The Open Master Hearing Aid (openMHA)

4.10.0

Getting Started

Kompetenzzentrum für Hörgeräte-Systemtechnik

The Open Master Hearing Aid (openMHA) – Getting Started

HörTech gGmbH
Marie-Curie-Str. 2
D–26129 Oldenburg

Contents

1 Requirements

1.1 Required Programs

Please install the following software to work with this guide:

- Operating System
  - **Linux:** Ubuntu 18.04, 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://de.mathworks.com/downloads/
- JACK Audio Connection Kit
  - **Linux**
    → `sudo apt install jackd2 qjackctl`
  - **Windows**
    http://jackaudio.org/downloads/
  - **macOS**
    http://jackaudio.org/downloads/

1.2 Update to Latest Version

This guide was released with openMHA version 4.10.0. If you have already installed openMHA on your system, make sure that you are using the latest version by repeating the installation as described in https://github.com/HoerTech-gGmbH/openMHA/blob/master/INSTALLATION.md.

1.3 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). But 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.

2 Getting Started

2.1 Starting openMHA

After openMHA and its dependencies have been installed (see section 1.1) 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**
Type now **`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**.

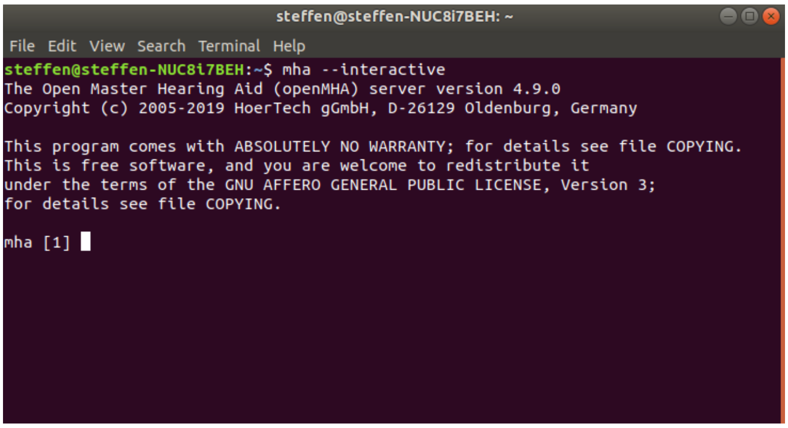

Figure 1 - Linux Terminal: Type `mha --interactive`

. . . .

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.

3 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.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 4 for your specific operating system)

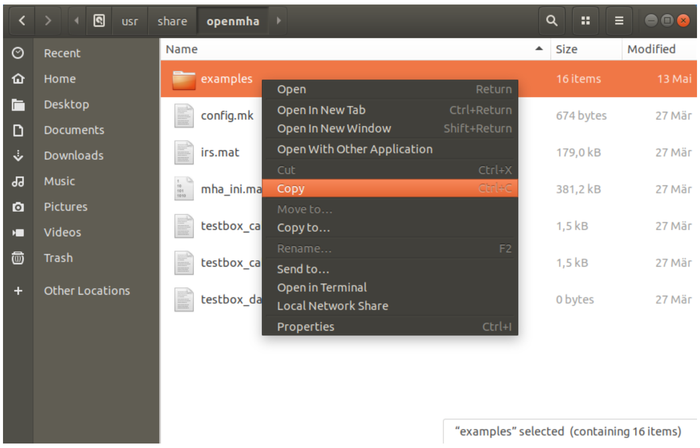

Figure 2 - Copy the examples folder from the protected directory (e.g. /usr/share/openmha/

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

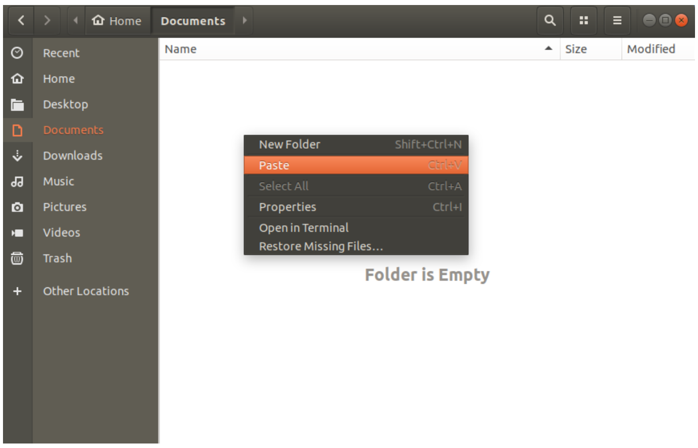

Figure 3 - Paste examples into folder inside a non-protected directory (e.g. /home/YourUserName/Documents

4. Open your **terminal** (see 2.1)
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**

3.1 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.cfg`
Start the openMHA signal processing by:
→ `cmd=start`
and then exit openMHA by typing:
→ `cmd=quit`

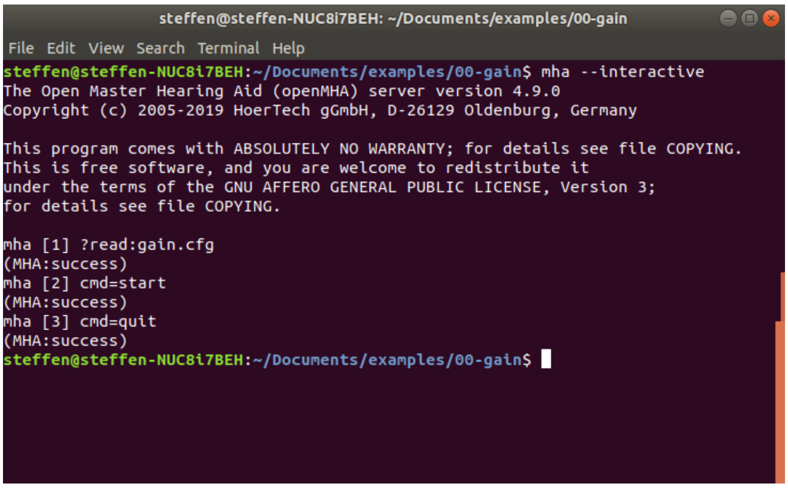

Figure 4 Interactive mode: Applying gain to a static audio signal

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".

3.2 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.cfg:

```
#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 signal of JACK
srate = 44100
#Again, we want to use the plugin "mhachain"
mhalib = mhachain
Here we will only use one plugin "gain"
mha.algos=[ gain ]
#Set max and min gain factors in dB
mha.gain.min=-20
mha.gain.max=20
#two gain factors (left and right)
mha.gain.gains=[ -10 10 ]
#In this example, we load the IO library that connects
#the MHA to the Jack audio server.
iolib = MHAIOJackdb
```

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/*.

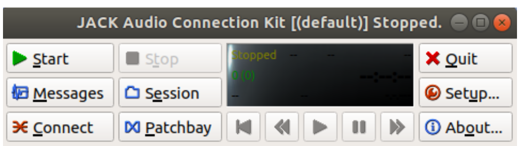

Figure 5 JACK Audio Connection Kit: GUI

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

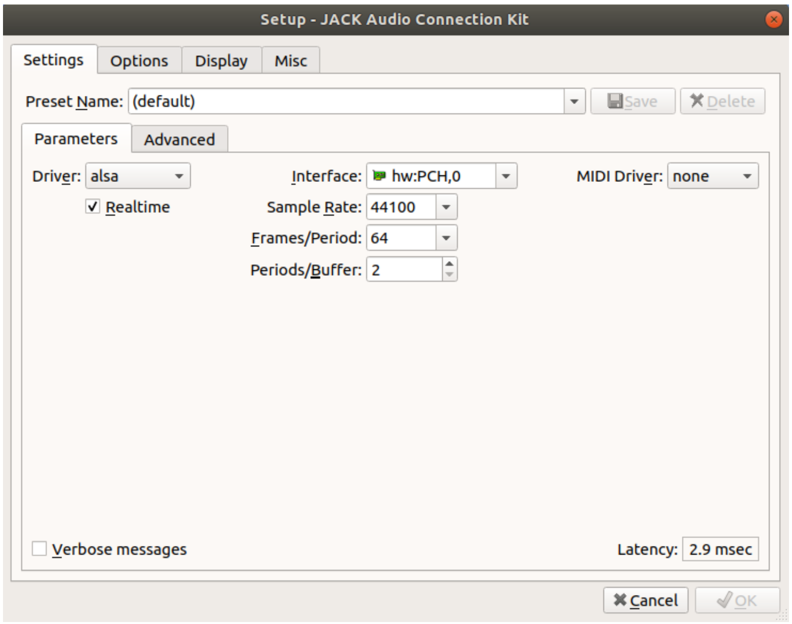

Figure 6 Jack GUI: Setup

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)

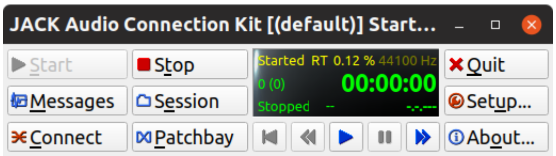

Figure 7 Jack GUI with running server

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 8).

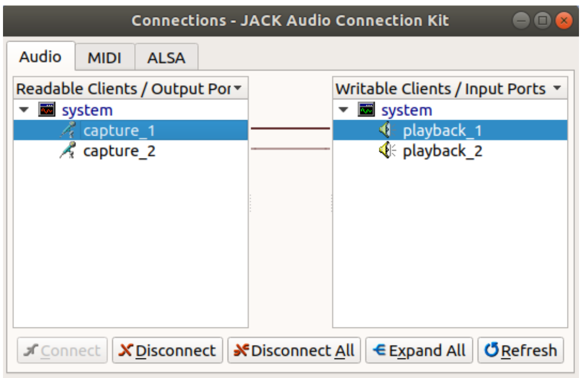

Figure 8 Jack GUI: Setup

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 2.1)
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.cfg`
9. → `cmd=start`

   openMHA is now applying a gain to your own voice input. In order to close openMHA type:
10. → `cmd=quit`

3.2.1 Adjusting the fragment size

If the required fragment size is not supported by the audio hardware, double buffering can be used in the openMHA framework. Decoupling of the JACK fragment size and the openMHA fragment size can be reached by inserting the double buffer plugin *db* between the framework and the algorithm. This is demonstrated in gain_live_double.cfg. **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.**

4 Control Frequency Shifter using Octave/Matlab GUI

1. **End** all running **mha processes**
2. **Open Matlab or Octave**
3. **Linux and Windows:** 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. Use the Command Window to enable communication with openMHA through java by typing:
   - **Linux**:
     → `javaaddpath('/usr/lib/openmha/mfiles/mhactl_java.jar')`
   - **Windows**:
     → `javaaddpath('C:\Program Files\openMHA\mfiles\mhactl_java.jar')`
   - **macOS**:
     → `javaaddpath('/usr/local/lib/openmha/mfiles/mhactl_java.jar')`
7. In order to start a new openMHA instance type
   → `openmha = mha_start;`
8. In order to read in the configuration file type:
   → `mha_query(openmha,'','read:fshift_live.cfg');`
9. **Start JACK Server** using **JACK Control**
   (Setting: Sample Rate = 44100, Frames/Period = 64)

10. Start the mha process by typing
    → `mha_set(openmha, 'cmd', 'start' );`

11. **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.

12. 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**

13. 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.

5 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 and Windows:** 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. Use the Command Windows to enable communication with openMHA through java by typing:
   - **Linux**:
     `javaaddpath('/usr/lib/openmha/mfiles/mhactl_java.jar')`
   - **Windows**:
     `javaaddpath('C:\Program Files\openMHA\mfiles\mhactl_java.jar')`
   - **macOS**:
     `javaaddpath('/usr/local/lib/openmha/mfiles/mhactl_java.jar')`
7. In order to start openMHA type `openmha = mha_start;`
8. Read in configuration into mha by
   `mha_query(openmha,'','read:dynamiccompression_live.cfg');`
9. **Start JACK Server** using **JACK Control**
   (Setting: Sample Rate = 44100, Frames/Period = 64)
10. In order to start the mha process type `mha_set(openmha,'cmd','start');`
11. In order to read out the current gaintable and relevant paramters type the following:
    ```
    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');
    ```

12. Plot the I/O characteristics
    `level_in = ((1:size(gaintable,2))-1) * gtstep + gtmin;`
    `level_out = level_in + gaintable;`
13. In order to plot input and output level, type `figure, plot(level_in,level_out)`
14. You can design your own gaintable in Matlab 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. noise gate, compressive region, output limit:
      `gaintable_new =repmat([-50,30:-2:0,-4:-4:-32],18,1);`
    - e.g. compress high frequency band only: ...
15. In order to apply the new gaintable type
    `mha_set(openmha,'mha.overlapadd.mhachain.dc.gtdata',gaintable_new);`
16. You can stop openMHA using `mha_set(openmha,'cmd','quit')`
17. More complex gaintable example would be
    `openmha = mha_start([],{},{'?read:dynamiccompression_live.cfg'})`
    `mha_set(openmha,'cmd','start')`
18. Start fitting GUI by typing `mhacontrol(openmha)`

6 Dealing with 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:

```
%Start openMHA process
openmha = mha_start;
% Read configuration file
mha_query(openmha,'','read:coherence_live.cfg');
% Start configuration file
mha_set(openmha,'cmd','start')

%% Label center frequencies and set gain factor of ac_proc

% Label center frequencies
freqs=mha_get(openmha,'mha.overlapadd.mhachain.coherence.cf');
% Set gain factor in dB - default value was choosen to be 6
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 choosen 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.

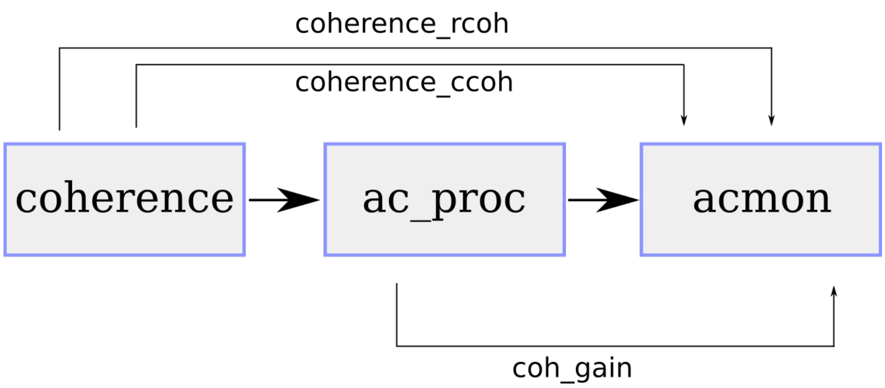

Figure 9 Schematics of AC variable data stream between plugins: coherence, ac_proc, acmon

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

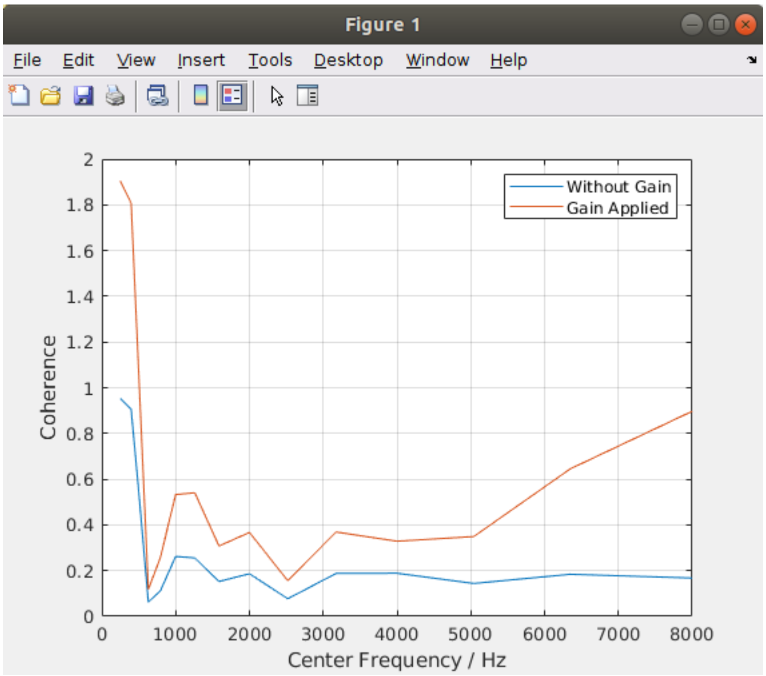

Figure 10 Matlab: Coherence plotted as a function of frequency

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