openSMILE:) The Munich Open-Source Large-scale Multimedia Feature Extractor

A tutorial for version 2.1

Introduction

The openSMILE feature extraction and audio analysis tool enables you to extract large audio (and recently also video) feature spaces incrementally and fast, and apply machine learning methods to classify and analyze your data in real-time. It combines acoustic features from Music Information Retrieval and Speech Processing, as well as basic computer vision features. Large, standard acoustic feature sets are included and usable out-of-the-box to ensure comparable standards in feature extraction in related research.

The purpose of this article is to briefly introduce openSMILE, it's features, potentials, and intended use-cases as well as to give a hands-on tutorial packed with examples that should get you started quickly with using openSMILE.

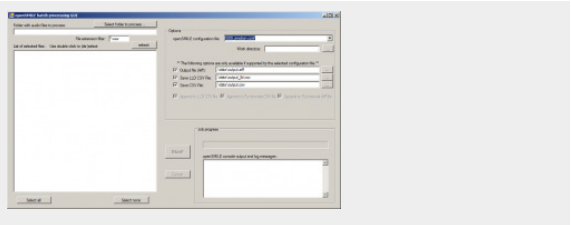
About openSMILE

SMILE is originally an acronym for *Speech & Music Interpretation by Large-space feature Extraction*. Due to the recent addition of video-processing in version 2.0, the acronym openSMILE evolved to open-*Source Media Interpretation by Large-space feature Extraction*.

The development of the toolkit has been started at Technische Universität München (TUM) for the EU-FP7 research project SEMAINE. The original primary focus was on state-of-the-art acoustic emotion recognition for emotionally aware, interactive virtual agents. After the project, openSMILE has been continuously extended to a universal audio analysis toolkit. It has been used and evaluated extensively in the series of INTERSPEECH challenges on emotion, paralinguistics, and speaker states and traits: From the first INTERSPEECH 2009 Emotion Challenge up to the upcoming Challenge at INTERSPEECH 2015 (see openaudio.eu for a summary of the challenges). Since 2013 the code-base has been transferred to audEERING and the development is continued by them under a dual-license model – keeping openSMILE free for the research community.

openSMILE is written in C++ and is available as both a standalone command-line executable as well as a dynamic library. The main features of openSMILE are its capability of on-line incremental processing and its modularity. Feature extractor components can be freely interconnected to create new and custom features, all via a simple text-based configuration file. New components can be added to openSMILE via an easy binary plug-in interface and an extensive internal API. Scriptable batch feature extraction is supported just as well as live on-line extraction from live recorded audio streams. This enables you to build and design systems on off-line databases, and then use exactly the same code to run your developed system in an interactive on-line prototype or even product.

openSMILE is intended as a toolkit for researchers and developers, but not for end-users. It thus cannot be configured through a Graphical User Interface (GUI). However, it is a fast, scalable, and highly flexible command-line backend application, on which several front-end applications could be based. Such examples are network interface components, and in the latest release of openSMILE (version 2.1) a batch feature extraction GUI for Windows platforms:



As seen in the above figure, the GUI allows to easily choose a configuration file, the desired output files and formats, and to select files and folders on which to run the analysis.

Made popular in the field of speech emotion recognition and paralinguistic speech analysis, openSMILE is now beeing widely used in this community. According to google scholar the two papers on openSMILE ([Eyben10] and [Eyben13a]) are currently cited over 380 times. Research teams across the globe are using it for several tasks, including paralinguistic speech analysis, such as alcohol intoxication detection, in VoiceXML telephony-based spoken dialogue systems — as implemented by the HALEF framework, natural, speech enabled virtual agent systems, and human behavioural signal processing, to name only a few examples.

Key Features

The key features of openSMILE are:

- It is **cross-platform** (Windows, Linux, Mac, **new in 2.1: Android**)
- It offers **both incremental processing and batch processing**.
- It efficiently extracts a **large number of features very fast** by re-using already computed values.
- It has multi-threading support for parallel feature extraction and classification.
- It is extensible with new custom components and plug-ins.
- It supports **audio file in- and output** as well as **live sound recording and playback**.
- The computation of MFCC, PLP, (log-)energy, and delta regression coefficients is **fully HTK compatible**.
- It has a wide range of **general audio signal processing**components:
  - Windowing functions (Hamming, Hann, Gauss, Sine, …),
  - Fast-Fourier Transform,
  - Pre-emphasis filter,
  - Finit-Impulse-Response (FIR) filterbanks,
  - Autocorrelation,
  - Cepstrum,
  - Overlap-add re-synthesis,
- … and **speech-related acoustic descriptors**:
  - Signal energy,
  - Loudness based on a simplified sub-band auditory model,
  - Mel-/Bark-/Octave-scale spectra,
  - MFCC and PLP-CC,
  - Pitch (ACF and SHS algorithms and Viterbi smoothing),
  - Voice quality (Jitter, Shimmer, HNR),
  - Linear Predictive Coding (LPC),
  - Line Spectral Pairs (LSP),
  - Formants,
  - Spectral shape descriptors (Roll-off, slope, etc.),
- … and **music-related descriptors**:
  - Pitch classes (semitone spectrum),
  - CHROMA and CENS features.
- It supports **multi-modal fusion** on the feature level through openCV integration.
- Several **post-processing**methods for low-level descriptors are included:
  - Moving average smoothing,
  - Moving average mean subtraction and variance normalization (e.g. for on-line Cepstral mean subtraction),
  - On-line histogram equalization (experimental),
  - Delta regression coefficients of arbitrary order,
  - Binary operations to re-combine descriptors.
- A **wide range** of **statistical functionals**for feature summarization is supported, e.g.:
  - Means, Extremes,
  - Moments,
  - Segment statistics,
  - Sample-values,
  - Peak statistics,
  - Linear and quadratic regression,
  - Percentiles,
  - Durations,
  - Onsets,
  - DCT coefficients,
  - Zero-crossings.
- Generic and **popular data file formats**are supported:
  - Hidden Markov Toolkit (HTK) parameter files (read/write)
  - WEKA Arff files (currently only non-sparse) (read/write)
  - Comma separated value (CSV) text (read/write)
  - LibSVM feature file format (write)

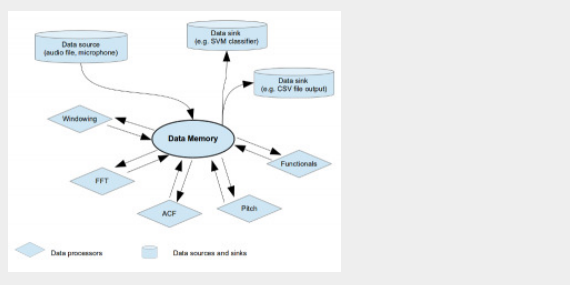
In the latest release (2.1) the new features are:

- **Integration and improvement of the emotion recognition models from openEAR,**
- **LSTM-RNN based voice-activity detector prototype models included,**
- **Fast linear SVMsink component** which supports linear kernel SVM models trained with the WEKA SMO classifier,
- LSTM-RNN JSON network file support for networks trained with the CURRENNT toolkit,
- Spectral harmonics descriptors,
- **Android support**,
- Improvements to configuration files and command-line options,
- Improvements and fixes.

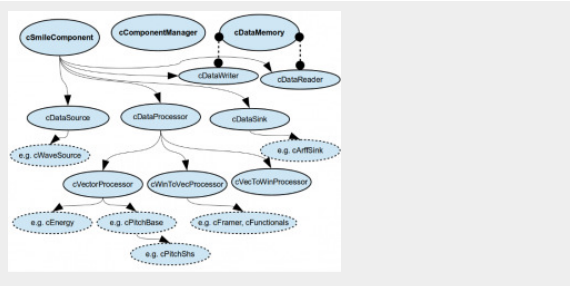
openSMILE's architecture

openSMILE has a very modular architecture, designed for incremental data-flow.
A central *dataMemory* component hosts shared memory buffers (known as dataMemory levels) to which a single component can write data and one or more other components can read data from. There are data-source components, which read data from files or other external sources and introduce them to the dataMemory. Then there are data-processor components, which read data, modify them, and save it to a new buffer – these are the actual feature extractor components. In the end data-

sink components read the final data and save them to files or digest it in other ways (classifiers etc.):



As all components which process data and connect to the dataMemory share some common functionality, they are all derived from a single base class **cSmileComponent**. The following figure shows the class hierarchy, and the connections between the cDataWriter and cDataReader components to the dataMemory (dotted lines).



Getting openSMILE and the documentation

The latest openSMILE packages can be downloaded here.
At the time of writing the most recent release is 2.1. Grab the complete package of the latest release. This includes the source code, the binaries for Linux and Windows. Some most up-to-date releases might not always include a full-blown set of binaries for all platforms, so sometimes you might have to compile from source, if you want the latest cutting-edge version.

While the tutorial in the next section should give you a good quick-start, it does not and can not cover every detail of openSMILE. For learning more and getting further help, there are three main resources:
The first is the openSMILE documentation, called the openSMILE book. It contains detailed instructions on how to install, compile, and use openSMILE and introduces you to the basics of openSMILE. However, it might not be the most up-to-date resource for the newest features. Thus, the second resource, is the on-line help built into the binaries. This provides the most up-to-date documentation of available components and their options and features. We will tell you how to use the on-line help in the next section. If you cannot find your answer in neither of these resources, you can ask for help in the discussion forums on the openSMILE website or *read the source-code*.

Quick-start tutorial

You can't wait to get openSMILE and try it out on your own data? Then this is your section. In the following the basic concepts of openSMILE are described, pre-built use-cases of automatic, on-line voice activity detection and speech emotion recognition are presented, and the concept of configuration files and the data-flow architecture are explained.

a. Basic concepts

Please refer to the openSMILE book for detailed installation and compilation instructions. Here we assume that you have a compiled SMILExtract binary (optionally with PortAudio support, if you want to use the live audio recording examples below), with which you can run:

```
SMILExtract -h
SMILExtract -H cWaveSource
```

to see general usage instructions (first line) and the on-line help for the cWaveSource component (second line), for example.

However, from this on-line help it is hard to get a general picture of the openSMILE concepts. We thus describe briefly how to use openSMILE for the most common tasks.

Very loosely said, the SMILExtract binaries can be seen as a special kind of code interpreter which executes custom configuration scripts. What openSMILE actually does in the end when you invoke it is only controlled by this configuration script. So, in order to do something with openSMILE you need:

- The binary SMILExtract,
- a (set of) configuration file(s),
- and optionally other files, such as classification models, etc.

The configuration file defines all the components that are to be used as well as their data-flow interconnections. All the components are iteratively run in the "*tick-loop*", i.e. a run method (*tick()*) of each component is called in every loop iteration. Each component then checks if there are new data to process, and if yes, processes the data, and makes them available for other components to process them further. Every component returns a status value,

which indicates whether the component has processed data or not. If no component has had any further data to process, the end of the data input (EOI) is assumed. All components are switched to an EOI state and the tick-loop is executed again to process data which require special attention at the end of the input, such as delta-regression coefficients. Since version 2.0-rc1, multi-pass processing is supported, i.e. providing a feature to enable re-running of the whole processing. It is not encouraged to use this, since it breaks incremental processing, but for some experiments it might be necessary.

The minimal, generic use-case scenario for openSMILE is thus as follows:

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

Each configuration file can define additional command-line options. Most prominent examples are the options for in- and output files (-I and -O). These options are not shown when the normal help is invoked with the -h option. To show the options defined by a configuration file, use this command-line:

```
SMILExtract -cmdHelp -C config/my_configfile.conf
```

The default command-line for processing audio files for feature extraction is:

```
SMILExtract -C config/my_configfile.conf -I input_file.wav -O output_file
```

This runs SMILExtract with the configuration given in *my_configfile.conf*.

The following two sections will show you how to quickly get some advanced applications running as pre-configured use-cases for voice activity detection and speech emotion recognition.

b. Use-case: The openSMILE voice-activity detector

The latest openSMILE release (2.1) contains a research prototype of an intelligent, data-drive voice-activity detector (VAD) based on Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN), similar to the system introduced in [Eyben13b].

The VAD examples are contained in the folder scripts/vad. A README in that folder describes further details. Here we give a brief tutorial on how to use the two included use-case examples:

- **vad_opensource.conf**: Runs the LSTM-RNN VAD and dumps the activations (voice probability) for each frame to a CSV text file. To run the example on a wave file, type:

```
cd scripts/vad;
SMILExtracct -I ../../example-audio/media-interpretation.wav \
             -C vad_opensoure.conf -csvoutput vad.csv
```

  This will write the VAD probabilities scaled to the range -1 to +1 (2nd column) and the corresponding timestamps (1st column) to vad.csv. A VAD probability greater 0 indicates voice presence.
- **vad_segmeter.conf**: Runs the VAD on an input wave file, and automatically extract voice segments to new wave files. Optionally the raw voicing probabilities as in the above example can be saved to file. To run the example on a wave file, type:

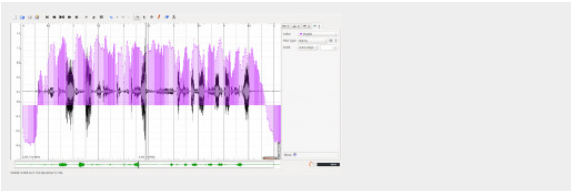
```
cd scripts/vad;
mkdir -p voice_segments
SMILExtract -I ../../example-audio/media-interpretation.wav -C vad_segmenter.conf \
            -waveoutput voice_segments/segment_
```

  This will create a new wave file (numbered consecutively, starting at 1). The vad_segmenter.conf optionally supports output to CSV with the *-csvoutput filename* option. The start and end times (in seconds) of the voice segments relative to the start of the input file can be optionally dumped with the *-saveSegmentTimes filename* option. The columns of the output file are: segment filename, start (sec.), end (sec.), length of segment as number of raw (10ms) frames.

To visualise the VAD output over the waveform, we recommend using Sonic-visualiser. If you have sonc-visualiser installed (on Linux) you can open both the wave-file and the VAD output with this command:

```
sonic-visualiser example-audio/media-interpretation.wav vad.csv
```

An annotation layer import dialog should appear. The first column should be detected as Time and the second column as value. If this is not the case, select these values manually, and specify that timing is specified explicitly (should be the default) and click OK. You should see something like this:



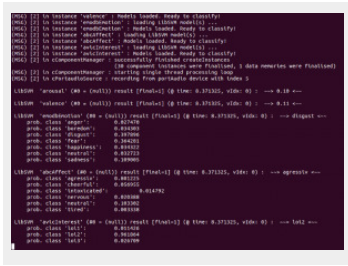
c. Use-case: Automatic speech emotion recognition

As of version 2.1, openSMILE supports running the emotion recognition models from the openEAR toolkit [Eyben09] in live emotion recognition demo. In order to start this live speech emotion recognition demo, download the speech emotion recognition models and unzip them in the top-level folder of the openSMILE package. A folder named models should be created there which contains a README.txt, and a sub-folder *emo*. If this is the case, you are ready to run the demo. Type:

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

to run it. The classification output will be shown on the console.

NOTE: *This example requires that you are running a binary with PortAudio support enabled. Refer to the openSMILE book for details on how to compile your binary with portaudio support for Linux. For Windows pre-compiled binaries (SMILExtractPA*.exe) are included, which should be used instead of the standard SMILExtract.exe for the above example.*



If you want to choose a different audio recording device, use

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

To see a list of available devices and their IDs, type:

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

Note: If you have a different directory layout or have installed SMILExtract in a system path, you must make sure that the models are located in a directory named "models" located in the directory from where you call the binary, or you must adapt the path to the models in the configuration file (emobase_live4.conf).

In openSMILE 2.1, the emotion recognition models can also be used **for off-line/batch analysis**. Two configuration files are provided for this purpose: *config/emobase_live4_batch.conf* and *config/emobase_live4_batch_single.conf*.
The latter of the two will compute a single feature vector for the input file and return a single result. Use this, if your audio files are already chunked into short phrases or sentences. The first, *emobase_live4_batch.conf* will run an energy based segementation on the input and will return a result for every segment. Use this for longer, un-cut audio files. To run analyis in batch mode, type:

```
SMILExtract -C config/emobase_live4_batch(_single).conf -I example-audio/opensmile.wav > result.txt
```

This will redirect the result(s) from SMILExtract's standard output (console) to the file *result.txt*. The file is by default in a machine parseable format, where key=value tokens are separated by :: and a single result is given on each line, for example:

```
SMILE-RESULT::ORIGIN=libsvm::TYPE=regression::COMPONENT=arousal::VIDX=0::NAME=(null)::
     VALUE=1.237816e-01
SMILE-RESULT::ORIGIN=libsvm::TYPE=regression::COMPONENT=valence::VIDX=0::NAME=(null)::
     VALUE=1.825088e-01
SMILE-RESULT::ORIGIN=libsvm::TYPE=classification::COMPONENT=emodbEmotion::VIDX=0::
     NAME(null)::CATEGORY_IDX=2::CATEGORY=disgust::PROB=0;anger:0.033040::
     PROB=1;boredom:0.210172::PROB=2;disgust:0.380724::PROB=3;fear:0.031658::
     PROB=4;happiness:0.016040::PROB=5;neutral:0.087751::PROB=6;sadness:0.240615
SMILE-RESULT::ORIGIN=libsvm::TYPE=classification::COMPONENT=abcAffect::VIDX=0::
    NAME=(null)::CATEGORY_IDX=0::CATEGORY=agressiv::PROB=0;agressiv:0.614545::
    PROB=1;cheerful:0.229169::PROB=2;intoxicated:0.037347::PROB=3;nervous:0.011133::
    PROB=4;neutral:0.091070::PROB=5;tired:0.016737
SMILE-RESULT::ORIGIN=libsvm::TYPE=classification::COMPONENT=avicInterest::VIDX=0::
    NAME=(null)::CATEGORY_IDX=1::CATEGORY=loi2::PROB=0;loi1:0.006460::
    PROB=1;loi2:0.944799::PROB=2;loi3:0.048741
```

The above example is the result of the analysis of the file *example-audio/media-interpretation.wav*.

d. Understanding configuration files

The above, pre-configured examples are a good quick-start to show the diverse potential of the tool. We will now take a deeper look at openSMILE configuration files. First, we will use simple, small configuration files, and modify these in order to understand the basic concepts of these files. Then, we will show you how to write your own configuration files from scratch.

The demo files used in this section are provided in the 2.1 release package in the folder *config/demo*. We will first start with *demo1_energy.conf*. This file extracts basic frame-wise logarithmic energy. To run this file on one of the included audio examples in the folder *example-audio*, type the following command:

```
SMILExtract -C config/demo/demo1_energy.conf -I example-audio/
```

openSMILE

```
.wav -O energy.csv
```

This will create a file called energy.csv. Its content should look similar to this:

```
frameIndex;frameTime;pcm_LOGenergy
0;0.000000;-1.383729e+01
1;0.010000;-1.344107e+01
2;0.020000;-1.362546e+01
3;0.030000;-1.371792e+01
4;0.040000;-1.256511e+01
5;0.050000;-1.210091e+01
6;0.060000;-1.216670e+01
7;0.070000;-1.241860e+01
8;0.080000;-1.232635e+01
9;0.090000;-1.222799e+01
10;0.100000;-1.289224e+01
11;0.110000;-1.241491e+01
12;0.120000;-1.244319e+01
13;0.130000;-1.286947e+01
14;0.140000;-1.316939e+01
15;0.150000;-1.261712e+01
16;0.160000;-1.229953e+01
17;0.170000;-1.183391e+01
18;0.180000;-1.149516e+01
```

The second example we will discuss here, is the audio recorder example (audiorecorder.conf).

NOTE: *This example requires that you are running a binary with PortAudio support enabled. Refer to the openSMILE book for details on how to compile your binary with portaudio support for Linux. For Windows pre-compiled binaries (SMILExtractPA*.exe) are included, which should be used instead of the standard SMILExtract.exe for the following example.*

This example implements a simple live audio recorder. Audio is recorded from the default audio device to an uncompressed PCM wave file. To run the example and record to *rec.wav*, type:

```
SMILExtract -C config/demo/audiorecorder.conf -O rec.wav
```

Modifiying existing configuration files is the fasted way to create custom extraction scripts. We will now change the *demo1_energy.conf* file to extract Root-Mean-Square (RMS) energy instead of logarithmic energy. This can be achieved by changing the respective options in the section of the *cEnergy* component (identified by the section heading [energy:cEnergy]) from

```
rms = 0
log = 1
```

to

```
rms = 1
log = 0
```

As a second example, we will merge *audiorecorder.conf* and *demo1_energy.conf* to create a configuration file which computes the frame-wise RMS energy from live audio input. First, we start with concatenating the two files. On Linux, type:

```
cat config/demo/audiorecorder.conf config/demo/demo1_energy.conf > config/demo/live_energy.conf
```

On Windows, use a text editor such as Notepad++ to combine the files via copy and paste. Now we must remove the cWaveSource component from the original *demo1_energy.conf*, as this should be replaced by the cPortaudioSource component of the *audiorecorder.conf* file. To do this, we search for the line

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

and comment it out by prefixing it with a ; or the C-style // or the script- and INI-style #. We also remove the corresponding configuration file section for waveSource. We do the same for the waveSink component and the corresponding section, the leave only the output of the computed frame-wise energy to a CSV file. Theoretically, we could also leave the waveSink section and component, but we would need to change the command-line option defined for the output filename, as this is the same for the CSV output and the wave-file output without any changes. In this case we should replace the filename option in the waveSink section by:

```
filename = \cm[waveoutput{output.wav}:name of output wave file]
```

Now, run your new configuration file with:

```
SMILExtract -C config/demo/live_energy.conf -O live_energy.csv
```

and inspect the contents of the *live_energy.csv* file with a text editor.

openSMILE configuration files are made up of sections, similar to INI files. Each section is identified by a header which takes the form:

```
[instancename:cComponentType]
```

The first part (*instancename*) is a custom-chosen name for the section. It must be unique throughout the whole configuration file and all included sub-files. The second part defines the type of this configuration section and thereby its allowed contents. The configuration section typename must be one of the available component names (from the list printed by the command

SMILExtract -L), as configuration file sections are linked to component instances.
The contents of each section are lines of *key=value* pairs, until the next section header is found. Besides simple key=value pairs as in INI files, a more advanced structure is supported by openSMILE. The key can be a hierarchical value build of key1.subkey, for example, or an array such as
keyarray[0] and keyarray[1]. On the other side, the value field can also denote an array of values, if the values are separated by a semi-colon (;). Quotes for the values are not needed and not yet supported, and multi-line values are not allowed.
Boolean flags are always expressed as numeric values with 1 for on or true and 0 for off or false.
The *keys* are referred to as the configuration options of the components, i.e. those listed by the on-line help (SMILExtract -H cComponentType).

Since version 2.1, configuration sections can be split into multiple parts across the configuration file. That is, the same header (same instancename and typename) may occur more than once. In that case all options from all occurrences will be joint.

There is one configuration section that must always be present: that of the *component manager*:

```
[componentInstances:cComponentManager]
instance[dataMemory].type = cDataMemory
instance[instancename].type = cComponentType
instance[instancename2].type = cComponentType2
...
```

The component manager is the main instance which creates all component instances of the currently loaded configuration, makes them read their configuration settings from the parsed configuration file (through the *configManager* component), and runs the *tick-loop*, i.e. the loop where data are processed incrementally by calling each component once to process newly available data frames. Each component that shall be included in the configuration, must be listed in this section, and for each component listed there, a corresponding configuration file section with the same *instancename* and of the same component type must exist. The only exception is the first line, which instantiates the central *dataMemory* component. It must be always present in the instance list, but no configuration file section has to be supplied for it.

Each component that processes data has a data-reader and/or a data-writer sub-component, which are configurable via the reader and writer objects. The only options of interest to us now in these objects are the dmLevel options. These options configure the data-flow connections in your configuration file, i.e. they define in which order data is processed by the components, or in other words, which component is connected with which other component:

Each component that modifies data or creates data (i.e. reading it from external sources etc.), will write its data to a unique dataMemory location (called *level*). The name of this location is defined in the configuration file via the option *writer.dmLevel=name_of_evel*. The level names must be unique and only one single component can write to each level. Multiple components can, however, read from a single level, enabling re-use of already computed data by multiple components.
E.g. we typically have a wave source component which reads audio data from an uncompressed audio file (see also the demo1_energy.conf file):

```
[wavesource:cWaveSource]
writer.dmLevel = wave
filename = input.wav
```

The above reads data from input.wav into the dataMemory level *wave*.
If next we want to chunk the audio data into overlapping analysis windows of 20ms length at a rate of 10ms, we need a *cFramer* component:

```
[framer:cFramer]
reader.dmLevel = wave
writer.dmLevel = frames20ms
frameSize = 0.02
frameStep = 0.01
```

The crucial line in the above code is the line which sets the reader dataMemory level (reader.dmLevel = wave) to the output level of the wave source component – effectively connecting the framer to the wave source component.

To create new configuration files from scratch, a configuration file template generator is available. We will use it to create a configuration for computing magnitude spectra via the Fast-Fourier Transform (FFT). The template file generator requires a list of components that we want to have in the configuration file, so we must build this list first. In openSMILE most processing steps are wrapped in individual components to increase flexibility and re-usability of intermediate data.
For our example we thus need the following components:

- An audio file reader (cWaveSource),
- a component which generates short-time analysis frames (cFramer),
- a component which applies a windowing function to these frames such as a Hamming window (cWindower),
- a component which performs a FFT (cTranformFFT),

- a component which computes spectral magnitudes from the complex FFT result (cFFTmagphase),
- and finally a component which writes the magnitude spectra to a CSV file (cCsvSink).

The generate our configuration file template, we thus run (note, that the component names are case sensitive!):

```
SMILExtract -l 0 -logfile my_fft_magnitude.conf -cfgFileTemplate -configDflt cWaveSource,cFramer,
    cWindower,cTransformFFT,cFFTmagphase,cCsvSink
```

The switch *-cfgFileTemplate* enables the template file output, and makes *-configDflt* accept a comma separated list of component names. If *-configDflt* is used by itself, it will print only the default configuration section of a single component (of which the name is given as argument to that option). This invocation of SMILExtract prints the configuration file template to the log (i.e., standard error and to the (log-)file given by the *-logfile* option). The switch *-l 0* suppresses all other log messages (by setting the log-level to 0), leaving only the configuration file template lines in the specified file.

The file generated by the above command cannot be used as is, yet. We need to update the data-flow connections first. In our example this is trivial, as one component always reads from the previous one, except for the wave source, which has no reader. We have to change:

```
[waveSource:cWaveSource]
writer.dmLevel = < >
```

to

```
[waveSource:cWaveSource]
writer.dmLevel = wave
```

The same for the framer, resulting in:

```
[framer:cFramer]
reader.dmLevel = wave
writer.dmLevel = frames
```

and for the windower:

```
[windower:cWindower]
reader.dmLevel = frames
writer.dmLevel = windowed
...
winFunc = Hamming
...
```

where we also change the windowing function from the default (Hanning) to Hamming,
and in the same fashion we go down all the way to the csvSink component:

```
[transformFFT:cTransformFFT]
reader.dmLevel = windowed
writer.dmLevel = fftcomplex

...

[fFTmagphase:cFFTmagphase]
reader.dmLevel = fftcomplex
writer.dmLevel = fftmag

...

[csvSink:cCsvSink]
reader.dmLevel = fftmag
```

The configuration file can now be used with the command:

```
SMILExtract -C my_fft_magnitude.conf
```

However, if you run the above, you will most likely get an error message that the file *input.wav* is not found. This is good news, as it first of all means you have configured the data-flow correctly. In case you did not, you will get error messages about missing data memory levels, etc. The missing file problem is due to the hard-coded input file name with the option *filename = input.wav* in the wave source section. If you change this line to *filename = example-audio/opensmile.wav* your configuration will run without errors. It writes the result to a file called *smileoutput.csv*.

To avoid having to change the filenames in the configuration file for every input file you want to process, openSMILE provides a very convenient feature: it allows you to define command-line options in the configuration files. In order to use this feature you replace the value of the *filename* by the command *\cm[]*, e.g. for the input file:

```
filename = \cm[inputfile(I){input.wav}:input filename]
```

and for the output file:

```
filename = \cm[outputfile(O){output.csv}:output filename]
```

The syntax of the \cm command is: [longoptionName(shortOption-1charOnly){default value}:description for on-line help].

e. Reference feature sets

A major advantage of openSMILE over related feature extraction toolkits is that is comes with several reference and baseline feature sets which were used for the INTERSPEECH Challenges (2009-2014) on Emotion, Paralinguistics and Speaker States and Traits, as well as the Audio-Visual Emotion Challenges (AVEC) from 2011-2013. All of the INTERSPEECH configuration files are found under *config/ISxx_*.conf*.
All the INTERSPEECH Challenge configuration files follow a common standard regarding the data output

options they define. The default output file option (-O) defines the name of the WEKA ARFF file to which functionals are written. To save the data in CSV format additionally, use the option *-csvoutput filename*. To disable the default ARFF output, use *-O ?*.
To enable saving of intermediate parameters, frame-wise Low-Level Descriptors (LLD), in CSV format the option
-lldoutput filename can be used. By default, lines are appended to the functions ARFF and CSV files is they exist, but the LLD files will be overwritten. To change this behaviour, the boolean (1/0) options -appendstaticarff 1/0, -appendstaticcsv 1/0, and -appendlld 0/1 are provided.

Besides the Challenge feature sets, openSMILE 2.1 is capable of extracting parameters for the Geneva Minimalistic Acoustic Parameter Set (GeMAPS — submitted for publication as [Eyben14], configuration files will be available together with publication of the article), which is a small set of acoustic paramters relevant for affective voice research. It was standardized and agreed upon by several research teams, including linguists, psychologists, and engineers.

Besides these large-scale brute-forced acoustic feature sets, several other configuration files are provided for extracting individual LLD. These include Mel-Frequency Cepstral Coefficients (MFCC*.conf) and Perceptual Linear Predictive Coding Cepstral Coefficients (PLP*.conf), as well as the fundamental frequency and loudness (prosodyShsViterbiLoudness.conf, or smileF0.conf for fundamental frequency only).

Conclusion and summary

We have introduced openSMILE version 2.1 in this article and have given a hands-on practical guide on how to use it to extract audio features of out-of-the-box baseline feature sets, as well as customized acoustic descriptors. It was also shown how to use the voice activity detector, and pre-trained emotion models from the openEAR toolkit for live, incremental emotion recognition. The openSMILE toolkit features a large collection of baseline acoustic feature sets for paralinguistic speech and music analysis and a flexible and complete framework for audio analysis. In future work, more efforts will be put in documentation, speed-up of the underlying framework, and the implementation of new, robust acoustic and visual descriptors.

Acknowledgements

This research was supported by an ERC Advanced Grant in the European Community's 7th Framework Programme under grant agreement 230331-PROPEREMO (Production and perception of emotion: an affective sciences approach) to Klaus Scherer and by the National Center of Competence in Research (NCCR) Affective Sciences financed by the Swiss National Science Foundation (51NF40-104897) and hosted by the University of Geneva.

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement No.\ 338164 (ERC Starting Grant iHEARu).

The authors would like to thank audEERING UG (haftungsbeschränkt) for providing up-to-date pre-release documentation, computational resources, and great support in maintaining the free open-source releases.

Papers

[Eyben09] F. Eyben, M. Wöllmer, and B. Schuller, "openEAR - Introducing the Munich Open-Source Emotion and Affect Recognition Toolkit," in Proceedings 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops (ACII 2009), vol. I, Amsterdam, The Netherlands, pp. 576-581, HUMAINE Association, IEEE, September 2009.

[Eyben10] F. Eyben, M. Wöllmer, and B. Schuller, "openSMILE - The Munich Versatile and Fast Open-Source Audio Feature Extractor," in Proceedings of the 18th ACM International Conference on Multimedia (ACM'MM 2010), Florence, Italy, pp. 1459-1462, ACM, October 2010.

[Eyben13a] F. Eyben, F. Weninger, F. Groß, and B. Schuller, "Recent Developments in openSMILE, the Munich Open-Source Multimedia Feature Extractor," in Proceedings of the 21st ACM International Conference on Multimedia (ACM'MM 2013), Barcelona, Spain, pp. 835-838, ACM, October 2013.

[Eyben13b] Eyben, F.; Weninger, F.; Squartini, S.; Schuller, B., "Real-life voice activity detection with LSTM Recurrent Neural Networks and an application to Hollywood movies," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 483-487, 26-31 May 2013. doi: 10.1109/ICASSP.2013.6637694

[Eyben14] F. Eyben et al.: "The Geneva Minimalistic Acoustic Parameter Set (GeMAPS) for Voice Research and Affective Computing", submitted to IEEE Transactions on Affective Computing. 2015.

Authors: Florian Eyben and Björn Schuller
Affiliations: F. Eyben: Technische Universität München, Munich, Germany; B. Schuller: Chair of Complex and

Intelligent Systems (CIS), University of Passau, Passau, Germany and Department of Computing, Imperial College London, London, UK.

Call for Workshop Proposals @ ACM Multimedia 2015

We invite proposals for Workshops to be held at the ACM Multimedia 2015 Conference. Accepted workshops will take place in conjunction with the main conference, which is scheduled for October 26-30, 2015, in Brisbane, Australia.

We solicit proposals for two different kinds of workshops: regular workshops and data challenge.

Regular Workshops

The regular workshops should offer a forum for discussions of broad range of emerging and specialized topics of interest to the SIG Multimedia community. There are a number of important issues to be considered when generating a workshop proposal:

1. The topic of the proposed workshop should offer a perspective distinct from and complementary to the research themes of the main conference. We therefore strongly advise to carefully review the themes of the main conference (which can be found here), when generating a proposal.

2. The SIG Multimedia community expects the workshop program to be part of the program to nurture and to grow the workshop research theme towards becoming mainstream in the multimedia research field and one of the themes of the main conference in the future.

3. Interdisciplinary theme workshops are strongly encouraged.

4. Workshops should offer a discussion forum of a different type than that of the main conference. In particular, they should avoid becoming “mini-conferences” with accompanying keynote presentations and best paper awards. While formal presentation of ideas through regular oral sessions are allowed, we strongly encourage organizers to propose alternate ways to allow participants to discuss open issues, key methods and important research topics related to the workshop theme. Examples are panels, group brainstorming sessions, mini-tutorials around key ideas and proof of concept demonstration sessions.

Data Challenge Workshops

We are also seeking organizers to propose Challenge-Based Workshops. Both academic and corporate organizers are welcome.

Data Challenge workshops are solicited from both academic and corporate organizers. The organizers should provide a dataset that is exemplar of the complexities of current and future multimodal/ multimedia problems, and one or more multimodal/ multimedia tasks whose performance can be objectively measured. Participants in the challenge will evaluate their methods against the challenge data in order to identify areas of strengths and weakness. Best performing participating methods will be presented in the form of papers and oral/poster presentations at the workshop.

More information

For details on submitting workshops proposals and the evaluation criteria, please check the following site:

http://www.acmmm.org/2015/
call-for-workshop-proposals/

Important dates:

- Proposal Submission: February 10, 2015
- Notification of Acceptance February 27, 2015

Looking forward to receiving many excellent submissions!

Alan Hanjalic, Lexing Xie and Svetha Venkatesh
Workshops Chairs, ACM Multimedia 2015

MPEG Column: 110th MPEG Meeting

– original posts here by Multimedia Communication blog, Christian Timmerer, AAU/bitmovin

The 110th MPEG meeting was held at the Strasbourg Convention and Conference Centre featuring the following highlights: