

# SpeakerID: A Speaker Identification API and its Tools

Ville Hautamäki

student number: 124508

14.6.2002

University of Joensuu

Department of Computer Science

Special Project in Computer Science

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation Guide</b>	<b>2</b>
2.1	Installation to Linux . . . . .	2
2.2	Installation to Solaris (cs) . . . . .	2
<b>3</b>	<b>User's Guide</b>	<b>3</b>
3.1	Adding Users to the Speaker Database . . . . .	3
3.2	Identifying an Unknown User from a Speaker Database . . . . .	5
3.3	Removing Users from Speaker Database . . . . .	7
3.4	Extracting Cepstral Features from an Audio File . . . . .	7
3.5	Common Command line Options . . . . .	7
3.6	Supported Audio File Formats . . . . .	9
<b>4</b>	<b>General Overview of Library Modules</b>	<b>11</b>
4.1	External Libraries . . . . .	11
4.2	Add a New Speaker . . . . .	12
4.3	Identify an Unknown Speaker . . . . .	12
4.4	Pre-processing . . . . .	13
4.5	Feature Extraction . . . . .	13
4.6	Modeling . . . . .	15
4.7	Speaker Database Handling . . . . .	15
4.8	Matching . . . . .	16
<b>5</b>	<b>API Guide</b>	<b>16</b>
5.1	Functions in the Library . . . . .	16
5.2	Data Types and Structures . . . . .	20
5.2.1	Parameters struct . . . . .	20
5.2.2	Modeling struct . . . . .	21

5.2.3	Matching struct . . . . .	22
5.2.4	Decisionlogic struct . . . . .	23
5.2.5	Featextract struct . . . . .	24
5.2.6	Personal struct . . . . .	27
5.2.7	Nbest struct . . . . .	27
5.3	Internal File Formats . . . . .	28
5.3.1	Model File Format . . . . .	28
5.3.2	Speaker Database File Format . . . . .	29
<b>6</b>	<b>Testing</b>	<b>29</b>
	<b>References</b>	<b>30</b>

*"My voice is my passport, verify me."*

– Text dependent speaker verification  
password from the movie: Sneakers (1992)

# 1 Introduction

Speaker recognition is usually a general name referring to two different subtasks: *speaker identification* (SI) and *speaker verification* (SV) [4]. Speaker database can be either *closed-set* or *open-set*. Closed-set means that the unknown speaker is tested against the known speakers in the speaker database and open-set mean that unknown speaker might not exist in the speaker database of known speakers. This project work is concentrated on creation of sufficient general API to handle closed-set speaker identification problem. Also we created a few commandline tools to actually perform speaker identification task, and as such to provide tools to demonstrate the API.

Our work is based on the Special Project in Computer Science by Teemu Kilpeläinen and as such great deal of source code is actually written by him.

Figure 1: Schematic diagram of the closed-set speaker identification system. [4]

Figure 1 describes the general framework of the speaker identification system. This work includes all the parts mentioned in this figure. In Figure 1 the black arrows

describe the *training procedure* and the white arrows describe the *testing procedure*, which is sometimes called *matching*. Training creates the speaker database of known speakers by their speech samples. Matching tests unknown speaker against trained database of known speakers.

## 2 Installation Guide

This is a short installation guide. Installation in Linux and Solaris (cs).

### 2.1 Installation to Linux

Make sure, that you have `fftw`<sup>1</sup> and `libsndfile`<sup>2</sup> installed. Installation of those external libraries is easy, just uncompress them in their respective directories and then type as root:

```
./configure
make
make install
```

Then you just need to uncompress `speakerid-*.tar.gz` file to its own directory. Change to `speakerid/src/` directory and then type `make` to compile. To install compiled binaries to `/usr/local/bin` type as root `make install` and you are done.

### 2.2 Installation to Solaris (cs)

Make sure, that you have `fftw` and `libsndfile` installed. Installation of those external libraries is easy, just uncompress them in their respective directories and then type as root:

```
./configure
make Makefile
make install
```

In cs these external libraries are already installed. But problem is that, you cannot use them as dynamic loadable libraries.

---

<sup>1</sup>(<http://www.fftw.org>)

<sup>2</sup>(<http://www.zip.com.au/~erikd/libsndfile/>)

Then you just need to uncompress `speakerid-*.tar.gz` file to its own directory. Change to `speakerid/src/` directory. As you cannot use external libraries as dynamically loadable, you have to make sure that `LINKOPT` is correctly specified in `Makefile.solaris` with `-static` option, otherwise `libsndfile` does not work. Change to `speakerid/src/` directory and then type `make -f Makefile.solaris` to compile. To install compiled binaries to `/usr/local/bin` type as root `make -f Makefile.solaris install` and you are done.

### 3 User's Guide

There are two ways to use SpeakerID package: i) to compile all files and link them as a dynamic library and ii) to . use ready made programs that demonstrate API usage.

Actual building a dynamically loadable library is not covered by this documentation, but library is designed to be used either as a dynamic library or a static one. Static library compilation is easy, you just compile all library files with application C source files and then link all resulting object files together. In unix systems dynamic library is `.so` and in win32 it is called `.dll`. Both mean basically the same thing, basic functionality is hidden in library, and programmer provides it's own user interface to library functionality. Libraries, and it's API is further discussed in section 5.

Other possibility is to use ready made programs that demonstrate API usage, and thus provide user interface to it. In this section we concentrate on how to use these programs in unix environment. Library and programs have been developed in Linux environment, and Linux C library is somewhat different than Solaris one, which we have also tested these programs on. So it is prudent to suppose that also some strange "bugs" might crop up in different unix systems. Mandatory options of each program are listed in the heading.

In this section all executable included binaries are explained in their respective Sections. Binaries, except `removeuser` uses common commandline handling routines, those are described shortly in Section 3.5. Further description of all the options, and what they mean in this library is in Section 5.2.

#### 3.1 Adding Users to the Speaker Database

**`adduser -i inputfile -s speakerdb -I speakerID other options`**

Use `adduser` program to add user to speaker database. When `speakerdb` does not exist `adduser` will automatically create new one. Program's mandatory commandline options are: input audio file, speaker database file and unique speaker id. File names are strings and speaker id is integer.

Audio file contains voice sample from the user, which is used in training his/her speaker model. Speaker database file has to be given also to program, as new speaker is added to it. Speaker ID is an integer value that is unique between speakers. The id is used as a key when doing searches from speaker database. Option `-p <model path>` sets path, where model files reside, if it is not set, then `adduser` sets current working directory as that path. Note that feature is not part of the `speakerID` library but it is the feature of `adduser`

Here is example on how to run:

```
[villeh@cspc105 src]$ ./adduser -i spkr25_train.wav -I 25 -s test.db
```

First program will print all parameters (given or default), for example:

```
adduser: Fri May 31 15:46:38 EET DST 2002
By Ville Hautamäki, Teemu Kilpeläinen, Tomi Kinnunen, Ismo Kärkkäinen
Inputfile: ../spkr11_train.wav
SpeakerDB: 256/test.db
Path to Models: 256/
Window len: 30
Window shift: 10
Remove sil: 0
Remove DC: 1
Window function: 1
High emph coeff: 0.970000
Feature type: 1
Num. Mel filters: 0
Include delta: 0
Include delta delta: 0
Include 0th coeff: 0
MFCC ord: 12
Lifter type: 1
```

```
Absolute magnitude: 1
Model size: 256
Max iterations: 100
Stop threshold: 0.000010
Num. initial sol.: 5
Matching func: 1
Using CB in matching: 1
Max iterations in CB matching: 5
Decision func: 1
Number of best matches: 0
Confidence measure: 1
User's ID: 11
User's name: Foobar
User's gender: 1
User's age: 15
User's dialect: 2
```

Program will make model, which will have user id as its file name. It will also add reference to model file to speaker database file. Speaker database file is in *comma separated values* (CSV) format. Because speaker database is in CSV format user can edit the file with normal ASCII editor, like emacs. Further description of the format is in section 5.3.2.

Section 3.5 explains command line options in more detail.

### 3.2 Identifying an Unknown User from a Speaker Database

**identifyuser -i inputfile -s speakerdb other options**

Use `identifyuser` program to identify an unknown user against known speakers in the speaker database. Program's mandatory commandline options are: input audio file and speaker database file. Audio file contains a voice sample from user. Speaker database file has to be given also to program. Here is example on how to run:

```
[villeh@cspc105 src]$ ./identifyuser -i spkr1_hidas1.wav -s test.db
```

First program will print all parameters (given or default), for example:



```
identifyuser: Fri May 31 15:47:25 EET DST 2002
By Ville Hautamäki, Teemu Kilpeläinen, Tomi Kinnunen, Ismo Kärkkäinen
Inputfile: spkr1_hidas1.wav
SpeakerDB: ../../wavs/test.db
Path to Models: (null)
Window len: 30
Window shift: 10
Remove sil: 0
Remove DC: 1
Window function: 1
High emph coeff: 0.970000
Feature type: 1
Num. Mel filters: 0
Include delta: 0
Include delta delta: 0
Include 0th coeff: 0
MFCC ord: 12
Lifter type: 1
Absolute magnitude: 1
Model size: 64
Max iterations: 100
Stop threshold: 0.000010
Num. initial sol.: 5
Matching func: 1
Using CB in matching: 1
Max iterations in CB matching: 5
Decision func: 1
Number of best matches: 0
Confidence measure: 1
```

Program will print out an unsorted list of speakers in the database. For each speaker in the database match score against unknown sample is printed.

Section 3.5 explains commandline options in more detail.

### 3.3 Removing Users from Speaker Database

#### **removeuser -s speakerdb -I speakerID**

Program **removeuser** removes user from a speaker database. Mandatory options are speaker database file and speaker ID. Speaker ID is the integer value unique for each speaker. Note when the speakers entry is deleted from the database, his/her model file is retained. Leaving model file untouched does not pose any problems in usage. For each model system assigns speaker id as a model file name. Removed user's model file will be rewritten with new file when new user is allocated same user id.

```
[villeh@cspc105 src]$ ./removeuser -I 25 -s test.db
```

### 3.4 Extracting Cepstral Features from an Audio File

#### **featextract -i inputfile -o outputfile -s speakerdb other options**

Program **featextract** uses just the feature extraction part of the speakerID library, it outputs cepstral features in an ASCII output file. Program's mandatory commandline options are: input audio file, speaker database file and output feature file. Audio file contains a voice sample from user. Speaker database file has to be given also to program, due to the implementation reasons. Output feature file will contain cepstral features in ASCII format similar to model file described in Section 5.3.1. Here is example on how to run:

```
[villeh@cspc105 src]$ ./featextract -i spkr1_hidas1.wav -o output.txt -s test.db
```

### 3.5 Common Command line Options

Following describes command line option in more detail. All options are so called long options, starting with two hyphens(--) following option string. Some frequently used options also have one character shorthand they should be called only with one hyphen (-). Further description of all the options, and what they mean in this library is in Section 5.2.

Table 1: Basic options

Option	Default	Description
-h -help	CWD	Print help and exit
-V -version		Print version and exit
-i STRING -input=STRING		Audio inputfile containing speakers sample
-s STRING -speakerdb=STRING		Speaker database file
-p STRING -modelpath=STRING		Path to speaker model files
-o STRING -featurefile=STRING		Feature extraction output file (debug)

Table 2: Modeling, matching and decision logic options

Option	Default	Description
-model_size=INT	64	Model size (clusters per speaker)
-max_iterations=INT	100	Stopping criterion of GLA, Max iterations
-stop_threshold=DOUBLE	0.00001	Stopping criterion of GLA, Min error, threshold
-initial_solutions=INT	5	Number of initial solutions (RGLA)
-matching_function=INT	1 = AVDIST	Select matching function
-using_cb	on	Create codebook also for matching
-maxiter=INT	5	Max iterations for codebook generation
-cb_size	0	Intermediate codebook size 0 is 1/10 Cepstrum size
-decision_function=INT	1 = NBEST	Select decision function
-nbest=INT	0	Number of best matches 0 = all matches
-confidence_measure=INT	1 = NNRAIO	Select confidence measure

*"You mean you saw the man?*

*You can identify the murderer?"*

– from the movie: Murder on the Orient Express (1974)

Table 3: Feature extraction options

Option	Default	Description
<code>-win_length=INT</code>	30	Window length in ms
<code>-win_shift_ms=INT</code>	10	Window shift in ms
<code>-remove_sil</code>	off	Remove silence
<code>-sil_thr</code>	0.4	Silence removal coefficient
<code>-remove_dc_offset</code>	off	Remove DC-offset from input signal
<code>-win_function_type=INT</code>	1 = HAMMING	Select window function
<code>-high_emph_coeff=DOUBLE</code>	0.97	High emphasis coefficient
<code>-feat_type=INT</code>	1 = MFCC	Feature-type
<code>-num_mel_filters=INT</code>	0	Number of mel-filters 0 = auto detected
<code>-include_delta</code>	off	Include delta cepstral coefficients
<code>-include_delta_delta</code>	off	Include delta delta cepstral coefficients
<code>-include_0th_coeff</code>	off	include 0th cepstral coefficient
<code>-MFCC_ord=INT</code>	12	MFCC-order = dimension of result vector
<code>-lifter_type=INT</code>	1 = truncated	Cepstral lifter function type
<code>-absolute_magnitude</code>	on	Absolute magnitude

Table 4: Speaker options

Option	Default	Description
<code>-I INT -id=INT</code>		Unique speaker ID
<code>-N STRING -name=STRING</code>		Speakers name
<code>-gender=INT</code>		Speakers gender
<code>-age=INT</code>		Speakers age
<code>-dialect=INT</code>		Speakers dialect

### 3.6 Supported Audio File Formats

SpeakerID library uses libsndfile as a lowlevel audio file handling library. Thus all audio file formats supported by libsndfile are also supported by speakerID. Here is the list of supported formats [1]:

- Microsoft WAV 8, 16, 24 and 32 bit integer PCM.
- Microsoft 32 bit floating point PCM.

- Microsoft IMA/DVI ADPCM WAV format (16 bits per sample compressed to 4 bits per sample); mono and stereo only.
- Microsoft ADPCM WAV format (16 bits per sample compressed to 4 bits per sample); mono and stereo only.
- Microsoft 8 bit A-law and u-law formats (16 bits per sample compressed to 8 bits per sample); single or multi channel.
- Microsoft WAV files using GSM 6.10 encoding; mono only.
- Apple/SGI AIFF and AIFC uncompressed 8, 16, 24 and 32 bit integer PCM.
- Apple Macintosh OSX AIFC files with little endian PCM data (16, 24 and 32 bit).
- Apple/SGI AIFC files containing 32 bit floating point data.
- Sun/NeXT AU/SND format (big endian 8, 16, 24 and 32 bit PCM, 8 bit u-law and 8 bit A-law).
- Dec AU format (little endian 8, 16, 24 and 32 bit PCM, 8 bit u-law and 8 bit A-law).
- Headerless 8kHz 8bit u-law encoded AU/SND files (mono only).
- G721 and G723 ADPCM encoded AU/SND files (mono only).
- RAW header-less PCM files of 8, 16, 24 and 32 bits. The 16, 24 and 32 bit files may be big or little endian byte ordering. The 8 bit samples may be signed or unsigned values.
- Amiga uncompressed IFF / 8SVX / 16SV PCM files (8 and 16 bit)(mono only).
- Ensoniq PARIS big and little endian, 16 and 24 bit PCM files (.PAF) .
- Sphere NIST 16, 24 and 32 bit PCM files. Big and little endian supported.
- Propellorhead's Reason REX files (read only).
- IRCAM (Berkeley, CARL) files with 16 and 32 bit PCM, A-law, u-law and float data.

## 4 General Overview of Library Modules

User of the library needs only include `speakerid.h` file to his own project. In `speakerid.c` all the high level operations are implemented. Library has two fundamental operations (others also exist), adding new speakers and identifying unknown speaker.

### 4.1 External Libraries

Following source files are modified very slightly by me or not at all. Note though, that Teemu Kilpeläinen's Special Project in Computer Science worked as a basis for my work.

#### Teemu Kilpeläinen

`mel-functions.c`, `mel-functions.h`

#### Tomi Kinnunen and Ismo Kärkkäinen

`clustering.c`, `clustering.h`, `distortion.c`, `distortion.h`, `fvec.c`, `fvec.h`,  
`matrix.c`, `matrix.h`, `misc.c`, `misc.h`, `pattern.c`, `pattern.h`, `memctrl.c`, `memctrl.h`,  
`textfile.c`, `textfile.h`

#### Timo Kaukoranta

`random.c`, `random.h`

#### The GNU Project

`getopt_long()` can be found from [3].

`getopt1.c`, `getopt.c`, `getopt.h`

#### Matteo Frigo and Steven G. Johnson

`libfftw.so` in unix and `libfftw.dll` in windows [2].

**Eric de Castro Lopo**

`libsndfile.so` in unix and `libsndfile.dll` in windows [1].

## 4.2 Add a New Speaker

Figure 2: A high level diagram of adding a new speaker

Figure 2 describes the general outline of the speaker adding procedure, in which audio input file is first preprocessed. The feature extraction procedure is performed on the preprocessed signal. Output of the feature extraction module is a set of feature vectors. These feature vectors are fed into modeling procedure, which creates speaker model from the vectors. After that operation speaker database is updated. Individual parts of this procedure are explained in more detail in this Section.

## 4.3 Identify an Unknown Speaker

Figure 3: A high level diagram of identifying an unknown speaker

Figure 3 describes the general outline of identifying unknown speakers against speaker database. Audio input file is first preprocessed. The feature extraction procedure is performed on the preprocessed signal. Output of the feature extraction module is a set of feature vectors. These feature vectors are fed directly to matching procedure. Other option is to first model quickly feature vectors, that is, to create a codebook of them and then fed them into matching procedure. First modeling and then matching will considerably speed up the matching. In matching module, features are compared

with one by one to each model in speaker database. Score of each comparison is returned in an *Nbest list*. Nbest list is further explained in the Section 5.2.7.

#### 4.4 Pre-processing

Figure 4: More detailed look on the pre-processing procedures

Figure 4 shows two basic procedures of signal pre-processing phase, namely DC component remove and high emphasis filtering. All pre-processing functionality is implemented in `preprocess.c` file. All pre-processing is done on audio signal, output is still processed audio signal. This means that these filters are global to the whole signal, contrary to rest of the signal processing in this library, which filter short term frames from the signal.

High emphasis filters transfer function is defined as:

$$H(z) = 1 - \alpha z^{-1}. \quad (1)$$

Where  $H(z)$  is the transfer function and  $\alpha$  is user defined scalar variable, by default it is set to 0.97.

DC component remove function centers signal around 0. Procedure is also called signal bias removal.

#### 4.5 Feature Extraction

Figure 5: More detailed look on the feature extraction process

Figure 5 shows the general outline of the feature extraction module, which is implemented in `feature_extraction.c`. This module gets as parameters pointer to the input signal and pointer to the `Parameter` struct. First module creates a frequency



spaced triangular *mel filter-bank*, which is implemented in `mel-functions.c`. Number of filters in mel filterbank is either user selected or calculated from signals sample rate.

Next signal is framed to short overlapping segments, called *frames*. Each frame is also windowed in time domain by either of following window functions: Hamming, Barlett, Hanning and Blackman. Hamming is default windowing function used. Framing and related functions are implemented in `frame.c`.

For each frame, FFT is calculated, using an external library called *Fastest Fourier Transform in the West* (FFTW). Either *absolute magnitude spectrum* or *power spectrum* is calculated from FFT frame, thus effectively throwing out complex numbers. This feature is implemented in `feature_extraction.c` Absolute magnitude is the default setting.

Figure 6: Example of absolute manitude spectral frame

Figure 6 is an example of absolute magnitude frame. Where X-axis describes the frequency scale and Y-axis describes the magnitude of the corresponding frequency.

Spectral frame is then filtered with the mel filterbank. Then logarithm is taken from the filtered frame, which is in turn transformed by *discrete cosine transform* (DCT), also from `mel-functions.c`.

Output of the DCT is then *liftered* with either raised sine, half raised sine or it is just truncated. Liftering is implemented in `feature_extraction.c`. Output is a of set cepstral feature vectors in `PATTERN` data structure by Tomi Kinnunen, from `pattern.c`.

## 4.6 Modeling

Figure 7: More detailed look on the modeling process

Figure 7 shows how higher level modeling procedure works. Clustering means that we use some method to represent cepstral feature vectors with smaller number of *codevectors*. General outline of this procedure is implemented in `speakerid.c` and more detailed implementation is by Tomi Kinnunen and Ismo Kärkkäinen in `clustering.c`. For speaker model generation we use *Randomized GLA* and for intermediate codebook in matching we use GLA. GLA is used, because codebook should be generated quickly. Number of iterations and number of initial solutions is user selectable. Output codebook is in `PATTERN` format, and thus written into file with functions from `pattern.c`.

## 4.7 Speaker Database Handling

Figure 8: More detailed look on the speaker database handling

Figure 8 shows how speaker database is actually implemented. Retrieval of individual speakers from the database and updating the whole database is done with functions implemented in `speakerid.c`. Database data structure is a doubly linked list. Each list `Element` contains all information in `Personal` struct, which is explained in more detail in Section 5.2.6. Module `csv.c` handles actual database file reading, parsing and writing. Database file format is described in Section 5.3.2.

## 4.8 Matching

Figure 9: More detailed look on the matching procedure

Figure 9 shows more detailed look on the matching procedure. General level of matching is implemented in the `speakerid.c`. Cepstrum is the PATTERN of feature vectors, it can be either raw features from feature extraction procedure or intermediate codebook created from the raw vectors. `MeanSquaredError` function from `pattern.c` calculates MSE value between Cepstrum and each model one by one. MSE value and user id are stored in Nbest structure. Nbest handling is implemented in `speakerid.c`. Matching outputs Nbest list.

## 5 API Guide

SpeakerID library consists of functions that operate the speaker database, two structures `Parameters` and `Personal` and some internal files. In this section those features are discussed in more detail.

### 5.1 Functions in the Library

#### `int create_speakerdb(Parameters *paraptr)`

Creates a new speaker database file. This function is not really useful, as `add_speaker()` will automatically create new speaker database file, if it does not exist. Takes as a parameter pointer to parameter struct. Field `speaker_db` in that struct has to be set!

Example on how to use this function:

```
#include "speakerid.h"
...

if (!paraptr->speaker_db) {
    printf("No Speaker DB set \n");
```

```

        exit(-1);
    }

    create_speakerdb(paraptr);

```

Function returns 1 on error and 0 on success.

**int add\_speaker(Parameters \*paraptr, Personal \*perptr)**

Creates a new speaker model, and adds reference to that model to speaker database. Takes as a parameters pointer to **Parameters** struct and pointer to **Personal** struct, containing personal information about the speaker. Mandatory fields in the **Parameters** struct are: **instr**, **speaker\_db** and **path\_models**. **instr** has to contain path to the audio file containing a voice sample from the speaker. In **Personal** struct all fields are mandatory, but only **id** has to be set to something sensible.

*Note:* user of the library has to make sure, that all id's are unique, this library does not do that checking for the user.

Example on how to use this function:

```

#include "speakerid.h"
...

if (!paraptr->speaker_db || !paraptr->instr || !paraptr->path_models) {
    printf("Some mandatory paths are not set\n");
    exit(-1);
}

if (!perptr->id) {
    printf("User ID not set");
    exit(-1);
}

add_speaker(paraptr, perptr);

```

Function returns 1 on error and 0 on success.

**int remove\_speaker(Parameters \*paraptr, int id)**

Removes the speaker from speaker database. Takes as parameters a pointer to **Parameters** struct and user id of the speaker marked for removal. Mandatory field in the **Parameters** struct is **speaker\_db**. Function does not remove model file, from the filesystem. This does not pose any problem as new files with the same id overwrite the old one.

Example on how to use this function:

```
#include "speakerid.h"
...

if (!paraptr->speaker_db) {
    printf("No Speaker DB set \n");
    exit(-1);
}

if (!perptr->id) {
    printf("User ID not set");
    exit(-1);
}

remove_speaker(paraptr, perptr->id);
```

Function returns 1 on error and 0 on success. Error in this context usually means, that user by that id does not exist in the database.

**Nbest \*find\_most\_similar\_spkr(Parameters \*paraptr)**

Returns an **Nbest** list of known speakers against unknown sample. Takes as a parameter pointer to the **Parameters** struct. Mandatory fields in the **Parameters** struct are: **speaker\_db** and **instr**. **instr** has to be the path to audio file, containing voice sample from the speaker.

Function returns linked list of **Nbest** structures, where each struct contains information from one speaker model in the database. This is list is unsorted. In **Nbest** structure **match\_score** fields contains raw match information (mean squared error between unknown speaker and known speaker from the speaker database). In typical speaker identification application, speaker with the lowest

`match_score` is decided as the unknown speaker. The `id` field contains the unique id number of the identified speaker. Explanation of `Nbest` structure can be found in this same section.

When irrecoverable error is detected function returns `NULL`.

Use `remove_nbest()` function to remove `Nbest` structures from the memory.

Example on how to use this function:

```
#include "speakerid.h"
...

Nbest *listhead;

if (!paraptr->speaker_db) {
    printf("No Speaker DB set \n");
    exit(-1);
}

listhead = find_most_similar_spkr(paraptr);
if (!listhead) {
    printf("Error detected\n");
    exit(-1);
}
print_nbest(listhead, stdout);
remove_nbest(listhead);
```

#### **void remove\_nbest(Nbest \*nbestptr)**

Removes `Nbest` list from the memory. Takes as a parameter pointer to the first element in the `Nbest` list.

Function does not return anything.

#### **void print\_nbest(Nbest \*listhead)**

Prints `Nbest` list to the `stdout`. Takes as a parameter pointer to the first element in the `Nbest` list. This function is mostly useful in the debugging.

Function does not return anything.

**int print\_features(Parameters \*paraptr)**

Prints feature vector table. Takes as a parameter pointer to **Parameters** struct. Mandatory parameters are **instr**, which is path to input audio file and **featurefile**, which is a path to output file. This function is useful in debugging purposes.

Function returns 0 on success and 1 on any error.

**char \*get\_version(void)**

Returns version string of the library.

**5.2 Data Types and Structures**

**Parameters** structure contains fields: **featptr**, **modelptr**, **matchptr** and **deciptr**. These structures, in turn, give parameters to different submodules in SpeakerID library. If these pointers are set to NULL, then default values will be set to those parameters. Usually it is not necessary to change the default values. Other fields in the **Parameter** struct should be usually set (depending on the API function you are using) excluding **infile** and **dbfile**, which are for internal use only. In **Personal** struct all fields should be set. Validity of parameters are not usually checked, except in case of file names.

**5.2.1 Parameters struct**

```
struct Parameters {
    char *instr;
    char *speaker_db;
    char *path_models;
    char *featurefile;
    Featextract *featptr;
    Modeling *modelptr;
    Matching *matchptr;
    Decisionlogic *deciptr;
};
```

**char \*instr**

String, which contains path to input audio file. Mandatory parameter.

**char \*speaker\_db**

String, which contains path to speaker database file. Mandatory parameter.

**char \*path\_models**

String, which contains path to models directory. In UNIX path has to end with / character and in Windows it has to end with \. Mandatory parameter.

**char \*featurefile**

String, which contains name of the feature extraction output file, when user just wants to generate feature vectors and nothing more. This parameter is used internally in `featextract.c`. Usually no need to worry, set to NULL if in doubt.

**Featextract \*featptr**

Pointer to **Featextract** structure. That structure contains parameters specific to feature extraction. If set to NULL default parameters will be set to all feature extraction parameters.

**Modeling \*modelptr**

Pointer to **Modeling** structure. That structure contains parameters specific to modeling. If set to NULL default parameters will be set to all modeling parameters.

**Matching \*matchptr**

Pointer to **Matching** structure. That structure contains parameters specific to matching. If set to NULL default parameters will be set to all matching parameters.

**Decisionlogic \*deciptr**

Pointer to **Decisionlogic** structure. That structure contains parameters specific to decision logic. If set to NULL default parameters will be set to all decision logic parameters.

**5.2.2 Modeling struct**

```
struct Modeling {  
    int model_size;
```



```
int max_iterations;  
double stop_threshold;  
int num_initial_solutions;  
};
```

**int model\_size**

Size of the speaker model. This corresponds to number of code vectors in the speaker model. Default value is 64. Rule of the thumb is that larger model improves identification rate but slows down the modeling phase. However this is not a problem as modeling is done offline.

**int max\_iterations**

Maximum number of GLA iterations. This is one of the GLA stopping criterions used. Default value is 100. Rule of the thumb is that more iterations improves identification rate but slows down the modeling phase. However this is not a problem as modeling is done offline.

**double stop\_threshold**

Minimum change between GLA iterations. Modeling stops either when **max\_iterations** iterations are reached or change in the quality of the produced clustering is under this threshold. Default is 0.0001.

**int num\_initial\_solutions**

Initial number of solutions for RGLA. Normal GLA might get stuck to local minimum. For that reason, RGLA starts with number of different initial solutions and returns the best solution. Default is 10.

**5.2.3 Matching struct**

```
struct Matching {  
    int matching_function;  
    int using_cb;  
    int maxiter;  
};
```

**int matching\_function**

Selects matching function to use. At this moment only distance between sets known speaker in the database and the unknown speakers feature vectors is implemented. Other future possibilities are for example weighted matching.

**int using\_cb**

If `using_cb` is set, then library creates intermediate codebook before matching and uses that instead of feature vectors. `model`. This significantly speeds up matching. If using this feature speedwise bottleneck is creation of intermediate codebook. By default this flag is set on.

**int maxiter**

Number of iterations when creating the intermediate codebook. Default is 5. Rule of thumb is that lower the number faster is the matching, but identification rate usually drops at the same time.

**int cb\_size**

Size of the intermediate codebook. If it is set to 0, then codebook size is set to 1/10 of feature set. This affects modeling time. By default this is set to 0.

**5.2.4 Decisionlogic struct**

```
struct Decisionlogic {  
    int decision_function;  
    int num_best;  
    int confidence_measure;  
};
```

**int decision\_function**

Sets decision function used. Right now only Nbest list is supported, which is set with 1. Nbest means in this context, that decision logic will send  $n$  best matches to application layer. These are represented in the form of linear linked list of `Nbest` structs, which is explained in more detail in 5.2.7.

**int num\_best**

Sets the number of best matches. This feature is not yet implemented. Value 0 means keep all matches. Default is 0.

**int confidence\_measure**

Selects used confidence function. Distances returned by matching function are unnormalized, meaning that for one set of speakers close to other speaker could be something like 1000 and for other just 10. Thus we need somekind of normalized score. We define confidence here to be value  $a$  in range  $[0...1]$ . Where 1 is absolute confidence and 0 is no confidence at all. And as such should be read, that at what confidence matched speaker is correct match. This feature is not yet implemented. Default is 1.

**5.2.5 Featextract struct**

```
struct Featextract {  
    // Global parameters for whole signal  
    int win_length;  
    int win_shift_ms;  
    int remove_sil;  
    double sil_thr;  
    int remove_dc_offset;  
    int win_function_type;  
    double high_emph_coeff;  
    // Cepstral coefficient parameters  
    int feat_type;  
    int num_mel_filters;  
    int include_delta;  
    int include_delta_delta;  
    int include_0th_coeff;  
    int MFCC_ord;  
    int lifter_type;  
    int absolute_magnitude;  
};
```

**int win\_length**

Analysis window length in ms. Default is 30.

**int win\_shift\_ms**

Analysis window shift in ms. Default is 10.

**int remove\_sil**

Silent frames are removed as a post processing feature if this option is set. Idea works like this, in cepstrum feature vector  $c_0$  coefficient contains log total energy of the frame. We assume that silent frames have low energy content, so those frames with sufficiently low  $c_0$  can be dropped. First we calculate threshold  $s$  (given below) and then drop every cepstrum vector if its  $c_0$  is below  $s$ .

User should try removing silence when identifying takes too much time, as matching procedure takes most of the time here. And less feature vectors are given to matching less time matching takes.

By default silence removal is set to off.

**double sil\_thr**

User defined threshold coefficient for silence removal. It works like this:

$$s = \alpha * \frac{\sum_{j=1}^n c_{0,j}}{n}. \quad (2)$$

Where  $\alpha$  is the user defined threshold coefficient,  $c_{0,j}$  is the  $j$ :th  $c_0$  coefficient in the cepstrum and  $s$  is the calculated threshold. Smaller  $\alpha$  more frames are detected as silence and thus removed. Default  $\alpha$  is 1.0.

**int remove\_dc\_offset**

If set removes DC offset (signal bias, zero frequency) from signal. Default is on.

**int win\_function\_type**

Windowing function type. Possible types are: 1 = Hamming, 2 = Rectangular, 3 = Barlett and 4 = Blackman. Default is 1.

**double high\_emph\_coeff**

High emphasis coefficient. Default is 0.97.

**int feat\_type**

Feature type. Only MFCC is supported at this time, which is set with 1.

**int num\_mel\_filters**

Set the number of mel filters. If set to 0 system will calculate appropriate number of mel filters with the following equation:

$$m = \lceil 3 * \log f_s \rceil, \quad (3)$$

where  $f_s$  is signal sample rate and  $m$  is the number of mel filters in mel filter bank. Default is 0.

#### **int include\_delta**

If set calculates delta coefficients. This feature is not implemented yet. Default is off.

#### **int include\_delta\_delta**

If set calculates delta delta coefficients. This feature is not implemented yet. Default is off.

#### **int include\_0th\_coeff**

If set includes 0th cepstral coefficient. 0th coefficient describes log-total energy of the frame, and thus contains no usefull speaker dependent information. Also 0th coefficient has much bigger absolute magnitude than other cepstral coefficients, so it would dominate in modeling or and matching process. It is not usually needed and thus default is off.

#### **int MFCC\_ord**

MFCC order. When 0th coefficient is removed, MFCC order is the dimension of the feature vector. When 0th coefficient is included, dimension is MFCC + 1. Default is 12.

#### **int lifter\_type**

Sets lifter function type. 1 = truncated, 2 = raised sine and 3 = half raised sine. Default is 1.

#### **int absolute\_magnitude**

Sets whether magnitude spectrum or power spectrum (which is squared magnitude) is used. Default is on.

### 5.2.6 Personal struct

Fields in this struct are copied directly to speaker database in `add_user()`.

```
struct Personal {  
    int id;  
    char *name;  
    int gender;  
    int age;  
    int dialect;  
};
```

#### **int id**

Unique speaker ID. ID at value 0 is defined to NULL, meaning no user should have that value. This option is mandatory.

#### **char \*name**

Sets speakers name. Put the whole name here, with first and last name.

#### **int gender**

Sets speakers gender. 0 = male and 1 = female.

#### **int age**

Sets users age.

#### **int dialect**

Sets users dialect.

### 5.2.7 Nbest struct

Figure 10: Nbest structures organized in linear linked list

Figure 10 shows how, **Nbest** list is organized. List does not have head element, nor is it doubly linked. At this moment list is not ordered in anyway. Last element in the list has **next** set to NULL.

```
struct Nbest {  
    double match_score;  
    double confidence;  
    int id;  
    Nbest *next;  
};
```

**double match\_score**

Is set to raw distance between unknown speaker and model. Lower value is better.

**double confidence**

Confidence value at range [0...1]. Larger the value better confidence, that identified speaker is correctly identified. Feature is not implemented yet.

**int id**

Unique ID of matched model.

**Nbest \*next**

Pointer to the next element in the **Nbest** list.

## 5.3 Internal File Formats

### 5.3.1 Model File Format

Each cluster representative (codebook centroid) is in one row of the ASCII file. Vector components are separated with a single space. All scalars are floating point numbers. See example of actual model file:

```
-5.05449 -0.64882 0.76464 -0.88285 -0.08399 -0.79391 0.99402 -0.38675 0.07316  
1.38093 -0.55842 0.61479 -0.71368 0.15056 -0.69219 0.53067 -0.15405 0.46711  
6.24849 -0.91968 2.24709 -2.13802 -1.67578 -1.77014 0.34853 -1.01528 -0.07106  
8.00430 -2.03435 -1.59680 -3.63113 1.15400 -0.88487 -0.14261 -1.03327 0.51821
```

### 5.3.2 Speaker Database File Format

Table 5: Explanation of different fields in the speaker database file

Speaker ID	Model file	Name	Gender	Age	Dialect
------------	------------	------	--------	-----	---------

Speaker database is in comma separated value (CSV) format, where one speaker takes one row. Table 5 shows the ordering of the fields in speaker database CSV file. First two fields are most important, rest can be anything. First field tells unique speaker ID and the next path to speakers model file. See example of actual speaker database file:

```
25,4/25,Foobar,1,15,2
23,4/23,Foobar,1,15,2
21,4/21,Foobar,1,15,2
19,4/19,Foobar,1,15,2
```

## 6 Testing

Figure 11: Results of identification rate when varying codebook size

Figure 11 shows *identification rate* as a function of codebook size. Identification rate is calculated with correctly identified divided by total number of speakers. In test



run, we used speaker database consisting of 25 speakers. X axis in the Figure 11 is in logarithmic scale, so inconsistency with small codebooks is shown here. From codebook size 64 onwards identification rate was 100we conclude that SpeakerID works as expected.

## References

- [1] de Castro Lopo E. *Libsndfile*, Project library website, no affiliate, <http://www.zip.com.au/~erikd/libsndfile/>, 6.6.2002.
- [2] Frigo M. and Johnson S.G. *FFTW*, FFTW Home Page, Massachusetts Institute of Technology, <http://www.fftw.org/>, 11.6.2002.
- [3] The GNU Project. *GNU*, The Gnu Project Homepage, <http://www.gnu.org>, 11.6.2002.
- [4] Kinnunen T. *Vector Quantization in Speaker Recognition*, Research website, University of Joensuu, <http://cs.joensuu.fi/pages/tkinnu/research/index.html>, 1.6.2002.

*"You are such a fine speaker that  
I'm afraid you may actually end in convincing yourself."*  
– from the movie: Mansfield Park (1999)