**
 → `addpath('/usr/local/lib/openmha/mfiles/')`
6. In order to start a new openMHA instance type
 → `openmha = mha_start;`
7. In order to read in the configuration file type:
 → `mha_query(openmha, '', 'read:fshift_live.cfg');`
8. **Start JACK Server using JACK Control**
 (Setting: Sample Rate = 44100, Frames/Period = 64)
9. Start the mha process by typing
 → `mha_set(openmha, 'cmd', 'start');`
10. **JACK Control:** Connect the "capture" and "playback" channels of the sound card to the MHA "in" and "out" channels. Connect a microphone to the soundcard.
11. Start GUI by typing
 → `mhagui_generic(openmha)` into the **Command Window**
 (a) **mha ->open sub-parser**

- (b) **mhachain ->open sub-parser**
 - (c) **fshift_hilbert ->open sub-parser**
 - (d) **df -> open vector<float>control**
12. Change the settings df, fmin and fmax in the GUI and listen to the processed microphone sound. You can not only move the sliders using the mouse cursor or up- and down-arrow keys, but also replace the numbers directly by typing new numbers and pressing **Enter** in the text fields of the GUI. These settings control a frequency shifter, which band is shifted and how much.

6 Control Dynamic Compression using Octave/Matlab GUI

1. **End** all running **mha processes** (You can type **killall mha** in the terminal to any running mha processes [Linux/macOS only])
2. **Open Matlab or Octave**
3. **Linux:** Set LD_LIBRARY_PATH to empty by typing
 → `setenv('LD_LIBRARY_PATH', '')`
 into the **Command Window**
MacOS only:
 Type:
 → `setenv('PATH', [getenv('PATH') ':/usr/local/bin']);`
 into the **Command Window**
4. Use the Matlab/Octave "**Current Folder**" Section to navigate to:
 - **Linux:** `/usr/share/openmha/examples/01-dynamic-compression`
 - **Windows:** `C:\Program Files\openMHA\examples\01-dynamic-compression`
 - **macOS:** `/usr/local/share/openmha/examples/01-dynamic-compression`
5. In order to use the Matlab functions of openMHA type the following using the **Command Window**:
 - **Linux:** `addpath('/usr/lib/openmha/mfiles')`
 - **Windows:** `addpath('C:\Program Files\openMHA\mfiles')`
 - **macOS:** `addpath('/usr/local/lib/openmha/mfiles/')`
6. In order to start openMHA type `openmha = mha_start;`
7. Read in configuration into mha by
`mha_query(openmha, '', 'read:dynamiccompression_live.cfg');`
8. **Start JACK Server using JACK Control**
 (Setting: Sample Rate = 44100, Frames/Period = 64)
9. In order to start the mha process type `mha_set(openmha, 'cmd', 'start');`
10. The current gaintable `gtdata` and relevant parameters such as `gtmin` and `gtstep` (more information on these can be found under http://www.openmha.org/docs/openMHA_plugins.pdf#subsection.2.1) can be read out by:

```
gaintable = mha_get(openmha, 'mha.overlapadd.mhachain.dc.gtdata');
gtmin = mha_get(openmha, 'mha.overlapadd.mhachain.dc.gtmin');
gtstep = mha_get(openmha, 'mha.overlapadd.mhachain.dc.gtstep');
```

The variables `gtmin` (minimum input level) and `gtstep` (input level increment) expect vectors, but we have assigned scalar values to `gtmin` and `gtstep` previously. Scalar values will be interpreted as vectors of length 1 by the MHA when assigned to vector variables. The dc plugin allows vectors of length 1 for variables `gtmin` and `gtstep`, in this case the same value applies to every channel/band.

It is also possible to have different `gtmin/gtstep` values for each channel/band by assigning a vector of values to these variables where the number of elements in the vector is equal to the number of channels/bands.

11. You can design your own gaintable in Matlab, e.g. noise gate, compressive region, output limit

```
gaintable = repmat([-50, 30:-2:0, -4:-4:-32], 18, 1);
```

with

```
gtmin = zeros(1, size(gaintable, 1));
```

and

```
gtstep = 4*ones(1, size(gaintable, 1));
```

This would result in an input/output characteristic which is the same for every channel/band.

12. Plot the I/O characteristics

```
level_in = ((1:size(gaintable, 2))-1) .* gtstep'+gtmin';
```

```
level_out = level_in + gaintable;
```

13. In order to apply the gaintable type

```
mha_set(openmha, 'mha.overlapadd.mhachain.dc.gtdata', gaintable);
```

14. In order to plot input and output level in Matlab, type

```
figure, plot(level_in', level_out')
```

Note: You can also use the mfile tool `dc_plot_io.m`. For this type:

```
figure, dc_plot_io(gtmin, gtstep, gaintable, level_in);
```

15. You can design more gaintables in Matlab, e.g. by using `gaintable_new = [...]`

- e.g. Squash all input levels to the same output level, infinite compression:

```
gaintable_new = 65.*ones(18, 1) - level_in;
```

- e.g. compress high frequency band only: ...

16. In order to apply the new gaintable type

```
mha_set(openmha, 'mha.overlapadd.mhachain.dc.gtdata', gaintable_new);
```

17. The fitting GUI can be started by typing `mhacontrol(openmha)`

18. You can stop openMHA using `mha_set(openmha, 'cmd', 'quit')`

7 Using AC Variables

The objective of this section is to learn about dealing with AC-Variables in combination with Matlab.

What are AC Variables?

Sometimes plugin algorithms need to share more information than just the current audio signal. openMHA supports this by providing a mechanism to share any type of additional data between plugins in the form of **algorithm communication variables** or **AC variables**. Further information on the purpose of AC Variables can be found in the *Application Manual* section 2.2.

.....

The coherence between two live microphone signals is investigated. A JACK server is used to connect both microphone signals to openMHA.

1. **Start JACK Server** using **JACK Control**
(Setting: Sample Rate = 44100, Frames/Period = 64)
Note: You need to have two microphone inputs available for this task
2. Start Matlab/Octave and the "**Current Folder**" control to navigate to:
 - **Linux**: /usr/share/openmha/examples/15-ac-variables
 - **Windows**: C:\Program Files\openMHA\examples\15-ac-variables
 - **macOS**: /usr/local/share/openmha/examples/15-ac-variables
3. Open the Matlab script **acmatlab.m**

The most important lines of **acmatlab.m** are shown below:

```

1 %Start openMHA process
2 openmha = mha_start;
3 % Read configuration file
4 mha_query(openmha, '', 'read:coherence_live.cfg');
5 % Start configuration file
6 mha_set(openmha, 'cmd', 'start')
7
8 %% Label center frequencies and set gain factor of ac_proc
9
10 % Label center frequencies
11 freqs=mha_get(openmha, 'mha.overlapadd.mhachain.coherence.cf');
12 % Set gain factor in dB - default value was chosen to be 6
13 mha_set(openmha, 'mha.overlapadd.mhachain.coh_gain.gain.gains', 6);

```

Explanation

The configuration used is called *coherence_live.cfg*. It uses the plugin **mhachain** in order to connect three plugins:

1. coherence
2. ac_proc:coh_gain
3. acmon in series.

Within the configuration file this is denoted by:

```
mha.overlapadd.mhachain.algos = [coherence ac_proc:coh_gain acmon]
```

The purpose of each plugin is the following:

coherence: This plugin measures the coherence between the two microphone input signals.

ac_proc:coh_gain: The real name of the plugin is *ac_proc*, however in this case the alias *coh_gain* is used. This plugin interprets the AC variable data stream received from *coherence* as an audio signal. The plugin itself can load another plugin. In this case the gain plugin is loaded. The gain factor was chosen to be 6 dB. This means that the AC variable output "signal" is amplified by 6 dB. (line 12) Outside the plugin *ac_proc* the signal is provided as AC variable stream.

acmon: This plugin is used to convert incoming AC variable data streams into monitor variables. In this case the output of *coherence* as well as *ac_proc:coh_gain* is used.

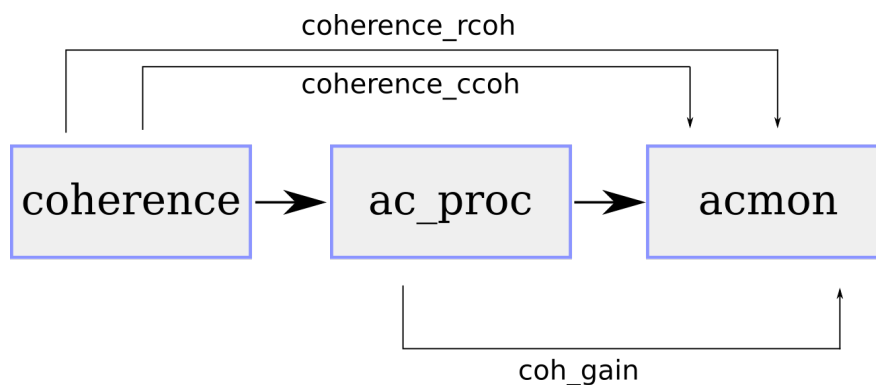


Figure 10 Schematics of AC variable data stream between plugins: coherence, ac_proc, acmon

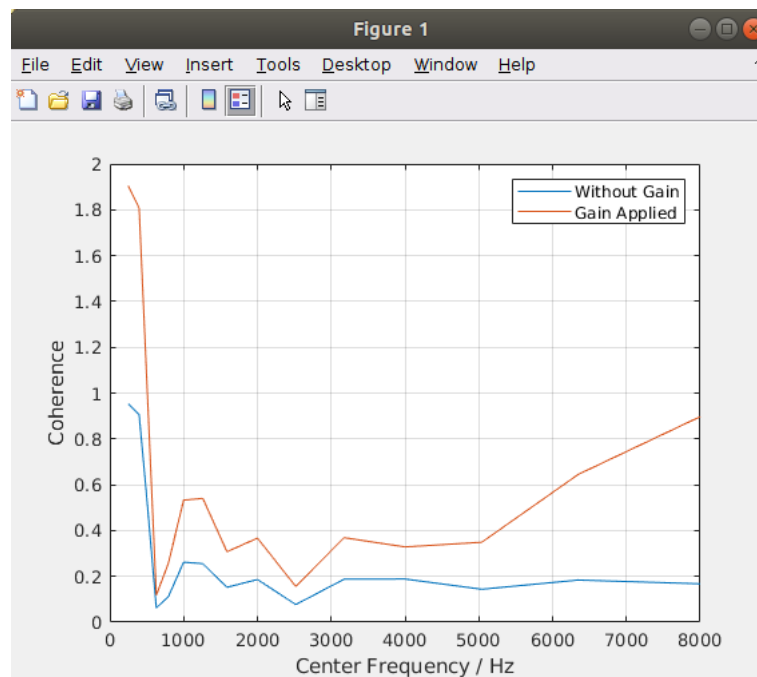
4. Start the Matlab script **acmatlab.m**

Figure 11 Matlab: Coherence plotted as a function of frequency

The purpose of this example is to show that the plugin `ac_proc` can be used to apply common signal processing operations such as a gain to an originally non-audio signal such as an AC variable data stream.

8 Writing your own Configuration Script

This section will offer a guide on how to write your own openMHA script. For doing so some basic parameters need to be set. In this example we want to write an example script for using the *sine* plugin.

First it is necessary to fix the following parameters:

1. **Number of input channels** (`nchannels_in`)
2. **Fragment size** (`fragsize`)
3. **Sampling rate in Hz** (`srate`)

For an input file with two channels and a sampling rate of 44.1 kHz. This could for example look like:

```
#The number of channels we want to process
nchannels_in = 2
#Number of frames to be processed in each block.
fragsize = 64
#Sampling rate. Has to be the same as the input file
srate = 44100
```

4. **Plugin(s):** Next we need to tell the MHA which plugin(s) to load with the `mhalib` variable. As mentioned in the beginning of the section the sine plugin will be used as an example:

```
#We want to use the plugin "sine"
mhalib = sine
```

5. **Configuration variables:** Each plugin has *configuration variables* which can be adjusted accordingly. For each plugin the configuration variables are listed in the *plugin manual* (http://www.openmha.org/docs/openMHA_plugins.pdf#subsection.16.3 for sine plugin). These variables can be adjusted according to your requirements, such as the frequency of the sine wave (`f`), the RMS level (`lev`) or if the input signal should be added to the sine wave or replaced completely (`mode`). An extract from the *plugin manual* is shown below in Figure 12.

Name	Type	Description	Default
mhaconfig_in	parser	Input configuration	(see below)
mhaconfig_out	parser	Output configuration	(see below)
lev	float	sine RMS level in dB SPL FF	0
f	float	Frequency in Hz Range: [0,[0
mode	keyword_list	Replace input signal with tone or mix tone into input signal Range: [replace mix]	replace
channels	vector<int>	List of audio channels to feed with tone (all other audio channels are not affected)	[]

Figure 12 Extract from *Plugin Manual*: Configuration variables of the *sine* plugin

Although the plugin is used by `mhalib = sine`, the variables of the plugin are changed by setting `mha.<variable_name> = value`. This could for example look like:

```
#Adjust configuration variables
#frequency
mha.f = 440
#RMS Level
mha.lev = 100
#Operating mode (can be changed to replace[default])
mha.mode = mix
#Channels on which the sine plugin should operate
mha.channels=[0]
```

Here the frequency was adjusted to 440 Hz and the RMS level of the sine tone was changed to 100 dB. The operating mode was set to `mix`, such that the input is mixed with the sine signal (instead of being replaced). The variable `channels` was adjusted to `[0]`. This means that the sine plugin is only operating on the first channel. In order to operate on both channels `mha.channels = [0 1]` can be used.

6. **Audio Back-End:** openMHA supports different audio back-ends, such as sound drivers, JACK audio servers, sound files or networks. This can be adjusted by choosing the so called IO plugin library (`iolib`). :

For static audio files: e.g. `iolib = MHAIOFile`

For live JACK signals: `iolib = MHAIOJack`

In this case we want to work with a static audio file such that we need to choose:

```
#For live input use 'MHAIOJack'
iolib = MHAIOFile
```

7. Input (`io.in`)

For static audio files: `io.in = name_of_file.wav`

For live JACK signals: `io.con_in = [system:capture_1 system:capture_2]`
(for two input channels)

```
#Define the name of the input file
#The input file needs to be in the same directory
#as the .cfg file itself
io.in = 1speaker_diffNoise_2ch.wav
```

Note: In this case the input file needs to be in the same directory as the `.cfg` file itself. If this is not the case the absolute or relative (to the `.cfg` file) path can be used, e.g.

- `../../Folder/1speaker_diffNoise_2ch.wav` (*relative path*)
- `/home/UserName/Folder/1speaker_diffNoise_2ch.wav` (*absolute path*)

8. Output (io.out)

For static audio files: e.g. `io.out = name_of_file_out.wav`

For live JACK signals: `io.con_out = [system:playback_1 system:playback_2]`
(for two output channels)

```
#Define name of output file
io.out = lspeaker_diffNoise_2ch_OUT.wav
```

How to run your script:

1. Start openMHA in the **same directory** as your .cfg file
2. `→?read:your_script.cfg`
3. Start the openMHA signal processing by:
 `→ cmd=start`
4. In order to exit openMHA type:
 `→ cmd=quit`