.

# openEAR - An introductory tutorial

for the Munich open Emotion and Affect Recognition toolkit

Florian Eyben, Martin Woellmer, and Bjoern Schuller

E-mails: last-name at tum.de

Institute for Human-Machine Communication
Technische Universitaet Muenchen (TUM)
D-80333 Munich, Germany
`http://www.mmk.ei.tum.de`

`www.openaudio.eu`

# openEAR - An introductory tutorial

This document provides a brief overview over openEAR's capabilities and should get you started in doing the following:

- Obtaining openEAR
- Compiling and installing
- Running openEAR 's live emotion recognition
- Extracting features
- Building custom models

## 1 Obtaining openEAR

The latest stable release of openEAR can be found on `http://www.openaudio.eu/`. For development releases and additional files, please check out the openEAR page on SourceForge.net: `https://sourceforge.net/projects/openart/`[1] For the cutting edge version of the feature extractor openSMILE, you can check out the code from the SVN:

```
svn co https://opensmile.svn.sourceforge.net/svnroot/opensmile opensmile
```

Note that openSMILE is maintained as a separate project on SourceForge.net. This project contains only the feature extractor and no model files and other tools for emotion recognition.

You can chose between binary releases for Linux and Windows, or get the source code and compile it yourself. The latter is the recommended way for Linux/Unix and MacOS systems. A binary release contains the main executable `SMILExtract` or `SMILExtract.exe`, example configuration files in the `config/` folder, pre-trained models in the `models/` folder, and scripts for visualisation and model-training in the `scripts/` folder.

The binary releases are ready to use. For Linux, you must choose the executable you want to use. Executables for different architectures are provided in the `linux_bin` directory. It should be obvious from the file name, which to use. Rename or copy the executable which suits your platform to `SMILExtract` in the top-level directory. Executables which are linked against PortAudio carry a `_pa` in the file name. For Windows, if you want to use PortAudio recording for live emotion recognition you must rename `SMILExtract_pa.exe` to `SMILExtract.exe` and `openSmileLib_pa.dll` to `openSmileLib.dll`. To test if your release works, change to the top-level directory of the distribution and type

```
./SMILExtract -h
```

in the shell prompt on Unix systems or

```
SMILExtract -h
```

[1] Please note that *openart* is the old name of the project. We have requested a project rename on SourceForge a couple of months ago, however this has not been done yet. Once the project is renamed by SourceForge, the new URL will be: `https://sourceforge.net/projects/openear/`

in the Windows command-line prompt. If you see the usage information everything is working. You can then skip to section 3.

*Note for Linux:* The binaries contained in the release are statically linked binaries, i.e. the libopensmile is linked into the binary. The binaries do only link against libc6 and pthreads. You cannot use binary plugins with these binaries! In order to use plugins, you must compile the source code to obtain a binary linking dynamically to libopensmile (see section 2.1). Further note that NO binary release with PortAudio support is provided for Linux. In order to use PortAudio recording you must compile from the source code (see section 2.1).

If you have obtained a source release, read the next section on how to compile and install it.

## 2 Compiling and installing openEAR

The core of openEAR compiles without any third-party dependencies, except for *pthreads* on Unix systems. You can use the core version to extract features and build custom models. You can not do live audio recording/playback, and echo cancellation. You need the PortAudio[2] library to support audio recording and playback on Linux, Windows, and MacOS. For echo cancellation you have to install the Speex codec library[3].

### 2.1 Compiling on Linux/Mac

This section describes how to compile and install the core of openEAR on Unix systems. You need to have the following packages installed: `autotools` (i.e. automake, autoconf, libtool, and m4), `make`, GNU C and C++ compiler `gcc` and `g++`. You will also want to install *perl5* in order to run the scripts for model training.

Start by unpacking the openEAR package to a directory to which you have write access:

```
tar -zxvf openEAR-0.1.0.tar.gz
```

Then change to the newly created directory:

```
cd openEAR-0.1.0/
```

and run:

```
./autogen.sh
```

Note: if you cannot run `autogen.sh` then run it either as `sh autogen.sh` or change the executable permission using the command `chmod +x autogen.sh`.

If you get errors and warnings during the first run of `autogen`, run it a second time. Even if you still get warnings now, it should be fine to proceed. Configure openEAR with

```
./configure
```

or, if you want to install to a custom directory (not the default location /usr or /usr/local), use

```
./configure --prefix=/directory/prefix/to/install/to
```

On modern CPUs it is recommended to create an optimised executable by using the following compiler flags: `-O2 -mfpmath=sse -march=native`, e.g. when calling configure:

```
./configure CXXFLAGS=‘‘-O2 -mfpmath=sse -march=native’’
```

[2]`http://www.portaudio.com/`

[3]`http://speex.org/`

To create statically linked binaries (i.e. with libopensmile linked statically), use the following option to configure:

```
./configure --enable-shard=no
```

*Warning:* openSMILE plugins will not work with statically linked binaries.
Next, compile openEAR with

```
make -j4 ; make
```

You are now ready to install openEAR by typing

```
make install
```

Please note that this currently only installs the openSMILE feature extractor binary `SMILExtract` and the feature extractor's library `libopensmile.so`. Model files and configuration files still remain in the build directory. Therefore, the following chapters will assume that all commands are entered in the top-level directory of the source distribution.

The reason for splitting openSMILE into a small binary linking to the dynamic library containing all the functionality is the support of binary plugins. Thus, the plugins can link to libopensmile and can be loaded at runtime into SMILExtract. How to use plugins is, however, not part of this introductory tutorial. More documentation on plugins will be added in the future.

If you have installed openSMILE to a non-default path, you must set you library path to include the newly installed libopensmile before running the SMILExtract binary:

```
export LD_LIBRARY_PATH=/path/to/your/smile/lib
```

## 2.2 Compiling on Linux/Mac with PortAudio

To compile openEAR with PortAudio support, the easiest way is to install the latest version of PortAudio via your distribution's package manager (be sure to install a development version, including development headers). You can then run the same steps as in section 2.1, the configure script should automatically detect your installation of PortAudio.

If you cannot install packages on your system or do not have access to a PortAudio package, unpack the file `thirdparty/portaudio.tgz` into the openEAR top-level directory. Then read the PortAudio compilation instructions and compile and install PortAudio according to these instructions. You can the continue with the steps listed in section 2.1. If you need to install PortAudio to a non-standard location, then you can pass the path to your installation to openEAR's configure script:

```
./configure --with-portaudio=/path/to/your/portaudio
```

## 2.3 Compiling on Windows

For compiling on Microsoft Windows (XP and Vista) there are two possible ways:

- Using Mingw32 and MSYS
- Using Visual Studio 2005 or 2008 (Express)

The preferred and supported way is using Visual Studio. If, however, you want to use Mingw32, please refer to `http://www.mingw.org/wiki/msys` for how to correctly set up your Mingw32 and MSYS system with all necessary development tools (autoconf, automake, libtool, and m4 as included in the MSYS DTK). You can then follow the Unix installation instructions in sections 2.1 and 2.2.

For compiling with Visual Studio several Project files are provided in the folder `ide/vs2005/`. To build the openSMILE feature extractor you need to build two projects: the library and the executable. For the standalone version (without PortAudio support, i.e. without live audio recording/playback) these two projects would be (in correct order) `openSmileLib.vcproj` and `openSmile.vcproj`. After successfully building both projects you should have an `openSmileLib.dll` and a `SMILExtract.exe` in the top-level directory of the source tree (NOT in the ide/vs2005/Debug or Release folder!). You can now copy these two files to a directory in your path, e.g. `C:\Windows\system32`.

To build the Visual Studio Projects you need the Windows Platform SDK. The Platform SDK should be installed in `C:\Program Files\Microsoft Platform SDK`. If you have it installed in a different location you must change the additonal include directories in the C/C++ compiler settings and the additional library directories in the Linker settings tab to match your path.

### 2.4 Compiling on Windows with PortAudio

Please obtain PortAudio from `http://www.portaudio.com/`. You will also find compilation and installation instructions for Windows there. Unpack the Windows PortAudio source tree to a subdirectory of the openEAR distribution, which you call `portaudio`. If you don't unpack PortAudio to this location, then you need to open the Visual Studio Project files mentioned in the next paragraph and adjust the Include and Linker paths for PortAudio. To build PortAudio without the DirectX, ASIO, and wasapi APIs, add `PA_NO_DS` and `PA_NO_ASIO` to the preprocessor defines (C/C++ settings tab, preprocessor) and disable all the .cpp files in the related hostapi project folders. Make sure you compile the Release configuration of PortAudio.

To compile with PortAudio support, i.e. with live audio recording/playback, you need to compile the following two projects after you have compiled PortAudio (and possibly adjusted the paths) `openSmileLibPA.vcproj` and `openSmilePA.vcproj`. After successfully building both projects you should have an `openSmileLib.dll` and a `SMILExtract.exe` in the top-level directory of the source tree (NOT in the ide/vs2005/Debug or Release folder!). Also make sure, that the PortAudio dll file is either in your path or in the same directory as `SMILExtract.exe`. The PortAudio dll can be found in the directory `portaudio/build/msvc/Win32/Release`, if you have successfully built the PortAudio distribution.

*Please note:* the PortAudio versions of the openSMILE projects assume that the dll is called `portaudio_x86.dll`. This name, however, might be different, depending on your architecture. Thus, you should this and change the name of the import library in the Linker, advanced settings tab.

## 3 Running openEAR 's live emotion recognition

Running openEAR in live emotion recognition mode with 4 pre-trained models (AVIC, ABC, EmoDB, SAL) is very easy. If you have installed and set up openEAR correctly you only need to run it with the right configuration file, which is also provided in the openEAR distribution:

```
SMILExtract -C config/emobase_live4.conf
```

Once the model sets have been loaded, you will see turn-wise output of interest (AVIC), various affective states (ABC), emotion (EmoDB), and continuous arousal/valence from the SAL corpus (models provided by the EU-funded research project SEMAINE). To terminate the program in live recognition mode, press Ctrl+C.

# 4 Extracting features with SMILExtract

Extracting features with openSMILE is easy once you have written your configuration file. To get you started quickly we have provided some example configuration files in the `config/` folder, which you may also modify to suit your needs. These are briefly explained in section 4.1. Writing your own configuration file is also not complicated as you will see in section 4.2.

## 4.1 Using the example configuration files

### 4.1.1 Extracting MFCC features

To extract HTK compatible MFCC features for speech recognition and emotion recognition you can use the following command:

```
SMILExtract -C config/MFCC12_E_D_A.conf -I input.wav -O output.htk
```

You will get 39 features per vector, 12 MFCC, logarithmic energy, and delta and delta delta coefficients appended to MFCC and energy. The frame size is set to 25 ms at a rate of 10 ms. A Hamming function is used to window the frames and a pre-emphasis with $k = 0.97$ is applied. The MFCC 1-12 are computed from 26 Mel-bands computed from the FFT power spectrum. The frequency range of the Mel-spectrum is set from 0 to 8 kHz. You can edit the configuration file to change these options.

In order to additionally apply a Cepstral Mean Subtraction to the above features on a per-input base, use the following command:

```
SMILExtract -C config/MFCC12_E_D_A.conf -I input.wav -O output.htk
```

### 4.1.2 Extracting features for emotion recognition

A baseline set of 988 acoustic features for emotion recognition can be extracted using the following command:

```
SMILExtract -C config/emobase.conf -I input.wav -O output.arff
```

This will produce an ARFF file with a header containing all the feature names and one instance, containing a feature vector for the given input file. To append more instances to the same ARFF file, simply run the above command again for different (or the same) input files. The ARFF file will have a dummy class label called emotion, containing one class *unknown* by default. To change this behaviour and assign custom classes and class labels to an individual instance, use a command-line like the following:

```
   SMILExtract -C config/emobase.conf -I inputN.wav -O output.arff -instname
inputN -classes {anger,fear,disgust} -classlabel anger
```

Thereby the parameter `-classes` specifies the list of nominal classes including the {} characters, or can be set to *numeric* for a numeric (regression) class. The parameter `-classlabel` specifies the class label/value of the instance computed from the currently given input (-I). For further information on these parameters, please take a look at the configuration file `emobase.conf` where these command-line parameters are defined.

The feature set specified by `emobase.conf` contains the following low-level descriptors (LLD): Intensity, Loudness, 12 MFCC, Pitch ($F_0$), Probability of voicing, $F_0$ envelope, 8 LSF (Line Spectral Frequencies), Zero-Crossing Rate. Delta regression coefficients are computed from these LLD, and the following functionals are applied to the LLD and the delta coefficients:

Max./Min. value and respective relative position within input, range, arithmetic mean, 2 linear regression coefficients and linear and quadratic error, standard deviation, skewness, kurtosis, quartile 1–3, and 3 inter-quartile ranges.

To extract the feature set used for the Interspeech 2009 Emotion Challenge (384 features), you can use the configuration file `emo_IS09.conf`:

```
SMILExtract -C config/emo_IS09.conf -I input.wav -O outputIS09.arff
```

The same options for specifying instance name, classes, and the class label as above are available.

For extracting a larger feature set with more functionals and more LLD enabled (total 6 552 features), use the configuration file `config/emo_large.conf`.

### 4.1.3 Batch extraction of features for emotion recognition

Batch extraction of features for a whole corpus of emotional speech is possible using the perl script `scripts/modeltrain/stddirectory_smileextract.pl`. The corpus has to be in the format described in section 5.1. The script takes 3 arguments as follows:

```
    stddirectory_smileextract.pl <corpus base path>
<opensmile config file (file name only, no path!)>
<output arff>
```

## 4.2 Understanding and modifying openSMILE configuration files

openSMILE configuration files follow an INI-style file format. The file is divided into sections, which are introduced by a section header:

```
[section1:sectionType]
```

The section header, opposed to standard INI-format, always contains two parts, the section name (first part) and the section type (second part). The two parts of the section header are separated by a colon (:). Next, a section contains attributes (which are defined by the section type, a description of the available types can be seen using the `-H` command-line option). Attributes are given as name = value pairs. An example of a generic config file section is given here:

```
[instancename:configType]     <-- this specifies the header
variable1 = value             <-- example of a string variable
variable2 = 7.8               <-- example of a "numeric" variable
variable3 = X                 <-- example of a "char" variable
subconf.var1 = myname         <-- example of a variable in a sub type
myarr[0] = value0             <-- example of an array
myarr[1] = value1
anotherarr = value0;value1    <-- example of an implicit array
noarray = value0\;value1      <-- use \; to quote the separator ';'
strArr[name1] = value1        <-- associative arrays, name=value pairs
strArr[name2] = value2
; comments may be expressed by ; // or # at the beginning of the line
; multi-line comments are not yet supported,
; however, if the will be, it will be C style /* and */
```

Principally the config type names can be any arbitrary names. However, for consistency the names of the components and their corresponding configuration type names are identical. Thus, to configure a component `cWaveSource` you need a configuration section of `cWaveSource`.

In every openSMILE configuration file there is one mandatory section, which configures the component manager. This is the component, which instantiates and runs all other components. The following sub-section will describe this section in detail.

### 4.2.1 Enabling components

The components which will be run, can be specified by configuring the `cComponentManager` component, as shown in the following listing (the section always has to be called `componentInstances`):

```
[componentInstances:cComponentManager]       <--- don't change this
; one data memory component must always be specified!
; the default name is 'dataMemory'
; if you call your data memory instance 'dataMemory',
; you will not have to specify the reader.dmInstance variables
; for all other components!
; NOTE: you may specify more than one data memory component
; configure the default data memory:
instance[dataMemory].type=cDataMemory
; configure an example data source (name = source1):
instance[source1].type=cExampleSource
```

The associative array `instance` is used to configure the list of components. The component instance names are specified by the array keys. They can contain all characters except for ], however, it is recommended to only use alphanumeric characters, _, and -. The component types (i.e. which component to instantiate), are given as value to the `type` option.

**Note:** for each component instance specified int the `instance` array a configuration section in the file *must* exist (*except for the data memory components!*), even if it is empty (e.g. if you want to use default values only). In this case you need to specify only the header line `[name:type]`.

### 4.2.2 Configuring components

The parameters of each component can be set in the configuration section corresponding to the specific component. For a wave source, for example, (which you instantiate with the line

```
instance[source1].type = cWaveSource
```

in the component manager configuration) you would add the following section (note that the name of the configuration section must match the name of the component instance, and the name of the configuration type must match the component's type name):

```
[source1:cWaveSource]
 ; the following sets the level this component writes to
 ; the level will be created by this component
 ; no other components may write to a level having the same name
writer.dmLevel = wave
filename = input.wav
```

This sets the file name of the wave source to `input.wav`. Further, it specifies that this wave source component should write to a data memory level called `wave`. Each openSMILE component, which processes data has at least a data reader (of type cDataReader), a data writer (of type cDataWriter), or both. These sub-components handle the interface to the data memory component(s). The most important option, which is mandatory, is `dmLevel`, which specifies the level to write to or to read from. Writing is only possible to one level and only one component may write to each level. We would like to note at this point that the levels do not have to be specified implicitly by configuring the data memory – in fact, the data memory is the only component which does not have and does not require a section in the configuration file – rather, the levels are created implicitly through `writer.dmLevel = newlevel`. Reading is possible from more than one level. Thereby, the input data will be concatenated frame-wise to one single frame containing data from all input levels. To specify reading from multiple levels, separate the level names with the array separator ';', e.g.:

```
reader.dmLevel = level1;level2
```

The next example shows the configuration of a `cFramer` component `frame`, which creates (overlapping) frames from raw wave input, as read by the wave source:

```
[frame:cFramer]
reader.dmLevel=wave
writer.dmLevel=frames
frameSize = 0.0250
frameStep = 0.010
```

The component reads from the level `wave`, and writes to the level `frames`. It will create frames of 25 ms length at a rate of 10 ms. The actual frame length in samples depends on the sampling rate, which will be read from meta-information contained in the `wave` level. For more examples please refer to the example configuration files in the `config/` directory.

### 4.2.3 Including other configuration files

To include other configuration files into the main configuration file use the following command on a separate line at the location where you want to include the other file:

```
\{path/to/config.file.to.include}
```

This include command can be used anywhere in the configuration file (as long it is on a separate line). It simply copies the lines of the included file into the main file while loading the configuration file into openSMILE.

### 4.2.4 Linking to command-line options

openSMILE allows to define new command-line options for the `SMILExtract` binary in the configuration file. To do so, use the following syntax:

```
[exampleSection:exampleType]
myAttrib1 = \cm[longoption(shortopt){default}:descr. text]
myAttrib2 = \cm[longoption{default}:descr. text]
```

The `shortopt` argument is optional. Note, that either `default` and/or `descr. text` are required to define a *new* option. If neither of the two is specified, the option will not be added to the command-line parser. Use this feature to reference options that were already added, i.e. if you want to use existing options or link to options you already defined in the configuration file at a prior location:

```
[exampleSection2:exampleType]
myAttrib2 = \cm[longoption]
```

### 4.2.5 Defining variables

This feature is not yet supported, but will be added soon. This should help avoid duplicate values and increase maintainability of configuration files.

### 4.3 Visualising extracted data with Matlab

If you have saved data to a matrix file using the component `cDatadumpSink`, then you can load this data into Matlab and plot it using the Matlab script `scripts/vis.m`. The script supports plotting the whole matrix as a greyscale image with optional range normalisation and 2D plotting of one or more feature contours. You will find more documentation in script file itself.

## 5 Building custom models

In order to run the model building scripts you need to have Perl 5 installed on your system. Look for ActiveState Perl for Windows. On Unix system Perl is usually installed by default.

Model building is most easy if you have a Weka ARFF file which contains features from all instances you want to build a model from. If this is the case, you can skip to section 5.2.

### 5.1 Building models directly from a corpus

In order to automatically extract features and build a classifier model from these features the emotion corpus you want to use must be in a simple, standard format which will be briefly described in the following:

- All audio recordings must be available in uncompressed wave format, preferably 16 kHz sampling rate and mono or stereo.
- All turns for one emotion class should be in a folder named after the class, e.g. all turns for *anger* should be in folder `anger/`, e.g. `anger/turn_anger1.wav`, `anger/turn_anger2.wav`.

Once you have converted your corpus to the format described above, you can run the model builder script `makemodel.pl` in the directory `scripts/modeltrain`:

```
   cd scripts/modeltrain
perl makemodel.pl path/to/your/corpus/ emobase.conf
```

The second parameter specifies the configuration file for feature extraction to use. You must only give the file name, without the directory. The path to the `config/` directory is automatically added, thus you can use only configuration files in the `config/` directory.

A few additional options for model building can be specified by modifying variables in the model builder script `makemodel.pl`:

- if you set `$do_select` to 1, a CFS feature selection using the WEKA command-line will be performed by the script `fsel.pl` (you need to adjust your WEKA paths in the script `fsel.pl` by setting `wekapath` and `wekacmd` variables).
- if you set `$regression` to 1, the script will build a regression model instead of a classification model. This is currently only supported with ARFF files (see section 5.2), since the feature extraction script does not support regression targets yet.

To use your new model, you must adapt the `emobase_live4.conf` configuration file (or create your own), to load your new model by creating a new `cLibsvmliveSink` section, adjusting the paths to your model, scale, etc. files, and adding a new instance entry for the `cLibsvmliveSink` to the componentManager configuration section.

### 5.2 Building models from Weka ARFF files

If you have an ARFF file containing features and an emotion class label or regression target label, you can directly run the model builder perl script `scripts/modeltrain/makemodel.pl` with the following command-line:

```
    cd scripts/modeltrain
perl makemodel.pl your_arff_file.arff
```

This will create a directory called `your_arff_file` in the subdirectory `built_models`. The directory will contain the LibSVM model file, a scale file, a class label file, and optionally a feature selection file.

### 5.3 Using existing LibSVM models

You may use an existing LibSVM models with openEAR. In order to do so you need to modify the paths to the model, scale, and classes file (and optionally feature selection file) to match your paths. See the `cLibsvmliveSink` sections in the `emobase_live4.conf` file for further information.

## 6 Controlling output and logging

The openSMILE feature extractor has quite comprehensive debug output and logging functionality. By default a log file called `smile.log` is created in the directory from where the executable is called and all log messages are saved to this file. You can change the logfile name with the command-line option `-logfile`, which takes the new file name as argument. If you don't want logging to a file you can specify `-nologfile`. Additionally the log messages are shown in the console window. To disable this, specify the `-noconsoleoutput` option.

You can control the verbosity of openSMILE via the option `-l <verbosity level>`. For normal informative messages a verbosity of 2 or 3 is sufficient. Verbosity levels up to 6 are implemented. To enable *a lot* more debug messages use the `-d` option, if your binary was compiled with debug output support (the binaries in the release are).

## 7 Getting more information

If you require more information on openEAR, or are interested in developing your own components, please refer to the documentation in the `doc/` folder, or contact Dipl.-Ing. Florian Eyben (eyben at tum.de).

If you are interested in integrating openEAR into commercial software products, please be aware that openEAR and openSMILE are under the GPL, which requires your system to be available under the terms of the GPL, too. If you do not want that, please contact Dipl.-Ing. Florian Eyben (eyben at tum.de), Dipl.-Ing. Martin Wöllmer (woellmer at tum.de), or Dr.-Ing. Björn Schuller (schuller at tum.de) to enquire terms and conditions for a commercial licensing of the openEAR code.

## 8 Acknowledgement

The development of openSMILE and openEAR has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 211486 (SEMAINE).