Spatial_Audio_Framework

Generated by Doxygen 1.8.17

Fri Apr 17 2020 17:12:09

1 Spatial_Audio_Framework

A cross-platform Spatial Audio Framework for developing spatial audio related applications.

git: `https://github.com/leomccormack/Spatial_Audio_Framework`, doxygen: `http←://research.spa.aalto.fi/projects/spatial_audio_framework`

1.1 Prerequisites

The framework requires the following libraries:

- Any Library/Libraries conforming to the `CBLAS` and `LAPACK` standards
- (Optional) `netCDF` for reading `SOFA` files

The rationale for the former requirement is that the framework employs the use of CBLAS/LAPACK routines for tackling all of the linear algebra operations, which are used quite prolifically throughout the code. Therefore, a performance library, which conforms to the CBLAS/LAPACK standards, is required by most of the framework modules. In principle, any such library (or combination of libraries) may be employed for this task, and if you've worked with such libraries before, then you probably already know what to do. However, using a custom Intel MKL library for this requirement is still generally recommended, as this is the approach used by the developers.

1.2 Available CBLAS/LAPACK options

Define one of the following preprocessor definitions, which will dictate which library/libraries you should link to your project:

```
SAF_USE_INTEL_MKL
SAF_USE_OPEN_BLAS_AND_LAPACKE
SAF_USE_ATLAS
```

- SAF_USE_INTEL_MKL - to use `Intel MKL`, or a ∗∗custom Intel MKL library∗∗ (recommended for x86←_64/amd64).
- SAF_USE_OPEN_BLAS_AND_LAPACKE - to use `OpenBLAS` and the LAPACKE interface (recommended for ARM)
- SAF_USE_ATLAS - to use `ALTAS` which is not recommended, since some LAPACK functions are missing. However, if you don't mind loosing some framework functionality, then ATLAS may still be a good choice for your particular project.

MacOSX users only: If you do not define one of the above flags, then SAF will use `Apple Accelerate` for CBLAS/LAPACK and also vDSP for the FFT. However, note that Intel MKL is still the more recommended option, as it is generally faster than Accelerate.

1.3 Enable SOFA support (Optional)

In order to use the built-in SOFA reader (framework/modules/saf_hrir/saf_sofa_reader.h), your project must also link against the netCDF library (including its dependencies). For those already familar with building and linking this particular library, you know what to do. However, for convenience, suggested platform specfic instructions have been provided below.

Note that the following preprocessor definition is also required:
```
SAF_ENABLE_SOFA_READER
```

1.3.1 Windows (64-bit) and MacOSX users

For convenience, the following statically built libraries are included in the "dependencies" folder; simply link your project against them:
```
libszip.lib; libzlib.lib; libhdf5.lib; libhdf5_hl.lib; netcdf.lib; # Win64
netcdf; hdf5; hdf5_hl; z; # MacOSX
```

Also, make sure to add the appropriate 'include' and 'lib' directories to your project's header and library search paths, respectively.

1.3.2 Linux (amd64) and Raspberry Pi (ARM) users

For ubuntu based distros, you may install netCDF and its dependencies with the following:
```
sudo apt-get install libhdf5-dev
sudo apt-get install libnetcdf-dev libnetcdff-dev
```

Then simply add the appropriate directory to the header search path, and add this linker flag to your project (or wherever it was installed):
```
-L/lib/x86_64-linux-gnu -lnetcdf
```

1.4 Using the framework

Once a CBLAS/LAPACK flag is defined (see above), and the correct libraries are linked to your project, you can now add the files found in the "framework" folder to your project. Then add the following directory to your header search paths:
```
Spatial_Audio_Framework/framework/include
```

The framework's master include header is then:
```
#include "saf.h"
```

Detailed instructions regarding how to use the functions offered by each of the framework's module, is provided in the main header file for the respective module (e.g. "/modules/saf_sh/saf_sh.h", or "/modules/saf_vbap/saf_vbap.↩ h").

1.4.1 Documentation

Documentation generated using Doxygen may also be found here.

Alternatively, you may compile the most recent documentation (HTML) yourself using the following command:
```
doxygen doxygen/doxygen_config
# optional, to build the pdf version:
cd doxygen/latex
make
```

1.4.2 Examples

Many examples have been included in the repository, which may also serve as a starting point for learning the framework:

- **ambi_bin** - a binaural Ambisonic decoder with built-in rotator. It includes the following decoding approaches: least-squares (LS), spatial re-sampling (SPR), Time-alignment (TA) [1], Magnitude Least-Squares (MagLS) [2].
- **ambi_dec** - a frequency-dependent Ambisonic decoder. Including the following decoding approaches: sampling ambisonic decoder (SAD), AllRAD [3], Energy-Preserving decoder (EPAD) [4], Mode-Matching decoder (MMD).
- **ambi_drc** - a frequency-dependent dynamic range compressor (DRC) for Ambisonic signals, based on the design proposed in [5].
- **ambi_enc** - a simple Ambisonic encoder.
- **array2sh** - converts microphone array signals into spherical harmonic signals (aka Ambisonic signals), based on theoretical descriptions [6,7]. More details found in [8].
- **beamformer** - a beamforming example with several different beamforming options.
- **binauraliser** - convolves input audio with interpolated HRTFs, which can be optionally loaded from a SOFA file.
- **dirass** - a sound-field visualiser based on re-assigning the energy of beamformers. This re-assignment is based on the DoA estimates extracted from spatially-localised active-intensity vectors, which are biased towards each beamformer direction [9].
- **panner** - a frequency-dependent VBAP panner [10], which permits source loudness compensation as a function of the room [11].
- **matrixconv** - a basic matrix convolver with an optional partitioned convolution mode.
- **multiconv** - a basic multi-channel convolver with an optional partitioned convolution mode.
- **powermap** - sound-field visualiser based on beamformer (PWD, MVDR) energy or sub-space methods (M↩ USIC).
- **rotator** - rotates spherical harmonic signals (aka Ambisonic signals) given yaw-pitch-roll angles [12].
- **sldoa** - a sound-field visualiser based on directly depicting the DoA estimates extracted from multiple spatially-localised active-intensity vectors; as proposed in [8].

Note that these examples have also been integrated into VST audio plug-ins using the JUCE framework and can be found `here`.

1.5 Contributing

Suggestions and contributions to the code are both welcomed and encouraged. It should be highlighted that, in general, the framework has been designed to be highly modular with plenty of room for expansion. Therefore:

- if you are researcher who has, for example, developed a new spatial-audio related method and want to integrate it into the framework... or
- if you notice that an existing piece of code can be rewritten to make it clearer/faster, or to fix a bug...

then please feel free to do so. Although, note that if you wish to make a substantial change or addition, then consider first discussing it by raising a github "issue" or contacting the developers directly via email; we may be able to help.

Note that we also very much appreciate feedback from developers who are using this framework, so if you are using it for an interesting project, then please let us know :-)

1.6 Developers

- **Leo McCormack** - C programmer and algorithm design (contact: leo.mccormack@aalto.fi)
- **Symeon Delikaris-Manias** - algorithm design
- **Archontis Politis** - algorithm design

1.7 License

This framework is provided under the ISC license. However, it also includes a modified version of the 'alias-free STFT' implementation by Juha Vilkamo (MIT license); kissFFT (BSD 3-clause license) by Mark Borgerding; and the 'convhull_3d' 3-D Convex Hull implementation by Leo McCormack (MIT license).

1.8 References

[1] Zaunschirm M, Scho"rkhuber C, Ho"ldrich R. **Binaural rendering of Ambisonic signals by head-related impulse response time alignment and a diffuseness constraint**.
The Journal of the Acoustical Society of America. 2018 Jun 19;143(6):3616-27.

[2] Scho"rkhuber C, Zaunschirm M, Ho"ldrich R. **Binaural Rendering of Ambisonic Signals via Magnitude Least Squares**.
InProceedings of the DAGA 2018 (Vol. 44, pp. 339-342).

[3] Zotter F, Frank M. **All-round ambisonic panning and decoding**.
Journal of the audio engineering society. 2012 Nov 26;60(10):807-20.

[4] Zotter F, Pomberger H, Noisternig M. **Energy-preserving ambisonic decoding**.
Acta Acustica united with Acustica. 2012 Jan 1;98(1):37-47.

[5] McCormack L, Va"lima"ki V. **FFT-based Dynamic Range Compression**.
In Proceedings of the 14th Sound and Music Computing Conference, July 5–8, Espoo, Finland, At Espoo, Finland 2017

[6] Williams EG. **Fourier acoustics: sound radiation and nearfield acoustical holography**.
Elsevier; 1999 Jun 10.

[7] Rafaely B. **Fundamentals of spherical array processing**.
Berlin: Springer; 2015 Feb 18.

[8] McCormack L, Delikaris-Manias S, Farina A, Pinardi D, Pulkki V. **Real-Time Conversion of Sensor Array Signals into Spherical Harmonic Signals with Applications to Spatially Localized Sub-Band Sound-Field Analysis**.
In Audio Engineering Society Convention 144 2018 May 14. Audio Engineering Society.

[9] McCormack L, Politis A, Pulkki V. **Sharpening of Angular Spectra Based on a Directional Re-assignment Approach for Ambisonic Sound-field Visualisation**.
In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2019 Apr 17 (pp. 576-580). IEEE.

[10] Pulkki V. **Virtual sound source positioning using vector base amplitude panning**.
Journal of the audio engineering society. 1997 Jun 1;45(6):456-66.

[11] Laitinen MV, Vilkamo J, Jussila K, Politis A, Pulkki V. **Gain normalization in amplitude panning as a function of frequency and room reverberance**.
In Audio Engineering Society Conference: 55th International Conference: Spatial Audio 2014 Aug 26. Audio Engineering Society.

[12] Ivanic J, Ruedenberg K. **Rotation Matrices for Real Spherical Harmonics. Direct Determination by Recursion**.
The Journal of Physical Chemistry A. 1998 Nov 5;102(45):9099-100.

2 Acquiring and linking a custom Intel MKL library tailored for SAF

The Spatial_Audio_Framework (SAF) requires any library (or combination of libraries) which supports the CBLAS and LAPACK standards. Personally, we like Intel MKL for this task, as it works well and also has an optimised FFT implementation which SAF can use too.

Note that you may link directly to the Intel MKL static or dynamic libraries, but these are stupidly huge. This is why we like to use the MKL builder to build a custom MKL library, and below are instructions for Windows/MacOSX/Linux users rocking x86_64/amd64 CPUs.

2.1 Windows (64-bit) users

Note: use the "x64 Developer Command Prompt for VS.exe" (open as administrator) to run the following commands.

1. Install `Intel MKL`.
2. The required custom library may be obtained by first copying the included "∗∗dependencies/saf_mkl_list∗∗" file into the MKL "builder" folder:

```
xcopy saf_mkl_list C:\Program Files (x86)\IntelSWTools\compilers_and_libraries\windows\mkl\tools\builder /R
```

1. EITHER (The blue pill): to generate and copy the "∗∗saf_mkl_custom.dll∗∗" library to a system path folder, and the "∗∗saf_mkl_custom.lib∗∗" for you to use locally, enter the following commands:

```
cd /Program Files (x86)/IntelSWTools/compilers_and_libraries/windows/mkl/tools/builder
nmake intel64 interface=lp64 threading=sequential name=saf_mkl_custom export=saf_mkl_list
xcopy saf_mkl_custom.dll C:\Windows\System32 /R
xcopy saf_mkl_custom.lib C:\Users\[YOUR WORKING DIRECTORY]\Spatial_Audio_Framework\dependencies\Win64\lib /R
```

1. OR (The red pill): you may instead build a threaded version of the library (which involves some additional steps):

```
cd /Program Files (x86)/IntelSWTools/compilers_and_libraries/windows/mkl/tools/builder
nmake intel64 interface=lp64 threading=parallel name=saf_mkl_custom export=saf_mkl_list
xcopy saf_mkl_custom.dll C:\Windows\System32 /R
xcopy saf_mkl_custom.lib C:\Users\[YOUR WORKING DIRECTORY]\Spatial_Audio_Framework\dependencies\Win64\lib /R
cd C:/Program Files (x86)/IntelSWTools/compilers_and_libraries/windows/compiler/lib/intel64/
xcopy libiomp5md.lib C:\Users\[YOUR WORKING DIRECTORY]\Spatial_Audio_Framework\dependencies\Win64\lib /R
cd C:/Program Files (x86)/IntelSWTools/compilers_and_libraries/windows/redist/intel64/compiler
xcopy libiomp5md.dll C:\Windows\System32 /R
```

1. Add the following header search path to your project:

```
C:/Program Files (x86)/IntelSWTools/compilers_and_libraries/windows/mkl/include
```

1. Add the following library search path to your project:

```
C:/Users/[YOUR WORKING DIRECTORY]/SDKs/Spatial_Audio_Framework/dependencies/Win64/lib
```

1. link your project against the following libraries (note that the second library is only needed if you built the threaded version):
   ```
   saf_mkl_custom.lib
   libiomp5md.lib
   ```
2. Add "SAF_USE_INTEL_MKL" to your pre-processor definitions.

Note: If you built the threaded version of the library, then there are some more pre-processors defintions you can use. See below.

2.2 MacOSX users

By default, the framework will use Apple's Accelerate library for the BLAS/LAPACK routines and FFT, so you may ignore all of these steps if you wish. However, Mac users may still elect to use Intel MKL, as is it often faster than Accelerate.

1. Install Intel MKL. Optionally, you may want to add the MKL global environment variables using this command:

```
source /opt/intel/compilers_and_libraries/mac/mkl/bin/mklvars.sh intel64
```

1. The required custom library may be obtained by first copying the included "∗∗dependencies/saf_mkl_list∗∗" file into the MKL "builder" folder:

```
sudo cp saf_mkl_list /opt/intel/compilers_and_libraries/mac/mkl/tools/builder
```

1. EITHER (The blue pill): to generate and copy the "∗∗saf_mkl_custom.dylib∗∗" library to a system path folder, ready for you to use, enter the following commands:

```
cd /opt/intel/compilers_and_libraries/mac/mkl/tools/builder
sudo make intel64 interface=lp64 threading=sequential name=libsaf_mkl_custom export=saf_mkl_list
sudo cp saf_mkl_custom.dylib /usr/local/lib
```

1. OR (The red pill): you may instead build a threaded version of the library (which involves some additional steps):

```
cd /opt/intel/compilers_and_libraries/mac/mkl/tools/builder
sudo make intel64 interface=lp64 threading=parallel name=libsaf_mkl_custom export=saf_mkl_list
sudo cp saf_mkl_custom.dylib /usr/local/lib
sudo cp /opt/intel/compilers_and_libraries/mac/compiler/lib/libiomp5.dylib /usr/local/lib
```

1. Add the following header search path to your project:

```
/opt/intel/compilers_and_libraries/mac/mkl/include
```

1. Then add the following linker flags to your project (note that the second library is only needed if you built the threaded version):

```
-L/usr/local/lib -lsaf_mkl_custom
-L/usr/local/lib -liomp5
```

1. Finally, add "SAF_USE_INTEL_MKL" to your project's pre-processor definitions.

Note: If you built the threaded version of the library, then there are some more pre-processors defintions you can use. See below.

2.3 Linux (amd64) users

0. Add the following directories to the header search paths:
```
Spatial_Audio_Framework/framework/include
Spatial_Audio_Framework/dependencies/Linux/include
```

1. Add the following directory to the library search paths:

```
Spatial_Audio_Framework/dependencies/Linux/lib
```

1. Install Intel MKL.
2. The required "∗∗saf_mkl_custom.so∗∗" file may be generated using Intel's custom dll builder.
3. First copy the included "dependencies/saf_mkl_list" file to this folder:

```
/opt/intel/compilers_and_libraries/linux/mkl/tools/builder
```

1. To generate and copy this library to a system path folder, use the following commands (root permissions required):

```
cd /opt/intel/compilers_and_libraries/linux/mkl/tools/builder
sudo make intel64 interface=lp64 threading=sequential name=libsaf_mkl_custom export=saf_mkl_list
sudo cp saf_mkl_custom.so /usr/lib
```

1. Then add the following linker flag to your project:

```
-L/usr/lib -lsaf_mkl_custom
```

1. Add "SAF_USE_INTEL_MKL" to your pre-processor definitions.

2.4 Intel MKL threading options (Optional)

If you built a threaded version of the custom library, then there are some additional options you may specify. More information can be found here.

Note by default: MKL_NUM_THREADS = [number of CPU cores], MKL_DYNAMIC = 1.

You may also change these at run time using the following functions:
```
/* for example: */
MKL_Set_Num_Threads(2);
MKL_Set_Dynamic(1);
```

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Data Structure Documentation

5.1 _cdf4sap_cmplx_data Struct Reference

Main data structure for the Covariance Domain Framework for Spatial Audio Processing (CDF4SAP), for complex-valued matrices.

Data Fields

- int **nXcols**
- int **nYcols**

- float_complex ∗ **Cr_cmplx**
- float_complex ∗ **lambda**
- float_complex ∗ **U_Cy**
- float_complex ∗ **S_Cy**
- float_complex ∗ **S_Cx**
- float_complex ∗ **Ky**
- float_complex ∗ **U_Cx**
- float_complex ∗ **Kx**
- float_complex ∗ **Kx_reg_inverse**
- float_complex ∗ **U**
- float_complex ∗ **V**
- float_complex ∗ **P**
- float ∗ **s_Cx**
- float_complex ∗ **G_hat**
- float_complex ∗ **Cx_QH**
- float_complex ∗ **GhatH_Ky**
- float_complex ∗ **QH_GhatH_Ky**
- float_complex ∗ **KxH_QH_GhatH_Ky**
- float_complex ∗ **lambda_UH**
- float_complex ∗ **P_Kxreginverse**
- float_complex ∗ **Cx_MH**
- float_complex ∗ **Cy_tilde**
- float_complex ∗ **G_M**

5.1.1 Detailed Description

Main data structure for the Covariance Domain Framework for Spatial Audio Processing (CDF4SAP), for complex-valued matrices.

Definition at line 62 of file saf_cdf4sap.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_cdf4sap/saf_cdf4sap.c

5.2 _cdf4sap_data Struct Reference

Main data structure for the Covariance Domain Framework for Spatial Audio Processing (CDF4SAP), for real-valued matrices.

Data Fields

- int **nXcols**
- int **nYcols**
- float ∗ **lambda**
- float ∗ **U_Cy**
- float ∗ **S_Cy**
- float ∗ **Ky**
- float ∗ **U_Cx**
- float ∗ **S_Cx**
- float ∗ **s_Cx**
- float ∗ **Kx**
- float ∗ **Kx_reg_inverse**
- float ∗ **U**
- float ∗ **V**
- float ∗ **P**
- float ∗ **G_hat**
- float ∗ **Cx_QH**
- float ∗ **GhatH_Ky**
- float ∗ **QH_GhatH_Ky**
- float ∗ **KxH_QH_GhatH_Ky**
- float ∗ **lambda_UH**
- float ∗ **P_Kxreginverse**
- float ∗ **Cx_MH**
- float ∗ **Cy_tilde**
- float ∗ **G_M**

5.2.1 Detailed Description

Main data structure for the Covariance Domain Framework for Spatial Audio Processing (CDF4SAP), for real-valued matrices.

Definition at line 44 of file saf_cdf4sap.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_cdf4sap/saf_cdf4sap.c

5.3 _ch_vertex Struct Reference

vertex structure, used by convhull_3d

```
#include <convhull_3d.h>
```

Data Fields

-
```
union {
   struct {
      CH_FLOAT x
      CH_FLOAT y
      CH_FLOAT z
   }
   CH_FLOAT v [3]
};
```

5.3.1 Detailed Description

vertex structure, used by convhull_3d

Definition at line 61 of file convhull_3d.h.

The documentation for this struct was generated from the following file:

- framework/resources/convhull_3d/convhull_3d.h

5.4 _saf_fft_data Struct Reference

Data structure for complex-complex FFT transforms.

Data Fields

- int **N**
- float **Scale**
- int **useKissFFT_flag**
- kiss_fft_cfg **kissFFThandle_fwd**
- kiss_fft_cfg **kissFFThandle_bkw**

5.4.1 Detailed Description

Data structure for complex-complex FFT transforms.

Definition at line 77 of file saf_fft.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_fft.c

5.5 _saf_rfft_data Struct Reference

Data structure for real-(half)complex FFT transforms.

Data Fields

- int **N**
- float **Scale**
- int **useKissFFT_flag**
- kiss_fftr_cfg **kissFFThandle_fwd**
- kiss_fftr_cfg **kissFFThandle_bkw**

5.5.1 Detailed Description

Data structure for real-(half)complex FFT transforms.

Definition at line 57 of file saf_fft.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_fft.c

5.6 _safMatConv_data Struct Reference

Data structure for the matrix convolver.

Data Fields

- int **hopSize**
- int **fftSize**
- int **nBins**
- int **length_h**
- int **nCHin**
- int **nCHout**
- int **numFilterBlocks**
- int **numOvrlpAddBlocks**
- int **usePartFLAG**
- void ∗ **hFFT**
- float ∗ **x_pad**
- float ∗ **y_pad**
- float ∗ **hx_n**
- float ∗ **z_n**
- float ∗ **ovrlpAddBuffer**
- float ∗ **y_n_overlap**
- float_complex ∗ **H_f**
- float_complex ∗ **X_n**
- float_complex ∗ **HX_n**
- float_complex ∗∗ **Hpart_f**

5.6.1 Detailed Description

Data structure for the matrix convolver.

Definition at line 37 of file saf_matrixConv.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_matrixConv.c

5.7 _safMulConv_data Struct Reference

Data structure for the multi-channel convolver.

Data Fields

- int **hopSize**
- int **fftSize**
- int **nBins**
- int **length_h**
- int **nCH**
- int **numOvrlpAddBlocks**
- int **numFilterBlocks**
- int **usePartFLAG**
- void ∗ **hFFT**
- float ∗ **x_pad**
- float ∗ **z_n**
- float ∗ **ovrlpAddBuffer**
- float ∗ **hx_n**
- float ∗ **y_n_overlap**
- float_complex ∗ **X_n**
- float_complex ∗ **HX_n**
- float_complex ∗ **Z_n**
- float_complex ∗ **H_f**
- float_complex ∗ **Hpart_f**

5.7.1 Detailed Description

Data structure for the multi-channel convolver.

Definition at line 239 of file saf_matrixConv.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_matrixConv.c

5.8 afHybrid Struct Reference

Data structure for the hybrid filtering employed by afSTFTlib.

Data Fields

- int **inChannels**
- int **outChannels**
- int **hopSize**
- float **hybridCoeffs** [3]
- complexVector ∗∗ **analysisBuffer**
- int **loopPointer**

5.8.1 Detailed Description

Data structure for the hybrid filtering employed by afSTFTlib.

The purpose of this filtering is to further divide the 4 lowest FFT-bins, to improve the frequency resolution at low-frequencies. For example, 129 bins would become 133 hybrid-bins.

Definition at line 105 of file afSTFTlib.c.

The documentation for this struct was generated from the following file:

- framework/resources/afSTFT/afSTFTlib.c

5.9 afSTFT Struct Reference

Main data structure for afSTFTlib.

Data Fields

- int **inChannels**
- int **outChannels**
- int **hopSize**
- int **hLen**
- int **pr**
- int **LDmode**
- int **hopIndexIn**
- int **hopIndexOut**
- int **totalHops**
- float * **protoFilter**
- float * **protoFilterI**
- float ** **inBuffer**
- float * **fftProcessFrameTD**
- float ** **outBuffer**
- int **log2n**
- void * **hSafFFT**
- float_complex * **fftProcessFrameFD**
- float * **tempHopBuffer**
- void * **h_afHybrid**
- int **hybridMode**

5.9.1 Detailed Description

Main data structure for afSTFTlib.

Definition at line 70 of file afSTFTlib.c.

The documentation for this struct was generated from the following file:

- framework/resources/afSTFT/afSTFTlib.c

5.10 complexVector Struct Reference

Complex data type used by afSTFTlib.

```
#include <afSTFTlib.h>
```

Data Fields

- float ∗ **re**
- float ∗ **im**

5.10.1 Detailed Description

Complex data type used by afSTFTlib.

Definition at line 64 of file afSTFTlib.h.

The documentation for this struct was generated from the following file:

- framework/resources/afSTFT/afSTFTlib.h

5.11 float_w_idx Struct Reference

Helper struct for qsort in convhull_3d_build()

Data Fields

- CH_FLOAT **val**
- int **idx**

5.11.1 Detailed Description

Helper struct for qsort in convhull_3d_build()

Definition at line 80 of file convhull_3d.c.

The documentation for this struct was generated from the following file:

- framework/resources/convhull_3d/convhull_3d.c

5.12 int_w_idx Struct Reference

Helper struct for qsort in convhull_3d_build()

Data Fields

- int **val**
- int **idx**

5.12.1 Detailed Description

Helper struct for qsort in convhull_3d_build()

Definition at line 87 of file convhull_3d.c.

The documentation for this struct was generated from the following file:

- framework/resources/convhull_3d/convhull_3d.c

5.13 saf_sort_double Struct Reference

Helper struct for sorting a vector of doubles using 'qsort'.

Data Fields

- double **val**
- int **idx**

5.13.1 Detailed Description

Helper struct for sorting a vector of doubles using 'qsort'.

Definition at line 46 of file saf_sort.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_sort.c

5.14 saf_sort_float Struct Reference

Helper struct for sorting a vector of floats using 'qsort'.

Data Fields

- float **val**
- int **idx**

5.14.1 Detailed Description

Helper struct for sorting a vector of floats using 'qsort'.

Definition at line 38 of file saf_sort.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_sort.c

5.15 saf_sort_int Struct Reference

Helper struct for sorting a vector of integers using 'qsort'.

Data Fields

- int **val**
- int **idx**

5.15.1 Detailed Description

Helper struct for sorting a vector of integers using 'qsort'.

Definition at line 30 of file saf_sort.c.

The documentation for this struct was generated from the following file:

- framework/modules/saf_utilities/saf_sort.c

6 File Documentation

6.1 framework/include/saf.h File Reference

Main include header for the Spatial_Audio_Framework.

```
#include "../modules/saf_utilities/saf_utilities.h"
#include "../resources/afSTFT/afSTFTlib.h"
#include "../modules/saf_cdf4sap/saf_cdf4sap.h"
#include "../modules/saf_hoa/saf_hoa.h"
#include "../modules/saf_hrir/saf_hrir.h"
#include "../modules/saf_sh/saf_sh.h"
#include "../modules/saf_vbap/saf_vbap.h"
```

Macros

- #define SAF_MODULE_UTILITIES
  SAF Module: Utilities.
- #define SAF_MODULE_AFSTFT
  SAF Module: afSTFTlib.
- #define SAF_MODULE_CDF4SAP
  SAF Module: CDF4SAP.
- #define SAF_MODULE_HOA
  SAF Module: HOA.
- #define SAF_MODULE_HRIR
  SAF Module: HRIR.
- #define SAF_MODULE_SH
  SAF Module: SH.
- #define SAF_MODULE_VBAP
  SAF Module: VBAP.
- #define SAF_MODULE_SOFA_READER
  SAF Module: SOFA Reader.

6.1.1 Detailed Description

Main include header for the Spatial_Audio_Framework.

6.1.1.1 Core modules

afSTFTlib, saf_cdf4sap, saf_hoa, saf_hrir, saf_sh, saf_utilities, saf_vbap

6.1.1.2 Optional modules

saf_sofa_reader

Author

Leo McCormack

Date

06.04.2018

6.1.2 Macro Definition Documentation

6.1.2.1 SAF_MODULE_AFSTFT

`#define SAF_MODULE_AFSTFT`

SAF Module: afSTFTlib.

The Alias-free STFT implementation by Juha Vilkamo, with some minor changes. The Original source code can be found here:

- `https://github.com/jvilkamo/afSTFT`

6.1.2.2 Dependencies saf_utilities

Definition at line 76 of file saf.h.

6.1.2.3 SAF_MODULE_CDF4SAP `#define SAF_MODULE_CDF4SAP`

SAF Module: CDF4SAP.

Covarience Domain Framework for Spatial Audio Processing (CDF4SAP). A C implementation of the framework described in [1].

6.1.2.4 Dependencies saf_utilities

See also

[1] Vilkamo, J., Bäckström, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

Definition at line 92 of file saf.h.

6.1.2.5 SAF_MODULE_HOA `#define SAF_MODULE_HOA`

SAF Module: HOA.

A collection of higher-order Ambisonics related functions. Some of which are derived from the Matlab library by Archontis Politis, found here:

- `https://github.com/polarch/Higher-Order-Ambisonics`

6.1.2.6 Dependencies saf_utilities, saf_vbap, saf_sh

Definition at line 105 of file saf.h.

6.1.2.7 SAF_MODULE_HRIR `#define SAF_MODULE_HRIR`

SAF Module: HRIR.

A collection of head-related impulse-response (HRIR) functions. Including estimation of the interaural time differences (ITDs), conversion of HRIRs to HRTF filterbank coefficients, and HRTF interpolation utilising amplitude-normalised VBAP gains.

6.1.2.8 Dependencies saf_utilities, afSTFTlib

Definition at line 119 of file saf.h.

6.1.2.9 SAF_MODULE_SH `#define SAF_MODULE_SH`

SAF Module: SH.

A collection of spherical harmonic related functions. Many of which have been derived from Matlab libraries by Archontis Politis; found here:

- https://github.com/polarch/Spherical-Harmonic-Transform
- https://github.com/polarch/Array-Response-Simulator
- https://github.com/polarch/Spherical-Array-Processing

6.1.2.10 Dependencies saf_utilities

Definition at line 134 of file saf.h.

6.1.2.11 SAF_MODULE_SOFA_READER `#define SAF_MODULE_SOFA_READER`

SAF Module: SOFA Reader.

A simple SOFA file reader that returns only the bare minimum needed to load HRIR data.

6.1.2.12 Enable instructions Add this pre-processor definition to your project: SAF_ENABLE_SOFA_READER

6.1.2.13 Dependencies netcdf library

Definition at line 166 of file saf.h.

6.1.2.14 SAF_MODULE_UTILITIES `#define SAF_MODULE_UTILITIES`

SAF Module: Utilities.

Contains a collection of useful memory allocation functions and cross- platform complex number wrappers. Optimised linear algebra routines utilising BLAS and LAPACK are also included.

6.1.2.15 Dependencies

A performance library comprising CBLAS and LAPACK routines is required by the module and, thus, also by the SAF framework as a whole. Add one of the following FLAGS to your project's preprocessor definitions list, in order to enable one of these suitable performance libraries, which must also be linked correctly to your project.

- SAF_USE_INTEL_MKL: to enable Intel's Math Kernal Library with Fortran LAPACK interface
- SAF_USE_ATLAS: to enable ATLAS BLAS routines and ATLAS's CLAPACK interface
- SAF_USE_OPENBLAS_WITH_LAPACKE: to enable OpenBLAS with LAPACKE interface

See also

More information can be found here: `https://github.com/leomccormack/Spatial_Audio_Framework`

Note

MacOSX users only: saf_utilities will employ Apple's Accelerate library by default, if none of the above FLAGS are defined.

Definition at line 63 of file saf.h.

6.1.2.16 SAF_MODULE_VBAP `#define SAF_MODULE_VBAP`

SAF Module: VBAP.

VBAP functions largely derived from the MATLAB library by Archontis Politis, found here:

- `https://github.com/polarch/Vector-Base-Amplitude-Panning`

6.1.2.17 Dependencies

saf_utilities

Definition at line 147 of file saf.h.

6.2 framework/modules/saf_cdf4sap/saf_cdf4sap.c File Reference

Public part of the Covariance Domain Framework module (saf_cdf4sap)

```
#include "saf_cdf4sap.h"
#include "../saf_utilities/saf_utilities.h"
```

Data Structures

- struct _cdf4sap_data

  Main data structure for the Covariance Domain Framework for Spatial Audio Processing (CDF4SAP), for real-valued matrices.
- struct _cdf4sap_cmplx_data

  Main data structure for the Covariance Domain Framework for Spatial Audio Processing (CDF4SAP), for complex-valued matrices.

Functions

- void cdf4sap_create (void **const phCdf, int nXcols, int nYcols)

    Creates an instance of the Covariance Domain Framework (REAL)
- void cdf4sap_cmplx_create (void **const phCdf, int nXcols, int nYcols)

    Creates an instance of the Covariance Domain Framework (COMPLEX)
- void cdf4sap_destroy (void **const phCdf)

    Destroys an instance of the Covariance Domain Framework (REAL)
- void cdf4sap_cmplx_destroy (void **const phCdf)

    Destroys an instance of the Covariance Domain Framework (COMPLEX)
- void formulate_M_and_Cr (void *const hCdf, float *Cx, float *Cy, float *Q, int useEnergyFLAG, float reg, float *M, float *Cr)

    Computes the optimal mixing matrices (REAL)
- void formulate_M_and_Cr_cmplx (void *const hCdf, float_complex *Cx, float_complex *Cy, float_complex *Q, int useEnergyFLAG, float reg, float_complex *M, float *Cr)

    Computes the optimal mixing matrices (COMPLEX)

6.2.1 Detailed Description

Public part of the Covariance Domain Framework module (saf_cdf4sap)

Covariance Domain Framework for Spatial Audio Processing (CDF4SAP). This is a direct C port of the Matlab function given in [1], which was originally written by Juha Vilkamo. The algorithm is explained in further detail in [2].

See also

[1] Vilkamo, J., Ba"ckstro"m, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

[2] Vilkamo, J., & Ba"ckstro"m, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time–Frequency Domain Spatial Audio. John Wiley & Sons.

Author

Leo McCormack

Date

25.11.2016

6.2.2 Function Documentation

6.2.2.1 cdf4sap_cmplx_create()

```
void cdf4sap_cmplx_create (
        void **const phCdf,
        int nXcols,
        int nYcols )
```

Creates an instance of the Covariance Domain Framework (COMPLEX)

Allocates memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for COMPLEX-VALUED input/output matrices. For REAL-VALUED input/output matrices use cdf4sap_create().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle
in	*nXcols*	Number of columns/rows in square input matrix 'Cx'
in	*nYcols*	Number of columns/rows in square input matrix 'Cy'

Definition at line 131 of file saf_cdf4sap.c.

6.2.2.2 cdf4sap_cmplx_destroy()

```
void cdf4sap_cmplx_destroy (
        void **const phCdf )
```

Destroys an instance of the Covariance Domain Framework (COMPLEX)

Frees memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for COMPLEX-VALUED input/output matrices. For REAL-VALUED input/output matrices use cdf4sap_destroy().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle

Definition at line 219 of file saf_cdf4sap.c.

6.2.2.3 cdf4sap_create()

```
void cdf4sap_create (
        void **const phCdf,
        int nXcols,
        int nYcols )
```

Creates an instance of the Covariance Domain Framework (REAL)

Allocates memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for REAL-VALUED input/output matrices. For COMPLEX-VALUED input/output matrices use cdf4sap_cmplx_create().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle
in	*nXcols*	Number of columns/rows in square input matrix 'Cx'
in	*nYcols*	Number of columns/rows in square input matrix 'Cy'

Definition at line 79 of file saf_cdf4sap.c.

6.2.2.4 cdf4sap_destroy()

```
void cdf4sap_destroy (
    void **const phCdf )
```

Destroys an instance of the Covariance Domain Framework (REAL)

Frees memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for REAL-VALUED input/output matrices. For COMPLEX-VALUED input/output matrices use cdf4sap_cmplx_destroy().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle

Definition at line 184 of file saf_cdf4sap.c.

6.2.2.5 formulate_M_and_Cr()

```
void formulate_M_and_Cr (
    void *const hCdf,
    float * Cx,
    float * Cy,
    float * Q,
    int useEnergyFLAG,
    float reg,
    float * M,
    float * Cr )
```

Computes the optimal mixing matrices (REAL)

This function solves the problem of determining the optimal mixing matrices 'M' and 'Cr', such that the covariance matrix of the output: y_out = M*x + Mr*decorrelated(x), is aligned with the target matrix 'Cy', given the covariance matrix of input x, Cx=x*x^H, and a prototype decoding matrix Q. For the derivation and a more detailed description, the reader is referred to [1,2].

Note

Use this function for REAL-VALUED input/output matrices. For COMPLEX-VALUED input/output use formulate_M_and_Cr_cmplx().

For an example of how to use this function, one may refer to the implementation of the parametric binaural Ambisonic decoder (described in [3]) found here: https://github.com/leomccormack/CroPaC-Binaural

Parameters

in	*hCdf*	Covariance Domain Framework handle

Parameters

in	*Cx*	Covariance matrix of input 'x'; FLAT: nXcols x nXcols
in	*Cy*	Target covariance matrix; FLAT: nXcols x nXcols
in	*Q*	Prototype matrix; FLAT: nYcols x nXcols
in	*useEnergyFLAG*	Set to '0' to apply energy compensation to 'M' instead of outputing 'Cr'. Set to '1' to output 'Cr'
in	*reg*	Regularisation term (suggested: 0.2f)
out	*M*	Mixing matrix; FLAT: nYcols x nXcols
out	*Cr*	Mixing matrix residual; FLAT: nYcols x nYcols

See also

[1] Vilkamo, J., Ba"ckstro"m, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

[2] Vilkamo, J., & Ba"ckstro"m, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time–Frequency Domain Spatial Audio. John Wiley & Sons.

[3] McCormack, L., Delikaris-Manias, S. (2019). "Parametric first-order ambisonic decoding for headphones utilising the Cross-Pattern Coherence algorithm". inProc 1st EAA Spatial Audio Signal Processing Symposium, Paris, France.

Definition at line 255 of file saf_cdf4sap.c.

6.2.2.6 formulate_M_and_Cr_cmplx()

```
void formulate_M_and_Cr_cmplx (
    void *const hCdf,
    float_complex * Cx,
    float_complex * Cy,
    float_complex * Q,
    int useEnergyFLAG,
    float reg,
    float_complex * M,
    float * Cr )
```

Computes the optimal mixing matrices (COMPLEX)

This function solves the problem of determining the optimal mixing matrices 'M' and 'Cr', such that the covariance matrix of the output: y_out = M∗x + Mr∗decorrelated(x), is aligned with the target matrix 'Cy', given the covariance matrix of input x, Cx=x∗x^H, and a prototype decoding matrix Q. For the derivation and a more detailed description, the reader is referred to [1,2].

Note

Use this function for COMPLEX-VALUED input/output matrices. For REAL-VALUED input/output use formulate_M_and_Cr().

For an example of how to use this function, one may refer to the implementation of the parametric binaural Ambisonic decoder (described in [3]) found here: https://github.com/leomccormack/CroPaC-Binaural

Parameters

in	*hCdf*	Covariance Domain Framework handle
in	*Cx*	Covariance matrix of input 'x'; FLAT: nXcols x nXcols
in	*Cy*	Target covariance matrix; FLAT: nXcols x nXcols
in	*Q*	Prototype matrix; FLAT: nYcols x nXcols
in	*useEnergyFLAG*	Set to '0' to apply energy compensation to 'M' instead of outputing 'Cr'. Set to '1' to output 'Cr'
in	*reg*	Regularisation term (suggested: 0.2f)
out	*M*	Mixing matrix; FLAT: nYcols x nXcols
out	*Cr*	Mixing matrix residual; FLAT: nYcols x nYcols

See also

[1] Vilkamo, J., Ba"ckstro"m, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

[2] Vilkamo, J., & Ba"ckstro"m, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time–Frequency Domain Spatial Audio. John Wiley & Sons.

[3] McCormack, L., Delikaris-Manias, S. (2019). "Parametric first-order ambisonic decoding for headphones utilising the Cross-Pattern Coherence algorithm". inProc 1st EAA Spatial Audio Signal Processing Symposium, Paris, France.

Definition at line 387 of file saf_cdf4sap.c.

6.3 framework/modules/saf_cdf4sap/saf_cdf4sap.h File Reference

Public part of the Covariance Domain Framework module (saf_cdf4sap)

```
#include "../saf_utilities/saf_complex.h"
```

Functions

- void cdf4sap_create (void ∗∗const phCdf, int nXcols, int nYcols)

  Creates an instance of the Covariance Domain Framework (REAL)
- void cdf4sap_cmplx_create (void ∗∗const phCdf, int nXcols, int nYcols)

  Creates an instance of the Covariance Domain Framework (COMPLEX)
- void cdf4sap_destroy (void ∗∗const phCdf)

  Destroys an instance of the Covariance Domain Framework (REAL)
- void cdf4sap_cmplx_destroy (void ∗∗const phCdf)

  Destroys an instance of the Covariance Domain Framework (COMPLEX)
- void formulate_M_and_Cr (void ∗const hCdf, float ∗Cx, float ∗Cy, float ∗Q, int useEnergyFLAG, float reg, float ∗M, float ∗Cr)

  Computes the optimal mixing matrices (REAL)
- void formulate_M_and_Cr_cmplx (void ∗const hCdf, float_complex ∗Cx, float_complex ∗Cy, float_complex ∗Q, int useEnergyFLAG, float reg, float_complex ∗M, float ∗Cr)

  Computes the optimal mixing matrices (COMPLEX)

6.3.1 Detailed Description

Public part of the Covariance Domain Framework module (saf_cdf4sap)

Covariance Domain Framework for Spatial Audio Processing (CDF4SAP). This is a direct C port of the Matlab function given in [1], which was originally written by Juha Vilkamo. The algorithm is explained in further detail in [2].

See also

[1] Vilkamo, J., Ba"ckstro"m, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

[2] Vilkamo, J., & Ba"ckstro"m, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time–Frequency Domain Spatial Audio. John Wiley & Sons.

Author

Leo McCormack

Date

25.11.2016

6.3.2 Function Documentation

6.3.2.1 cdf4sap_cmplx_create()

```
void cdf4sap_cmplx_create (
        void **const phCdf,
        int nXcols,
        int nYcols )
```

Creates an instance of the Covariance Domain Framework (COMPLEX)

Allocates memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for COMPLEX-VALUED input/output matrices. For REAL-VALUED input/output matrices use cdf4sap_create().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle
in	*nXcols*	Number of columns/rows in square input matrix 'Cx'
in	*nYcols*	Number of columns/rows in square input matrix 'Cy'

Definition at line 131 of file saf_cdf4sap.c.

6.3.2.2 cdf4sap_cmplx_destroy()

```
void cdf4sap_cmplx_destroy (
        void **const phCdf )
```

Destroys an instance of the Covariance Domain Framework (COMPLEX)

Frees memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for COMPLEX-VALUED input/output matrices. For REAL-VALUED input/output matrices use cdf4sap_destroy().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle

Definition at line 219 of file saf_cdf4sap.c.

6.3.2.3 cdf4sap_create()

```
void cdf4sap_create (
        void **const phCdf,
        int nXcols,
        int nYcols )
```

Creates an instance of the Covariance Domain Framework (REAL)

Allocates memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for REAL-VALUED input/output matrices. For COMPLEX-VALUED input/output matrices use cdf4sap_cmplx_create().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle
in	*nXcols*	Number of columns/rows in square input matrix 'Cx'
in	*nYcols*	Number of columns/rows in square input matrix 'Cy'

Definition at line 79 of file saf_cdf4sap.c.

6.3.2.4 cdf4sap_destroy()

```
void cdf4sap_destroy (
        void **const phCdf )
```

Destroys an instance of the Covariance Domain Framework (REAL)

Frees memory for a Covariance Domain Framework for Spatial Audio Processing (CDF4SAP) handle.

Note

Use this function for REAL-VALUED input/output matrices. For COMPLEX-VALUED input/output matrices use cdf4sap_cmplx_destroy().

Parameters

in	*phCdf*	The address (&) of the CDF4SAP handle

Definition at line 184 of file saf_cdf4sap.c.

6.3.2.5 formulate_M_and_Cr()

```
void formulate_M_and_Cr (
        void *const hCdf,
        float * Cx,
        float * Cy,
        float * Q,
        int useEnergyFLAG,
        float reg,
        float * M,
        float * Cr )
```

Computes the optimal mixing matrices (REAL)

This function solves the problem of determining the optimal mixing matrices 'M' and 'Cr', such that the covariance matrix of the output: y_out = M∗x + Mr∗decorrelated(x), is aligned with the target matrix 'Cy', given the covariance matrix of input x, Cx=x∗x^H, and a prototype decoding matrix Q. For the derivation and a more detailed description, the reader is referred to [1,2].

Note

Use this function for REAL-VALUED input/output matrices. For COMPLEX-VALUED input/output use formulate_M_and_Cr_cmplx().

For an example of how to use this function, one may refer to the implementation of the parametric binaural Ambisonic decoder (described in [3]) found here: `https://github.com/leomccormack/CroPaC-Binaural`

Parameters

in	*hCdf*	Covariance Domain Framework handle
in	*Cx*	Covariance matrix of input 'x'; FLAT: nXcols x nXcols
in	*Cy*	Target covariance matrix; FLAT: nXcols x nXcols
in	*Q*	Prototype matrix; FLAT: nYcols x nXcols
in	*useEnergyFLAG*	Set to '0' to apply energy compensation to 'M' instead of outputing 'Cr'. Set to '1' to output 'Cr'
in	*reg*	Regularisation term (suggested: 0.2f)
out	*M*	Mixing matrix; FLAT: nYcols x nXcols
out	*Cr*	Mixing matrix residual; FLAT: nYcols x nYcols

See also

[1] Vilkamo, J., Ba"ckstro"m, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

[2] Vilkamo, J., & Ba"ckstro"m, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time–Frequency Domain Spatial Audio. John Wiley & Sons.

[3] McCormack, L., Delikaris-Manias, S. (2019). "Parametric first-order ambisonic decoding for headphones utilising the Cross-Pattern Coherence algorithm". inProc 1st EAA Spatial Audio Signal Processing Symposium, Paris, France.

Definition at line 255 of file saf_cdf4sap.c.

6.3.2.6 formulate_M_and_Cr_cmplx()

```
void formulate_M_and_Cr_cmplx (
        void *const hCdf,
        float_complex * Cx,
        float_complex * Cy,
        float_complex * Q,
        int useEnergyFLAG,
        float reg,
        float_complex * M,
        float * Cr )
```

Computes the optimal mixing matrices (COMPLEX)

This function solves the problem of determining the optimal mixing matrices 'M' and 'Cr', such that the covariance matrix of the output: y_out = M∗x + Mr∗decorrelated(x), is aligned with the target matrix 'Cy', given the covariance matrix of input x, Cx=x∗x^H, and a prototype decoding matrix Q. For the derivation and a more detailed description, the reader is referred to [1,2].

Note

Use this function for COMPLEX-VALUED input/output matrices. For REAL-VALUED input/output use formulate_M_and_Cr().

For an example of how to use this function, one may refer to the implementation of the parametric binaural Ambisonic decoder (described in [3]) found here: https://github.com/leomccormack/CroPaC-Binaural

Parameters

in	*hCdf*	Covariance Domain Framework handle
in	*Cx*	Covariance matrix of input 'x'; FLAT: nXcols x nXcols
in	*Cy*	Target covariance matrix; FLAT: nXcols x nXcols
in	*Q*	Prototype matrix; FLAT: nYcols x nXcols
in	*useEnergyFLAG*	Set to '0' to apply energy compensation to 'M' instead of outputing 'Cr'. Set to '1' to output 'Cr'
in	*reg*	Regularisation term (suggested: 0.2f)
out	*M*	Mixing matrix; FLAT: nYcols x nXcols
out	*Cr*	Mixing matrix residual; FLAT: nYcols x nYcols

See also

[1] Vilkamo, J., Ba"ckstro"m, T., & Kuntz, A. (2013). Optimized covariance domain framework for time–frequency processing of spatial audio. Journal of the Audio Engineering Society, 61(6), 403-411.

[2] Vilkamo, J., & Ba"ckstro"m, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time–Frequency Domain Spatial Audio. John Wiley & Sons.

[3] McCormack, L., Delikaris-Manias, S. (2019). "Parametric first-order ambisonic decoding for headphones utilising the Cross-Pattern Coherence algorithm". inProc 1st EAA Spatial Audio Signal Processing Symposium, Paris, France.

Definition at line 387 of file saf_cdf4sap.c.

6.4 framework/modules/saf_hoa/saf_hoa.c File Reference

Public part of the higher-order Ambisonics module (saf_hoa)

```
#include "saf_hoa.h"
#include "saf_hoa_internal.h"
```

Functions

- void getRSH (int N, float ∗dirs_deg, int nDirs, float ∗Y)

  Computes REAL spherical harmonics [1] for multiple directions on the sphere.

- void getRSH_recur (int N, float ∗dirs_deg, int nDirs, float ∗Y)

  Computes REAL spherical harmonics [1] for multiple directions on the sphere.

- void getMaxREweights (int order, int diagMtxFlag, float ∗a_n)

  Computes the weights required to manipulate a hyper-cardioid beam-pattern, such that it has maximum energy in the given look-direction.

- void getLoudspeakerAmbiDecoderMtx (float ∗ls_dirs_deg, int nLS, LOUDSPEAKER_AMBI_DECODER_M←ETHODS method, int order, int enableMaxReWeighting, float ∗decMtx)

  Computes an ambisonic decoding matrix of a specific order, for a specific loudspeaker layout.

- void getBinauralAmbiDecoderMtx (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, BI←NAURAL_AMBI_DECODER_METHODS method, int order, float ∗freqVector, float ∗itd_s, float ∗weights, int enableDiffCovMatching, int enableMaxReWeighting, float_complex ∗decMtx)

  Computes binaural ambisonic decoding matrices (one per frequency) at a specific order, for a given HRTF set.

- void getBinauralAmbiDecoderFilters (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int fftSize, float fs, BINAURAL_AMBI_DECODER_METHODS method, int order, float ∗itd_s, float ∗weights, int enableDiffCov←Matching, int enableMaxReWeighting, float ∗decFilters)

  Computes ambisonic decoding filters for a given HRTF set.

- void applyDiffCovMatching (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Imposes a diffuse-field covariance constraint on a given binaural decoding matrix [1].

6.4.1 Detailed Description

Public part of the higher-order Ambisonics module (saf_hoa)

A collection of Ambisonics related functions. Many of which are derived from the Matlab library by Archontis Politis [1].

See also

[1] https://github.com/polarch/Higher-Order-Ambisonics

Author

Leo McCormack

Date

19.03.2018

6.4.2 Function Documentation

6.4.2.1 applyDiffCovMatching()

```
void applyDiffCovMatching (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        int order,
        float * weights,
        float_complex * decMtx )
```

Imposes a diffuse-field covariance constraint on a given binaural decoding matrix [1].

Note

decMtx is altered in-place.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
in,out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Zaunschirm M, Scho"rkhuber C, Ho"ldrich R. Binaural rendering of Ambisonic signals by head-related impulse response time alignment and a diffuseness constraint. The Journal of the Acoustical Society of America. 2018 Jun 19;143(6):3616-27

Definition at line 347 of file saf_hoa.c.

6.4.2.2 getBinauralAmbiDecoderFilters()

```
void getBinauralAmbiDecoderFilters (
    float_complex * hrtfs,
    float * hrtf_dirs_deg,
    int N_dirs,
    int fftSize,
    float fs,
    BINAURAL_AMBI_DECODER_METHODS method,
    int order,
    float * itd_s,
    float * weights,
    int enableDiffCM,
    int enableMaxrE,
    float * decFilters )
```

Computes ambisonic decoding filters for a given HRTF set.

Parameters

in	*hrtfs*	The HRTFs; FLAT: (fftSize/2+1) x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions
in	*fftSize*	FFT size
in	*fs*	Sampling rate
in	*method*	Decoding method (see BINAURAL_AMBI_DECODER_METHODS enum)
in	*order*	Decoding order
in	*itd_s*	Only needed for BINAURAL_DECODER_TA decoder (can set to NULL if using different method); N_dirs x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
in	*enableDiffCM*	Set to '0' to disable diffuse correction, '1' to enable
in	*enableMaxrE*	Set to '0' to disable maxRE weighting, '1' to enable
out	*decFilters*	Decoding filters; FLAT: NUM_EARS x (order+1)^2 x fftSize

Definition at line 297 of file saf_hoa.c.

6.4.2.3 getBinauralAmbiDecoderMtx()

```
void getBinauralAmbiDecoderMtx (
    float_complex * hrtfs,
    float * hrtf_dirs_deg,
    int N_dirs,
    int N_bands,
    BINAURAL_AMBI_DECODER_METHODS method,
```

```
int order,
float * freqVector,
float * itd_s,
float * weights,
int enableDiffCM,
int enableMaxrE,
float_complex * decMtx )
```

Computes binaural ambisonic decoding matrices (one per frequency) at a specific order, for a given HRTF set.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands/bins
in	*method*	Decoding method (see BINAURAL_AMBI_DECODER_METHODS enum)
in	*order*	Decoding order
in	*freqVector*	Only needed for BINAURAL_DECODER_TA or BINAURAL_DECODER_MAGLS decoders (set to NULL if using a different method); N_bands x 1
in	*itd_s*	Only needed for BINAURAL_DECODER_TA decoder (set to NULL if using different method); N_dirs x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
in	*enableDiffCM*	Set to '0' to disable diffuse correction, '1' to enable
in	*enableMaxrE*	Set to '0' to disable maxRE weighting, '1' to enable
out	*decMtx*	Decoding matrices (one per frequency); FLAT: N_bands x NUM_EARS x (order+1)^2

Definition at line 225 of file saf_hoa.c.

6.4.2.4 getLoudspeakerAmbiDecoderMtx()

```
void getLoudspeakerAmbiDecoderMtx (
float * ls_dirs_deg,
int nLS,
LOUDSPEAKER_AMBI_DECODER_METHODS method,
int order,
int enableMaxrE,
float * decMtx )
```

Computes an ambisonic decoding matrix of a specific order, for a specific loudspeaker layout.

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES [azi elev]; FLAT: nLS x 2
in	*nLS*	Number of loudspeakers
in	*method*	Decoding method (see "LOUDSPEAKER_AMBI_DECODER_METHODS" enum)
in	*order*	Decoding order
in	*enableMaxrE*	Set to '0' to disable, '1' to enable
out	*decMtx*	Decoding matrix; FLAT: nLS x (order+1)^2

Definition at line 161 of file saf_hoa.c.

6.4.2.5 getMaxREweights()

```
void getMaxREweights (
        int order,
        int diagMtxFlag,
        float * a_n )
```

Computes the weights required to manipulate a hyper-cardioid beam-pattern, such that it has maximum energy in the given look-direction.

Traditionally, due to the back lobes of beamformers when panning a source via Ambisonics encoding/decoding, there is unwanted energy given to loudspeakers directly opposite the true source direction. This max_rE weighting [1] essentially spatially "tapers" the spherical harmonic patterns used to generate said beams, reducing the contribution of the higher orders to the beam patterns. This results in worse spatial selectivity, as the width of the beam pattern main lobe is widened, however, the back lobes are also reduced; thus mitigating the aforementioned problem.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*diagMtxFlag*	Set to '0' if you want the weights to be returned as a vector, or to '1' as a diagonal matrix instead.
out	*a_n*	The max_rE weights, as a vector/diagonal matrix; (order+1)^2 x 1 OR FLAT: (order+1)^2 x (order+1)^2

See also

[1] Zotter F, Frank M. All-round ambisonic panning and decoding. Journal of the audio engineering society. 2012 Nov 26; 60(10):807-20.

Definition at line 127 of file saf_hoa.c.

6.4.2.6 getRSH()

```
void getRSH (
        int order,
        float * dirs_deg,
        int nDirs,
        float * Y )
```

Computes REAL spherical harmonics [1] for multiple directions on the sphere.

The real spherical harmonics are computed WITHOUT the 1/sqrt(4∗pi) term. i.e. max(omni) = 1. Also, compared to getRSH_recur(), this function uses unnorm_legendreP() and double precision, so is more suitable for determining 'Y' in an initialisation stage. This version is indeed slower, but more precise; especially for high orders.

Note

This function is mainly intended for Ambisonics, due to the omission of the 1/sqrt(4∗pi) scaling, and the directions are given in [azimuth elevation] (degrees). In Ambisonics literature, the format convention of 'Y' is referred to as ACN/N3D

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_deg*	Directions on the sphere [azi, ELEVATION] convention, in DEGREES; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITHOUT the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 34 of file saf_hoa.c.

6.4.2.7 getRSH_recur()

```
void getRSH_recur (
        int order,
        float * dirs_deg,
        int nDirs,
        float * Y )
```

Computes REAL spherical harmonics [1] for multiple directions on the sphere.

The real spherical harmonics are computed WITHOUT the 1/sqrt(4∗pi) term. i.e. max(omni) = 1. Also, Compared to getRSH(), this function uses unnorm_legendreP_recur() and single precision, so is more suitable for determining 'Y' in a real-time loop. It sacrifices some precision, as numerical error propogates through the recursion, but it is faster.

Note

This function is mainly intended for Ambisonics, due to the omission of the 1/sqrt(4∗pi) scaling, and the directions are given in [azimuth elevation] (degrees). In Ambisonics literature, the format convention of 'Y' is referred to as ACN/N3D

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_deg*	Directions on the sphere [azi, ELEVATION] convention, in DEGREES; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITHOUT the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 65 of file saf_hoa.c.

6.5 framework/modules/saf_hoa/saf_hoa.h File Reference

Public part of the higher-order Ambisonics module (saf_hoa)

```
#include "../saf_utilities/saf_complex.h"
#include "../saf_utilities/saf_error.h"
```

Enumerations

- enum _LOUDSPEAKER_AMBI_DECODER_METHODS {
LOUDSPEAKER_DECODER_DEFAULT, LOUDSPEAKER_DECODER_SAD, LOUDSPEAKER_DECODER_MMD,
LOUDSPEAKER_DECODER_EPAD,
LOUDSPEAKER_DECODER_ALLRAD }

    Ambisonic decoding options for loudspeaker playback.
- enum _BINAURAL_AMBI_DECODER_METHODS {
BINAURAL_DECODER_DEFAULT, BINAURAL_DECODER_LS, BINAURAL_DECODER_LSDIFFEQ,
BINAURAL_DECODER_SPR,
BINAURAL_DECODER_TA, BINAURAL_DECODER_MAGLS }

    Ambisonic decoding options for binaural/headphone playback.

Functions

- void getRSH (int order, float ∗dirs_deg, int nDirs, float ∗Y)

    Computes REAL spherical harmonics [1] for multiple directions on the sphere.
- void getRSH_recur (int order, float ∗dirs_deg, int nDirs, float ∗Y)

    Computes REAL spherical harmonics [1] for multiple directions on the sphere.
- void getMaxREweights (int order, int diagMtxFlag, float ∗a_n)

    Computes the weights required to manipulate a hyper-cardioid beam-pattern, such that it has maximum energy in the given look-direction.
- void getLoudspeakerAmbiDecoderMtx (float ∗ls_dirs_deg, int nLS, LOUDSPEAKER_AMBI_DECODER_M↩ETHODS method, int order, int enableMaxrE, float ∗decMtx)

    Computes an ambisonic decoding matrix of a specific order, for a specific loudspeaker layout.
- void getBinauralAmbiDecoderMtx (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, BI↩NAURAL_AMBI_DECODER_METHODS method, int order, float ∗freqVector, float ∗itd_s, float ∗weights, int enableDiffCM, int enableMaxrE, float_complex ∗decMtx)

    Computes binaural ambisonic decoding matrices (one per frequency) at a specific order, for a given HRTF set.
- void getBinauralAmbiDecoderFilters (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int fftSize, float fs, BINAURAL_AMBI_DECODER_METHODS method, int order, float ∗itd_s, float ∗weights, int enableDiffCM, int enableMaxrE, float ∗decFilters)

    Computes ambisonic decoding filters for a given HRTF set.
- void applyDiffCovMatching (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

    Imposes a diffuse-field covariance constraint on a given binaural decoding matrix [1].

6.5.1 Detailed Description

Public part of the higher-order Ambisonics module (saf_hoa)

A collection of Ambisonics related functions. Many of which are derived from the Matlab library by Archontis Politis [1].

See also

[1] https://github.com/polarch/Higher-Order-Ambisonics

Author

Leo McCormack

Date

19.03.2018

6.5.2 Enumeration Type Documentation

6.5.2.1 _BINAURAL_AMBI_DECODER_METHODS `enum _BINAURAL_AMBI_DECODER_METHODS`

Ambisonic decoding options for binaural/headphone playback.

Note

A more detailed description of each method may be found in saf_hoa_internal.h.

See also

[1] Z. Ben-Hur, F. Brinkmann, J. Sheaffer, S. Weinzierl, and B. Rafaely, "Spectral equalization in binaural signals represented by order- truncated spherical harmonics" The Journal of the Acoustical Society of America, vol. 141, no. 6, pp. 4087–4096, 2017.

[2] B. Bernschutz, A. V. Giner, C. Po"rschmann, and J. Arend, "Binaural reproduction of plane waves with reduced modal order" Acta Acustica united with Acustica, vol. 100, no. 5, pp. 972–983, 2014.

[3] Zaunschirm M, Scho"rkhuber C, Ho"ldrich R. Binaural rendering of Ambisonic signals by head-related impulse response time alignment and a diffuseness constraint. The Journal of the Acoustical Society of America. 2018 Jun 19;143(6):3616-27

[4] Scho"rkhuber C, Zaunschirm M, Ho"ldrich R. Binaural Rendering of Ambisonic Signals via Magnitude Least Squares. InProceedings of the DAGA 2018 (Vol. 44, pp. 339-342).

Enumerator

BINAURAL_DECODER_DEFAULT	The default decoder is "BINAURAL_DECODER_LS".
BINAURAL_DECODER_LS	Least-squares (LS) decoder. The simplest binaural decoder.
BINAURAL_DECODER_LSDIFFEQ	Least-squares (LS) decoder with diffuse-field spectral equalisation [1].
BINAURAL_DECODER_SPR	Spatial resampling decoder (on the same lines as the virtual loudspeaker approach) [2].
BINAURAL_DECODER_TA	Time-alignment decoder [3]. Relies on discarding the phase information of the HRTFs, past the frequency at which humans are less sensitive to inter-aural time differences. Therefore, the least-squares fitting priorites matching the interaural level differences (ILDs), rather than the interaural time differences (ITDs).
BINAURAL_DECODER_MAGLS	Magnitude least-squares decoder [4]. On similar lines to the time-alignment decoder, but differing in its execution.

Definition at line 114 of file saf_hoa.h.

6.5.2.2 _LOUDSPEAKER_AMBI_DECODER_METHODS enum _LOUDSPEAKER_AMBI_DECODER_METHODS

Ambisonic decoding options for loudspeaker playback.

Note that all of these decoding options revert to "SAD" if the loudspeakers are uniformly distributed on the sphere. The benefits afforded by MMD, AllRAD, etc. relate to their improved performance when using irregular loudspeaker arrangements.

See also

[1] Zotter F, Pomberger H, Noisternig M. Energy–preserving ambisonic decoding. Acta Acustica united with Acustica. 2012 Jan 1; 98(1):37-47.

[2] Zotter F, Frank M. All-round ambisonic panning and decoding. Journal of the audio engineering society. 2012 Nov 26; 60(10):807-20.

Enumerator

LOUDSPEAKER_DECODER_DEFAULT	The default decoder is "LOUDSPEAKER_DECODER_SAD".
LOUDSPEAKER_DECODER_SAD	Sampling Ambisonic Decoder (SAD): transpose of the loudspeaker spherical harmonic matrix, scaled by the number of loudspeakers. This is the simplest decoding approach, as it simply relies on generating hyper- cardioid beamformers for each loudspeaker direction.
LOUDSPEAKER_DECODER_MMD	Mode-Matching Decoder (MMD): pseudo-inverse of the loudspeaker spherical harmonic matrix. Due to the pseudo-inverse, more signal energy is lent to regions on the surface of the sphere that are more sparsely populated with loudspeakers. Therefore, one must also be careful, as some loudspeakers may be given a huge amount of signal energy and wake the dead.
LOUDSPEAKER_DECODER_EPAD	Energy-Preserving Ambisonic Decoder (EPAD) [1].
LOUDSPEAKER_DECODER_ALLRAD	All-Round Ambisonic Decoder (AllRAD): SAD decoding to t-design, panned for the target loudspeaker directions using VBAP [2]. Perhaps the Ambisonic decoder we would most recommend for irregular loudspeaker layouts.

Definition at line 58 of file saf_hoa.h.

6.5.3 Function Documentation

6.5.3.1 applyDiffCovMatching()

```
void applyDiffCovMatching (
    float_complex * hrtfs,
    float * hrtf_dirs_deg,
    int N_dirs,
    int N_bands,
    int order,
    float * weights,
    float_complex * decMtx )
```

Imposes a diffuse-field covariance constraint on a given binaural decoding matrix [1].

Note

decMtx is altered in-place.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
in,out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Zaunschirm M, Scho"rkhuber C, Ho"ldrich R. Binaural rendering of Ambisonic signals by head-related impulse response time alignment and a diffuseness constraint. The Journal of the Acoustical Society of America. 2018 Jun 19;143(6):3616-27

Definition at line 347 of file saf_hoa.c.

6.5.3.2 getBinauralAmbiDecoderFilters()

```
void getBinauralAmbiDecoderFilters (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int fftSize,
        float fs,
        BINAURAL_AMBI_DECODER_METHODS method,
        int order,
        float * itd_s,
        float * weights,
        int enableDiffCM,
        int enableMaxrE,
        float * decFilters )
```

Computes ambisonic decoding filters for a given HRTF set.

Parameters

in	*hrtfs*	The HRTFs; FLAT: (fftSize/2+1) x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions
in	*fftSize*	FFT size
in	*fs*	Sampling rate
in	*method*	Decoding method (see BINAURAL_AMBI_DECODER_METHODS enum)
in	*order*	Decoding order
in	*itd_s*	Only needed for BINAURAL_DECODER_TA decoder (can set to NULL if using different method); N_dirs x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
in	*enableDiffCM*	Set to '0' to disable diffuse correction, '1' to enable
in	*enableMaxrE*	Set to '0' to disable maxRE weighting, '1' to enable
out	*decFilters*	Decoding filters; FLAT: NUM_EARS x (order+1)^2 x fftSize

Definition at line 297 of file saf_hoa.c.

6.5.3.3 getBinauralAmbiDecoderMtx()

```
void getBinauralAmbiDecoderMtx (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        BINAURAL_AMBI_DECODER_METHODS method,
        int order,
        float * freqVector,
        float * itd_s,
        float * weights,
        int enableDiffCM,
        int enableMaxrE,
        float_complex * decMtx )
```

Computes binaural ambisonic decoding matrices (one per frequency) at a specific order, for a given HRTF set.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands/bins
in	*method*	Decoding method (see BINAURAL_AMBI_DECODER_METHODS enum)
in	*order*	Decoding order
in	*freqVector*	Only needed for BINAURAL_DECODER_TA or BINAURAL_DECODER_MAGLS decoders (set to NULL if using a different method); N_bands x 1
in	*itd_s*	Only needed for BINAURAL_DECODER_TA decoder (set to NULL if using different method); N_dirs x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
in	*enableDiffCM*	Set to '0' to disable diffuse correction, '1' to enable
in	*enableMaxrE*	Set to '0' to disable maxRE weighting, '1' to enable
out	*decMtx*	Decoding matrices (one per frequency); FLAT: N_bands x NUM_EARS x (order+1)^2

Definition at line 225 of file saf_hoa.c.

6.5.3.4 getLoudspeakerAmbiDecoderMtx()

```
void getLoudspeakerAmbiDecoderMtx (
        float * ls_dirs_deg,
        int nLS,
        LOUDSPEAKER_AMBI_DECODER_METHODS method,
        int order,
        int enableMaxrE,
        float * decMtx )
```

Computes an ambisonic decoding matrix of a specific order, for a specific loudspeaker layout.

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES [azi elev]; FLAT: nLS x 2
in	*nLS*	Number of loudspeakers
in	*method*	Decoding method (see "LOUDSPEAKER_AMBI_DECODER_METHODS" enum)
in	*order*	Decoding order
in	*enableMaxrE*	Set to '0' to disable, '1' to enable
out	*decMtx*	Decoding matrix; FLAT: nLS x (order+1)^2

Definition at line 161 of file saf_hoa.c.

6.5.3.5 getMaxREweights()

```
void getMaxREweights (
        int order,
        int diagMtxFlag,
        float * a_n )
```

Computes the weights required to manipulate a hyper-cardioid beam-pattern, such that it has maximum energy in the given look-direction.

Traditionally, due to the back lobes of beamformers when panning a source via Ambisonics encoding/decoding, there is unwanted energy given to loudspeakers directly opposite the true source direction. This max_rE weighting [1] essentially spatially "tapers" the spherical harmonic patterns used to generate said beams, reducing the contribution of the higher orders to the beam patterns. This results in worse spatial selectivity, as the width of the beam pattern main lobe is widened, however, the back lobes are also reduced; thus mitigating the aforementioned problem.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*diagMtxFlag*	Set to '0' if you want the weights to be returned as a vector, or to '1' as a diagonal matrix instead.
out	*a_n*	The max_rE weights, as a vector/diagonal matrix; (order+1)^2 x 1 OR FLAT: (order+1)^2 x (order+1)^2

See also

[1] Zotter F, Frank M. All-round ambisonic panning and decoding. Journal of the audio engineering society. 2012 Nov 26; 60(10):807-20.

Definition at line 127 of file saf_hoa.c.

6.5.3.6 getRSH()

```
void getRSH (
        int order,
        float * dirs_deg,
        int nDirs,
        float * Y )
```

Computes REAL spherical harmonics [1] for multiple directions on the sphere.

The real spherical harmonics are computed WITHOUT the 1/sqrt(4∗pi) term. i.e. max(omni) = 1. Also, compared to getRSH_recur(), this function uses unnorm_legendreP() and double precision, so is more suitable for determining 'Y' in an initialisation stage. This version is indeed slower, but more precise; especially for high orders.

Note

This function is mainly intended for Ambisonics, due to the omission of the 1/sqrt(4∗pi) scaling, and the directions are given in [azimuth elevation] (degrees). In Ambisonics literature, the format convention of 'Y' is referred to as ACN/N3D

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_deg*	Directions on the sphere [azi, ELEVATION] convention, in DEGREES; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITHOUT the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 34 of file saf_hoa.c.

6.5.3.7 getRSH_recur()

```
void getRSH_recur (
        int order,
        float * dirs_deg,
        int nDirs,
        float * Y )
```

Computes REAL spherical harmonics [1] for multiple directions on the sphere.

The real spherical harmonics are computed WITHOUT the 1/sqrt(4∗pi) term. i.e. max(omni) = 1. Also, Compared to getRSH(), this function uses unnorm_legendreP_recur() and single precision, so is more suitable for determining 'Y' in a real-time loop. It sacrifices some precision, as numerical error propogates through the recursion, but it is faster.

Note

This function is mainly intended for Ambisonics, due to the omission of the 1/sqrt(4∗pi) scaling, and the directions are given in [azimuth elevation] (degrees). In Ambisonics literature, the format convention of 'Y' is referred to as ACN/N3D

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_deg*	Directions on the sphere [azi, ELEVATION] convention, in DEGREES; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITHOUT the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 65 of file saf_hoa.c.

6.6 framework/modules/saf_hoa/saf_hoa_internal.c File Reference

Internal part of the higher-order Ambisonics module (saf_hoa)

```
#include "saf_hoa.h"
#include "saf_hoa_internal.h"
```

Functions

- void getEPAD (int order, float ∗ls_dirs_deg, int nLS, float ∗decMtx)

  Computes the "Energy preserving Ambisonic decoder", as detailed in [1].
- void getAllRAD (int order, float ∗ls_dirs_deg, int nLS, float ∗decMtx)

  Computes the "All-round Ambisonics decoder", as detailed in [1].
- void getBinDecoder_LS (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Computes a standard least-squares (LS) binaural ambisonic decoder.
- void getBinDecoder_LSDIFFEQ (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Computes a least-squares (LS) binaural ambisonic decoder with diffuse-field equalisation.
- void getBinDecoder_SPR (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Computes a binaural ambisonic decoder based on spatial resampling (aka virtual loudspeaker decoding) [1].
- void getBinDecoder_TA (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗freqVector, float ∗itd_s, float ∗weights, float_complex ∗decMtx)

  Computes a binaural ambisonic decoder based on the time-alignment (TA) method described in [1].
- void getBinDecoder_MAGLS (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗freqVector, float ∗weights, float_complex ∗decMtx)

  Computes a binaural ambisonic decoder based on the magnitude least-squares (MagLS) method first described in [1], and with the algorithm given in [2].

6.6.1 Detailed Description

Internal part of the higher-order Ambisonics module (saf_hoa)

A collection of Ambisonics related functions. Many of which are derived from the Matlab library by Archontis Politis [1].

See also

[1] https://github.com/polarch/Higher-Order-Ambisonics

Author

Leo McCormack

Date

19.03.2018

6.6.2 Function Documentation

6.6.2.1 getAllRAD()

```
void getAllRAD (
        int order,
        float * ls_dirs_deg,
        int nLS,
        float * decMtx )
```

Computes the "All-round Ambisonics decoder", as detailed in [1].

Parameters

in	*order*	Decoding order
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES [azi elev]; FLAT: nLS x 2
in	*nLS*	Number of loudspeakers
out	*decMtx*	Decoding matrix; FLAT: nLS x (order+1)^2

See also

[1] Zotter, F., Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of the Audio Engineering Society, 60(10), 807:820

Definition at line 87 of file saf_hoa_internal.c.

6.6.2.2 getBinDecoder_LS()

```
void getBinDecoder_LS (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        int order,
        float * weights,
        float_complex * decMtx )
```

Computes a standard least-squares (LS) binaural ambisonic decoder.

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

This standard LS decoder typically exhibits strong timbral colourations in the output when using lower-order input. This is due to input order truncation, as the HRTF grid is typically of much higher modal order than that of the input. This colouration especially affects high- frequencies, since high-frequency energy is predominantly concentrated in the higher-order components. This actually gets worse the more HRTFs you have.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

Definition at line 137 of file saf_hoa_internal.c.

6.6.2.3 getBinDecoder_LSDIFFEQ()

```
void getBinDecoder_LSDIFFEQ (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        int order,
        float * weights,
        float_complex * decMtx )
```

Computes a least-squares (LS) binaural ambisonic decoder with diffuse-field equalisation.

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

This equalisation mitagates some of the timbral colourations exhibited by standard LS decoding; especially at lower input orders.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Z. Ben-Hur, F. Brinkmann, J. Sheaffer, S. Weinzierl, and B. Rafaely, "Spectral equalization in binaural signals represented by order- truncated spherical harmonics," The Journal of the Acoustical Society of America, vol. 141, no. 6, pp. 4087–4096, 2017.

Definition at line 205 of file saf_hoa_internal.c.

6.6.2.4 getBinDecoder_MAGLS()

```
void getBinDecoder_MAGLS (
            float_complex * hrtfs,
            float * hrtf_dirs_deg,
            int N_dirs,
            int N_bands,
            int order,
            float * freqVector,
            float * weights,
            float_complex * decMtx )
```

Computes a binaural ambisonic decoder based on the magnitude least-squares (MagLS) method first described in [1], and with the algorithm given in [2].

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

Mag-LS operates under similar principles held by the TA/TAC decoder, differing in the manner in which the phase is neglected at frequencies above 1.5kHz.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*freqVector*	Frequency vector; N_bands x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Scho"rkhuber C, Zaunschirm M, Ho"ldrich R. Binaural Rendering of Ambisonic Signals via Magnitude Least Squares. InProceedings of the DAGA 2018 (Vol. 44, pp. 339–342).

[2] Zotter, F., & Frank, M. (2019). Ambisonics. Springer Open.

Definition at line 500 of file saf_hoa_internal.c.

6.6.2.5 getBinDecoder_SPR()

```
void getBinDecoder_SPR (
            float_complex * hrtfs,
            float * hrtf_dirs_deg,
            int N_dirs,
            int N_bands,
            int order,
            float * weights,
            float_complex * decMtx )
```

Computes a binaural ambisonic decoder based on spatial resampling (aka virtual loudspeaker decoding) [1].

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

Like "getBinDecoder_LSDIFFEQ" this method mitagates some of the timbral colourations exhibited by standard LS decoding at lower input orders. However, it operates without equalisation. Instead, the modal order of the HRTF grid is brought closer to the decoding order, by simply reducing the number of HRTF points used, and calculating the LS decoder with this reduced number of HRTFs. Therefore, rather than assigning high-frequency energy to higher-order components and subsequently discarding it, due to order truncation, the energy is instead aliased back into the lower-order components and preserved.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] B. Bernschu"tz, A. V. Giner, C. Po"rschmann, and J. Arend, "Binaural reproduction of plane waves with reduced modal order" Acta Acustica united with Acustica, vol. 100, no. 5, pp. 972–983, 2014.

Definition at line 307 of file saf_hoa_internal.c.

6.6.2.6 getBinDecoder_TA()

```
void getBinDecoder_TA (
    float_complex * hrtfs,
    float * hrtf_dirs_deg,
    int N_dirs,
    int N_bands,
    int order,
    float * freqVector,
    float * itd_s,
    float * weights,
    float_complex * decMtx )
```

Computes a binaural ambisonic decoder based on the time-alignment (TA) method described in [1].

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

Since the standard LS decoder is unable to sufficiently fit lower-order spherical harmonics to the highly directive HRTF patterns, this approach addresses this by conducting preliminary time-alignment of the Head-related impulse responses (HRIRs), which aids the LS fitting. This method essentially exploits prior knowledge of the bandwidth in which the inter-aural level differences (ILDs) are the dominant localisation cues; which is above approximately 1.5 kHz. By discarding the phase information of the HRTFs at frequencies above 1.5 kHz, the LS fitting instead prioritises the delivery of the correct magnitude responses; rather than the phase. Thus it ultimately yields improved ILD cues and diminished inter-aural time difference (ITD) cues; but in a frequency range where ILD cues are more important for localisation. This method, therefore, mitagates some of the localisation deficiencies compared with the standard LS decoding at lower input orders.

The paper [1] also detailed a diffuse-field covariance contraint, and the original name was TAC (C=contrained), however, in this framework, this constraint is a seperate independent operation. One may impose it on any binaural decoder using the applyDiffCovMatching() function.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*freqVector*	Frequency vector; N_bands x 1
in	*itd_s*	Interaural time differences (ITDs), seconds; N_dirs x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Zaunschirm M, Scho"rkhuber C, Ho"ldrich R. Binaural rendering of Ambisonic signals by head-related impulse response time alignment and a diffuseness constraint. The Journal of the Acoustical Society of America. 2018 Jun 19;143(6):3616–27

Definition at line 407 of file saf_hoa_internal.c.

6.6.2.7 getEPAD()

```
void getEPAD (
    int order,
    float * ls_dirs_deg,
    int nLS,
    float * decMtx )
```

Computes the "Energy preserving Ambisonic decoder", as detailed in [1].

Note

The function has been written to also work when the number of spherical harmonic components exceeds the number of loudspeakers. In which case, the 'U' matrix from the SVD is truncated instead. However, ideally, nLS > nSH, like in the paper (and in general).

Parameters

in	*order*	Decoding order
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES [azi elev]; FLAT: nLS x 2
in	*nLS*	Number of loudspeakers
out	*decMtx*	Decoding matrix; FLAT: nLS x (order+1)^2

See also

[1] Zotter, F., Pomberger, H., Noisternig, M. (2012). Energy-Preserving Ambisonic Decoding. Acta Acustica United with Acustica, 98(1), 37:47

Definition at line 38 of file saf_hoa_internal.c.

6.7 framework/modules/saf_hoa/saf_hoa_internal.h File Reference

Internal part of the higher-order Ambisonics module (saf_hoa)

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "saf_hoa.h"
#include "../saf_sh/saf_sh.h"
#include "../saf_vbap/saf_vbap.h"
#include "../saf_utilities/saf_utilities.h"
```

Macros

- #define **NUM_EARS** 2

Functions

- void getEPAD (int order, float ∗ls_dirs_deg, int nLS, float ∗decMtx)

  Computes the "Energy preserving Ambisonic decoder", as detailed in [1].
- void getAllRAD (int order, float ∗ls_dirs_deg, int nLS, float ∗decMtx)

  Computes the "All-round Ambisonics decoder", as detailed in [1].
- void getBinDecoder_LS (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Computes a standard least-squares (LS) binaural ambisonic decoder.
- void getBinDecoder_LSDIFFEQ (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Computes a least-squares (LS) binaural ambisonic decoder with diffuse-field equalisation.
- void getBinDecoder_SPR (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗weights, float_complex ∗decMtx)

  Computes a binaural ambisonic decoder based on spatial resampling (aka virtual loudspeaker decoding) [1].
- void getBinDecoder_TA (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗freqVector, float ∗itd_s, float ∗weights, float_complex ∗decMtx)

  Computes a binaural ambisonic decoder based on the time-alignment (TA) method described in [1].
- void getBinDecoder_MAGLS (float_complex ∗hrtfs, float ∗hrtf_dirs_deg, int N_dirs, int N_bands, int order, float ∗freqVector, float ∗weights, float_complex ∗decMtx)

  Computes a binaural ambisonic decoder based on the magnitude least-squares (MagLS) method first described in [1], and with the algorithm given in [2].

6.7.1 Detailed Description

Internal part of the higher-order Ambisonics module (saf_hoa)

A collection of Ambisonics related functions. Many of which are derived from the Matlab library by Archontis Politis [1].

See also

[1] https://github.com/polarch/Higher-Order-Ambisonics

Author

Leo McCormack

Date

19.03.2018

6.7.2 Function Documentation

6.7.2.1 getAllRAD()

```
void getAllRAD (
    int order,
    float * ls_dirs_deg,
    int nLS,
    float * decMtx )
```

Computes the "All-round Ambisonics decoder", as detailed in [1].

Parameters

in	*order*	Decoding order
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES [azi elev]; FLAT: nLS x 2
in	*nLS*	Number of loudspeakers
out	*decMtx*	Decoding matrix; FLAT: nLS x (order+1)^2

See also

[1] Zotter, F., Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of the Audio Engineering Society, 60(10), 807:820

Definition at line 87 of file saf_hoa_internal.c.

6.7.2.2 getBinDecoder_LS()

```
void getBinDecoder_LS (
    float_complex * hrtfs,
    float * hrtf_dirs_deg,
    int N_dirs,
    int N_bands,
    int order,
    float * weights,
    float_complex * decMtx )
```

Computes a standard least-squares (LS) binaural ambisonic decoder.

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

This standard LS decoder typically exhibits strong timbral colourations in the output when using lower-order input. This is due to input order truncation, as the HRTF grid is typically of much higher modal order than that of the input. This colouration especially affects high- frequencies, since high-frequency energy is predominantly concentrated in the higher-order components. This actually gets worse the more HRTFs you have.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

Definition at line 137 of file saf_hoa_internal.c.

6.7.2.3 getBinDecoder_LSDIFFEQ()

```
void getBinDecoder_LSDIFFEQ (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        int order,
        float * weights,
        float_complex * decMtx )
```

Computes a least-squares (LS) binaural ambisonic decoder with diffuse-field equalisation.

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

This equalisation mitagates some of the timbral colourations exhibited by standard LS decoding; especially at lower input orders.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Z. Ben-Hur, F. Brinkmann, J. Sheaffer, S. Weinzierl, and B. Rafaely, "Spectral equalization in binaural signals represented by order- truncated spherical harmonics," The Journal of the Acoustical Society of America, vol. 141, no. 6, pp. 4087–4096, 2017.

Definition at line 205 of file saf_hoa_internal.c.

6.7.2.4 getBinDecoder_MAGLS()

```
void getBinDecoder_MAGLS (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        int order,
        float * freqVector,
        float * weights,
        float_complex * decMtx )
```

Computes a binaural ambisonic decoder based on the magnitude least-squares (MagLS) method first described in [1], and with the algorithm given in [2].

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

Mag-LS operates under similar principles held by the TA/TAC decoder, differing in the manner in which the phase is neglected at frequencies above 1.5kHz.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*freqVector*	Frequency vector; N_bands x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Scho"rkhuber C, Zaunschirm M, Ho"ldrich R. Binaural Rendering of Ambisonic Signals via Magnitude Least Squares. InProceedings of the DAGA 2018 (Vol. 44, pp. 339–342).

[2] Zotter, F., & Frank, M. (2019). Ambisonics. Springer Open.

Definition at line 500 of file saf_hoa_internal.c.

6.7.2.5 getBinDecoder_SPR()

```
void getBinDecoder_SPR (
        float_complex * hrtfs,
        float * hrtf_dirs_deg,
        int N_dirs,
        int N_bands,
        int order,
        float * weights,
        float_complex * decMtx )
```

Computes a binaural ambisonic decoder based on spatial resampling (aka virtual loudspeaker decoding) [1].

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

Like "getBinDecoder_LSDIFFEQ" this method mitagates some of the timbral colourations exhibited by standard LS decoding at lower input orders. However, it operates without equalisation. Instead, the modal order of the HRTF grid is brought closer to the decoding order, by simply reducing the number of HRTF points used, and calculating the LS decoder with this reduced number of HRTFs. Therefore, rather than assigning high-frequency energy to higher-order components and subsequently discarding it, due to order truncation, the energy is instead aliased back into the lower-order components and preserved.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] B. Bernschu"tz, A. V. Giner, C. Po"rschmann, and J. Arend, "Binaural reproduction of plane waves with reduced modal order" Acta Acustica united with Acustica, vol. 100, no. 5, pp. 972–983, 2014.

Definition at line 307 of file saf_hoa_internal.c.

6.7.2.6 getBinDecoder_TA()

```
void getBinDecoder_TA (
            float_complex * hrtfs,
            float * hrtf_dirs_deg,
            int N_dirs,
            int N_bands,
            int order,
            float * freqVector,
            float * itd_s,
            float * weights,
            float_complex * decMtx )
```

Computes a binaural ambisonic decoder based on the time-alignment (TA) method described in [1].

The binaural ambisonic decoder is computed for each frequency bin/band, ready to be applied to input SH signals in the time-frequency domain, or, take the inverse-FFT and apply it via matrix convolution.

Note

Since the standard LS decoder is unable to sufficiently fit lower-order spherical harmonics to the highly directive HRTF patterns, this approach addresses this by conducting preliminary time-alignment of the Head-related impulse responses (HRIRs), which aids the LS fitting. This method essentially exploits prior knowledge of the bandwidth in which the inter-aural level differences (ILDs) are the dominant localisation cues; which is above approximately 1.5 kHz. By discarding the phase information of the HRTFs at frequencies above 1.5 kHz, the LS fitting instead prioritises the delivery of the correct magnitude responses; rather than the phase. Thus it ultimately yields improved ILD cues and diminished inter-aural time difference (ITD) cues; but in a frequency range where ILD cues are more important for localisation. This method, therefore, mitagates some of the localisation deficiencies compared with the standard LS decoding at lower input orders.

The paper [1] also detailed a diffuse-field covariance contraint, and the original name was TAC (C=contrained), however, in this framework, this constraint is a seperate independent operation. One may impose it on any binaural decoder using the applyDiffCovMatching() function.

Parameters

in	*hrtfs*	The HRTFs; FLAT: N_bands x NUM_EARS x N_dirs
in	*hrtf_dirs_deg*	HRTF directions; FLAT: N_dirs x 2
in	*N_dirs*	Number of HRTF directions in set
in	*N_bands*	Number of frequency bands/bins
in	*order*	Decoding order
in	*freqVector*	Frequency vector; N_bands x 1
in	*itd_s*	Interaural time differences (ITDs), seconds; N_dirs x 1
in	*weights*	Integration weights (set to NULL if not available); N_dirs x 1
out	*decMtx*	Decoding matrix; FLAT: N_bands x NUM_EARS x (order+1)^2

See also

[1] Zaunschirm M, Scho"rkhuber C, Ho"ldrich R. Binaural rendering of Ambisonic signals by head-related impulse response time alignment and a diffuseness constraint. The Journal of the Acoustical Society of America. 2018 Jun 19;143(6):3616–27

Definition at line 407 of file saf_hoa_internal.c.

6.7.2.7 getEPAD()

```
void getEPAD (
    int order,
    float * ls_dirs_deg,
    int nLS,
    float * decMtx )
```

Computes the "Energy preserving Ambisonic decoder", as detailed in [1].

Note

The function has been written to also work when the number of spherical harmonic components exceeds the number of loudspeakers. In which case, the 'U' matrix from the SVD is truncated instead. However, ideally, nLS > nSH, like in the paper (and in general).

Parameters

in	*order*	Decoding order
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES [azi elev]; FLAT: nLS x 2
in	*nLS*	Number of loudspeakers
out	*decMtx*	Decoding matrix; FLAT: nLS x (order+1)^2

See also

[1] Zotter, F., Pomberger, H., Noisternig, M. (2012). Energy-Preserving Ambisonic Decoding. Acta Acustica United with Acustica, 98(1), 37:47

Definition at line 38 of file saf_hoa_internal.c.

6.8 framework/modules/saf_hrir/saf_default_hrirs.c File Reference

Default HRIR data.

```
#include "saf_hrir.h"
```

Variables

- const double **__default_hrirs** [836][2][1024]
- const double **__default_hrir_dirs_deg** [836][2]
- const int **__default_N_hrir_dirs** = 836
- const int **__default_hrir_len** = 1024
- const int **__default_hrir_fs** = 48000

6.8.1 Detailed Description

Default HRIR data.

The default HRIR set is a Genelec Aural ID of a KEMAR Dummy Head (@48kHz). Kindly provided by Aki Mäkivirta and Jaan Johansson

Author

Leo McCormack

Date

17.04.2020

6.9 framework/modules/saf_hrir/saf_hrir.c File Reference

Public part of the HRIR/HRTF processing module (saf_hrir)

```
#include "saf_hrir.h"
#include "saf_hrir_internal.h"
```

Functions

- void estimateITDs (float ∗hrirs, int N_dirs, int hrir_len, int fs, float ∗itds_s)

  Estimates the interaural time-differences (ITDs) for each HRIR in a set via the cross-correlation between the left and right IRs.
- void HRIRs2FilterbankHRTFs (float ∗hrirs, int N_dirs, int hrir_len, float_complex ∗hrtf_fb)

  Passes zero padded HRIRs through the afSTFT filterbank.
- void HRIRs2HRTFs (float ∗hrirs, int N_dirs, int hrir_len, int fftSize, float_complex ∗hrtfs)

  Converts a HRIR set to HRTFs, with a given FFT size.
- void diffuseFieldEqualiseHRTFs (int N_dirs, float ∗itds_s, float ∗centreFreq, int N_bands, float_complex ∗hrtfs)

  Applies diffuse-field equalisation to a set of HRTFs.
- void interpHRTFs (float_complex ∗hrtfs, float ∗itds, float ∗freqVector, float ∗vbap_gtable, int N_hrtf_dirs, int N_bands, int N_interp_dirs, float_complex ∗hrtfs_interp)

  Interpolates a set of HRTFs for specified directions; defined by an amplitude normalised VBAP interpolation table (see saf_vbap.h)
- void binauralDiffuseCoherence (float_complex ∗hrtfs, float ∗itds, float ∗freqVector, int N_hrtf_dirs, int N_↩ bands, float ∗HRTFcoh)

  Computes the binaural diffuse coherence per frequency for a given HRTF set, as in [1].

6.9.1 Detailed Description

Public part of the HRIR/HRTF processing module (saf_hrir)

A collection of head-related impulse-response (HRIR) functions. Including estimation of the interaural time differences (ITDs), conversion of HRIRs to HRTF filterbank coefficients, and HRTF interpolation utilising amplitude-normalised VBAP gains.

Author

Leo McCormack

Date

12.12.2016

6.9.2 Function Documentation

6.9.2.1 binauralDiffuseCoherence()

```
void binauralDiffuseCoherence (
    float_complex * hrtfs,
    float * itds,
    float * freqVector,
    int N_hrtf_dirs,
    int N_bands,
    float * HRTFcoh )
```

Computes the binaural diffuse coherence per frequency for a given HRTF set, as in [1].

Parameters

in	*hrtfs*	HRTFs as filterbank coeffs; FLAT: N_bands x 2 x N_hrtf_dirs
in	*itds*	The inter-aural time difference (ITD) for each HRIR; N_hrtf_dirs x 1
in	*freqVector*	Frequency vector; N_bands x 1
in	*N_hrtf_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands
out	*HRTFcoh*	Binaural coherence per frequency; N_bands x 1

See also

[1] A. Politis, "Diffuse-field coherence of sensors with arbitrary directional responses," arXiv preprint arXiv:1608.07713,2016.

Definition at line 257 of file saf_hrir.c.

6.9.2.2 diffuseFieldEqualiseHRTFs()

```
void diffuseFieldEqualiseHRTFs (
        int N_dirs,
        float * itds_s,
        float * centreFreq,
        int N_bands,
        float_complex * hrtfs )
```

Applies diffuse-field equalisation to a set of HRTFs.

Note

This function is NOT suitable for binaural room impulse responses (BRIRs).

Parameters

in	*N_dirs*	Number of HRTFs
in	*itds_s*	HRIR ITDs; N_dirs x 1
in	*centreFreq*	Frequency vector; N_bands x 1
in	*N_bands*	Number of frequency bands/bins
in,out	*hrtfs*	The HRTFs; FLAT: N_bands x 2 x N_dirs

Definition at line 149 of file saf_hrir.c.

6.9.2.3 estimateITDs()

```
void estimateITDs (
        float * hrirs,
        int N_dirs,
        int hrir_len,
        int fs,
        float * itds_s )
```

Estimates the interaural time-differences (ITDs) for each HRIR in a set via the cross-correlation between the left and right IRs.

Parameters

in	*hrirs*	HRIRs; FLAT: N_dirs x 2 x hrir_len
in	*N_dirs*	Number of HRIRs
in	*hrir_len*	Length of the HRIRs in samples
in	*fs*	Sampling rate of the HRIRs
out	*itds_s*	ITDs in seconds; N_dirs x 1

Definition at line 34 of file saf_hrir.c.

6.9.2.4 HRIRs2FilterbankHRTFs()

```
void HRIRs2FilterbankHRTFs (
        float * hrirs,
        int N_dirs,
```

```
int hrir_len,
float_complex * hrtf_fb )
```

Passes zero padded HRIRs through the afSTFT filterbank.

The filterbank coefficients are then normalised with the energy of an impulse, which is centered at approximately the beginning of the HRIR peak.

Note

This function is NOT suitable for binaural room impulse responses (BRIRs). Currently, this function is also hard-coded for 128 hop size with hybrid mode enabled. (133 bands in total)

Parameters

in	*hrirs*	HRIRs; FLAT: N_dirs x 2 x hrir_len
in	*N_dirs*	Number of HRIRs
in	*hrir_len*	Length of the HRIRs in samples
out	*hrtf_fb*	HRTFs as filterbank coeffs; FLAT: 133 x 2 x N_dirs

Definition at line 104 of file saf_hrir.c.

6.9.2.5 HRIRs2HRTFs()

```
void HRIRs2HRTFs (
float * hrirs,
int N_dirs,
int hrir_len,
int fftSize,
float_complex * hrtfs )
```

Converts a HRIR set to HRTFs, with a given FFT size.

Note

If the HRIRs are shorter than the FFT size (hrir_len<fftSize), then the HRIRs are zero-padded. If they are longer, then they are truncated.

Parameters

in	*hrirs*	HRIRs; FLAT: N_dirs x 2 x hrir_len
in	*N_dirs*	Number of HRIRs
in	*hrir_len*	Length of the HRIRs in samples
in	*fftSize*	FFT size
out	*hrtfs*	HRTFs; FLAT: (fftSize/2+1) x 2 x N_dirs

Definition at line 117 of file saf_hrir.c.

6.9.2.6 interpHRTFs()

```
void interpHRTFs (
    float_complex * hrtfs,
    float * itds,
    float * freqVector,
    float * vbap_gtable,
    int N_hrtf_dirs,
    int N_bands,
    int N_interp_dirs,
    float_complex * hrtf_interp )
```

Interpolates a set of HRTFs for specified directions; defined by an amplitude normalised VBAP interpolation table (see saf_vbap.h)

The interpolation is performed by applying interpolation gains to the HRTF magnitudes and HRIR inter-aural time differences separately. The inter-aural phase differences are then reintroduced for each frequency band. Note that this essentially a C implementation of a MatLab function by Archontis Politis.

Note

Use VBAPgainTable2InterpTable() to take a conventional energy- normalised VBAP gain table, and convert it to an amplitude-normalised interpolation table.

Parameters

in	*hrtfs*	HRTFs as filterbank coeffs; FLAT: N_bands x 2 x N_hrtf_dirs
in	*itds*	The inter-aural time difference (ITD) for each HRIR; N_hrtf_dirs x 1
in	*freqVector*	Frequency vector; N_bands x 1
in	*vbap_gtable*	Amplitude-Normalised VBAP gain table; FLAT: N_interp_dirs x N_hrtf_dirs
in	*N_hrtf_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands
in	*N_interp_dirs*	Number of interpolated hrtf positions
out	*hrtf_interp*	interpolated HRTFs; FLAT: N_bands x 2 x N_interp_dirs

Definition at line 198 of file saf_hrir.c.

6.10 framework/modules/saf_hrir/saf_hrir.h File Reference

Public part of the HRIR/HRTF processing module (saf_hrir)

```
#include "../saf_utilities/saf_utilities.h"
```

Functions

- void estimateITDs (float ∗hrirs, int N_dirs, int hrir_len, int fs, float ∗itds_s)

  Estimates the interaural time-differences (ITDs) for each HRIR in a set via the cross-correlation between the left and right IRs.
- void HRIRs2FilterbankHRTFs (float ∗hrirs, int N_dirs, int hrir_len, float_complex ∗hrtf_fb)

  Passes zero padded HRIRs through the afSTFT filterbank.
- void HRIRs2HRTFs (float ∗hrirs, int N_dirs, int hrir_len, int fftSize, float_complex ∗hrtfs)

  Converts a HRIR set to HRTFs, with a given FFT size.

- void diffuseFieldEqualiseHRTFs (int N_dirs, float ∗itds_s, float ∗centreFreq, int N_bands, float_complex ∗hrtfs)

    Applies diffuse-field equalisation to a set of HRTFs.
- void interpHRTFs (float_complex ∗hrtfs, float ∗itds, float ∗freqVector, float ∗vbap_gtable, int N_hrtf_dirs, int N_bands, int N_interp_dirs, float_complex ∗hrtf_interp)

    Interpolates a set of HRTFs for specified directions; defined by an amplitude normalised VBAP interpolation table (see saf_vbap.h)
- void binauralDiffuseCoherence (float_complex ∗hrtfs, float ∗itds, float ∗freqVector, int N_hrtf_dirs, int N_↩ bands, float ∗HRTFcoh)

    Computes the binaural diffuse coherence per frequency for a given HRTF set, as in [1].

Variables

- const double **__default_hrirs** [836][2][1024]
- const double **__default_hrir_dirs_deg** [836][2]
- const int **__default_N_hrir_dirs**
- const int **__default_hrir_len**
- const int **__default_hrir_fs**

6.10.1 Detailed Description

Public part of the HRIR/HRTF processing module (saf_hrir)

A collection of head-related impulse-response (HRIR) functions. Including estimation of the interaural time differences (ITDs), conversion of HRIRs to HRTF filterbank coefficients, and HRTF interpolation utilising amplitude-normalised VBAP gains.

Author

Leo McCormack

Date

12.12.2016

6.10.2 Function Documentation

6.10.2.1 binauralDiffuseCoherence()

```
void binauralDiffuseCoherence (
        float_complex * hrtfs,
        float * itds,
        float * freqVector,
        int N_hrtf_dirs,
        int N_bands,
        float * HRTFcoh )
```

Computes the binaural diffuse coherence per frequency for a given HRTF set, as in [1].

Parameters

in	*hrtfs*	HRTFs as filterbank coeffs; FLAT: N_bands x 2 x N_hrtf_dirs
in	*itds*	The inter-aural time difference (ITD) for each HRIR; N_hrtf_dirs x 1
in	*freqVector*	Frequency vector; N_bands x 1
in	*N_hrtf_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands
out	*HRTFcoh*	Binaural coherence per frequency; N_bands x 1

See also

[1] A. Politis, "Diffuse-field coherence of sensors with arbitrary directional responses," arXiv preprint arXiv:1608.07713,2016.

Definition at line 257 of file saf_hrir.c.

6.10.2.2 diffuseFieldEqualiseHRTFs()

```
void diffuseFieldEqualiseHRTFs (
        int N_dirs,
        float * itds_s,
        float * centreFreq,
        int N_bands,
        float_complex * hrtfs )
```

Applies diffuse-field equalisation to a set of HRTFs.

Note

This function is NOT suitable for binaural room impulse responses (BRIRs).

Parameters

in	*N_dirs*	Number of HRTFs
in	*itds_s*	HRIR ITDs; N_dirs x 1
in	*centreFreq*	Frequency vector; N_bands x 1
in	*N_bands*	Number of frequency bands/bins
in,out	*hrtfs*	The HRTFs; FLAT: N_bands x 2 x N_dirs

Definition at line 149 of file saf_hrir.c.

6.10.2.3 estimateITDs()

```
void estimateITDs (
        float * hrirs,
        int N_dirs,
        int hrir_len,
        int fs,
        float * itds_s )
```

Estimates the interaural time-differences (ITDs) for each HRIR in a set via the cross-correlation between the left and right IRs.

Parameters

in	*hrirs*	HRIRs; FLAT: N_dirs x 2 x hrir_len
in	*N_dirs*	Number of HRIRs
in	*hrir_len*	Length of the HRIRs in samples
in	*fs*	Sampling rate of the HRIRs
out	*itds_s*	ITDs in seconds; N_dirs x 1

Definition at line 34 of file saf_hrir.c.

6.10.2.4 HRIRs2FilterbankHRTFs()

```
void HRIRs2FilterbankHRTFs (
    float * hrirs,
    int N_dirs,
    int hrir_len,
    float_complex * hrtf_fb )
```

Passes zero padded HRIRs through the afSTFT filterbank.

The filterbank coefficients are then normalised with the energy of an impulse, which is centered at approximately the beginning of the HRIR peak.

Note

This function is NOT suitable for binaural room impulse responses (BRIRs). Currently, this function is also hard-coded for 128 hop size with hybrid mode enabled. (133 bands in total)

Parameters

in	*hrirs*	HRIRs; FLAT: N_dirs x 2 x hrir_len
in	*N_dirs*	Number of HRIRs
in	*hrir_len*	Length of the HRIRs in samples
out	*hrtf_fb*	HRTFs as filterbank coeffs; FLAT: 133 x 2 x N_dirs

Definition at line 104 of file saf_hrir.c.

6.10.2.5 HRIRs2HRTFs()

```
void HRIRs2HRTFs (
    float * hrirs,
    int N_dirs,
    int hrir_len,
    int fftSize,
    float_complex * hrtfs )
```

Converts a HRIR set to HRTFs, with a given FFT size.

Note

If the HRIRs are shorter than the FFT size (hrir_len<fftSize), then the HRIRs are zero-padded. If they are longer, then they are truncated.

Parameters

in	*hrirs*	HRIRs; FLAT: N_dirs x 2 x hrir_len
in	*N_dirs*	Number of HRIRs
in	*hrir_len*	Length of the HRIRs in samples
in	*fftSize*	FFT size
out	*hrtfs*	HRTFs; FLAT: (fftSize/2+1) x 2 x N_dirs

Definition at line 117 of file saf_hrir.c.

6.10.2.6 interpHRTFs()

```
void interpHRTFs (
    float_complex * hrtfs,
    float * itds,
    float * freqVector,
    float * vbap_gtable,
    int N_hrtf_dirs,
    int N_bands,
    int N_interp_dirs,
    float_complex * hrtf_interp )
```

Interpolates a set of HRTFs for specified directions; defined by an amplitude normalised VBAP interpolation table (see saf_vbap.h)

The interpolation is performed by applying interpolation gains to the HRTF magnitudes and HRIR inter-aural time differences separately. The inter-aural phase differences are then reintroduced for each frequency band. Note that this essentially a C implementation of a MatLab function by Archontis Politis.

Note

Use VBAPgainTable2InterpTable() to take a conventional energy- normalised VBAP gain table, and convert it to an amplitude-normalised interpolation table.

Parameters

in	*hrtfs*	HRTFs as filterbank coeffs; FLAT: N_bands x 2 x N_hrtf_dirs
in	*itds*	The inter-aural time difference (ITD) for each HRIR; N_hrtf_dirs x 1
in	*freqVector*	Frequency vector; N_bands x 1
in	*vbap_gtable*	Amplitude-Normalised VBAP gain table; FLAT: N_interp_dirs x N_hrtf_dirs
in	*N_hrtf_dirs*	Number of HRTF directions
in	*N_bands*	Number of frequency bands
in	*N_interp_dirs*	Number of interpolated hrtf positions
out	*hrtf_interp*	interpolated HRTFs; FLAT: N_bands x 2 x N_interp_dirs

Definition at line 198 of file saf_hrir.c.

6.11 framework/modules/saf_hrir/saf_hrir_internal.c File Reference

Internal part of the HRIR/HRTF processing module (saf_hrir)

```
#include "saf_hrir.h"
#include "saf_hrir_internal.h"
```

Functions

- static void afAnalyse (float ∗inTD, int nSamplesTD, int nCH, float_complex ∗outTF)
  Passes input time-domain data through afSTFT.
- void FIRtoFilterbankCoeffs (float ∗hIR, int N_dirs, int nCH, int ir_len, int nBands, float_complex ∗hFB)
  Converts FIR filters into Filterbank Coefficients.

6.11.1 Detailed Description

Internal part of the HRIR/HRTF processing module (saf_hrir)

A collection of head-related impulse-response (HRIR) functions. Including estimation of the interaural time differences (ITDs), conversion of HRIRs to HRTF filterbank coefficients, and HRTF interpolation utilising amplitude-normalised VBAP gains.

Author

Leo McCormack

Date

12.12.2016

6.11.2 Function Documentation

6.11.2.1 afAnalyse()

```
static void afAnalyse (
        float * inTD,
        int nSamplesTD,
        int nCH,
        float_complex * outTF ) [static]
```

Passes input time-domain data through afSTFT.

Copyright (c) 2015 Juha Vilkamo, MIT license

Note

Currently hard coded for a 128 hop size with hybrid mode enabled

Definition at line 40 of file saf_hrir_internal.c.

6.11.2.2 FIRtoFilterbankCoeffs()

```
void FIRtoFilterbankCoeffs (
        float * hIR,
        int N_dirs,
        int nCH,
        int ir_len,
        int N_bands,
        float_complex * hFB )
```

Converts FIR filters into Filterbank Coefficients.

Note

This is currently hard coded for a 128 hop size with hybrid mode enabled (see afSTFTlib.h).

Parameters

in	*hIR*	Time-domain FIR; FLAT: N_dirs x nCH x ir_len
in	*N_dirs*	Number of FIR sets
in	*nCH*	Number of channels per FIR set
in	*ir_len*	Length of the FIR
in	*N_bands*	Number of time-frequency domain bands
out	*hFB*	The FIRs as Filterbank coefficients; FLAT: N_bands x nCH x N_dirs

Definition at line 96 of file saf_hrir_internal.c.

6.12 framework/modules/saf_hrir/saf_hrir_internal.h File Reference

Internal part of the HRIR/HRTF processing module (saf_hrir)

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "saf_hrir.h"
#include "../../resources/afSTFT/afSTFTlib.h"
#include "../saf_utilities/saf_utilities.h"
```

Macros

- #define **NUM_EARS** 2

Functions

- void FIRtoFilterbankCoeffs (float ∗hIR, int N_dirs, int nCH, int ir_len, int N_bands, float_complex ∗hFB)
  Converts FIR filters into Filterbank Coefficients.

6.12.1 Detailed Description

Internal part of the HRIR/HRTF processing module (saf_hrir)

A collection of head-related impulse-response (HRIR) functions. Including estimation of the interaural time differences (ITDs), conversion of HRIRs to HRTF filterbank coefficients, and HRTF interpolation utilising amplitude-normalised VBAP gains.

Author

Leo McCormack

Date

12.12.2016

6.12.2 Function Documentation

6.12.2.1 FIRtoFilterbankCoeffs()

```
void FIRtoFilterbankCoeffs (
    float * hIR,
    int N_dirs,
    int nCH,
    int ir_len,
    int N_bands,
    float_complex * hFB )
```

Converts FIR filters into Filterbank Coefficients.

Note

This is currently hard coded for a 128 hop size with hybrid mode enabled (see afSTFTlib.h).

Parameters

in	*hIR*	Time-domain FIR; FLAT: N_dirs x nCH x ir_len
in	*N_dirs*	Number of FIR sets
in	*nCH*	Number of channels per FIR set
in	*ir_len*	Length of the FIR
in	*N_bands*	Number of time-frequency domain bands
out	*hFB*	The FIRs as Filterbank coefficients; FLAT: N_bands x nCH x N_dirs

Definition at line 96 of file saf_hrir_internal.c.

6.13 framework/modules/saf_hrir/saf_sofa_reader.c File Reference

A simple sofa reader, which returns only the bare minimum.

```
#include "saf_sofa_reader.h"
#include "saf_hrir.h"
```

6.13.1 Detailed Description

A simple sofa reader, which returns only the bare minimum.

Note

This (optional) SOFA reader, requires netcdf to be linked.

Author

Leo McCormack

Date

21.11.2017

6.14 framework/modules/saf_hrir/saf_sofa_reader.h File Reference

A simple sofa reader, which returns only the bare minimum.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../saf_utilities/saf_utilities.h"
```

Functions

- void loadSofaFile (char ∗sofa_filepath, float ∗∗hrirs, float ∗∗hrir_dirs_deg, int ∗N_hrir_dirs, int ∗hrir_len, int ∗hrir_fs)

    A bare-bones SOFA file reader.

6.14.1 Detailed Description

A simple sofa reader, which returns only the bare minimum.

Note

This (optional) SOFA reader, requires netcdf to be linked to the project.

Author

Leo McCormack

Date

21.11.2017

6.14.2 Function Documentation

6.14.2.1 loadSofaFile()

```
void loadSofaFile (
    char * sofa_filepath,
    float ** hrirs,
    float ** hrir_dirs_deg,
    int * N_hrir_dirs,
    int * hrir_len,
    int * hrir_fs )
```

A bare-bones SOFA file reader.

Allocates memory and copies the values of the essential data contained in a SOFA file to the output arguments.

Note

The hrirs are returned as NULL if the file does not exist.

Parameters

in	*sofa_filepath*	Directory/file_name of the SOFA file you wish to load. Optionally, you may set this as NULL, and the function will return the default HRIR data.
out	*hrirs*	(&) the HRIR data; FLAT: N_hrir_dirs x 2 x hrir_len
out	*hrir_dirs_deg*	(&) the HRIR positions; FLAT: N_hrir_dirs x 2
out	*N_hrir_dirs*	(&) number of HRIR positions
out	*hrir_len*	(&) length of the HRIRs, in samples
out	*hrir_fs*	(&) sampling rate of the HRIRs

6.15 framework/modules/saf_sh/saf_sh.c File Reference

Public part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

```
#include "saf_sh.h"
#include "saf_sh_internal.h"
```

Functions

- static double Jn (int n, double z)

  Wrapper for the xylindrical bessel function of the first kind
- static double Yn (int n, double z)

  Wrapper for the xylindrical bessel function of the second kind.
- void yawPitchRoll2Rzyx (float yaw, float pitch, float roll, int rollPitchYawFLAG, float R[3][3])

  Constructs a 3x3 rotation matrix from the Euler angles, using the yaw-pitch-roll (zyx) convention.
- void unitSph2Cart (float azi_rad, float elev_rad, float xyz[3])

  Converts spherical coordinates to cartesian coordinates of unit length.
- void unitCart2Sph (float xyz[3], float AziElev_rad[2])

  Converts cartesian coordinates of unit length to spherical coordinates.
- void unitCart2Sph_aziElev (float xyz[3], float ∗azi_rad, float ∗elev_rad)

  Converts cartesian coordinates of unit length to spherical coordinates.
- void unnorm_legendreP (int n, double ∗x, int lenX, double ∗y)

  Calculates unnormalised legendre polynomials up to order N, for all values in vector x [1].
- void unnorm_legendreP_recur (int n, float ∗x, int lenX, float ∗Pnm_minus1, float ∗Pnm_minus2, float ∗Pnm)

  Calculates unnormalised legendre polynomials up to order N, for all values in vector x.
- void getSHreal (int order, float ∗dirs_rad, int nDirs, float ∗Y)

  Computes REAL spherical harmonics [1] for each direction on the sphere.
- void getSHreal_recur (int N, float ∗dirs_rad, int nDirs, float ∗Y)

  Computes REAL spherical harmonics [1] for each direction on the sphere.
- void getSHcomplex (int order, float ∗dirs_rad, int nDirs, float_complex ∗Y)

  Computes COMPLEX spherical harmonics [1] for each direction on the sphere.
- void complex2realSHMtx (int order, float_complex ∗T_c2r)

  Computes a complex to real spherical harmonic transform matrix.
- void real2complexSHMtx (int order, float_complex ∗T_r2c)

  Computes a real to complex spherical harmonic transform matrix.
- void complex2realCoeffs (int order, float_complex ∗C_N, int K, float ∗R_N)

  Converts SH coefficients from the complex to real basis.

- void getSHrotMtxReal (float Rxyz[3][3], float ∗RotMtx, int L)

  Generates a real-valued spherical harmonic rotation matrix [1] (assumes ACN channel ordering convention)
- void computeVelCoeffsMtx (int sectorOrder, float_complex ∗A_xyz)

  Computes the matrices that generate the coefficients of the beampattern of order (sectorOrder+1) that is essentially the product of a pattern of order=sectorOrder and a dipole.
- float computeSectorCoeffsEP (int orderSec, float_complex ∗A_xyz, SECTOR_PATTERNS pattern, float ∗sec_dirs_deg, int nSecDirs, float ∗sectorCoeffs)

  Computes the beamforming matrices of sector and velocity coefficients for ENERGY-preserving (EP) sectors for real SH.
- float computeSectorCoeffsAP (int orderSec, float_complex ∗A_xyz, SECTOR_PATTERNS pattern, float ∗sec_dirs_deg, int nSecDirs, float ∗sectorCoeffs)

  Computes the beamforming matrices of sector and velocity coefficients for AMPLITUDE-preserving (AP) sectors for real SH.
- void beamWeightsCardioid2Spherical (int N, float ∗b_n)

  Generates spherical coefficients for cardioids.
- void beamWeightsHypercardioid2Spherical (int N, float ∗b_n)

  Generates beamweights in the SHD for hypercardioid beampatterns.
- void beamWeightsMaxEV (int N, float ∗b_n)

  Generates beamweights in the SHD for maximum energy-vector beampatterns.
- void beamWeightsVelocityPatternsReal (int order, float ∗b_n, float azi_rad, float elev_rad, float_complex ∗A←_xyz, float ∗velCoeffs)

  Generates beamforming coefficients for velocity patterns (REAL)
- void beamWeightsVelocityPatternsComplex (int order, float ∗b_n, float azi_rad, float elev_rad, float_complex ∗A_xyz, float_complex ∗velCoeffs)

  Generates beamforming coefficients for velocity patterns (COMPLEX)
- void rotateAxisCoeffsReal (int order, float ∗c_n, float theta_0, float phi_0, float ∗c_nm)

  Generates spherical coefficients for a rotated axisymmetric pattern (REAL)
- void rotateAxisCoeffsComplex (int order, float ∗c_n, float theta_0, float phi_0, float_complex ∗c_nm)

  Generates spherical coefficients for a rotated axisymmetric pattern (COMPLEX)
- void checkCondNumberSHTReal (int order, float ∗dirs_rad, int nDirs, float ∗w, float ∗cond_N)

  Computes the condition numbers for a least-squares SHT.
- void generatePWDmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nGrid_dirs, float ∗pmap)

  Generates a powermap based on the energy of plane-wave decomposition (PWD)/ hyper-cardioid beamformers.
- void generateMVDRmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nGrid_dirs, float regPar, float ∗pmap, float_complex ∗w_MVDR_out)

  Generates a powermap based on the energy of adaptive minimum variance distortion-less response (MVDR) beamformers.
- void generateCroPaCLCMVmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nGrid_dirs, float regPar, float lambda, float ∗pmap)

  (EXPERIMENTAL) Generates a powermap utilising the CroPaC LCMV post-filter described in [1]
- void generateMUSICmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nSources, int nGrid_dirs, int logScaleFlag, float ∗pmap)

  Generates an activity-map based on the sub-space multiple-signal classification (MUSIC) method.
- void generateMinNormMap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nSources, int nGrid_dirs, int logScaleFlag, float ∗pmap)

  Generates an activity-map based on the sub-space minimum-norm (MinNorm) method.
- void bessel_Jn (int N, double ∗z, int nZ, double ∗J_n, double ∗dJ_n)

  Computes the (cylindrical) Bessel function of the first kind: Jn.
- void bessel_Yn (int N, double ∗z, int nZ, double ∗Y_n, double ∗dY_n)

  Computes the (cylindrical) Bessel function of the second kind: Yn.
- void hankel_Hn1 (int N, double ∗z, int nZ, double_complex ∗H_n1, double_complex ∗dH_n1)

  Computes the (cylindrical) Hankel function of the first kind: Hn1.

- void hankel_Hn2 (int N, double ∗z, int nZ, double_complex ∗H_n2, double_complex ∗dH_n2)

    Computes the (cylindrical) Hankel function of the second kind: Hn2.
- void bessel_jn (int N, double ∗z, int nZ, int ∗maxN, double ∗j_n, double ∗dj_n)

    Computes the spherical Bessel function of the first kind: jn.
- void bessel_in (int N, double ∗z, int nZ, int ∗maxN, double ∗i_n, double ∗di_n)

    Computes the modified spherical Bessel function of the first kind: in.
- void bessel_yn (int N, double ∗z, int nZ, int ∗maxN, double ∗y_n, double ∗dy_n)

    Computes the spherical Bessel function of the second kind (Neumann): yn.
- void bessel_kn (int N, double ∗z, int nZ, int ∗maxN, double ∗k_n, double ∗dk_n)

    Computes the modified spherical Bessel function of the second kind: kn.
- void hankel_hn1 (int N, double ∗z, int nZ, int ∗maxN, double_complex ∗h_n1, double_complex ∗dh_n1)

    Computes the spherical Hankel function of the first kind: hn1.
- void hankel_hn2 (int N, double ∗z, int nZ, int ∗maxN, double_complex ∗h_n2, double_complex ∗dh_n2)

    Computes the spherical Hankel function of the second kind: hn2.
- void cylModalCoeffs (int order, double ∗kr, int nBands, ARRAY_CONSTRUCTION_TYPES arrayType, double_complex ∗b_N)

    Calculates the modal coefficients for open/rigid cylindrical arrays.
- float sphArrayAliasLim (float r, float c, int maxN)

    Returns a simple estimate of the spatial aliasing limit (the kR = maxN rule)
- void sphArrayNoiseThreshold (int maxN, int Nsensors, float r, float c, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, float maxG_db, float ∗f_lim)

    Computes the frequncies (per order), at which the noise of a SHT of a SMA exceeds a specified maximum level.
- void sphModalCoeffs (int order, double ∗kr, int nBands, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, double_complex ∗b_N)

    Calculates the modal coefficients for open/rigid spherical arrays.
- void sphScattererModalCoeffs (int order, double ∗kr, double ∗kR, int nBands, double_complex ∗b_N)

    Calculates the modal coefficients for a rigid spherical scatterer with omni-directional sensors.
- void sphScattererDirModalCoeffs (int order, double ∗kr, double ∗kR, int nBands, double dirCoeff, double_complex ∗b_N)

    Calculates the modal coefficients for a rigid spherical scatterer with directional sensors.
- void sphDiffCohMtxTheory (int order, float ∗sensor_dirs_rad, int N_sensors, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, double ∗kr, double ∗kR, int nBands, double ∗M_diffcoh)

    Calculates the theoretical diffuse coherence matrix for a spherical array.
- void simulateCylArray (int order, double ∗kr, int nBands, float ∗sensor_dirs_rad, int N_sensors, float ∗src_dirs_deg, int N_srcs, ARRAY_CONSTRUCTION_TYPES arrayType, float_complex ∗H_array)

    Simulates a cylindrical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the cylinder.
- void simulateSphArray (int order, double ∗kr, double ∗kR, int nBands, float ∗sensor_dirs_rad, int N_sensors, float ∗src_dirs_deg, int N_srcs, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, float_complex ∗H_array)

    Simulates a spherical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the sphere.
- void evaluateSHTfilters (int order, float_complex ∗M_array2SH, int nSensors, int nBands, float_complex ∗H_array, int nDirs, float_complex ∗Y_grid, float ∗cSH, float ∗lSH)

    Generates some objective measures, which evaluate the performance of the spatial encoding filters.

Variables

- const float wxyzCoeffs [4][4]

    First-order ACN/N3D to WXYZ conversion matrix.

6.15.1 Detailed Description

Public part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

A collection of spherical harmonic related functions. Many of which have been derived from Matlab libraries by Archontis Politis [1-3].

See also

[1] https://github.com/polarch/Spherical-Harmonic-Transform

[2] https://github.com/polarch/Array-Response-Simulator

[3] https://github.com/polarch/Spherical-Array-Processing

Author

Leo McCormack

Date

22.05.2016

6.15.2 Function Documentation

6.15.2.1 beamWeightsCardioid2Spherical()

```
void beamWeightsCardioid2Spherical (
    int N,
    float * b_n )
```

Generates spherical coefficients for cardioids.

For a specific order N of a higher order cardioid of the form D(theta)=(1/2)^N * (1+cos(theta))^N, generate the beamweights for the same pattern in the SHD. Because the pattern is axisymmetric only the N+1 coefficients of m=0 are returned.

Parameters

in	*N*	Order of spherical harmonic expansion
out	*b_n*	Beamformer weights; (N+1) x 1

Definition at line 789 of file saf_sh.c.

6.15.2.2 beamWeightsHypercardioid2Spherical()

```
void beamWeightsHypercardioid2Spherical (
    int N,
    float * b_n )
```

Generates beamweights in the SHD for hypercardioid beampatterns.

The hypercardioid is the pattern that maximises the directivity-factor for a certain SH order N. The hypercardioid is also the plane-wave decomposition beamformer in the SHD, also called 'regular' because the beamweights are just the SH values on the beam-direction. Since the pattern is axisymmetric only the N+1 coefficients of m=0 are returned.

Parameters

in	*N*	Order of spherical harmonic expansion
out	*b_n*	Beamformer weights; (N+1) x 1

Definition at line 806 of file saf_sh.c.

6.15.2.3 beamWeightsMaxEV()

```
void beamWeightsMaxEV (
    int N,
    float * b_n )
```

Generates beamweights in the SHD for maximum energy-vector beampatterns.

Generate the beamweights for the a maximum energy-vector beampattern in the SHD. This pattern originates from ambisonic-related research and it maximises the ambisonic energy-vector, which is essentially the directional centroid of the squared pattern. IT can also be seen as the pattern that maximizes the acoustic intensity vector of a diffuse field weighted with this pattern. In practice it is almost the same as a supercardioid that maximizes front-back power ratio for a certain order, and it can be used as such. Because the pattern is axisymmetric only the N+1 coefficients of m=0 are returned. Details for their theory can be found e.g. in [1].

Parameters

in	*N*	Order of spherical harmonic expansion
out	*b_n*	Beamformer weights; (N+1) x 1

See also

[1] Zotter, F., Pomberger, H. and Noisternig, M., 2012. Energy- preserving ambisonic decoding. Acta Acustica united with Acustica, 98(1), pp.37-47.

Definition at line 824 of file saf_sh.c.

6.15.2.4 beamWeightsVelocityPatternsComplex()

```
void beamWeightsVelocityPatternsComplex (
    int order,
    float * b_n,
    float azi_rad,
    float elev_rad,
    float_complex * A_xyz,
    float_complex * velCoeffs )
```

Generates beamforming coefficients for velocity patterns (COMPLEX)

If the sound-field is weighted with an axisymmetric spatial distribution described by the N+1 SH coefficients b_n, then the beamweights capturing the velocity signals for the weighted sound-field are of an order one higher than the weighting pattern, and can be derived from it. This type of beamforming has some applications for spatial sound reproduction and acoustic analysis, see [1].

Parameters

in	*order*	Order of spherical harmonic expansion
in	*b_n*	Axisymmetric beamformer weights; (order+1) x 1
in	*azi_rad*	Orientation, azimuth in RADIANS
in	*elev_rad*	Orientation, ELEVATION in RADIANS
in	*A_xyz*	Velocity coefficients; see computeVelCoeffsMtx(); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
out	*velCoeffs*	Beamforming coefficients for velocity patterns; FLAT: (order+2)^2 x 3

See also

[1] Politis, A. and Pulkki, V., 2016. Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409.

Definition at line 872 of file saf_sh.c.

6.15.2.5 beamWeightsVelocityPatternsReal()

```
void beamWeightsVelocityPatternsReal (
    int order,
    float * b_n,
    float azi_rad,
    float elev_rad,
    float_complex * A_xyz,
    float * velCoeffs )
```

Generates beamforming coefficients for velocity patterns (REAL)

If the sound-field is weighted with an axisymmetric spatial distribution described by the N+1 SH coefficients b_n, then the beamweights capturing the velocity signals for the weighted sound-field are of an order one higher than the weighting pattern, and can be derived from it. This type of beamforming has some applications for spatial sound reproduction and acoustic analysis, see [1].

Parameters

in	*order*	Order of spherical harmonic expansion
in	*b_n*	Axisymmetric beamformer weights; (order+1) x 1
in	*azi_rad*	Orientation, azimuth in RADIANS
in	*elev_rad*	Orientation, ELEVATION in RADIANS
in	*A_xyz*	Velocity coefficients; see computeVelCoeffsMtx(); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
out	*velCoeffs*	Beamforming coefficients for velocity patterns; FLAT: (order+2)^2 x 3

See also

[1] Politis, A. and Pulkki, V., 2016. Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409.

Definition at line 851 of file saf_sh.c.

6.15.2.6 bessel_in()

```
void bessel_in (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * i_n,
    double * di_n )
```

Computes the modified spherical Bessel function of the first kind: in.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*i_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*di_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1523 of file saf_sh.c.

6.15.2.7 bessel_Jn()

```
void bessel_Jn (
    int N,
    double * z,
    int nZ,
    double * J_n,
    double * dJ_n )
```

Computes the (cylindrical) Bessel function of the first kind: Jn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*J_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dJ_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1346 of file saf_sh.c.

6.15.2.8 bessel_jn()

```
void bessel_jn (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * j_n,
    double * dj_n )
```

Computes the spherical Bessel function of the first kind: jn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*j_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dj_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1468 of file saf_sh.c.

6.15.2.9 bessel_kn()

```
void bessel_kn (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * k_n,
    double * dk_n )
```

Computes the modified spherical Bessel function of the second kind: kn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*k_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dk_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1628 of file saf_sh.c.

6.15.2.10 bessel_Yn()

```
void bessel_Yn (
    int N,
    double * z,
    int nZ,
    double * Y_n,
    double * dY_n )
```

Computes the (cylindrical) Bessel function of the second kind: Yn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*Y_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dY_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1377 of file saf_sh.c.

6.15.2.11 bessel_yn()

```
void bessel_yn (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * y_n,
    double * dy_n )
```

Computes the spherical Bessel function of the second kind (Neumann): yn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*y_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dy_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1578 of file saf_sh.c.

6.15.2.12 checkCondNumberSHTReal()

```
void checkCondNumberSHTReal (
    int order,
    float * dirs_rad,
    int nDirs,
    float * w,
    float * cond_N )
```

Computes the condition numbers for a least-squares SHT.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
in	*w*	Integration weights; nDirs x 1
out	*cond_N*	Condition numbers; (order+1) x 1

Definition at line 957 of file saf_sh.c.

6.15.2.13 complex2realCoeffs()

```
void complex2realCoeffs (
    int order,
    float_complex * C_N,
    int K,
    float * R_N )
```

Converts SH coefficients from the complex to real basis.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*C_N*	Complex coeffients; FLAT: (order+1)^2 x K
in	*K*	Number of columns
out	*R_N*	Real coefficients; FLAT: (order+1)^2 x K

Definition at line 531 of file saf_sh.c.

6.15.2.14 complex2realSHMtx()

```
void complex2realSHMtx (
    int order,
    float_complex * T_c2r )
```

Computes a complex to real spherical harmonic transform matrix.

Computes the unitary transformation matrix T_c2r. It expresses the real spherical harmonics with respect to the complex ones, so that r_N = T_c2r ∗ y_N, where r_N and y_N is are the real and complex SH vectors, respectively.

Parameters

in	*order*	Order of spherical harmonic expansion
out	*T_c2r*	Transformation matrix for complex->real; FLAT: (order+1)^2 x (order+1)^2

Definition at line 467 of file saf_sh.c.

6.15.2.15 computeSectorCoeffsAP()

```
float computeSectorCoeffsAP (
    int orderSec,
    float_complex * A_xyz,
    SECTOR_PATTERNS pattern,
    float * sec_dirs_deg,
    int nSecDirs,
    float * sectorCoeffs )
```

Computes the beamforming matrices of sector and velocity coefficients for AMPLITUDE-preserving (AP) sectors for real SH.

This partitioning of the sound-field into spatially-localised sectors has been used for parametric sound-field reproduction in [1] and visualision in [2,3].

Parameters

in	*orderSec*	Order of sector patterns
in	*A_xyz*	Velocity coefficients (see "computeVelCoeffsMtx"); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
in	*pattern*	See "SECTOR_PATTERNS" enum for the options
in	*sec_dirs_deg*	Sector directions [azi elev], in DEGREES; FLAT: nSecDirs x 2
in	*nSecDirs*	Number of sectors
out	*sectorCoeffs*	The sector coefficients; FLAT: (nSecDirs*4) x (orderSec+2)^2

Returns

Normalisation coefficient

See also

[1] Politis, A., Vilkamo, J., & Pulkki, V. (2015). Sector-based parametric sound field reproduction in the spherical harmonic domain. IEEE Journal of Selected Topics in Signal Processing, 9(5), 852-866.

[2] McCormack, L., Politis, A., and Pulkki, V. (2019). "Sharpening of angular spectra based on a directional re-assignment approach for ambisonic sound-field visualisation". IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[3] McCormack, L., Delikaris-Manias, S., Politis, A., Pavlidi, D., Farina, A., Pinardi, D. and Pulkki, V., 2019. Applications of Spatially Localized Active-Intensity Vectors for Sound-Field Visualization. Journal of the Audio Engineering Society, 67(11), pp.840-854.

Definition at line 736 of file saf_sh.c.

6.15.2.16 computeSectorCoeffsEP()

```
float computeSectorCoeffsEP (
    int orderSec,
    float_complex * A_xyz,
    SECTOR_PATTERNS pattern,
    float * sec_dirs_deg,
    int nSecDirs,
    float * sectorCoeffs )
```

Computes the beamforming matrices of sector and velocity coefficients for ENERGY-preserving (EP) sectors for real SH.

This partitioning of the sound-field into spatially-localised sectors has been used for parametric sound-field reproduction in [1] and visualisation in [2,3].

Parameters

in	*orderSec*	Order of sector patterns
in	*A_xyz*	Velocity coefficients (see "computeVelCoeffsMtx"); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
in	*pattern*	See "SECTOR_PATTERNS" enum for the options
in	*sec_dirs_deg*	Sector directions [azi elev], in DEGREES; FLAT: nSecDirs x 2
in	*nSecDirs*	Number of sectors
out	*sectorCoeffs*	The sector coefficients; FLAT: (nSecDirs*4) x (orderSec+2)^2

Returns

Normalisation coefficient

See also

[1] Politis, A., Vilkamo, J., & Pulkki, V. (2015). Sector-based parametric sound field reproduction in the spherical harmonic domain. IEEE Journal of Selected Topics in Signal Processing, 9(5), 852-866.

[2] McCormack, L., Politis, A., and Pulkki, V. (2019). "Sharpening of angular spectra based on a directional re-assignment approach for ambisonic sound-field visualisation". IEEE International Conference ' on Acoustics, Speech and Signal Processing (ICASSP).

[3] McCormack, L., Delikaris-Manias, S., Politis, A., Pavlidi, D., Farina, A., Pinardi, D. and Pulkki, V., 2019. Applications of Spatially Localized Active-Intensity Vectors for Sound-Field Visualization. Journal of the Audio Engineering Society, 67(11), pp.840-854.

Definition at line 670 of file saf_sh.c.

6.15.2.17 computeVelCoeffsMtx()

```
void computeVelCoeffsMtx (
    int sectorOrder,
    float_complex * A_xyz )
```

Computes the matrices that generate the coefficients of the beampattern of order (sectorOrder+1) that is essentially the product of a pattern of order=sectorOrder and a dipole.

It is used in beamWeightsVelocityPatterns(). For the derivation of the matrices see [1].

Parameters

in	*sectorOrder*	Order of patterns
out	*A_xyz*	Velocity coefficients; FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3

See also

[1] Politis, A. and Pulkki, V., 2016. Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409

Definition at line 638 of file saf_sh.c.

6.15.2.18 cylModalCoeffs()

```
void cylModalCoeffs (
    int order,
    double * kr,
    int nBands,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double_complex * b_N )
```

Calculates the modal coefficients for open/rigid cylindrical arrays.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 1794 of file saf_sh.c.

6.15.2.19 evaluateSHTfilters()

```
void evaluateSHTfilters (
    int order,
    float_complex * M_array2SH,
    int nSensors,
    int nBands,
    float_complex * H_array,
    int nDirs,
    float_complex * Y_grid,
    float * cSH,
    float * lSH )
```

Generates some objective measures, which evaluate the performance of the spatial encoding filters.

This analysis is performed by comparing the spatial resolution of the spherical harmonic components generated by the encoding filters, with the ideal SH components. For more information, the reader is directed to [1,2].

Parameters

in	*order*	Transform/encoding order
in	*M_array2SH*	Encoding matrix per frequency; FLAT: nBands x (order+1)^2 x nSensors
in	*nSensors*	Number of sensors
in	*nBands*	Number of frequency bands/bins
in	*H_array*	Measured/modelled array responses for many directions; FLAT: nBands x nSensors x nDirs
in	*nDirs*	Number of directions the array was measured/modelled
in	*Y_grid*	Spherical harmonics weights for each grid direction; FLAT: nDirs x (order+1)^2
out	*cSH*	Absolute values of the spatial correlation per band and order; FLAT: nBands x (order+1)
out	*lSH*	Level difference per band and order; FLAT: nBands x (order+1)

See also

[1] Moreau, S., Daniel, J., Bertet, S., 2006, 3D sound field recording with higher order ambisonics–objective measurements and validation of spherical microphone. In Audio Engineering Society Convention 120.

[2] Politis, A., Gamper, H. (2017). "Comparing Modelled And Measurement- Based Spherical Harmonic Encoding Filters For Spherical Microphone Arrays. In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA).

Definition at line 2307 of file saf_sh.c.

6.15.2.20 generateCroPaCLCMVmap()

```
void generateCroPaCLCMVmap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nGrid_dirs,
    float regPar,
    float lambda,
    float * pmap )
```

(EXPERIMENTAL) Generates a powermap utilising the CroPaC LCMV post-filter described in [1]

The spatial post-filter is estimated for all directions on the grid, and is used to supress reverb/noise interference that may be present in an MVDR map. Unlike in the paper, the second column for the contraints 'A', is Y.∗diag(Cx), rather than utilising a maximum energy beamformer. The post- filters are then applied to the MVDR powermap map derived in the sherical harmonic domain, rather than an MVDR beamformer generated directly in the microphone array signal domain, like in the paper. Otherwise, the algorithm is the same.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nGrid_dirs*	Number of grid directions
in	*regPar*	Regularisation parameter, for diagonal loading of Cx
in	*lambda*	Parameter controlling how harsh CroPaC is applied, 0..1; 0: fully CroPaC, 1: fully MVDR
out	*pmap*	Resulting CroPaC LCMV powermap; nGrid_dirs x 1

See also

[1] Delikaris-Manias, S., Vilkamo, J., & Pulkki, V. (2016). Signal- dependent spatial filtering based on weighted-orthogonal beamformers in the spherical harmonic domain. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), 24(9), 1507-1519.

Definition at line 1136 of file saf_sh.c.

6.15.2.21 generateMinNormMap()

```
void generateMinNormMap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nSources,
    int nGrid_dirs,
    int logScaleFlag,
    float * pmap )
```

Generates an activity-map based on the sub-space minimum-norm (MinNorm) method.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nSources*	Number of sources present in sound scene
in	*nGrid_dirs*	Number of grid directions
in	*logScaleFlag*	'1' log(pmap), '0' pmap.
out	*pmap*	Resulting MinNorm pseudo-spectrum; nGrid_dirs x 1

Definition at line 1290 of file saf_sh.c.

6.15.2.22 generateMUSICmap()

```
void generateMUSICmap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nSources,
    int nGrid_dirs,
    int logScaleFlag,
    float * pmap )
```

Generates an activity-map based on the sub-space multiple-signal classification (MUSIC) method.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nSources*	Number of sources present in sound scene

Parameters

in	*nGrid_dirs*	Number of grid directions
in	*logScaleFlag*	'1' log(pmap), '0' pmap.
out	*pmap*	Resulting MUSIC pseudo-spectrum; nGrid_dirs x 1

Definition at line 1241 of file saf_sh.c.

6.15.2.23 generateMVDRmap()

```
void generateMVDRmap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nGrid_dirs,
    float regPar,
    float * pmap,
    float_complex * w_MVDR )
```

Generates a powermap based on the energy of adaptive minimum variance distortion-less response (MVDR) beamformers.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nGrid_dirs*	Number of grid directions
in	*regPar*	Regularisation parameter, for diagonal loading of Cx
out	*pmap*	Resulting MVDR powermap; nGrid_dirs x 1
out	*w_MVDR*	(Optional) weights will be copied to this, unless it's NULL; FLAT: nSH x nGrid_dirs || NULL

Definition at line 1070 of file saf_sh.c.

6.15.2.24 generatePWDmap()

```
void generatePWDmap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nGrid_dirs,
    float * pmap )
```

Generates a powermap based on the energy of plane-wave decomposition (PWD)/ hyper-cardioid beamformers.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nGrid_dirs*	Number of grid directions
out	*pmap*	Resulting PWD powermap; nGrid_dirs x 1

Definition at line 1028 of file saf_sh.c.

6.15.2.25 getSHcomplex()

```
void getSHcomplex (
    int order,
    float * dirs_rad,
    int nDirs,
    float_complex * Y )
```

Computes COMPLEX spherical harmonics [1] for each direction on the sphere.

The real spherical harmonics are computed WITH the 1/sqrt(4∗pi) term. i.e. max(omni) = 1/sqrt(4∗pi) + i0. This function employs unnorm_legendreP() and double precision.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITH the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 416 of file saf_sh.c.

6.15.2.26 getSHreal()

```
void getSHreal (
    int order,
    float * dirs_rad,
    int nDirs,
    float * Y )
```

Computes REAL spherical harmonics [1] for each direction on the sphere.

The real spherical harmonics are computed WITH the 1/sqrt(4∗pi) term. i.e. max(omni) = 1/sqrt(4∗pi). Compared to getSHreal_recur(), this function employs unnorm_legendreP() and double precision, which is slower but more precise.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITH the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 292 of file saf_sh.c.

6.15.2.27 getSHreal_recur()

```
void getSHreal_recur (
    int order,
    float * dirs_rad,
    int nDirs,
    float * Y )
```

Computes REAL spherical harmonics [1] for each direction on the sphere.

The real spherical harmonics are computed WITH the 1/sqrt(4∗pi) term. i.e. max(omni) = 1/sqrt(4∗pi). Compared to getSHreal(), this function employs unnorm_legendreP_recur() and single precision, which is faster but less precise.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITH the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 354 of file saf_sh.c.

6.15.2.28 getSHrotMtxReal()

```
void getSHrotMtxReal (
    float R[3][3],
    float * RotMtx,
    int L )
```

Generates a real-valued spherical harmonic rotation matrix [1] (assumes ACN channel ordering convention)

Note that the normalisation convention does not matter, as e.g. only dipoles are used to rotated dipoles, quadrapoles to rotate quadrapoles etc.

Parameters

in	*R*	zyx rotation matrix; 3 x 3
in	*L*	Order of spherical harmonic expansion
out	*RotMtx*	SH domain rotation matrix; FLAT: (L+1)^2 x (L+1)^2

See also

[1] Ivanic, J., Ruedenberg, K. (1998). Rotation Matrices for Real Spherical Harmonics. Direct Determination by Recursion Page: Additions and Corrections. Journal of Physical Chemistry A, 102(45), 9099?9100.

Definition at line 562 of file saf_sh.c.

6.15.2.29 hankel_Hn1()

```
void hankel_Hn1 (
    int N,
    double * z,
    int nZ,
    double_complex * Hn1_n,
    double_complex * dHn1_n )
```

Computes the (cylindrical) Hankel function of the first kind: Hn1.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*Hn1_n*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dHn1_n*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1408 of file saf_sh.c.

6.15.2.30 hankel_hn1()

```
void hankel_hn1 (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double_complex * h_n1,
    double_complex * dh_n1 )
```

Computes the spherical Hankel function of the first kind: hn1.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*h_n1*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dh_n1*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1678 of file saf_sh.c.

6.15.2.31 hankel_Hn2()

```
void hankel_Hn2 (
    int N,
    double * z,
    int nZ,
    double_complex * Hn2_n,
    double_complex * dHn2_n )
```

Computes the (cylindrical) Hankel function of the second kind: Hn2.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*Hn2_n*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dHn2_n*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1437 of file saf_sh.c.

6.15.2.32 hankel_hn2()

```
void hankel_hn2 (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double_complex * h_n2,
    double_complex * dh_n2 )
```

Computes the spherical Hankel function of the second kind: hn2.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*h_n2*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dh_n2*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1736 of file saf_sh.c.

6.15.2.33 real2complexSHMtx()

```
void real2complexSHMtx (
    int order,
    float_complex * T_r2c )
```

Computes a real to complex spherical harmonic transform matrix.

Computes the unitary transformation matrix T_r2c the expresses the complex spherical harmonics with respect to the real ones, so that y_N = T_r2c ∗ r_N, where r_N and y_N are the real and complex SH vectors, respectively.

Parameters

in	*order*	Order of spherical harmonic expansion
out	*T_r2c*	Transformation matrix for real->complex; FLAT: (order+1)^2 x (order+1)^2

Definition at line 499 of file saf_sh.c.

6.15.2.34 rotateAxisCoeffsComplex()

```
void rotateAxisCoeffsComplex (
    int order,
    float * c_n,
    float theta_0,
    float phi_0,
    float_complex * c_nm )
```

Generates spherical coefficients for a rotated axisymmetric pattern (COMPLEX)

Parameters

in	*order*	Order of spherical harmonic expansion
in	*c_n*	Coefficients describing a rotationally symmetric pattern order N, expressed as a sum of spherical harmonics of degree m=0; (N+1) x 1
in	*theta←↩ _0*	POLAR rotation for the pattern, in RADIANS
in	*phi_0*	Azimuthal rotation for the pattern, in RADIANS
out	*c_nm*	Coefficients of rotated pattern expressed as a sum of SHs; (N+1)^2 x 1

Definition at line 932 of file saf_sh.c.

6.15.2.35 rotateAxisCoeffsReal()

```
void rotateAxisCoeffsReal (
    int order,
    float * c_n,
    float theta_0,
    float phi_0,
    float * c_nm )
```

Generates spherical coefficients for a rotated axisymmetric pattern (REAL)

Parameters

in	*order*	Order of spherical harmonic expansion
in	*c_n*	Coefficients describing a rotationally symmetric pattern order N, expressed as a sum of spherical harmonics of degree m=0; (N+1) x 1
in	*theta↩_0*	POLAR rotation for the pattern, in RADIANS
in	*phi_0*	Azimuthal rotation for the pattern, in RADIANS
out	*c_nm*	Coefficients of rotated pattern expressed as a sum of SHs; (N+1)^2 x 1

Definition at line 912 of file saf_sh.c.

6.15.2.36 simulateCylArray()

```
void simulateCylArray (
    int order,
    double * kr,
    int nBands,
    float * sensor_dirs_rad,
    int N_sensors,
    float * src_dirs_deg,
    int N_srcs,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    float_complex * H_array )
```

Simulates a cylindrical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the cylinder.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*sensor_dirs_rad*	Spherical coords of the sensors in RADIANS, [azi ELEV]; FLAT: N_sensors x 2
in	*N_sensors*	Number of sensors
in	*src_dirs_deg*	Spherical coords of the plane waves in DEGREES, [azi ELEV]; FLAT: N_srcs x 2
in	*N_srcs*	Number sources (DoAs of plane waves)
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
out	*H_array*	Simulated array response for each plane wave; FLAT: nBands x N_sensors x N_srcs

Definition at line 2175 of file saf_sh.c.

6.15.2.37 simulateSphArray()

```
void simulateSphArray (
    int order,
    double * kr,
    double * kR,
    int nBands,
    float * sensor_dirs_rad,
```

```
int N_sensors,
float * src_dirs_deg,
int N_srcs,
ARRAY_CONSTRUCTION_TYPES arrayType,
double dirCoeff,
float_complex * H_array )
```

Simulates a spherical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the sphere.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗array_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius, set to NULL if not needed
in	*nBands*	Number of frequency bands/bins
in	*sensor_dirs_rad*	Spherical coords of the sensors in RADIANS, [azi ELEV]; FLAT: N_sensors x 2
in	*N_sensors*	Number of sensors
in	*src_dirs_deg*	Spherical coords of the plane waves in DEGREES, [azi ELEV]; FLAT: N_srcs x 2
in	*N_srcs*	Number sources (DoAs of plane waves)
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
out	*H_array*	Simulated array response for each plane wave; FLAT: nBands x N_sensors x N_srcs

Definition at line 2227 of file saf_sh.c.

6.15.2.38 sphArrayAliasLim()

```
float sphArrayAliasLim (
float r,
float c,
int maxN )
```

Returns a simple estimate of the spatial aliasing limit (the kR = maxN rule)

Parameters

in	*r*	Array radius, meters
in	*c*	Speed of sound, m/s
in	*maxN*	Order

Returns

Spatial aliasing limit estimate, in Hz

Definition at line 1859 of file saf_sh.c.

6.15.2.39 sphArrayNoiseThreshold()

```
void sphArrayNoiseThreshold (
    int maxN,
    int Nsensors,
    float r,
    float c,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    float maxG_db,
    float * f_lim )
```

Computes the frequncies (per order), at which the noise of a SHT of a SMA exceeds a specified maximum level.

Computes the frequencies that the noise in the output channels of a spherical microphone array (SMA), after performing the spherical harmonic transform (SHT) and equalisation of the output signals, reaches a certain user-defined threshold maxG_db. The frequencies are computed only at the lower range of each order, where its response decays rapidly, ignoring for example the nulls of an open array at the higher frequencies. The estimation of the limits are based on a linear approximation of the log-log response found e.g. in [1]

Parameters

in	*maxN*	Maximum order of the array
in	*Nsensors*	Number of sensors
in	*r*	Mic radius, meters
in	*c*	Speed of sound, m/s
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
in	*maxG_db*	Max allowed amplification for the noise level, maxG_db = 20∗log10(maxG)
out	*f_lim*	Noise limit estimate; (maxN+1) x 1

See also

[1] Politis, A., Vilkamo, J., & Pulkki, V. (2015). Sector-based parametric sound field reproduction in the spherical harmonic domain. IEEE Journal of Selected Topics in Signal Processing, 9(5), 852-866.

Definition at line 1869 of file saf_sh.c.

6.15.2.40 sphDiffCohMtxTheory()

```
void sphDiffCohMtxTheory (
    int order,
    float * sensor_dirs_rad,
    int N_sensors,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    double * kr,
    double * kR,
    int nBands,
    double * M_diffcoh )
```

Calculates the theoretical diffuse coherence matrix for a spherical array.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)

Parameters

in	*sensor_dirs_rad*	Spherical coords of the sensors in RADIANS, [azi ELEV]; FLAT: N_sensors x 2
in	*N_sensors*	Number of sensors
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
in	*kr*	wavenumber∗sensor_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius, set to NULL if not applicable; nBands x 1
in	*nBands*	Number of frequency bands/bins
out	*M_diffcoh*	Theoretical diffuse coherence matrix per frequency; FLAT: N_sensors x N_sensors x nBands

Definition at line 2097 of file saf_sh.c.

6.15.2.41 sphModalCoeffs()

```
void sphModalCoeffs (
    int order,
    double * kr,
    int nBands,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    double_complex * b_N )
```

Calculates the modal coefficients for open/rigid spherical arrays.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 1897 of file saf_sh.c.

6.15.2.42 sphScattererDirModalCoeffs()

```
void sphScattererDirModalCoeffs (
    int order,
    double * kr,
    double * kR,
    int nBands,
    double dirCoeff,
    double_complex * b_N )
```

Calculates the modal coefficients for a rigid spherical scatterer with directional sensors.

Assumes all sensors are placed the same distance from the scatterer, w.r.t. the origin

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗array_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*dirCoeff*	Directivity coefficient, 1: omni, 0.5: card, 0:dipole
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 2030 of file saf_sh.c.

6.15.2.43 sphScattererModalCoeffs()

```
void sphScattererModalCoeffs (
    int order,
    double * kr,
    double * kR,
    int nBands,
    double_complex * b_N )
```

Calculates the modal coefficients for a rigid spherical scatterer with omni-directional sensors.

Assumes all sensors are placed the same distance from the scatterer, w.r.t. the origin

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗array_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 1981 of file saf_sh.c.

6.15.2.44 unitCart2Sph()

```
void unitCart2Sph (
    float xyz[3],
    float AziElev_rad[2] )
```

Converts cartesian coordinates of unit length to spherical coordinates.

Parameters

in	*xyz*	Unit cartesian coords, xyz
out	*AziElev_rad*	Azimuth and elevation in radians

Definition at line 137 of file saf_sh.c.

6.15.2.45 unitCart2Sph_aziElev()

```
void unitCart2Sph_aziElev (
    float xyz[3],
    float * azi_rad,
    float * elev_rad )
```

Converts cartesian coordinates of unit length to spherical coordinates.

Parameters

in	*xyz*	Unit cartesian coords, xyz
out	*azi_rad*	(&) azimuth in radians
out	*elev_rad*	(&) elevation in radians

Definition at line 148 of file saf_sh.c.

6.15.2.46 unitSph2Cart()

```
void unitSph2Cart (
    float azi_rad,
    float elev_rad,
    float xyz[3] )
```

Converts spherical coordinates to cartesian coordinates of unit length.

Parameters

in	*azi_rad*	Azimuth in radians
in	*elev_rad*	Elevation in radians
out	*xyz*	Unit cartesian coords, xyz; 3 x 1

Definition at line 125 of file saf_sh.c.

6.15.2.47 unnorm_legendreP()

```
void unnorm_legendreP (
    int n,
    double * x,
    int lenX,
    double * y )
```

Calculates unnormalised legendre polynomials up to order N, for all values in vector x [1].

Note

This INCLUDES the Condon-Shortley phase term. It is functionally identical to Matlab's legendre function (with default settings ['unnorm']).

Parameters

in	*n*	Order of legendre polynomial
in	*x*	Vector of input values; lenX x 1
in	*lenX*	Number of input values
out	*y*	Resulting unnormalised legendre values for each x value; FLAT: (n+1) x lenX

See also

[1] M, Abramowitz., I.A. Stegun. (1965). "Handbook of Mathematical Functions: Chapter 8", Dover Publications.

Definition at line 160 of file saf_sh.c.

6.15.2.48 unnorm_legendreP_recur()

```
void unnorm_legendreP_recur (
    int n,
    float * x,
    int lenX,
    float * Pnm_minus1,
    float * Pnm_minus2,
    float * Pnm )
```

Calculates unnormalised legendre polynomials up to order N, for all values in vector x.

It uses a recursive approach, which makes it more suitable for computing the legendre values in a real-time loop.

Note

This does NOT INCLUDE the Condon-Shortley phase term.

Parameters

in	*n*	Order of legendre polynomial
in	*x*	Vector of input values; lenX x 1
in	*lenX*	Number of input values
in	*Pnm_minus1*	Previous Pnm, (not used for n=1); FLAT: (n+1) x lenX
in	*Pnm_minus2*	Previous previous Pnm, (not used for n=0); FLAT: (n+1) x lenX
out	*Pnm*	Resulting unnormalised legendre values for each x value; FLAT: (n+1) x lenX

Definition at line 236 of file saf_sh.c.

6.15.2.49 yawPitchRoll2Rzyx()

```
void yawPitchRoll2Rzyx (
    float yaw,
    float pitch,
    float roll,
    int rollPitchYawFLAG,
    float R[3][3] )
```

Constructs a 3x3 rotation matrix from the Euler angles, using the yaw-pitch-roll (zyx) convention.

Parameters

in	*yaw*	Yaw angle in radians
in	*pitch*	Pitch angle in radians
in	*roll*	Roll angle in radians
in	*rollPitchYawFLAG*	'1' to use Rxyz, i.e. apply roll, pitch and then yaw, '0' Rzyx / y-p-r
out	*R*	zyx rotation matrix; 3 x 3

Definition at line 69 of file saf_sh.c.

6.15.3 Variable Documentation

6.15.3.1 wxyzCoeffs `const float wxyzCoeffs[4][4]`

Initial value:
```
= { {3.544907701811032f, 0.0f, 0.0f, 0.0f},
    {0.0f, 0.0f, 0.0f, 2.046653415892977f},
    {0.0f, 2.046653415892977f, 0.0f, 0.0f},
    {0.0f, 0.0f, 2.046653415892977f, 0.0f} }
```

First-order ACN/N3D to WXYZ conversion matrix.

Definition at line 39 of file saf_sh.c.

6.16 framework/modules/saf_sh/saf_sh.h File Reference

Public part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

`#include "../saf_utilities/saf_complex.h"`

Macros

- #define **ORDER2NSH**(order) ((order+1)∗(order+1))

Enumerations

- enum _ARRAY_CONSTRUCTION_TYPES { ARRAY_CONSTRUCTION_OPEN, ARRAY_CONSTRUCTION_OPEN_DIRECTI
ARRAY_CONSTRUCTION_RIGID, ARRAY_CONSTRUCTION_RIGID_DIRECTIONAL }

  Microphone/Hydrophone array contruction types.
- enum _SECTOR_PATTERNS { SECTOR_PATTERN_PWD, SECTOR_PATTERN_MAXRE, SECTOR_PATTERN_CARDIOID
}

  Sector pattern designs for directionally-contraining sound-fields [1].

Functions

- void yawPitchRoll2Rzyx (float yaw, float pitch, float roll, int rollPitchYawFLAG, float R[3][3])

  Constructs a 3x3 rotation matrix from the Euler angles, using the yaw-pitch-roll (zyx) convention.
- void unitSph2Cart (float azi_rad, float elev_rad, float xyz[3])

  Converts spherical coordinates to cartesian coordinates of unit length.
- void unitCart2Sph (float xyz[3], float AziElev_rad[2])

  Converts cartesian coordinates of unit length to spherical coordinates.
- void unitCart2Sph_aziElev (float xyz[3], float ∗azi_rad, float ∗elev_rad)

  Converts cartesian coordinates of unit length to spherical coordinates.
- void unnorm_legendreP (int n, double ∗x, int lenX, double ∗y)

  Calculates unnormalised legendre polynomials up to order N, for all values in vector x [1].
- void unnorm_legendreP_recur (int n, float ∗x, int lenX, float ∗Pnm_minus1, float ∗Pnm_minus2, float ∗Pnm)

  Calculates unnormalised legendre polynomials up to order N, for all values in vector x.
- void getSHreal (int order, float ∗dirs_rad, int nDirs, float ∗Y)

  Computes REAL spherical harmonics [1] for each direction on the sphere.
- void getSHreal_recur (int order, float ∗dirs_rad, int nDirs, float ∗Y)

  Computes REAL spherical harmonics [1] for each direction on the sphere.
- void getSHcomplex (int order, float ∗dirs_rad, int nDirs, float_complex ∗Y)

  Computes COMPLEX spherical harmonics [1] for each direction on the sphere.
- void complex2realSHMtx (int order, float_complex ∗T_c2r)

  Computes a complex to real spherical harmonic transform matrix.
- void real2complexSHMtx (int order, float_complex ∗T_r2c)

  Computes a real to complex spherical harmonic transform matrix.
- void complex2realCoeffs (int order, float_complex ∗C_N, int K, float ∗R_N)

  Converts SH coefficients from the complex to real basis.
- void getSHrotMtxReal (float R[3][3], float ∗RotMtx, int L)

  Generates a real-valued spherical harmonic rotation matrix [1] (assumes ACN channel ordering convention)
- void computeVelCoeffsMtx (int sectorOrder, float_complex ∗A_xyz)

  Computes the matrices that generate the coefficients of the beampattern of order (sectorOrder+1) that is essentially the product of a pattern of order=sectorOrder and a dipole.
- float computeSectorCoeffsEP (int orderSec, float_complex ∗A_xyz, SECTOR_PATTERNS pattern, float ∗sec_dirs_deg, int nSecDirs, float ∗sectorCoeffs)

  Computes the beamforming matrices of sector and velocity coefficients for ENERGY-preserving (EP) sectors for real SH.
- float computeSectorCoeffsAP (int orderSec, float_complex ∗A_xyz, SECTOR_PATTERNS pattern, float ∗sec_dirs_deg, int nSecDirs, float ∗sectorCoeffs)

  Computes the beamforming matrices of sector and velocity coefficients for AMPLITUDE-preserving (AP) sectors for real SH.
- void beamWeightsCardioid2Spherical (int N, float ∗b_n)

  Generates spherical coefficients for cardioids.
- void beamWeightsDolphChebyshev2Spherical (int N, int paramType, float arrayParam, float ∗b_n)

  Generates beamweights in the SHD for Dolph-Chebyshev beampatterns, with mainlobe and sidelobe control [1].
- void beamWeightsHypercardioid2Spherical (int N, float ∗b_n)

  Generates beamweights in the SHD for hypercardioid beampatterns.
- void beamWeightsMaxEV (int N, float ∗b_n)

  Generates beamweights in the SHD for maximum energy-vector beampatterns.
- void beamWeightsVelocityPatternsReal (int order, float ∗b_n, float azi_rad, float elev_rad, float_complex ∗A←_xyz, float ∗velCoeffs)

  Generates beamforming coefficients for velocity patterns (REAL)
- void beamWeightsVelocityPatternsComplex (int order, float ∗b_n, float azi_rad, float elev_rad, float_complex ∗A_xyz, float_complex ∗velCoeffs)

Generates beamforming coefficients for velocity patterns (COMPLEX)
- void rotateAxisCoeffsReal (int order, float ∗c_n, float theta_0, float phi_0, float ∗c_nm)

  Generates spherical coefficients for a rotated axisymmetric pattern (REAL)
- void rotateAxisCoeffsComplex (int order, float ∗c_n, float theta_0, float phi_0, float_complex ∗c_nm)

  Generates spherical coefficients for a rotated axisymmetric pattern (COMPLEX)
- void checkCondNumberSHTReal (int order, float ∗dirs_rad, int nDirs, float ∗w, float ∗cond_N)

  Computes the condition numbers for a least-squares SHT.
- void generatePWDmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nGrid_dirs, float ∗pmap)

  Generates a powermap based on the energy of plane-wave decomposition (PWD)/ hyper-cardioid beamformers.
- void generateMVDRmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nGrid_dirs, float regPar, float ∗pmap, float_complex ∗w_MVDR)

  Generates a powermap based on the energy of adaptive minimum variance distortion-less response (MVDR) beamformers.
- void generateCroPaCLCMVmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nGrid_dirs, float regPar, float lambda, float ∗pmap)

  (EXPERIMENTAL) Generates a powermap utilising the CroPaC LCMV post-filter described in [1]
- void generateMUSICmap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nSources, int nGrid_dirs, int logScaleFlag, float ∗pmap)

  Generates an activity-map based on the sub-space multiple-signal classification (MUSIC) method.
- void generateMinNormMap (int order, float_complex ∗Cx, float_complex ∗Y_grid, int nSources, int nGrid_dirs, int logScaleFlag, float ∗pmap)

  Generates an activity-map based on the sub-space minimum-norm (MinNorm) method.
- void bessel_Jn (int N, double ∗z, int nZ, double ∗J_n, double ∗dJ_n)

  Computes the (cylindrical) Bessel function of the first kind: Jn.
- void bessel_Yn (int N, double ∗z, int nZ, double ∗Y_n, double ∗dY_n)

  Computes the (cylindrical) Bessel function of the second kind: Yn.
- void hankel_Hn1 (int N, double ∗z, int nZ, double_complex ∗Hn1_n, double_complex ∗dHn1_n)

  Computes the (cylindrical) Hankel function of the first kind: Hn1.
- void hankel_Hn2 (int N, double ∗z, int nZ, double_complex ∗Hn2_n, double_complex ∗dHn2_n)

  Computes the (cylindrical) Hankel function of the second kind: Hn2.
- void bessel_jn (int N, double ∗z, int nZ, int ∗maxN, double ∗j_n, double ∗dj_n)

  Computes the spherical Bessel function of the first kind: jn.
- void bessel_in (int N, double ∗z, int nZ, int ∗maxN, double ∗i_n, double ∗di_n)

  Computes the modified spherical Bessel function of the first kind: in.
- void bessel_yn (int N, double ∗z, int nZ, int ∗maxN, double ∗y_n, double ∗dy_n)

  Computes the spherical Bessel function of the second kind (Neumann): yn.
- void bessel_kn (int N, double ∗z, int nZ, int ∗maxN, double ∗k_n, double ∗dk_n)

  Computes the modified spherical Bessel function of the second kind: kn.
- void hankel_hn1 (int N, double ∗z, int nZ, int ∗maxN, double_complex ∗h_n1, double_complex ∗dh_n1)

  Computes the spherical Hankel function of the first kind: hn1.
- void hankel_hn2 (int N, double ∗z, int nZ, int ∗maxN, double_complex ∗h_n2, double_complex ∗dh_n2)

  Computes the spherical Hankel function of the second kind: hn2.
- void cylModalCoeffs (int order, double ∗kr, int nBands, ARRAY_CONSTRUCTION_TYPES arrayType, double_complex ∗b_N)

  Calculates the modal coefficients for open/rigid cylindrical arrays.
- float sphArrayAliasLim (float r, float c, int maxN)

  Returns a simple estimate of the spatial aliasing limit (the kR = maxN rule)
- void sphArrayNoiseThreshold (int maxN, int Nsensors, float r, float c, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, float maxG_db, float ∗f_lim)

  Computes the frequncies (per order), at which the noise of a SHT of a SMA exceeds a specified maximum level.
- void sphModalCoeffs (int order, double ∗kr, int nBands, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, double_complex ∗b_N)

Calculates the modal coefficients for open/rigid spherical arrays.

- void sphScattererModalCoeffs (int order, double ∗kr, double ∗kR, int nBands, double_complex ∗b_N)

  Calculates the modal coefficients for a rigid spherical scatterer with omni-directional sensors.
- void sphScattererDirModalCoeffs (int order, double ∗kr, double ∗kR, int nBands, double dirCoeff, double_complex ∗b_N)

  Calculates the modal coefficients for a rigid spherical scatterer with directional sensors.
- void sphDiffCohMtxTheory (int order, float ∗sensor_dirs_rad, int N_sensors, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, double ∗kr, double ∗kR, int nBands, double ∗M_diffcoh)

  Calculates the theoretical diffuse coherence matrix for a spherical array.
- void simulateCylArray (int order, double ∗kr, int nBands, float ∗sensor_dirs_rad, int N_sensors, float ∗src_dirs_deg, int N_srcs, ARRAY_CONSTRUCTION_TYPES arrayType, float_complex ∗H_array)

  Simulates a cylindrical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the cylinder.
- void simulateSphArray (int order, double ∗kr, double ∗kR, int nBands, float ∗sensor_dirs_rad, int N_sensors, float ∗src_dirs_deg, int N_srcs, ARRAY_CONSTRUCTION_TYPES arrayType, double dirCoeff, float_complex ∗H_array)

  Simulates a spherical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the sphere.
- void evaluateSHTfilters (int order, float_complex ∗M_array2SH, int nSensors, int nBands, float_complex ∗H_array, int nDirs, float_complex ∗Y_grid, float ∗cSH, float ∗lSH)

  Generates some objective measures, which evaluate the performance of the spatial encoding filters.

6.16.1 Detailed Description

Public part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

A collection of spherical harmonic related functions. Many of which have been derived from Matlab libraries by Archontis Politis [1-3].

See also

[1] https://github.com/polarch/Spherical-Harmonic-Transform

[2] https://github.com/polarch/Array-Response-Simulator

[3] https://github.com/polarch/Spherical-Array-Processing

Author

Leo McCormack

Date

22.05.2016

6.16.2 Enumeration Type Documentation

6.16.2.1 _ARRAY_CONSTRUCTION_TYPES `enum _ARRAY_CONSTRUCTION_TYPES`

Microphone/Hydrophone array contruction types.

Enumerator

ARRAY_CONSTRUCTION_OPEN	Open array, omni-directional sensors.
ARRAY_CONSTRUCTION_OPEN_DIRECTIONAL	Open array, directional sensors.
ARRAY_CONSTRUCTION_RIGID	Rigid baffle, omni-directional sensors.
ARRAY_CONSTRUCTION_RIGID_DIRECTIONAL	Rigid baffle, directional sensors.

Definition at line 51 of file saf_sh.h.

6.16.2.2 _SECTOR_PATTERNS `enum _SECTOR_PATTERNS`

Sector pattern designs for directionally-contraining sound-fields [1].

See also

[1] Politis, A., & Pulkki, V. (2016). Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409

Enumerator

SECTOR_PATTERN_PWD	Plane-wave decomposition/Hyper-cardioid.
SECTOR_PATTERN_MAXRE	Spatially tapered hyper-cardioid, such that it has maximum energy concentrated in the look- direction.
SECTOR_PATTERN_CARDIOID	Cardioid pattern.

Definition at line 68 of file saf_sh.h.

6.16.3 Function Documentation

6.16.3.1 beamWeightsCardioid2Spherical()

```
void beamWeightsCardioid2Spherical (
        int N,
        float * b_n )
```

Generates spherical coefficients for cardioids.

For a specific order N of a higher order cardioid of the form D(theta)=(1/2)^N * (1+cos(theta))^N, generate the beamweights for the same pattern in the SHD. Because the pattern is axisymmetric only the N+1 coefficients of m=0 are returned.

Parameters

in	*N*	Order of spherical harmonic expansion
out	*b_n*	Beamformer weights; (N+1) x 1

Definition at line 789 of file saf_sh.c.

6.16.3.2 beamWeightsDolphChebyshev2Spherical()

```
void beamWeightsDolphChebyshev2Spherical (
    int N,
    int paramType,
    float arrayParam,
    float * b_n )
```

Generates beamweights in the SHD for Dolph-Chebyshev beampatterns, with mainlobe and sidelobe control [1].

Because the pattern is axisymmetric only the N+1 coefficients of m=0 are returned.

Note

NOT IMPLEMENTED YET

Parameters

in	*N*	Order of spherical harmonic expansion
in	*paramType*	'0' side-lobe level control, '1' mainlobe width control
in	*arrayParam*	Sidelobe level 1/R or mainlobe with 2∗a0
out	*b_n*	Beamformer weights; (N+1) x 1

See also

[1] Koretz, A. and Rafaely, B., 2009. Dolph-Chebyshev beampattern design for spherical arrays. IEEE Transactions on Signal Processing, 57(6), pp.2417-2420.

6.16.3.3 beamWeightsHypercardioid2Spherical()

```
void beamWeightsHypercardioid2Spherical (
    int N,
    float * b_n )
```

Generates beamweights in the SHD for hypercardioid beampatterns.

The hypercardioid is the pattern that maximises the directivity-factor for a certain SH order N. The hypercardioid is also the plane-wave decomposition beamformer in the SHD, also called 'regular' because the beamweights are just the SH values on the beam-direction. Since the pattern is axisymmetric only the N+1 coefficients of m=0 are returned.

Parameters

in	*N*	Order of spherical harmonic expansion
out	*b_n*	Beamformer weights; (N+1) x 1

Definition at line 806 of file saf_sh.c.

6.16.3.4 beamWeightsMaxEV()

```
void beamWeightsMaxEV (
    int N,
    float * b_n )
```

Generates beamweights in the SHD for maximum energy-vector beampatterns.

Generate the beamweights for the a maximum energy-vector beampattern in the SHD. This pattern originates from ambisonic-related research and it maximises the ambisonic energy-vector, which is essentially the directional centroid of the squared pattern. IT can also be seen as the pattern that maximizes the acoustic intensity vector of a diffuse field weighted with this pattern. In practice it is almost the same as a supercardioid that maximizes front-back power ratio for a certain order, and it can be used as such. Because the pattern is axisymmetric only the N+1 coefficients of m=0 are returned. Details for their theory can be found e.g. in [1].

Parameters

in	*N*	Order of spherical harmonic expansion
out	*b_n*	Beamformer weights; (N+1) x 1

See also

[1] Zotter, F., Pomberger, H. and Noisternig, M., 2012. Energy- preserving ambisonic decoding. Acta Acustica united with Acustica, 98(1), pp.37-47.

Definition at line 824 of file saf_sh.c.

6.16.3.5 beamWeightsVelocityPatternsComplex()

```
void beamWeightsVelocityPatternsComplex (
    int order,
    float * b_n,
    float azi_rad,
    float elev_rad,
    float_complex * A_xyz,
    float_complex * velCoeffs )
```

Generates beamforming coefficients for velocity patterns (COMPLEX)

If the sound-field is weighted with an axisymmetric spatial distribution described by the N+1 SH coefficients b_n, then the beamweights capturing the velocity signals for the weighted sound-field are of an order one higher than the weighting pattern, and can be derived from it. This type of beamforming has some applications for spatial sound reproduction and acoustic analysis, see [1].

Parameters

in	*order*	Order of spherical harmonic expansion
in	*b_n*	Axisymmetric beamformer weights; (order+1) x 1
in	*azi_rad*	Orientation, azimuth in RADIANS
in	*elev_rad*	Orientation, ELEVATION in RADIANS
in	*A_xyz*	Velocity coefficients; see computeVelCoeffsMtx(); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
out	*velCoeffs*	Beamforming coefficients for velocity patterns; FLAT: (order+2)^2 x 3

See also

[1] Politis, A. and Pulkki, V., 2016. Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409.

Definition at line 872 of file saf_sh.c.

6.16.3.6 beamWeightsVelocityPatternsReal()

```
void beamWeightsVelocityPatternsReal (
    int order,
    float * b_n,
    float azi_rad,
    float elev_rad,
    float_complex * A_xyz,
    float * velCoeffs )
```

Generates beamforming coefficients for velocity patterns (REAL)

If the sound-field is weighted with an axisymmetric spatial distribution described by the N+1 SH coefficients b_n, then the beamweights capturing the velocity signals for the weighted sound-field are of an order one higher than the weighting pattern, and can be derived from it. This type of beamforming has some applications for spatial sound reproduction and acoustic analysis, see [1].

Parameters

in	*order*	Order of spherical harmonic expansion
in	*b_n*	Axisymmetric beamformer weights; (order+1) x 1
in	*azi_rad*	Orientation, azimuth in RADIANS
in	*elev_rad*	Orientation, ELEVATION in RADIANS
in	*A_xyz*	Velocity coefficients; see computeVelCoeffsMtx(); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
out	*velCoeffs*	Beamforming coefficients for velocity patterns; FLAT: (order+2)^2 x 3

See also

[1] Politis, A. and Pulkki, V., 2016. Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409.

Definition at line 851 of file saf_sh.c.

6.16.3.7 bessel_in()

```
void bessel_in (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * i_n,
    double * di_n )
```

Computes the modified spherical Bessel function of the first kind: in.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*i_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*di_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1523 of file saf_sh.c.

6.16.3.8 bessel_Jn()

```
void bessel_Jn (
    int N,
    double * z,
    int nZ,
    double * J_n,
    double * dJ_n )
```

Computes the (cylindrical) Bessel function of the first kind: Jn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*J_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dJ_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1346 of file saf_sh.c.

6.16.3.9 bessel_jn()

```
void bessel_jn (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * j_n,
    double * dj_n )
```

Computes the spherical Bessel function of the first kind: jn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*j_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dj_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1468 of file saf_sh.c.

6.16.3.10 bessel_kn()

```
void bessel_kn (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * k_n,
    double * dk_n )
```

Computes the modified spherical Bessel function of the second kind: kn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*k_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dk_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1628 of file saf_sh.c.

6.16.3.11 bessel_Yn()

```
void bessel_Yn (
    int N,
    double * z,
    int nZ,
    double * Y_n,
    double * dY_n )
```

Computes the (cylindrical) Bessel function of the second kind: Yn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*Y_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dY←_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1377 of file saf_sh.c.

6.16.3.12 bessel_yn()

```
void bessel_yn (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double * y_n,
    double * dy_n )
```

Computes the spherical Bessel function of the second kind (Neumann): yn.

Computes the Bessel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*y_n*	Bessel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dy_n*	Bessel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1578 of file saf_sh.c.

6.16.3.13 checkCondNumberSHTReal()

```
void checkCondNumberSHTReal (
    int order,
    float * dirs_rad,
    int nDirs,
    float * w,
    float * cond_N )
```

Computes the condition numbers for a least-squares SHT.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2

Parameters

in	*nDirs*	Number of directions
in	*w*	Integration weights; nDirs x 1
out	*cond← _N*	Condition numbers; (order+1) x 1

Definition at line 957 of file saf_sh.c.

6.16.3.14 complex2realCoeffs()

```
void complex2realCoeffs (
    int order,
    float_complex * C_N,
    int K,
    float * R_N )
```

Converts SH coefficients from the complex to real basis.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*C_N*	Complex coeffients; FLAT: (order+1)^2 x K
in	*K*	Number of columns
out	*R_N*	Real coefficients; FLAT: (order+1)^2 x K

Definition at line 531 of file saf_sh.c.

6.16.3.15 complex2realSHMtx()

```
void complex2realSHMtx (
    int order,
    float_complex * T_c2r )
```

Computes a complex to real spherical harmonic transform matrix.

Computes the unitary transformation matrix T_c2r. It expresses the real spherical harmonics with respect to the complex ones, so that r_N = T_c2r ∗ y_N, where r_N and y_N is are the real and complex SH vectors, respectively.

Parameters

in	*order*	Order of spherical harmonic expansion
out	*T_c2r*	Transformation matrix for complex->real; FLAT: (order+1)^2 x (order+1)^2

Definition at line 467 of file saf_sh.c.

6.16.3.16 computeSectorCoeffsAP()

```
float computeSectorCoeffsAP (
    int orderSec,
```

```
float_complex * A_xyz,
SECTOR_PATTERNS pattern,
float * sec_dirs_deg,
int nSecDirs,
float * sectorCoeffs )
```

Computes the beamforming matrices of sector and velocity coefficients for AMPLITUDE-preserving (AP) sectors for real SH.

This partitioning of the sound-field into spatially-localised sectors has been used for parametric sound-field reproduction in [1] and visualisation in [2,3].

Parameters

in	*orderSec*	Order of sector patterns
in	*A_xyz*	Velocity coefficients (see "computeVelCoeffsMtx"); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
in	*pattern*	See "SECTOR_PATTERNS" enum for the options
in	*sec_dirs_deg*	Sector directions [azi elev], in DEGREES; FLAT: nSecDirs x 2
in	*nSecDirs*	Number of sectors
out	*sectorCoeffs*	The sector coefficients; FLAT: (nSecDirs*4) x (orderSec+2)^2

Returns

Normalisation coefficient

See also

[1] Politis, A., Vilkamo, J., & Pulkki, V. (2015). Sector-based parametric sound field reproduction in the spherical harmonic domain. IEEE Journal of Selected Topics in Signal Processing, 9(5), 852-866.

[2] McCormack, L., Politis, A., and Pulkki, V. (2019). "Sharpening of angular spectra based on a directional re-assignment approach for ambisonic sound-field visualisation". IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[3] McCormack, L., Delikaris-Manias, S., Politis, A., Pavlidi, D., Farina, A., Pinardi, D. and Pulkki, V., 2019. Applications of Spatially Localized Active-Intensity Vectors for Sound-Field Visualization. Journal of the Audio Engineering Society, 67(11), pp.840-854.

Definition at line 736 of file saf_sh.c.

6.16.3.17 computeSectorCoeffsEP()

```
float computeSectorCoeffsEP (
int orderSec,
float_complex * A_xyz,
SECTOR_PATTERNS pattern,
float * sec_dirs_deg,
int nSecDirs,
float * sectorCoeffs )
```

Computes the beamforming matrices of sector and velocity coefficients for ENERGY-preserving (EP) sectors for real SH.

This partitioning of the sound-field into spatially-localised sectors has been used for parametric sound-field reproduction in [1] and visualisation in [2,3].

Parameters

in	*orderSec*	Order of sector patterns
in	*A_xyz*	Velocity coefficients (see "computeVelCoeffsMtx"); FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3
in	*pattern*	See "SECTOR_PATTERNS" enum for the options
in	*sec_dirs_deg*	Sector directions [azi elev], in DEGREES; FLAT: nSecDirs x 2
in	*nSecDirs*	Number of sectors
out	*sectorCoeffs*	The sector coefficients; FLAT: (nSecDirs*4) x (orderSec+2)^2

Returns

Normalisation coefficient

See also

[1] Politis, A., Vilkamo, J., & Pulkki, V. (2015). Sector-based parametric sound field reproduction in the spherical harmonic domain. IEEE Journal of Selected Topics in Signal Processing, 9(5), 852-866.

[2] McCormack, L., Politis, A., and Pulkki, V. (2019). "Sharpening of angular spectra based on a directional re-assignment approach for ambisonic sound-field visualisation". IEEE International Conference ' on Acoustics, Speech and Signal Processing (ICASSP).

[3] McCormack, L., Delikaris-Manias, S., Politis, A., Pavlidi, D., Farina, A., Pinardi, D. and Pulkki, V., 2019. Applications of Spatially Localized Active-Intensity Vectors for Sound-Field Visualization. Journal of the Audio Engineering Society, 67(11), pp.840-854.

Definition at line 670 of file saf_sh.c.

6.16.3.18 computeVelCoeffsMtx()

```
void computeVelCoeffsMtx (
    int sectorOrder,
    float_complex * A_xyz )
```

Computes the matrices that generate the coefficients of the beampattern of order (sectorOrder+1) that is essentially the product of a pattern of order=sectorOrder and a dipole.

It is used in beamWeightsVelocityPatterns(). For the derivation of the matrices see [1].

Parameters

in	*sectorOrder*	Order of patterns
out	*A_xyz*	Velocity coefficients; FLAT: (sectorOrder+2)^2 x (sectorOrder+1)^2 x 3

See also

[1] Politis, A. and Pulkki, V., 2016. Acoustic intensity, energy-density and diffuseness estimation in a directionally-constrained region. arXiv preprint arXiv:1609.03409

Definition at line 638 of file saf_sh.c.

6.16.3.19 cylModalCoeffs()

```
void cylModalCoeffs (
    int order,
    double * kr,
    int nBands,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double_complex * b_N )
```

Calculates the modal coefficients for open/rigid cylindrical arrays.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 1794 of file saf_sh.c.

6.16.3.20 evaluateSHTfilters()

```
void evaluateSHTfilters (
    int order,
    float_complex * M_array2SH,
    int nSensors,
    int nBands,
    float_complex * H_array,
    int nDirs,
    float_complex * Y_grid,
    float * cSH,
    float * lSH )
```

Generates some objective measures, which evaluate the performance of the spatial encoding filters.

This analysis is performed by comparing the spatial resolution of the spherical harmonic components generated by the encoding filters, with the ideal SH components. For more information, the reader is directed to [1,2].

Parameters

in	*order*	Transform/encoding order
in	*M_array2SH*	Encoding matrix per frequency; FLAT: nBands x (order+1)^2 x nSensors
in	*nSensors*	Number of sensors
in	*nBands*	Number of frequency bands/bins
in	*H_array*	Measured/modelled array responses for many directions; FLAT: nBands x nSensors x nDirs
in	*nDirs*	Number of directions the array was measured/modelled
in	*Y_grid*	Spherical harmonics weights for each grid direction; FLAT: nDirs x (order+1)^2
out	*cSH*	Absolute values of the spatial correlation per band and order; FLAT: nBands x (order+1)
out	*lSH*	Level difference per band and order; FLAT: nBands x (order+1)

See also

[1] Moreau, S., Daniel, J., Bertet, S., 2006, 3D sound field recording with higher order ambisonics–objective measurements and validation of spherical microphone. In Audio Engineering Society Convention 120.

[2] Politis, A., Gamper, H. (2017). "Comparing Modelled And Measurement- Based Spherical Harmonic Encoding Filters For Spherical Microphone Arrays. In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA).

Definition at line 2307 of file saf_sh.c.

6.16.3.21 generateCroPaCLCMVmap()

```
void generateCroPaCLCMVmap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nGrid_dirs,
    float regPar,
    float lambda,
    float * pmap )
```

(EXPERIMENTAL) Generates a powermap utilising the CroPaC LCMV post-filter described in [1]

The spatial post-filter is estimated for all directions on the grid, and is used to supress reverb/noise interference that may be present in an MVDR map. Unlike in the paper, the second column for the contraints 'A', is Y.∗diag(Cx), rather than utilising a maximum energy beamformer. The post- filters are then applied to the MVDR powermap map derived in the sherical harmonic domain, rather than an MVDR beamformer generated directly in the microphone array signal domain, like in the paper. Otherwise, the algorithm is the same.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nGrid_dirs*	Number of grid directions
in	*regPar*	Regularisation parameter, for diagonal loading of Cx
in	*lambda*	Parameter controlling how harsh CroPaC is applied, 0..1; 0: fully CroPaC, 1: fully MVDR
out	*pmap*	Resulting CroPaC LCMV powermap; nGrid_dirs x 1

See also

[1] Delikaris-Manias, S., Vilkamo, J., & Pulkki, V. (2016). Signal- dependent spatial filtering based on weighted-orthogonal beamformers in the spherical harmonic domain. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), 24(9), 1507-1519.

Definition at line 1136 of file saf_sh.c.

6.16.3.22 generateMinNormMap()

```
void generateMinNormMap (
    int order,
    float_complex * Cx,
```

```
float_complex * Y_grid,
int nSources,
int nGrid_dirs,
int logScaleFlag,
float * pmap )
```

Generates an activity-map based on the sub-space minimum-norm (MinNorm) method.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nSources*	Number of sources present in sound scene
in	*nGrid_dirs*	Number of grid directions
in	*logScaleFlag*	'1' log(pmap), '0' pmap.
out	*pmap*	Resulting MinNorm pseudo-spectrum; nGrid_dirs x 1

Definition at line 1290 of file saf_sh.c.

6.16.3.23 generateMUSICmap()

```
void generateMUSICmap (
int order,
float_complex * Cx,
float_complex * Y_grid,
int nSources,
int nGrid_dirs,
int logScaleFlag,
float * pmap )
```

Generates an activity-map based on the sub-space multiple-signal classification (MUSIC) method.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nSources*	Number of sources present in sound scene
in	*nGrid_dirs*	Number of grid directions
in	*logScaleFlag*	'1' log(pmap), '0' pmap.
out	*pmap*	Resulting MUSIC pseudo-spectrum; nGrid_dirs x 1

Definition at line 1241 of file saf_sh.c.

6.16.3.24 generateMVDRmap()

```
void generateMVDRmap (
int order,
float_complex * Cx,
```

```
float_complex * Y_grid,
int nGrid_dirs,
float regPar,
float * pmap,
float_complex * w_MVDR )
```

Generates a powermap based on the energy of adaptive minimum variance distortion-less response (MVDR) beamformers.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nGrid_dirs*	Number of grid directions
in	*regPar*	Regularisation parameter, for diagonal loading of Cx
out	*pmap*	Resulting MVDR powermap; nGrid_dirs x 1
out	*w_MVDR*	(Optional) weights will be copied to this, unless it's NULL; FLAT: nSH x nGrid_dirs || NULL

Definition at line 1070 of file saf_sh.c.

6.16.3.25 generatePWDmap()

```
void generatePWDmap (
    int order,
    float_complex * Cx,
    float_complex * Y_grid,
    int nGrid_dirs,
    float * pmap )
```

Generates a powermap based on the energy of plane-wave decomposition (PWD)/ hyper-cardioid beamformers.

Parameters

in	*order*	Analysis order
in	*Cx*	Correlation/covarience matrix; FLAT: (order+1)^2 x (order+1)^2
in	*Y_grid*	Steering vectors for each grid direcionts; FLAT: (order+1)^2 x nGrid_dirs
in	*nGrid_dirs*	Number of grid directions
out	*pmap*	Resulting PWD powermap; nGrid_dirs x 1

Definition at line 1028 of file saf_sh.c.

6.16.3.26 getSHcomplex()

```
void getSHcomplex (
    int order,
    float * dirs_rad,
    int nDirs,
    float_complex * Y )
```

Computes COMPLEX spherical harmonics [1] for each direction on the sphere.

The real spherical harmonics are computed WITH the 1/sqrt(4*pi) term. i.e. max(omni) = 1/sqrt(4*pi) + i0. This function employs unnorm_legendreP() and double precision.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITH the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 416 of file saf_sh.c.

6.16.3.27 getSHreal()

```
void getSHreal (
    int order,
    float * dirs_rad,
    int nDirs,
    float * Y )
```

Computes REAL spherical harmonics [1] for each direction on the sphere.

The real spherical harmonics are computed WITH the 1/sqrt(4∗pi) term. i.e. max(omni) = 1/sqrt(4∗pi). Compared to getSHreal_recur(), this function employs unnorm_legendreP() and double precision, which is slower but more precise.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITH the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 292 of file saf_sh.c.

6.16.3.28 getSHreal_recur()

```
void getSHreal_recur (
    int order,
    float * dirs_rad,
    int nDirs,
    float * Y )
```

Computes REAL spherical harmonics [1] for each direction on the sphere.

The real spherical harmonics are computed WITH the 1/sqrt(4∗pi) term. i.e. max(omni) = 1/sqrt(4∗pi). Compared to getSHreal(), this function employs unnorm_legendreP_recur() and single precision, which is faster but less precise.

Parameters

in	*order*	Order of spherical harmonic expansion
in	*dirs_rad*	Directions on the sphere [azi, INCLINATION] convention, in RADIANS; FLAT: nDirs x 2
in	*nDirs*	Number of directions
out	*Y*	The SH weights [WITH the 1/sqrt(4∗pi)]; FLAT: (order+1)^2 x nDirs

See also

[1] Rafaely, B. (2015). Fundamentals of spherical array processing (Vol. 8). Berlin: Springer.

Definition at line 354 of file saf_sh.c.

6.16.3.29 getSHrotMtxReal()

```
void getSHrotMtxReal (
    float R[3][3],
    float * RotMtx,
    int L )
```

Generates a real-valued spherical harmonic rotation matrix [1] (assumes ACN channel ordering convention)

Note that the normalisation convention does not matter, as e.g. only dipoles are used to rotated dipoles, quadrapoles to rotate quadrapoles etc.

Parameters

in	*R*	zyx rotation matrix; 3 x 3
in	*L*	Order of spherical harmonic expansion
out	*RotMtx*	SH domain rotation matrix; FLAT: (L+1)^2 x (L+1)^2

See also

[1] Ivanic, J., Ruedenberg, K. (1998). Rotation Matrices for Real Spherical Harmonics. Direct Determination by Recursion Page: Additions and Corrections. Journal of Physical Chemistry A, 102(45), 9099?9100.

Definition at line 562 of file saf_sh.c.

6.16.3.30 hankel_Hn1()

```
void hankel_Hn1 (
    int N,
    double * z,
    int nZ,
    double_complex * Hn1_n,
    double_complex * dHn1_n )
```

Computes the (cylindrical) Hankel function of the first kind: Hn1.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*Hn1_n*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dHn1←↩ _n*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1408 of file saf_sh.c.

6.16.3.31 hankel_hn1()

```
void hankel_hn1 (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double_complex * h_n1,
    double_complex * dh_n1 )
```

Computes the spherical Hankel function of the first kind: hn1.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*h_n1*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dh_n1*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1678 of file saf_sh.c.

6.16.3.32 hankel_Hn2()

```
void hankel_Hn2 (
    int N,
    double * z,
    int nZ,
    double_complex * Hn2_n,
    double_complex * dHn2_n )
```

Computes the (cylindrical) Hankel function of the second kind: Hn2.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*Hn2_n*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dHn2↵ _n*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1437 of file saf_sh.c.

6.16.3.33 hankel_hn2()

```
void hankel_hn2 (
    int N,
    double * z,
    int nZ,
    int * maxN,
    double_complex * h_n2,
    double_complex * dh_n2 )
```

Computes the spherical Hankel function of the second kind: hn2.

Computes the Hankel values and their derivatives up to order N for all values in vector z

Parameters

in	*N*	Function order (highest is ∼30 given numerical precision)
in	*z*	Input values; nZ x 1
in	*nZ*	Number of input values
out	*maxN*	(&) maximum function order that could be computed <=N
out	*h_n2*	Hankel values (set as NULL if not required); FLAT: nZ x (N+1)
out	*dh_n2*	Hankel derivative values (set as NULL if not required); FLAT: nZ x (N+1)

Definition at line 1736 of file saf_sh.c.

6.16.3.34 real2complexSHMtx()

```
void real2complexSHMtx (
    int order,
    float_complex * T_r2c )
```

Computes a real to complex spherical harmonic transform matrix.

Computes the unitary transformation matrix T_r2c the expresses the complex spherical harmonics with respect to the real ones, so that y_N = T_r2c ∗ r_N, where r_N and y_N are the real and complex SH vectors, respectively.

Parameters

in	*order*	Order of spherical harmonic expansion
out	*T_r2c*	Transformation matrix for real->complex; FLAT: (order+1)^2 x (order+1)^2

Definition at line 499 of file saf_sh.c.

6.16.3.35 rotateAxisCoeffsComplex()

```
void rotateAxisCoeffsComplex (
    int order,
    float * c_n,
    float theta_0,
    float phi_0,
    float_complex * c_nm )
```

Generates spherical coefficients for a rotated axisymmetric pattern (COMPLEX)

Parameters

in	*order*	Order of spherical harmonic expansion
in	*c_n*	Coefficients describing a rotationally symmetric pattern order N, expressed as a sum of spherical harmonics of degree m=0; (N+1) x 1
in	*theta_0*	POLAR rotation for the pattern, in RADIANS
in	*phi_0*	Azimuthal rotation for the pattern, in RADIANS
out	*c_nm*	Coefficients of rotated pattern expressed as a sum of SHs; (N+1)^2 x 1

Definition at line 932 of file saf_sh.c.

6.16.3.36 rotateAxisCoeffsReal()

```
void rotateAxisCoeffsReal (
    int order,
    float * c_n,
    float theta_0,
    float phi_0,
    float * c_nm )
```

Generates spherical coefficients for a rotated axisymmetric pattern (REAL)

Parameters

in	*order*	Order of spherical harmonic expansion
in	*c_n*	Coefficients describing a rotationally symmetric pattern order N, expressed as a sum of spherical harmonics of degree m=0; (N+1) x 1
in	*theta_0*	POLAR rotation for the pattern, in RADIANS
in	*phi_0*	Azimuthal rotation for the pattern, in RADIANS
out	*c_nm*	Coefficients of rotated pattern expressed as a sum of SHs; (N+1)^2 x 1

Definition at line 912 of file saf_sh.c.

6.16.3.37 simulateCylArray()

```
void simulateCylArray (
    int order,
    double * kr,
    int nBands,
    float * sensor_dirs_rad,
    int N_sensors,
    float * src_dirs_deg,
    int N_srcs,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    float_complex * H_array )
```

Simulates a cylindrical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the cylinder.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*sensor_dirs_rad*	Spherical coords of the sensors in RADIANS, [azi ELEV]; FLAT: N_sensors x 2
in	*N_sensors*	Number of sensors
in	*src_dirs_deg*	Spherical coords of the plane waves in DEGREES, [azi ELEV]; FLAT: N_srcs x 2
in	*N_srcs*	Number sources (DoAs of plane waves)
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
out	*H_array*	Simulated array response for each plane wave; FLAT: nBands x N_sensors x N_srcs

Definition at line 2175 of file saf_sh.c.

6.16.3.38 simulateSphArray()

```
void simulateSphArray (
    int order,
    double * kr,
    double * kR,
    int nBands,
    float * sensor_dirs_rad,
    int N_sensors,
    float * src_dirs_deg,
    int N_srcs,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    float_complex * H_array )
```

Simulates a spherical microphone array, returning the transfer functions for each (plane wave) source direction on the surface of the sphere.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗array_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius, set to NULL if not needed
in	*nBands*	Number of frequency bands/bins
in	*sensor_dirs_rad*	Spherical coords of the sensors in RADIANS, [azi ELEV]; FLAT: N_sensors x 2

Parameters

in	*N_sensors*	Number of sensors
in	*src_dirs_deg*	Spherical coords of the plane waves in DEGREES, [azi ELEV]; FLAT: N_srcs x 2
in	*N_srcs*	Number sources (DoAs of plane waves)
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
out	*H_array*	Simulated array response for each plane wave; FLAT: nBands x N_sensors x N_srcs

Definition at line 2227 of file saf_sh.c.

6.16.3.39 sphArrayAliasLim()

```
float sphArrayAliasLim (
    float r,
    float c,
    int maxN )
```

Returns a simple estimate of the spatial aliasing limit (the kR = maxN rule)

Parameters

in	*r*	Array radius, meters
in	*c*	Speed of sound, m/s
in	*maxN*	Order

Returns

Spatial aliasing limit estimate, in Hz

Definition at line 1859 of file saf_sh.c.

6.16.3.40 sphArrayNoiseThreshold()

```
void sphArrayNoiseThreshold (
    int maxN,
    int Nsensors,
    float r,
    float c,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    float maxG_db,
    float * f_lim )
```

Computes the frequncies (per order), at which the noise of a SHT of a SMA exceeds a specified maximum level.

Computes the frequencies that the noise in the output channels of a spherical microphone array (SMA), after performing the spherical harmonic transform (SHT) and equalisation of the output signals, reaches a certain user-defined threshold maxG_db. The frequencies are computed only at the lower range of each order, where its response decays rapidly, ignoring for example the nulls of an open array at the higher frequencies. The estimation of the limits are based on a linear approximation of the log-log response found e.g. in [1]

Parameters

in	*maxN*	Maximum order of the array
in	*Nsensors*	Number of sensors
in	*r*	Mic radius, meters
in	*c*	Speed of sound, m/s
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
in	*maxG_db*	Max allowed amplification for the noise level, maxG_db = 20∗log10(maxG)
out	*f_lim*	Noise limit estimate; (maxN+1) x 1

See also

[1] Politis, A., Vilkamo, J., & Pulkki, V. (2015). Sector-based parametric sound field reproduction in the spherical harmonic domain. IEEE Journal of Selected Topics in Signal Processing, 9(5), 852-866.

Definition at line 1869 of file saf_sh.c.

6.16.3.41 sphDiffCohMtxTheory()

```
void sphDiffCohMtxTheory (
    int order,
    float * sensor_dirs_rad,
    int N_sensors,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    double * kr,
    double * kR,
    int nBands,
    double * M_diffcoh )
```

Calculates the theoretical diffuse coherence matrix for a spherical array.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*sensor_dirs_rad*	Spherical coords of the sensors in RADIANS, [azi ELEV]; FLAT: N_sensors x 2
in	*N_sensors*	Number of sensors
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
in	*kr*	wavenumber∗sensor_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius, set to NULL if not applicable; nBands x 1
in	*nBands*	Number of frequency bands/bins
out	*M_diffcoh*	Theoretical diffuse coherence matrix per frequency; FLAT: N_sensors x N_sensors x nBands

Definition at line 2097 of file saf_sh.c.

6.16.3.42 sphModalCoeffs()

```
void sphModalCoeffs (
    int order,
    double * kr,
    int nBands,
    ARRAY_CONSTRUCTION_TYPES arrayType,
    double dirCoeff,
    double_complex * b_N )
```

Calculates the modal coefficients for open/rigid spherical arrays.

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*arrayType*	See 'ARRAY_CONSTRUCTION_TYPES' enum
in	*dirCoeff*	Only for directional (open) arrays, 1: omni, 0.5: card, 0:dipole
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 1897 of file saf_sh.c.

6.16.3.43 sphScattererDirModalCoeffs()

```
void sphScattererDirModalCoeffs (
    int order,
    double * kr,
    double * kR,
    int nBands,
    double dirCoeff,
    double_complex * b_N )
```

Calculates the modal coefficients for a rigid spherical scatterer with directional sensors.

Assumes all sensors are placed the same distance from the scatterer, w.r.t. the origin

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗array_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
in	*dirCoeff*	Directivity coefficient, 1: omni, 0.5: card, 0:dipole
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 2030 of file saf_sh.c.

6.16.3.44 sphScattererModalCoeffs()

```
void sphScattererModalCoeffs (
    int order,
    double * kr,
```

```
double * kR,
int nBands,
double_complex * b_N )
```

Calculates the modal coefficients for a rigid spherical scatterer with omni-directional sensors.

Assumes all sensors are placed the same distance from the scatterer, w.r.t. the origin

Parameters

in	*order*	Max order (highest is ∼30 given numerical precision)
in	*kr*	wavenumber∗array_radius; nBands x 1
in	*kR*	wavenumber∗scatterer_radius; nBands x 1
in	*nBands*	Number of frequency bands/bins
out	*b_N*	Modal coefficients per kr and 0:order; FLAT: nBands x (order+1)

Definition at line 1981 of file saf_sh.c.

6.16.3.45 unitCart2Sph()

```
void unitCart2Sph (
float xyz[3],
float AziElev_rad[2] )
```

Converts cartesian coordinates of unit length to spherical coordinates.

Parameters

in	*xyz*	Unit cartesian coords, xyz
out	*AziElev_rad*	Azimuth and elevation in radians

Definition at line 137 of file saf_sh.c.

6.16.3.46 unitCart2Sph_aziElev()

```
void unitCart2Sph_aziElev (
float xyz[3],
float * azi_rad,
float * elev_rad )
```

Converts cartesian coordinates of unit length to spherical coordinates.

Parameters

in	*xyz*	Unit cartesian coords, xyz
out	*azi_rad*	(&) azimuth in radians
out	*elev_rad*	(&) elevation in radians

Definition at line 148 of file saf_sh.c.

6.16.3.47 unitSph2Cart()

```
void unitSph2Cart (
    float azi_rad,
    float elev_rad,
    float xyz[3] )
```

Converts spherical coordinates to cartesian coordinates of unit length.

Parameters

in	*azi_rad*	Azimuth in radians
in	*elev_rad*	Elevation in radians
out	*xyz*	Unit cartesian coords, xyz; 3 x 1

Definition at line 125 of file saf_sh.c.

6.16.3.48 unnorm_legendreP()

```
void unnorm_legendreP (
    int n,
    double * x,
    int lenX,
    double * y )
```

Calculates unnormalised legendre polynomials up to order N, for all values in vector x [1].

Note

This INCLUDES the Condon-Shortley phase term. It is functionally identical to Matlab's legendre function (with default settings ['unnorm']).

Parameters

in	*n*	Order of legendre polynomial
in	*x*	Vector of input values; lenX x 1
in	*lenX*	Number of input values
out	*y*	Resulting unnormalised legendre values for each x value; FLAT: (n+1) x lenX

See also

[1] M, Abramowitz., I.A. Stegun. (1965). "Handbook of Mathematical Functions: Chapter 8", Dover Publications.

Definition at line 160 of file saf_sh.c.

6.16.3.49 unnorm_legendreP_recur()

```
void unnorm_legendreP_recur (
    int n,
    float * x,
    int lenX,
```

```
float * Pnm_minus1,
float * Pnm_minus2,
float * Pnm )
```

Calculates unnormalised legendre polynomials up to order N, for all values in vector x.

It uses a recursive approach, which makes it more suitable for computing the legendre values in a real-time loop.

Note

This does NOT INCLUDE the Condon-Shortley phase term.

Parameters

in	*n*	Order of legendre polynomial
in	*x*	Vector of input values; lenX x 1
in	*lenX*	Number of input values
in	*Pnm_minus1*	Previous Pnm, (not used for n=1); FLAT: (n+1) x lenX
in	*Pnm_minus2*	Previous previous Pnm, (not used for n=0); FLAT: (n+1) x lenX
out	*Pnm*	Resulting unnormalised legendre values for each x value; FLAT: (n+1) x lenX

Definition at line 236 of file saf_sh.c.

6.16.3.50 yawPitchRoll2Rzyx()

```
void yawPitchRoll2Rzyx (
float yaw,
float pitch,
float roll,
int rollPitchYawFLAG,
float R[3][3] )
```

Constructs a 3x3 rotation matrix from the Euler angles, using the yaw-pitch-roll (zyx) convention.

Parameters

in	*yaw*	Yaw angle in radians
in	*pitch*	Pitch angle in radians
in	*roll*	Roll angle in radians
in	*rollPitchYawFLAG*	'1' to use Rxyz, i.e. apply roll, pitch and then yaw, '0' Rzyx / y-p-r
out	*R*	zyx rotation matrix; 3 x 3

Definition at line 69 of file saf_sh.c.

6.17 framework/modules/saf_sh/saf_sh_internal.c File Reference

Internal part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

```
#include "saf_sh.h"
#include "saf_sh_internal.h"
```

Functions

- float wigner_3j (int j1, int j2, int j3, int m1, int m2, int m3)

  Computes the Wigner 3j symbol through the Racah formula found in `http://mathworld.wolfram.com/Wigner3j-Symbol.html`, *Eq.7.*
- void gaunt_mtx (int N1, int N2, int N, float ∗A)

  Constructs a matrix of Guant coefficients.
- void **SPHI** (int N, double X, int ∗NM, double ∗SI, double ∗DI)
- void **SPHK** (int N, double X, int ∗NM, double ∗SK, double ∗DK)
- void **SPHJ** (int N, double X, int ∗NM, double ∗SJ, double ∗DJ)
- int **MSTA1** (double X, int MP)
- int **MSTA2** (double X, int N, int MP)
- double **ENVJ** (int N, double X)
- void **SPHY** (int N, double X, int ∗NM, double ∗SY, double ∗DY)
- float **getP** (int i, int l, int a, int b, float ∗∗R_1, float ∗∗R_lm1)
- float **getU** (int l, int m, int n, float ∗∗R_1, float ∗∗R_lm1)
- float **getV** (int l, int m, int n, float ∗∗R_1, float ∗∗R_lm1)
- float **getW** (int l, int m, int n, float ∗∗R_1, float ∗∗R_lm1)

6.17.1 Detailed Description

Internal part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

A collection of spherical harmonic related functions. Many of which have been derived from Matlab libraries by Archontis Politis [1-3].

See also

[1] `https://github.com/polarch/Spherical-Harmonic-Transform`

[2] `https://github.com/polarch/Array-Response-Simulator`

[3] `https://github.com/polarch/Spherical-Array-Processing`

Author

Leo McCormack

Date

22.05.2016

6.17.2 Function Documentation

6.17.2.1 gaunt_mtx()

```
void gaunt_mtx (
    int N1,
    int N2,
    int N,
    float * A )
```

Constructs a matrix of Guant coefficients.

Constructs the (N1+1)^2x(N2+1)^2x(N+1)^2 matrix of Gaunt coefficients which represent the integral of three spherical harmonics such as: G^q_{q',q''} = \int_\Omega Y_{q'}Y_{q''}Y^∗_{q} \mathrm{d}\Omega. With Gaunt coefficients, the spherical harmonic coefficients of the product of two spherical functions can be given directly as a linear relationship between the harmonic coefficients of the two functions.

Parameters

in	*N1*	Order of first harmonic coeffient
in	*N2*	Order of second harmonic coefficient
in	*N*	Target order
out	*A*	Gaunt matrix; FLAT: (N1+1)^2 x (N2+1)^2 x (N+1)^2

Definition at line 92 of file saf_sh_internal.c.

6.17.2.2 wigner_3j()

```
float wigner_3j (
    int j1,
    int j2,
    int j3,
    int m1,
    int m2,
    int m3 )
```

Computes the Wigner 3j symbol through the Racah formula found in `http://mathworld.wolfram.com/Wigner3j-Symbol.html`, Eq.7.

Parameters

in	*j1*	Wigner 3 j-symbol, j1
in	*j2*	Wigner 3 j-symbol, j2
in	*j3*	Wigner 3 j-symbol, j3
in	*m1*	Wigner 3 j-symbol, m1
in	*m2*	Wigner 3 j-symbol, m2
in	*m3*	Wigner 3 j-symbol, m3

Returns

wigner_3j symbol

Definition at line 37 of file saf_sh_internal.c.

6.18 framework/modules/saf_sh/saf_sh_internal.h File Reference

Internal part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#include <string.h>
#include <assert.h>
#include "saf_sh.h"
#include "../saf_utilities/saf_utilities.h"
```

Functions

- float wigner_3j (int j1, int j2, int j3, int m1, int m2, int m3)

    Computes the Wigner 3j symbol through the Racah formula found in http://mathworld.wolfram.com/↩ Wigner3j-Symbol.html, Eq.7.
- void gaunt_mtx (int N1, int N2, int N, float *A)

    Constructs a matrix of Guant coefficients.
- void **SPHI** (int N, double X, int *NM, double *SI, double *DI)
- void **SPHK** (int N, double X, int *NM, double *SK, double *DK)
- void **SPHJ** (int N, double X, int *NM, double *SJ, double *DJ)
- int **MSTA1** (double X, int MP)
- int **MSTA2** (double X, int N, int MP)
- double **ENVJ** (int N, double X)
- void **SPHY** (int N, double X, int *NM, double *SY, double *DY)
- float **getP** (int i, int l, int a, int b, float **R_1, float **R_lm1)
- float **getU** (int l, int m, int n, float **R_1, float **R_lm1)
- float **getV** (int l, int m, int n, float **R_1, float **R_lm1)
- float **getW** (int l, int m, int n, float **R_1, float **R_lm1)

6.18.1 Detailed Description

Internal part of the Spherical Harmonic Transform and Spherical Array Processing module (saf_sh)

A collection of spherical harmonic related functions. Many of which have been derived from Matlab libraries by Archontis Politis [1-3].

See also

[1] https://github.com/polarch/Spherical-Harmonic-Transform

[2] https://github.com/polarch/Array-Response-Simulator

[3] https://github.com/polarch/Spherical-Array-Processing

Author

Leo McCormack

Date

22.05.2016

6.18.2 Function Documentation

6.18.2.1 gaunt_mtx()

```
void gaunt_mtx (
    int N1,
    int N2,
    int N,
    float * A )
```

Constructs a matrix of Guant coefficients.

Constructs the (N1+1)^2x(N2+1)^2x(N+1)^2 matrix of Gaunt coefficients which represent the integral of three spherical harmonics such as: G^q_{q',q''} = \int_\Omega Y_{q'}Y_{q''}Y^*_{q} \mathrm{d}\Omega. With Gaunt coefficients, the spherical harmonic coefficients of the product of two spherical functions can be given directly as a linear relationship between the harmonic coefficients of the two functions.

Parameters

in	*N1*	Order of first harmonic coeffient
in	*N2*	Order of second harmonic coefficient
in	*N*	Target order
out	*A*	Gaunt matrix; FLAT: (N1+1)^2 x (N2+1)^2 x (N+1)^2

Definition at line 92 of file saf_sh_internal.c.

6.18.2.2 wigner_3j()

```
float wigner_3j (
        int j1,
        int j2,
        int j3,
        int m1,
        int m2,
        int m3 )
```

Computes the Wigner 3j symbol through the Racah formula found in http://mathworld.wolfram.com/Wigner3j-Symbol.html, Eq.7.

Parameters

in	*j1*	Wigner 3 j-symbol, j1
in	*j2*	Wigner 3 j-symbol, j2
in	*j3*	Wigner 3 j-symbol, j3
in	*m1*	Wigner 3 j-symbol, m1
in	*m2*	Wigner 3 j-symbol, m2
in	*m3*	Wigner 3 j-symbol, m3

Returns

wigner_3j symbol

Definition at line 37 of file saf_sh_internal.c.

6.19 framework/modules/saf_utilities/saf_complex.c File Reference

Contains wrappers for handling complex numbers across both C99- compliant compilers and Microsoft Visual Compiler (MSVC)

```
#include "saf_complex.h"
```

6.19.1 Detailed Description

Contains wrappers for handling complex numbers across both C99- compliant compilers and Microsoft Visual Compiler (MSVC)

Author

Leo McCormack

Date

11.07.2016

6.20 framework/modules/saf_utilities/saf_complex.h File Reference

Contains wrappers for handling complex numbers across both C99- compliant compilers and Microsoft Visual Compiler (MSVC)

6.20.1 Detailed Description

Contains wrappers for handling complex numbers across both C99- compliant compilers and Microsoft Visual Compiler (MSVC)

Author

Leo McCormack

Date

11.07.2016

6.21 framework/modules/saf_utilities/saf_decor.c File Reference

Collection of signal decorrelators.

```
#include "saf_decor.h"
#include "saf_utilities.h"
```

Functions

- static void randperm (int len, int ∗randperm)

  Random permutation of a vector of integers.
- void getDecorrelationDelays (int nChannels, float ∗freqs, int nFreqs, float fs, int maxTFdelay, int hopSize, int ∗delayTF)

  Returns delay values for multiple channels per frequency, such that once applied to an input signal (via simple frequency-dependent delay lines), the resulting signal is decorrelated w.r.t the original.
- void synthesiseNoiseReverb (int nCH, float fs, float ∗t60, float ∗fcen_oct, int nBands, int flattenFLAG, float ∗∗rir_filt, int ∗rir_len)

  Returns quick and dirty exponentially decaying noise bursts.

6.21.1 Detailed Description

Collection of signal decorrelators.

Author

Leo McCormack

Date

30.07.2018

6.21.2 Function Documentation

6.21.2.1 getDecorrelationDelays()

```
void getDecorrelationDelays (
        int nChannels,
        float * freqs,
        int nFreqs,
        float fs,
        int maxTFdelay,
        int hopSize,
        int * delayTF )
```

Returns delay values for multiple channels per frequency, such that once applied to an input signal (via simple frequency-dependent delay lines), the resulting signal is decorrelated w.r.t the original.

Note

This is a very basic algorithm and sounds particulary bad for transient signals. Consider using a transient detector to "duck" the decorrelated signal during such transients, to improve signal fidelity.

Parameters

in	*nChannels*	Number of channels
in	*freqs*	Centre frequencies; nFreqs x 1
in	*nFreqs*	Number of elements in frequency vector
in	*fs*	Sampling rate
in	*maxTFdelay*	Max number of time-slots to delay
in	*hopSize*	STFT hop size
out	*delayTF*	The resulting time delays per channel and frequency; FLAT: nFreq x nChannels

Definition at line 46 of file saf_decor.c.

6.21.2.2 synthesiseNoiseReverb()

```
void synthesiseNoiseReverb (
        int nChannels,
```

```
float fs,
float * t60,
float * fcen_oct,
int nBands,
int flattenFLAG,
float ** rir_filt,
int * rir_len )
```

Returns quick and dirty exponentially decaying noise bursts.

With long T60 times, it can be used to approximate the late reverberation tail of room impulse responses. With much shorter t60 times, it can be used for decorrelation purposes.

Parameters

in	*nChannels*	Number of channels
in	*fs*	Sampling rate
in	*t60*	T60 times (in seconds) per octave band; nBands x 1
in	*fcen_oct*	Octave band centre frequencies; nBands x 1
in	*nBands*	Number of octave bands
in	*flattenFLAG*	'0' nothing, '1' flattens the magnitude response to unity
out	*rir_filt*	(&) the shaped noise bursts; FLAT: nChannels x rir_len
out	*rir_len*	(&) length of filters, in samples

Definition at line 96 of file saf_decor.c.

6.22 framework/modules/saf_utilities/saf_decor.h File Reference

Collection of signal decorrelators.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
```

Functions

- void getDecorrelationDelays (int nChannels, float *freqs, int nFreqs, float fs, int maxTFdelay, int hopSize, int *delayTF)

  Returns delay values for multiple channels per frequency, such that once applied to an input signal (via simple frequency-dependent delay lines), the resulting signal is decorrelated w.r.t the original.
- void synthesiseNoiseReverb (int nChannels, float fs, float *t60, float *fcen_oct, int nBands, int flattenFLAG, float **rir_filt, int *rir_len)

  Returns quick and dirty exponentially decaying noise bursts.

6.22.1 Detailed Description

Collection of signal decorrelators.

Author

Leo McCormack

Date

30.07.2018

6.22.2 Function Documentation

6.22.2.1 getDecorrelationDelays()

```
void getDecorrelationDelays (
        int nChannels,
        float * freqs,
        int nFreqs,
        float fs,
        int maxTFdelay,
        int hopSize,
        int * delayTF )
```

Returns delay values for multiple channels per frequency, such that once applied to an input signal (via simple frequency-dependent delay lines), the resulting signal is decorrelated w.r.t the original.

Note

This is a very basic algorithm and sounds particulary bad for transient signals. Consider using a transient detector to "duck" the decorrelated signal during such transients, to improve signal fidelity.

Parameters

in	*nChannels*	Number of channels
in	*freqs*	Centre frequencies; nFreqs x 1
in	*nFreqs*	Number of elements in frequency vector
in	*fs*	Sampling rate
in	*maxTFdelay*	Max number of time-slots to delay
in	*hopSize*	STFT hop size
out	*delayTF*	The resulting time delays per channel and frequency; FLAT: nFreq x nChannels

Definition at line 46 of file saf_decor.c.

6.22.2.2 synthesiseNoiseReverb()

```
void synthesiseNoiseReverb (
        int nChannels,
```

```
float fs,
float * t60,
float * fcen_oct,
int nBands,
int flattenFLAG,
float ** rir_filt,
int * rir_len )
```

Returns quick and dirty exponentially decaying noise bursts.

With long T60 times, it can be used to approximate the late reverberation tail of room impulse responses. With much shorter t60 times, it can be used for decorrelation purposes.

Parameters

in	*nChannels*	Number of channels
in	*fs*	Sampling rate
in	*t60*	T60 times (in seconds) per octave band; nBands x 1
in	*fcen_oct*	Octave band centre frequencies; nBands x 1
in	*nBands*	Number of octave bands
in	*flattenFLAG*	'0' nothing, '1' flattens the magnitude response to unity
out	*rir_filt*	(&) the shaped noise bursts; FLAT: nChannels x rir_len
out	*rir_len*	(&) length of filters, in samples

Definition at line 96 of file saf_decor.c.

6.23 framework/modules/saf_utilities/saf_erb.c File Reference

A function to ascertain frequencies that fall within critical bands. [Equivalent-Rectangular Bandwidth (ERB)].

```
#include "saf_utilities.h"
```

Functions

- void findERBpartitions (float *centerFreq, int nBands, float maxFreqLim, int **erb_idx, float **erb_freqs, int *nERBBands)

    This function takes a frequency vector and groups its frequencies into critical bands [Equivalent-Rectangular Bandwidth (ERB)].

6.23.1 Detailed Description

A function to ascertain frequencies that fall within critical bands. [Equivalent-Rectangular Bandwidth (ERB)].

Author

Leo McCormack

Date

30.07.2018

6.23.2 Function Documentation

6.23.2.1 findERBpartitions()

```
void findERBpartitions (
    float * centerFreq,
    int nBands,
    float maxFreqLim,
    int ** erb_idx,
    float ** erb_freqs,
    int * nERBBands )
```

This function takes a frequency vector and groups its frequencies into critical bands [Equivalent-Rectangular Bandwidth (ERB)].

e.g.

- centerFreq[erb_idx[0]-1] -> centerFreq[erb_idx[1]-1] is ERB band 1
- centerFreq[erb_idx[1]-1] -> centerFreq[erb_idx[2]-1] is ERB band 2

Note

erb indices start from 1!

Parameters

in	*centerFreq*	Frequency vector; nBands x 1
in	*nBands*	Number of bins/bands in frequency vector
in	*maxFreqLim*	Past this frequency the bands are grouped into 1 band
out	*erb_idx*	(&) ERB indices (start from 1); nERBBands x 1
out	*erb_freqs*	(&) ERB frequencies; nERBBands x 1
out	*nERBBands*	(&) Number of ERB bands; 1 x 1

Definition at line 30 of file saf_erb.c.

6.24 framework/modules/saf_utilities/saf_erb.h File Reference

A function to ascertain frequencies that fall within critical bands. [Equivalent-Rectangular Bandwidth (ERB)].

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <float.h>
```

Functions

- void findERBpartitions (float ∗centerFreq, int nBands, float maxFreqLim, int ∗∗erb_idx, float ∗∗erb_freqs, int ∗nERBBands)

  This function takes a frequency vector and groups its frequencies into critical bands [Equivalent-Rectangular Bandwidth (ERB)].

6.24.1 Detailed Description

A function to ascertain frequencies that fall within critical bands. [Equivalent-Rectangular Bandwidth (ERB)].

Author

Leo McCormack

Date

30.07.2018

6.24.2 Function Documentation

6.24.2.1 findERBpartitions()

```
void findERBpartitions (
    float * centerFreq,
    int nBands,
    float maxFreqLim,
    int ** erb_idx,
    float ** erb_freqs,
    int * nERBBands )
```

This function takes a frequency vector and groups its frequencies into critical bands [Equivalent-Rectangular Bandwidth (ERB)].

e.g.

- centerFreq[erb_idx[0]-1] -> centerFreq[erb_idx[1]-1] is ERB band 1
- centerFreq[erb_idx[1]-1] -> centerFreq[erb_idx[2]-1] is ERB band 2

Note

erb indices start from 1!

Parameters

in	*centerFreq*	Frequency vector; nBands x 1
in	*nBands*	Number of bins/bands in frequency vector
in	*maxFreqLim*	Past this frequency the bands are grouped into 1 band
out	*erb_idx*	(&) ERB indices (start from 1); nERBBands x 1
out	*erb_freqs*	(&) ERB frequencies; nERBBands x 1
out	*nERBBands*	(&) Number of ERB bands; 1 x 1

Definition at line 30 of file saf_erb.c.

6.25 framework/modules/saf_utilities/saf_error.c File Reference

List of error and warning codes.

`#include "saf_error.h"`

Functions

- SAF_ERRORS saf_error_print (SAF_ERRORS err)
  Checks current error/warning code, and prints out a message if needed (used when in debug mode).

6.25.1 Detailed Description

List of error and warning codes.

Author

Leo McCormack

Date

05.08.2019

6.25.2 Function Documentation

6.25.2.1 saf_error_print()

```
SAF_ERRORS saf_error_print (
    SAF_ERRORS err )
```

Checks current error/warning code, and prints out a message if needed (used when in debug mode).

If there is no error/warning (SAF_ERROR__NO_ERROR), then the function does nothing. If there is a warning code, then an appropriate warning message is printed, and err is reset upon return (if needed). If there is an error code, then an appropriate error message is printed, and the program is terminated.

Parameters

in	*err*	SAF error code (see "SAF_ERRORS" enum)

Returns

SAF_ERROR__NO_ERROR

Definition at line 32 of file saf_error.c.

6.26 framework/modules/saf_utilities/saf_error.h File Reference

List of error and warning codes.

```
#include "stdio.h"
```

Enumerations

- enum _SAF_ERRORS {
  SAF_ERROR__NO_ERROR = 0, SAF_ERROR__ILLEGAL_INPUT_VALUE, SAF_ERROR__UNALLOCATED_FUNCTION_A
  SAF_ERROR__FAILED_TO_BUILD_CONVEX_HULL,
  SAF_WARNING__SOFA_FILE_NOT_FOUND, SAF_WARNING__UNABLE_TO_COMPUTE_BESSEL_FUNCTION_AT_SPE
  SAF_WARNING__FAILED_TO_COMPUTE_SVD, SAF_WARNING__FAILED_TO_COMPUTE_EVG,
  SAF_WARNING__FAILED_TO_SOLVE_LINEAR_EQUATION, SAF_WARNING__FAILED_TO_COMPUTE_CHOL
  }

  Error and warning codes.

Functions

- SAF_ERRORS saf_error_print (SAF_ERRORS err)

  Checks current error/warning code, and prints out a message if needed (used when in debug mode).

6.26.1 Detailed Description

List of error and warning codes.

Author

Leo McCormack

Date

05.08.2019

6.26.2 Enumeration Type Documentation

Enumerator

6.26.2.1 _SAF_ERRORS `enum _SAF_ERRORS`

Error and warning codes.

Error codes are considered fatal. Whereas warnings are given if alternative measures have taken place (due to some kind of strange behaviour), but the program may still continue.

Enumerator

SAF_ERROR__NO_ERROR	No error was encountered.
SAF_ERROR__ILLEGAL_INPUT_VALUE	One or more input variable is assigned an illegal value.
SAF_ERROR__UNALLOCATED_FUNCTION_AR↩ GUMENT	One or more input/output variable is NULL.
SAF_ERROR__FAILED_TO_BUILD_CONVEX_H↩ ULL	findLsTriplets - Failed to build Convex Hull.
SAF_WARNING__SOFA_FILE_NOT_FOUND	loadSofaFile(): sofa file was not found at the specified directory. Remember to include the ".sofa" suffix. In this case, the default HRIR set is loaded instead.
SAF_WARNING__UNABLE_TO_COMPUTE_BES↩ SEL_FUNCTION_AT_SPECIFIED_ORDER	bessel_jn(), bessel_in(), bessel_yn(), bessel_kn(), hankel_hn1(), or hankel_hn2(): Unable to compute the spherical Bessel/Hankel function at the specified order and input value. In this case, the Bessel/Hankel functions are returned at the maximum order that was possible, and this maximum order is returned by the function.
SAF_WARNING__FAILED_TO_COMPUTE_SVD	utility_?svd/utility_?pinv - The SVD failed to converge, or the input matrix contained illegal values so no solution was attempted. In these cases the function will zero all output matrices and vectors.
SAF_WARNING__FAILED_TO_COMPUTE_EVG	utility_?seig/utility_?eigmp/utility_?eig - Failed to compute all of the eigenvalues, no eigenvectors have been computed, or the input matrix contained illegal values so no solution was attempted. In these cases the function will zero all output matrices and vectors.
SAF_WARNING__FAILED_TO_SOLVE_LINEAR_↩ EQUATION	utility_?glslv/utility_?slslv - Input matrix was singular, solution not computed, or the input matrix contained illegal values so no solution was attempted. In these cases the function will zero the output matrix.
SAF_WARNING__FAILED_TO_COMPUTE_CHOL	utility_?chol - input matrix is not positive definite, and the Cholesky factorization could not be computed, or the input matrix contained illegal values so no solution was attempted. In these cases the function will zero the output matrix.

Definition at line 43 of file saf_error.h.

6.26.3 Function Documentation

6.26.3.1 saf_error_print()

```
SAF_ERRORS saf_error_print (
        SAF_ERRORS err )
```

Checks current error/warning code, and prints out a message if needed (used when in debug mode).

If there is no error/warning (SAF_ERROR__NO_ERROR), then the function does nothing. If there is a warning code, then an appropriate warning message is printed, and err is reset upon return (if needed). If there is an error code, then an appropriate error message is printed, and the program is terminated.

Parameters

in	*err*	SAF error code (see "SAF_ERRORS" enum)

Returns

SAF_ERROR__NO_ERROR

Definition at line 32 of file saf_error.c.

6.27 framework/modules/saf_utilities/saf_fft.c File Reference

Wrappers for optimised fast Fourier transform (FFT) routines.

```
#include "saf_utilities.h"
#include "saf_fft.h"
```

Data Structures

- struct _saf_rfft_data

  Data structure for real-(half)complex FFT transforms.
- struct _saf_fft_data

  Data structure for complex-complex FFT transforms.

Functions

- static int nextpow2 (int numsamp)

  A simple function which returns the next power of 2, taken from: *https://github.com/amaggi/legacy-code*.
- void getUniformFreqVector (int fftSize, float fs, float ∗freqVector)

  Calcuates the frequencies (in Hz) of uniformly spaced bins, for a given FFT size and sampling rate.
- void fftconv (float ∗x, float ∗h, int x_len, int h_len, int nCH, float ∗y)

  FFT-based convolution of signal 'x' with filter 'h'.
- void fftfilt (float ∗x, float ∗h, int x_len, int h_len, int nCH, float ∗y)

  FFT-based convolution for FIR filters.
- void hilbert (float_complex ∗x, int x_len, float_complex ∗y)

  Computes the discrete-time analytic signal via the Hilbert transform.
- void saf_rfft_create (void ∗∗const phFFT, int N)

  Creates an instance of saf_rfft; real<->half-complex (conjugate-symmetric) FFT.
- void saf_rfft_destroy (void ∗∗const phFFT)

Destroys an instance of saf_rfft.

- void saf_rfft_forward (void *const hFFT, float *inputTD, float_complex *outputFD)

  Performs the forward-FFT operation; use for real to complex (conjugate symmetric) transformations.
- void saf_rfft_backward (void *const hFFT, float_complex *inputFD, float *outputTD)

  Performs the backward-FFT operation; use for complex (conjugate symmetric) to real transformations.
- void saf_fft_create (void **const phFFT, int N)

  Creates an instance of saf_fft; complex<->complex FFT.
- void saf_fft_destroy (void **const phFFT)

  Destroys an instance of saf_fft.
- void saf_fft_forward (void *const hFFT, float_complex *inputTD, float_complex *outputFD)

  Performs the forward-FFT operation; use for complex to complex transformations.
- void saf_fft_backward (void *const hFFT, float_complex *inputFD, float_complex *outputTD)

  Performs the backward-FFT operation; use for complex to complex transformations.

6.27.1 Detailed Description

Wrappers for optimised fast Fourier transform (FFT) routines.

Note

If none of the supported optimised FFT implementations are linked, then saf_fft employs the highly respectable KissFFT from here (BSD 3-Clause License): https://github.com/mborgerding/kissfft

If linking Apple Accelerate: KissFFT is also used in cases where the FFT size is not 2^x.

6.27.1.1 Dependencies

Intel MKL, Apple Accelerate, or KissFFT (included in framework)

6.27.1.2 Example Usage

```
const int N = 256;                    // FFT size
float x_in[N];                     // input buffer (time-domain)
x_in[0] = ... x_in[N-1] =          // fill with data
float_complex x_out[(N/2+1)];      // output buffer (frequency-domain)
float test[N];                     // test (time-domain)
void *hFFT;                        // safFFT handle
safFFT_create(&hFFT, N);           // creates instance of safFFT
safFFT_forward(hFFT, x_in, x_out);  // perform forward transform
safFFT_backward(hFFT, x_out, test); // perform backwards transform
// 'x_in' should equal 'test' (given some numerical error)
safFFT_destroy(&hFFT);             // destroys instance of safFFT
```

Author

Leo McCormack

Date

06.04.2019

6.27.2 Function Documentation

6.27.2.1 fftconv()

```
void fftconv (
    float * x,
    float * h,
    int x_len,
    int h_len,
    int nCH,
    float * y )
```

FFT-based convolution of signal 'x' with filter 'h'.

Input channels and filters are zero padded to avoid circular convolution artefacts.

Note

The output must be of size: nCH x (x_len+h_len-1)

Parameters

in	*x*	Input(s); FLAT: nCH x x_len
in	*h*	Filter(s); FLAT: nCH x h_len
in	*x_len*	Length of input signal, in samples
in	*h_len*	Length of filter, in samples
in	*nCH*	Number of channels
out	*y*	Output signal(s); FLAT: nCH x (x_len+h_len-1)

Definition at line 125 of file saf_fft.c.

6.27.2.2 fftfilt()

```
void fftfilt (
    float * x,
    float * h,
    int x_len,
    int h_len,
    int nCH,
    float * y )
```

FFT-based convolution for FIR filters.

Similar to fftconv, other than only the first x_len samples of y are returned. It has parity with the 'fftfilt' function in Matlab, except it just uses one big FFT (i.e. no overlap-add).

Parameters

in	*x*	Input(s); FLAT: nCH x x_len
in	*h*	Filter(s); FLAT: nCH x h_len
in	*x_len*	Length of input signal, in samples
in	*h_len*	Length of filter, in samples
in	*nCH*	Number of channels
out	*y*	Output signal(s); FLAT: nCH x x_len

Definition at line 178 of file saf_fft.c.

6.27.2.3 getUniformFreqVector()

```
void getUniformFreqVector (
    int fftSize,
    float fs,
    float * freqVector )
```

Calcuates the frequencies (in Hz) of uniformly spaced bins, for a given FFT size and sampling rate.

Parameters

in	*fftSize*	FFT size
in	*fs*	Sampling rate
out	*freqVector*	0:fs/(fftSize/2):fs/2; (fftSize/2+1) x 1

Definition at line 113 of file saf_fft.c.

6.27.2.4 hilbert()

```
void hilbert (
    float_complex * x,
    int x_len,
    float_complex * y )
```

Computes the discrete-time analytic signal via the Hilbert transform.

The magnitude of the output is the envelope, and imaginary part is the actual Hilbert transform. (Functionally identical to Matlab's 'hilbert' function)

Parameters

in	*x*	Input; x_len x 1
in	*x_len*	Length of input signal, in samples
out	*y*	Output analytic signal; x_len x 1

Definition at line 198 of file saf_fft.c.

6.27.2.5 saf_fft_backward()

```
void saf_fft_backward (
    void *const hFFT,
    float_complex * inputFD,
    float_complex * outputTD )
```

Performs the backward-FFT operation; use for complex to complex transformations.

Parameters

in	*hFFT*	saf_fft handle
in	*inputFD*	Frequency-domain input; N x 1
out	*outputTD*	Time-domain output; N x 1

Definition at line 490 of file saf_fft.c.

6.27.2.6 saf_fft_create()

```
void saf_fft_create (
        void **const phFFT,
        int N )
```

Creates an instance of saf_fft; complex<->complex FFT.

Note

Only Even FFT sizes are supported.

Parameters

in	*phFFT*	(&) address of saf_fft handle
in	*N*	FFT size

Definition at line 388 of file saf_fft.c.

6.27.2.7 saf_fft_destroy()

```
void saf_fft_destroy (
        void **const phFFT )
```

Destroys an instance of saf_fft.

Parameters

in	*phFFT*	(&) address of saf_fft handle

Definition at line 431 of file saf_fft.c.

6.27.2.8 saf_fft_forward()

```
void saf_fft_forward (
        void *const hFFT,
        float_complex * inputTD,
        float_complex * outputFD )
```

Performs the forward-FFT operation; use for complex to complex transformations.

Parameters

in	*hFFT*	saf_fft handle
in	*inputTD*	Time-domain input; N x 1
out	*outputFD*	Frequency-domain output; N x 1

Definition at line 457 of file saf_fft.c.

6.27.2.9 saf_rfft_backward()

```
void saf_rfft_backward (
    void *const hFFT,
    float_complex * inputFD,
    float * outputTD )
```

Performs the backward-FFT operation; use for complex (conjugate symmetric) to real transformations.

Note

Only the first N/2 + 1 bins are needed to be passed in inputFD.

Parameters

in	*hFFT*	saf_rfft handle
in	*inputFD*	Frequency-domain input; (N/2 + 1) x 1
out	*outputTD*	Time-domain output; N x 1

Definition at line 356 of file saf_fft.c.

6.27.2.10 saf_rfft_create()

```
void saf_rfft_create (
    void **const phFFT,
    int N )
```

Creates an instance of saf_rfft; real<->half-complex (conjugate-symmetric) FFT.

Note

Only Even FFT sizes are supported.

Parameters

in	*phFFT*	(&) address of saf_rfft handle
in	*N*	FFT size

Definition at line 245 of file saf_fft.c.

6.27.2.11 saf_rfft_destroy()

```
void saf_rfft_destroy (
    void **const phFFT )
```

Destroys an instance of saf_rfft.

Parameters

in	*phFFT*	(&) address of saf_rfft handle

Definition at line 300 of file saf_fft.c.

6.27.2.12 saf_rfft_forward()

```
void saf_rfft_forward (
    void *const hFFT,
    float * inputTD,
    float_complex * outputFD )
```

Performs the forward-FFT operation; use for real to complex (conjugate symmetric) transformations.

Note

Only the first N/2 + 1 bins are returned in outputFD.

Parameters

in	*hFFT*	saf_rfft handle
in	*inputTD*	Time-domain input; N x 1
out	*outputFD*	Frequency-domain output; (N/2 + 1) x 1

Definition at line 325 of file saf_fft.c.

6.28 framework/modules/saf_utilities/saf_fft.h File Reference

Wrappers for optimised fast Fourier transform (FFT) routines.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <assert.h>
#include <limits.h>
#include "saf_complex.h"
```

Functions

- void getUniformFreqVector (int fftSize, float fs, float *freqVector)

  Calcuates the frequencies (in Hz) of uniformly spaced bins, for a given FFT size and sampling rate.
- void fftconv (float *x, float *h, int x_len, int h_len, int nCH, float *y)

  FFT-based convolution of signal 'x' with filter 'h'.
- void fftfilt (float *x, float *h, int x_len, int h_len, int nCH, float *y)

  FFT-based convolution for FIR filters.
- void hilbert (float_complex *x, int x_len, float_complex *y)

Computes the discrete-time analytic signal via the Hilbert transform.
- void saf_rfft_create (void **const phFFT, int N)

  Creates an instance of saf_rfft; real<->half-complex (conjugate-symmetric) FFT.
- void saf_rfft_destroy (void **const phFFT)

  Destroys an instance of saf_rfft.
- void saf_rfft_forward (void *const hFFT, float *inputTD, float_complex *outputFD)

  Performs the forward-FFT operation; use for real to complex (conjugate symmetric) transformations.
- void saf_rfft_backward (void *const hFFT, float_complex *inputFD, float *outputTD)

  Performs the backward-FFT operation; use for complex (conjugate symmetric) to real transformations.
- void saf_fft_create (void **const phFFT, int N)

  Creates an instance of saf_fft; complex<->complex FFT.
- void saf_fft_destroy (void **const phFFT)

  Destroys an instance of saf_fft.
- void saf_fft_forward (void *const hFFT, float_complex *inputTD, float_complex *outputFD)

  Performs the forward-FFT operation; use for complex to complex transformations.
- void saf_fft_backward (void *const hFFT, float_complex *inputFD, float_complex *outputTD)

  Performs the backward-FFT operation; use for complex to complex transformations.

6.28.1 Detailed Description

Wrappers for optimised fast Fourier transform (FFT) routines.

Note

If none of the supported optimised FFT implementations are linked, then saf_fft employs the highly respectable KissFFT from here (BSD 3-Clause License): https://github.com/mborgerding/kissfft

If linking Apple Accelerate: KissFFT is also used in cases where the FFT size is not 2^x.

6.28.1.1 Dependencies Intel MKL, Apple Accelerate, or KissFFT (included in framework)

6.28.1.2 Example Usage

```
const int N = 256;                  // FFT size
float x_in[N];                      // input buffer (time-domain)
x_in[0] = ... x_in[N-1] =           // fill with data
float_complex x_out[(N/2+1)];       // output buffer (frequency-domain)
float test[N];                      // test (time-domain)
void *hFFT;                         // safFFT handle
safFFT_create(&hFFT, N);            // creates instance of safFFT
safFFT_forward(hFFT, x_in, x_out);  // perform forward transform
safFFT_backward(hFFT, x_out, test); // perform backwards transform
// 'x_in' should equal 'test' (given some numerical error)
safFFT_destroy(&hFFT);              // destroys instance of safFFT
```

Author

Leo McCormack

Date

06.04.2019

6.28.2 Function Documentation

6.28.2.1 fftconv()

```
void fftconv (
    float * x,
    float * h,
    int x_len,
    int h_len,
    int nCH,
    float * y )
```

FFT-based convolution of signal 'x' with filter 'h'.

Input channels and filters are zero padded to avoid circular convolution artefacts.

Note

The output must be of size: nCH x (x_len+h_len-1)

Parameters

in	*x*	Input(s); FLAT: nCH x x_len
in	*h*	Filter(s); FLAT: nCH x h_len
in	*x_len*	Length of input signal, in samples
in	*h_len*	Length of filter, in samples
in	*nCH*	Number of channels
out	*y*	Output signal(s); FLAT: nCH x (x_len+h_len-1)

Definition at line 125 of file saf_fft.c.

6.28.2.2 fftfilt()

```
void fftfilt (
    float * x,
    float * h,
    int x_len,
    int h_len,
    int nCH,
    float * y )
```

FFT-based convolution for FIR filters.

Similar to fftconv, other than only the first x_len samples of y are returned. It has parity with the 'fftfilt' function in Matlab, except it just uses one big FFT (i.e. no overlap-add).

Parameters

in	*x*	Input(s); FLAT: nCH x x_len
in	*h*	Filter(s); FLAT: nCH x h_len
in	*x_len*	Length of input signal, in samples
in	*h_len*	Length of filter, in samples
in	*nCH*	Number of channels
out	*y*	Output signal(s); FLAT: nCH x x_len

Definition at line 178 of file saf_fft.c.

6.28.2.3 getUniformFreqVector()

```
void getUniformFreqVector (
    int fftSize,
    float fs,
    float * freqVector )
```

Calcuates the frequencies (in Hz) of uniformly spaced bins, for a given FFT size and sampling rate.

Parameters

in	*fftSize*	FFT size
in	*fs*	Sampling rate
out	*freqVector*	0:fs/(fftSize/2):fs/2; (fftSize/2+1) x 1

Definition at line 113 of file saf_fft.c.

6.28.2.4 hilbert()

```
void hilbert (
    float_complex * x,
    int x_len,
    float_complex * y )
```

Computes the discrete-time analytic signal via the Hilbert transform.

The magnitude of the output is the envelope, and imaginary part is the actual Hilbert transform. (Functionally identical to Matlab's 'hilbert' function)

Parameters

in	*x*	Input; x_len x 1
in	*x_len*	Length of input signal, in samples
out	*y*	Output analytic signal; x_len x 1

Definition at line 198 of file saf_fft.c.

6.28.2.5 saf_fft_backward()

```
void saf_fft_backward (
    void *const hFFT,
    float_complex * inputFD,
    float_complex * outputTD )
```

Performs the backward-FFT operation; use for complex to complex transformations.

Parameters

in	*hFFT*	saf_fft handle
in	*inputFD*	Frequency-domain input; N x 1
out	*outputTD*	Time-domain output; N x 1

Definition at line 490 of file saf_fft.c.

6.28.2.6 saf_fft_create()

```
void saf_fft_create (
    void **const phFFT,
    int N )
```

Creates an instance of saf_fft; complex<->complex FFT.

Note

Only Even FFT sizes are supported.

Parameters

in	*phFFT*	(&) address of saf_fft handle
in	*N*	FFT size

Definition at line 388 of file saf_fft.c.

6.28.2.7 saf_fft_destroy()

```
void saf_fft_destroy (
    void **const phFFT )
```

Destroys an instance of saf_fft.

Parameters

in	*phFFT*	(&) address of saf_fft handle

Definition at line 431 of file saf_fft.c.

6.28.2.8 saf_fft_forward()

```
void saf_fft_forward (
    void *const hFFT,
    float_complex * inputTD,
    float_complex * outputFD )
```

Performs the forward-FFT operation; use for complex to complex transformations.

Parameters

in	*hFFT*	saf_fft handle
in	*inputTD*	Time-domain input; N x 1
out	*outputFD*	Frequency-domain output; N x 1

Definition at line 457 of file saf_fft.c.

6.28.2.9 saf_rfft_backward()

```
void saf_rfft_backward (
    void *const hFFT,
    float_complex * inputFD,
    float * outputTD )
```

Performs the backward-FFT operation; use for complex (conjugate symmetric) to real transformations.

Note

Only the first N/2 + 1 bins are needed to be passed in inputFD.

Parameters

in	*hFFT*	saf_rfft handle
in	*inputFD*	Frequency-domain input; (N/2 + 1) x 1
out	*outputTD*	Time-domain output; N x 1

Definition at line 356 of file saf_fft.c.

6.28.2.10 saf_rfft_create()

```
void saf_rfft_create (
    void **const phFFT,
    int N )
```

Creates an instance of saf_rfft; real<->half-complex (conjugate-symmetric) FFT.

Note

Only Even FFT sizes are supported.

Parameters

in	*phFFT*	(&) address of saf_rfft handle
in	*N*	FFT size

Definition at line 245 of file saf_fft.c.

6.28.2.11 saf_rfft_destroy()

```
void saf_rfft_destroy (
    void **const phFFT )
```

Destroys an instance of saf_rfft.

Parameters

in	*phFFT*	(&) address of saf_rfft handle

Definition at line 300 of file saf_fft.c.

6.28.2.12 saf_rfft_forward()

```
void saf_rfft_forward (
    void *const hFFT,
    float * inputTD,
    float_complex * outputFD )
```

Performs the forward-FFT operation; use for real to complex (conjugate symmetric) transformations.

Note

Only the first N/2 + 1 bins are returned in outputFD.

Parameters

in	*hFFT*	saf_rfft handle
in	*inputTD*	Time-domain input; N x 1
out	*outputFD*	Frequency-domain output; (N/2 + 1) x 1

Definition at line 325 of file saf_fft.c.

6.29 framework/modules/saf_utilities/saf_filters.c File Reference

Contains a collection of filter design equations.

```
#include "saf_filters.h"
#include "saf_utilities.h"
```

Functions

- static void applyWindowingFunction (WINDOWING_FUNCTION_TYPES type, int winlength, float ∗x)

  Applies a windowing function (see WINDOWING_FUNCTION_TYPES enum) of length 'winlength', to vector 'x'.
- void getWindowingFunction (WINDOWING_FUNCTION_TYPES type, int winlength, float ∗win)

  Computes the weights of a specific windowing function.
- void getOctaveBandCutoffFreqs (float ∗centreFreqs, int nCentreFreqs, float ∗cutoffFreqs)

  Converts octave band CENTRE frequencies into CUTOFF frequencies.
- void flattenMinphase (float ∗x, int len)

  Equalises input sequence by its minimum phase form, in order to bring its magnitude response to unity.
- void biQuadCoeffs (BIQUAD_FILTER_TYPES filterType, float fc, float fs, float Q, float gain_dB, float b[3], float a[3])

  Calculates 2nd order IIR filter coefficients [1].

- void applyBiQuadFilter (float b[3], float a[3], float w_z_12[2], float ∗signal, int nSamples)

  Applies biQuad filter to an input signal using the direct form II difference equation: https://en.wikipedia.org/wiki/Digital_biquad_filter.
- void evalBiQuadTransferFunction (float b[3], float a[3], float ∗freqs, int nFreqs, float fs, float ∗magnitude_dB, float ∗phase_rad)

  Evaluates the 2nd order IIR transfer function at one or more frequencies, returning its magnitude and/or phase response.
- void FIRCoeffs (FIR_FILTER_TYPES filterType, int order, float fc1, float fc2, float fs, WINDOWING_FUNCTION_TYPES windowType, int scalingFLAG, float ∗h_filt)

  Computes FIR filter coefficients by windowing.
- void FIRFilterbank (int order, float ∗fc, int nCutoffFreq, float sampleRate, WINDOWING_FUNCTION_TYPES windowType, int scalingFLAG, float ∗filterbank)

  Computes a bank of FIR filter coefficients required to divide a signal into frequency bands.

6.29.1 Detailed Description

Contains a collection of filter design equations.

Author

Leo McCormack

Date

01.03.2019

6.29.2 Function Documentation

6.29.2.1 applyBiQuadFilter()

```
void applyBiQuadFilter (
        float b[3],
        float a[3],
        float w_z_12[2],
        float * signal,
        int nSamples )
```

Applies biQuad filter to an input signal using the direct form II difference equation: https://en.wikipedia.org/wiki/Digital_biquad_filter.

Note

input 'signal' is filtered in place (i.e. it becomes the output signal)

Parameters

in	*b*	b filter coefficients; 3 x 1
in	*a*	a filter coefficients; 3 x 1
in,out	*w_z_12*	Previous 2 wn samples (init as 0s); 2 x 1
in,out	*signal*	Signal to be filtered/filtered signal; nSamples x 1
in	*nSamples*	Number of samples in the signal

Definition at line 302 of file saf_filters.c.

6.29.2.2 biQuadCoeffs()

```
void biQuadCoeffs (
    BIQUAD_FILTER_TYPES filterType,
    float fc,
    float fs,
    float Q,
    float gain_dB,
    float b[3],
    float a[3] )
```

Calculates 2nd order IIR filter coefficients [1].

Parameters

in	*filterType*	See 'BIQUAD_FILTER_TYPES' enum
in	*fc*	Centre frequency, Hz
in	*fs*	Sampling frequency, Hz
in	*Q*	Q-factor
in	*gain_dB*	Gain, dB
out	*b*	b filter coefficients; 3 x 1
out	*a*	a filter coefficients; 3 x 1

See also

[1] Zo"lzer, U. (Ed.). (2011). DAFX: digital audio effects. John Wiley & Sons.

Definition at line 186 of file saf_filters.c.

6.29.2.3 evalBiQuadTransferFunction()

```
void evalBiQuadTransferFunction (
    float b[3],
    float a[3],
    float * freqs,
    int nFreqs,
    float fs,
    float * magnitude_dB,
    float * phase_rad )
```

Evaluates the 2nd order IIR transfer function at one or more frequencies, returning its magnitude and/or phase response.

Parameters

in	*b*	b filter coefficients; 3 x 1
in	*a*	a filter coefficients; 3 x 1
in	*freqs*	Frequencies at which to evaluate, Hz; nFreqs x 1
in	*nFreqs*	Number of frequencies at which to avaluate
in	*fs*	Sampling frequency, Hz
out	*magnitude_dB*	Magnitude, dB, at each frequency (set to NULL of not wanted); nFreqs x 1
out	*phase_rad*	Phase, radians, at each frequency (set to NULL of not wanted); nFreqs x 1

Definition at line 324 of file saf_filters.c.

6.29.2.4 FIRCoeffs()

```
void FIRCoeffs (
    FIR_FILTER_TYPES filterType,
    int order,
    float cutoff1,
    float cutoff2,
    float sampleRate,
    WINDOWING_FUNCTION_TYPES windowType,
    int scalingFLAG,
    float * filter )
```

Computes FIR filter coefficients by windowing.

When using the Hamming window, and scalingFLAG=1, the function is numerically identical to the default 'fir1' function in Matlab (when using it in single precision mode) [1].

Note

Input argument 'order' cannot be odd valued.

Some guidelines regarding the approx order (N) for certain filters. i.e. the orders where you actually get the expected -6dB attenuation at the cutoff frequency specified (fs=48kHz, Hamming window, scalingFLAG=1). (Use these figures only just to get a rough idea)

- LPF @ 100Hz - N∼1400
- LPF @ 250Hz - N∼550
- LPF @ 1kHz - N∼150
- LPF @ 4kHz - N∼40
- BPF @ 88-176Hz - N∼2500
- BPF @ 176-354Hz - N∼1600
- BPF @ 707-1410Hz - N∼400
- HPF @ 200Hz - N∼450
- HPF @ 4kHz - N∼60

Parameters

in	*filterType*	See 'FIR_FILTER_TYPES' enum
in	*order*	Filter order (N). Must be even.
in	*cutoff1*	Filter1 cutoff in Hz, for LPF/HPF, and lower cutoff for BPF/BSF
in	*cutoff2*	Filter2 cutoff in Hz, not needed for LPF/HPF, this is the upper cutoff for BPF/BSF
in	*sampleRate*	Sampling rate in Hz
in	*windowType*	See 'WINDOWING_FUNCTION_TYPES' enum
in	*scalingFLAG*	'0' none, '1' scaling applied to ensure passband is at 0dB
out	*filter*	Filter coefficients/weights/taps; (order+1) x 1

See also

[1] "Programs for Digital Signal Processing", IEEE Press John Wiley & Sons, 1979, pg. 5.2-1.

Definition at line 356 of file saf_filters.c.

6.29.2.5 FIRFilterbank()

```
void FIRFilterbank (
    int order,
    float * fc,
    int nCutoffFreqs,
    float sampleRate,
    WINDOWING_FUNCTION_TYPES windowType,
    int scalingFLAG,
    float * filterbank )
```

Computes a bank of FIR filter coefficients required to divide a signal into frequency bands.

Provided the order is sufficient, the sum of the bands should recontruct the original (although, shifted in time due to group delay) e.g fc[1] = { 1000 };

- Band1, &filter[0∗(order+1)] : LPF @ 1kHz
- Band2, &filter[1∗(order+1)] : HPF @ 1kHz

e.g fc[3] = { 1000, 2000, 4000 };

- Band1, &filter[0∗(order+1)] : LPF @ 1kHz
- Band2, &filter[1∗(order+1)] : BPF @ 1-2kHz
- Band3, &filter[2∗(order+1)] : BPF @ 2-4kHz
- Band4, &filter[3∗(order+1)] : HPF @ 4kHz

Parameters

in	*order*	Filter order. Must be even.
in	*fc*	Vector of cutoff frequencies; nCutoffFreqs x 1
in	*nCutoffFreqs*	Number of cutoff frequencies in vector 'fc'.
in	*sampleRate*	Sampling rate in Hz
in	*windowType*	See 'WINDOWING_FUNCTION_TYPES' enum
in	*scalingFLAG*	'0' none, '1' scaling applied to ensure passbands are at 0dB
out	*filterbank*	Filter coefficients/weights/taps; FLAT: (nCutoffFreqs+1) x (order+1)

Definition at line 450 of file saf_filters.c.

6.29.2.6 flattenMinphase()

```
void flattenMinphase (
    float * x,
    int len )
```

Equalises input sequence by its minimum phase form, in order to bring its magnitude response to unity.

Parameters

in,out	*x*	Input; len x 1
in	*len*	Length of input

Definition at line 138 of file saf_filters.c.

6.29.2.7 getOctaveBandCutoffFreqs()

```
void getOctaveBandCutoffFreqs (
    float * centreFreqs,
    int nCentreFreqs,
    float * cutoffFreqs )
```

Converts octave band CENTRE frequencies into CUTOFF frequencies.

The lower and upper CENTRE frequencies only have their upper and lower CUTOFF frequencies computed, respectively. e.g.:

- centreFreqs[6] = { 125, 250, 500, 1000, 2000, 4000 },

becomes:

- cutoffFreqs[5] = { 176, 354, 707, 1410, 2830 }

Passing cutoffFreqs[5] to FIRFilterbank(), will give filter coefficients for the following:

- Band1: LPF @ 176Hz
- Band2: BFP @ 176-354Hz
- Band3: BFP @ 354-707Hz
- Band4: BFP @ 707-1410Hz
- Band5: BFP @ 1410-2830Hz
- Band6: HPF @ 2830Hz

(Basically, band 125Hz also encapsulates everything down to DC, and band 4kHz also encapsulates everything up to Nyquist)

Note

cutoffFreqs vector is shorter than centreFreqs by 1 element.

Parameters

in	*centreFreqs*	Centre frequencies (octave bands); nCentreFreqs x 1
in	*nCentreFreqs*	Number of centre frequencies
out	*cutoffFreqs*	Cutoff frequencies, which encapsulate the specified centre frequencies by 1 octave; (nCentreFreqs-1) x 1

Definition at line 122 of file saf_filters.c.

6.29.2.8 getWindowingFunction()

```
void getWindowingFunction (
        WINDOWING_FUNCTION_TYPES type,
        int winlength,
        float * win )
```

Computes the weights of a specific windowing function.

Weights are symmetric if winlength is odd, and are asymmetric if winlength is even.

i.e. if winlength is even:

- index "winlength/2" = 1, and first value!=last value

if odd:

- index "(winlength-1)/2" = 1, and first value==last value

Parameters

in	*type*	See 'WINDOWING_FUNCTION_TYPES' enum
in	*winlength*	Window length in samples
out	*win*	Windowing function; winlength x 1

Definition at line 108 of file saf_filters.c.

6.30 framework/modules/saf_utilities/saf_filters.h File Reference

Contains a collection of filter design equations.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <float.h>
#include <math.h>
```

Enumerations

- enum _BIQUAD_FILTER_TYPES {
  BIQUAD_FILTER_LPF, BIQUAD_FILTER_HPF, BIQUAD_FILTER_PEAK, BIQUAD_FILTER_LOW_SHELF,
  BIQUAD_FILTER_HI_SHELF }

  Bi-quadratic (second-order) IIR filter design options.
- enum _FIR_FILTER_TYPES { FIR_FILTER_LPF, FIR_FILTER_HPF, FIR_FILTER_BPF, FIR_FILTER_BSF
  }

Finite Impulse Response (FIR) filter design options.

- enum _WINDOWING_FUNCTION_TYPES {
WINDOWING_FUNCTION_RECTANGULAR, WINDOWING_FUNCTION_HAMMING, WINDOWING_FUNCTION_HANN,
WINDOWING_FUNCTION_BARTLETT,
WINDOWING_FUNCTION_BLACKMAN, WINDOWING_FUNCTION_NUTTALL, WINDOWING_FUNCTION_BLACKMAN_NU
WINDOWING_FUNCTION_BLACKMAN_HARRIS }

  Windowing function types.

Functions

- void getWindowingFunction (WINDOWING_FUNCTION_TYPES type, int winlength, float ∗win)

  Computes the weights of a specific windowing function.
- void getOctaveBandCutoffFreqs (float ∗centreFreqs, int nCentreFreqs, float ∗cutoffFreqs)

  Converts octave band CENTRE frequencies into CUTOFF frequencies.
- void flattenMinphase (float ∗x, int len)

  Equalises input sequence by its minimum phase form, in order to bring its magnitude response to unity.
- void biQuadCoeffs (BIQUAD_FILTER_TYPES filterType, float fc, float fs, float Q, float gain_dB, float b[3], float a[3])

  Calculates 2nd order IIR filter coefficients [1].
- void applyBiQuadFilter (float b[3], float a[3], float w_z_12[2], float ∗signal, int nSamples)

  Applies biQuad filter to an input signal using the direct form II difference equation: `https://en.wikipedia.org/wiki/Digital_biquad_filter`.
- void evalBiQuadTransferFunction (float b[3], float a[3], float ∗freqs, int nFreqs, float fs, float ∗magnitude_dB, float ∗phase_rad)

  Evaluates the 2nd order IIR transfer function at one or more frequencies, returning its magnitude and/or phase response.
- void FIRCoeffs (FIR_FILTER_TYPES filterType, int order, float cutoff1, float cutoff2, float sampleRate, WINDOWING_FUNCTION_TYPES windowType, int scalingFLAG, float ∗filter)

  Computes FIR filter coefficients by windowing.
- void FIRFilterbank (int order, float ∗fc, int nCutoffFreqs, float sampleRate, WINDOWING_FUNCTION_TYPES windowType, int scalingFLAG, float ∗filterbank)

  Computes a bank of FIR filter coefficients required to divide a signal into frequency bands.

6.30.1 Detailed Description

Contains a collection of filter design equations.

Author

Leo McCormack

Date

01.03.2019

6.30.2 Enumeration Type Documentation

6.30.2.1 _BIQUAD_FILTER_TYPES `enum _BIQUAD_FILTER_TYPES`

Bi-quadratic (second-order) IIR filter design options.

Enumerator

BIQUAD_FILTER_LPF	low-pass filter
BIQUAD_FILTER_HPF	high-pass filter
BIQUAD_FILTER_PEAK	peaking filter
BIQUAD_FILTER_LOW_SHELF	low-shelving filter
BIQUAD_FILTER_HI_SHELF	high-shelving filter

Definition at line 45 of file saf_filters.h.

6.30.2.2 _FIR_FILTER_TYPES `enum _FIR_FILTER_TYPES`

Finite Impulse Response (FIR) filter design options.

Enumerator

FIR_FILTER_LPF	low-pass filter
FIR_FILTER_HPF	high-pass filter
FIR_FILTER_BPF	band-pass filter
FIR_FILTER_BSF	band-stop filter

Definition at line 57 of file saf_filters.h.

6.30.2.3 _WINDOWING_FUNCTION_TYPES `enum _WINDOWING_FUNCTION_TYPES`

Windowing function types.

Symmetric if winlength is odd, and asymmetric if winlength is even. Windows are evaluated: 0 <= n < winlength. Largely taken from: `https://en.wikipedia.org/wiki/Window_function`

Enumerator

WINDOWING_FUNCTION_RECTANGULAR	Rectangular.
WINDOWING_FUNCTION_HAMMING	Hamming.
WINDOWING_FUNCTION_HANN	Hann.
WINDOWING_FUNCTION_BARTLETT	Bartlett.
WINDOWING_FUNCTION_BLACKMAN	Blackman.
WINDOWING_FUNCTION_NUTTALL	Nuttall.
WINDOWING_FUNCTION_BLACKMAN_NUTTALL	Blackman-Nuttall.
WINDOWING_FUNCTION_BLACKMAN_HARRIS	Blackman-Harris.

Definition at line 72 of file saf_filters.h.

6.30.3 Function Documentation

6.30.3.1 applyBiQuadFilter()

```
void applyBiQuadFilter (
        float b[3],
        float a[3],
        float w_z_12[2],
        float * signal,
        int nSamples )
```

Applies biQuad filter to an input signal using the direct form II difference equation: https://en.wikipedia.org/wiki/Digital_biquad_filter.

Note

input 'signal' is filtered in place (i.e. it becomes the output signal)

Parameters

in	*b*	b filter coefficients; 3 x 1
in	*a*	a filter coefficients; 3 x 1
in,out	*w_z_12*	Previous 2 wn samples (init as 0s); 2 x 1
in,out	*signal*	Signal to be filtered/filtered signal; nSamples x 1
in	*nSamples*	Number of samples in the signal

Definition at line 302 of file saf_filters.c.

6.30.3.2 biQuadCoeffs()

```
void biQuadCoeffs (
        BIQUAD_FILTER_TYPES filterType,
        float fc,
        float fs,
        float Q,
        float gain_dB,
        float b[3],
        float a[3] )
```

Calculates 2nd order IIR filter coefficients [1].

Parameters

in	*filterType*	See 'BIQUAD_FILTER_TYPES' enum
in	*fc*	Centre frequency, Hz
in	*fs*	Sampling frequency, Hz
in	*Q*	Q-factor
in	*gain_dB*	Gain, dB
out	*b*	b filter coefficients; 3 x 1
out	*a*	a filter coefficients; 3 x 1

See also

[1] Zo"lzer, U. (Ed.). (2011). DAFX: digital audio effects. John Wiley & Sons.

Definition at line 186 of file saf_filters.c.

6.30.3.3 evalBiQuadTransferFunction()

```
void evalBiQuadTransferFunction (
    float b[3],
    float a[3],
    float * freqs,
    int nFreqs,
    float fs,
    float * magnitude_dB,
    float * phase_rad )
```

Evaluates the 2nd order IIR transfer function at one or more frequencies, returning its magnitude and/or phase response.

Parameters

in	*b*	b filter coefficients; 3 x 1
in	*a*	a filter coefficients; 3 x 1
in	*freqs*	Frequencies at which to evaluate, Hz; nFreqs x 1
in	*nFreqs*	Number of frequencies at which to avaluate
in	*fs*	Sampling frequency, Hz
out	*magnitude_dB*	Magnitude, dB, at each frequency (set to NULL of not wanted); nFreqs x 1
out	*phase_rad*	Phase, radians, at each frequency (set to NULL of not wanted); nFreqs x 1

Definition at line 324 of file saf_filters.c.

6.30.3.4 FIRCoeffs()

```
void FIRCoeffs (
    FIR_FILTER_TYPES filterType,
    int order,
    float cutoff1,
    float cutoff2,
    float sampleRate,
    WINDOWING_FUNCTION_TYPES windowType,
    int scalingFLAG,
    float * filter )
```

Computes FIR filter coefficients by windowing.

When using the Hamming window, and scalingFLAG=1, the function is numerically identical to the default 'fir1' function in Matlab (when using it in single precision mode) [1].

Note

Input argument 'order' cannot be odd valued.

Some guidelines regarding the approx order (N) for certain filters. i.e. the orders where you actually get the expected -6dB attenuation at the cutoff frequency specified (fs=48kHz, Hamming window, scalingFLAG=1). (Use these figures only just to get a rough idea)

- LPF @ 100Hz - N∼1400
- LPF @ 250Hz - N∼550
- LPF @ 1kHz - N∼150
- LPF @ 4kHz - N∼40
- BPF @ 88-176Hz - N∼2500
- BPF @ 176-354Hz - N∼1600
- BPF @ 707-1410Hz - N∼400
- HPF @ 200Hz - N∼450
- HPF @ 4kHz - N∼60

Parameters

in	*filterType*	See 'FIR_FILTER_TYPES' enum
in	*order*	Filter order (N). Must be even.
in	*cutoff1*	Filter1 cutoff in Hz, for LPF/HPF, and lower cutoff for BPF/BSF
in	*cutoff2*	Filter2 cutoff in Hz, not needed for LPF/HPF, this is the upper cutoff for BPF/BSF
in	*sampleRate*	Sampling rate in Hz
in	*windowType*	See 'WINDOWING_FUNCTION_TYPES' enum
in	*scalingFLAG*	'0' none, '1' scaling applied to ensure passband is at 0dB
out	*filter*	Filter coefficients/weights/taps; (order+1) x 1

See also

[1] "Programs for Digital Signal Processing", IEEE Press John Wiley & Sons, 1979, pg. 5.2-1.

Definition at line 356 of file saf_filters.c.

6.30.3.5 FIRFilterbank()

```
void FIRFilterbank (
        int order,
        float * fc,
        int nCutoffFreqs,
        float sampleRate,
        WINDOWING_FUNCTION_TYPES windowType,
        int scalingFLAG,
        float * filterbank )
```

Computes a bank of FIR filter coefficients required to divide a signal into frequency bands.

Provided the order is sufficient, the sum of the bands should recontruct the original (although, shifted in time due to group delay) e.g fc[1] = { 1000 };

- Band1, &filter[0∗(order+1)] : LPF @ 1kHz
- Band2, &filter[1∗(order+1)] : HPF @ 1kHz

e.g fc[3] = { 1000, 2000, 4000 };

- Band1, &filter[0∗(order+1)] : LPF @ 1kHz
- Band2, &filter[1∗(order+1)] : BPF @ 1-2kHz
- Band3, &filter[2∗(order+1)] : BPF @ 2-4kHz
- Band4, &filter[3∗(order+1)] : HPF @ 4kHz

Parameters

`in`	*order*	Filter order. Must be even.
`in`	*fc*	Vector of cutoff frequencies; nCutoffFreqs x 1
`in`	*nCutoffFreqs*	Number of cutoff frequencies in vector 'fc'.
`in`	*sampleRate*	Sampling rate in Hz
`in`	*windowType*	See 'WINDOWING_FUNCTION_TYPES' enum
`in`	*scalingFLAG*	'0' none, '1' scaling applied to ensure passbands are at 0dB
`out`	*filterbank*	Filter coefficients/weights/taps; FLAT: (nCutoffFreqs+1) x (order+1)

Definition at line 450 of file saf_filters.c.

6.30.3.6 flattenMinphase()

```
void flattenMinphase (
    float * x,
    int len )
```

Equalises input sequence by its minimum phase form, in order to bring its magnitude response to unity.

Parameters

`in,out`	*x*	Input; len x 1
`in`	*len*	Length of input

Definition at line 138 of file saf_filters.c.

6.30.3.7 getOctaveBandCutoffFreqs()

```
void getOctaveBandCutoffFreqs (
    float * centreFreqs,
    int nCentreFreqs,
    float * cutoffFreqs )
```

Converts octave band CENTRE frequencies into CUTOFF frequencies.

The lower and upper CENTRE frequencies only have their upper and lower CUTOFF frequencies computed, respectively. e.g.:

- centreFreqs[6] = { 125, 250, 500, 1000, 2000, 4000 },

becomes:

- cutoffFreqs[5] = { 176, 354, 707, 1410, 2830 }

Passing cutoffFreqs[5] to FIRFilterbank(), will give filter coefficients for the following:

- Band1: LPF @ 176Hz
- Band2: BFP @ 176-354Hz
- Band3: BFP @ 354-707Hz
- Band4: BFP @ 707-1410Hz
- Band5: BFP @ 1410-2830Hz
- Band6: HPF @ 2830Hz

(Basically, band 125Hz also encapsulates everything down to DC, and band 4kHz also encapsulates everything up to Nyquist)

Note

cutoffFreqs vector is shorter than centreFreqs by 1 element.

Parameters

in	*centreFreqs*	Centre frequencies (octave bands); nCentreFreqs x 1
in	*nCentreFreqs*	Number of centre frequencies
out	*cutoffFreqs*	Cutoff frequencies, which encapsulate the specified centre frequencies by 1 octave; (nCentreFreqs-1) x 1

Definition at line 122 of file saf_filters.c.

6.30.3.8 getWindowingFunction()

```
void getWindowingFunction (
        WINDOWING_FUNCTION_TYPES type,
        int winlength,
        float * win )
```

Computes the weights of a specific windowing function.

Weights are symmetric if winlength is odd, and are asymmetric if winlength is even.

i.e. if winlength is even:

- index "winlength/2" = 1, and first value!=last value

if odd:

- index "(winlength-1)/2" = 1, and first value==last value

Parameters

in	*type*	See 'WINDOWING_FUNCTION_TYPES' enum
in	*winlength*	Window length in samples
out	*win*	Windowing function; winlength x 1

Definition at line 108 of file saf_filters.c.

6.31 framework/modules/saf_utilities/saf_loudspeaker_presets.c File Reference

Comprises the directions of loudspeaker arrays and (nearly) uniform spherical grids.

```
#include "saf_loudspeaker_presets.h"
```

Variables

- const float __mono_dirs_deg [1][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a mono setup.
- const float __stereo_dirs_deg [2][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a stereo setup.
- const float __5pX_dirs_deg [5][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 5.x setup.
- const float __7pX_dirs_deg [7][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 7.x setup.
- const float __8pX_dirs_deg [8][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 8.x setup.
- const float __9pX_dirs_deg [9][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 9.x setup.
- const float __10pX_dirs_deg [10][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 10.x setup.
- const float __11pX_dirs_deg [11][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 11.x setup.
- const float __11pX_7_4_dirs_deg [11][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 7.4.x setup.
- const float __13pX_dirs_deg [13][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 13.x setup.
- const float __22pX_dirs_deg [22][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 22.x setup.
- const float __Aalto_MCC_dirs_deg [45][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the multi-channel anechoic chamber (MCC), at Aalto University.
- const float __Aalto_MCCsubset_dirs_deg [37][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the multi-channel anechoic chamber (MCC) sub-set, at Aalto University.
- const float __Aalto_Apaja_dirs_deg [29][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the audio-visual listening room (Apaja), at Aalto University.
- const float __Aalto_LR_dirs_deg [13][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the ITU standard listening room (LR), at Aalto University.

- const float __DTU_AVIL_dirs_deg [64][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the Audio Visual Immersion Lab (AVIL), at the Technical University of Denmark (DTU)
- const float __Zylia_Lab_dirs_deg [22][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the 22.x setup, at Zylia Labs.
- const float default_LScoords64_rad [64][2]

  Default Loudspeaker directions [azimuth, Elevation] in degrees.
- const float __Tdesign_degree_1_dirs_deg [2][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 1.
- const float __Tdesign_degree_2_dirs_deg [4][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 2.
- const float __Tdesign_degree_3_dirs_deg [6][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 3.
- const float __Tdesign_degree_4_dirs_deg [12][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 4.
- const float __Tdesign_degree_5_dirs_deg [12][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 5.
- const float __Tdesign_degree_6_dirs_deg [24][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 6.
- const float __Tdesign_degree_7_dirs_deg [24][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 7.
- const float __Tdesign_degree_8_dirs_deg [36][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 8.
- const float __Tdesign_degree_9_dirs_deg [48][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 9.
- const float __Tdesign_degree_10_dirs_deg [60][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 10.
- const float __Tdesign_degree_11_dirs_deg [70][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 11.
- const float __Tdesign_degree_12_dirs_deg [84][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 12.
- const float __Tdesign_degree_13_dirs_deg [94][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 13.
- const float __Tdesign_degree_14_dirs_deg [108][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 14.
- const float __Tdesign_degree_15_dirs_deg [120][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 15.
- const float __Tdesign_degree_16_dirs_deg [144][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 16.
- const float __Tdesign_degree_17_dirs_deg [156][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 17.
- const float __Tdesign_degree_18_dirs_deg [180][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 18.
- const float __Tdesign_degree_19_dirs_deg [204][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 19.
- const float __Tdesign_degree_20_dirs_deg [216][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 20.
- const float __Tdesign_degree_21_dirs_deg [240][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 21.
- const float __Tdesign_degree_30_dirs_deg [480][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 30.

- const float __Tdesign_degree_40_dirs_deg [840][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 40.
- const float __Tdesign_degree_50_dirs_deg [1296][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 50.
- const float __Tdesign_degree_100_dirs_deg [5100][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 100.
- const int __Tdesign_nPoints_per_degree [21]

  Number of points in each t-design (up to degree 21 only).
- const float ∗ __HANDLES_Tdesign_dirs_deg [21]

  minimum T-design HANDLES (up to degree 21 only).
- const float __SphCovering_4_dirs_deg [4][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 4 dirs.
- const float __SphCovering_5_dirs_deg [5][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 5 dirs.
- const float __SphCovering_6_dirs_deg [6][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 6 dirs.
- const float __SphCovering_7_dirs_deg [7][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 7 dirs.
- const float __SphCovering_8_dirs_deg [8][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 8 dirs.
- const float __SphCovering_9_dirs_deg [9][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 9 dirs.
- const float __SphCovering_10_dirs_deg [10][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 10 dirs.
- const float __SphCovering_11_dirs_deg [11][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 11 dirs.
- const float __SphCovering_12_dirs_deg [12][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 12 dirs.
- const float __SphCovering_13_dirs_deg [13][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 13 dirs.
- const float __SphCovering_14_dirs_deg [14][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 14 dirs.
- const float __SphCovering_15_dirs_deg [15][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 15 dirs.
- const float __SphCovering_16_dirs_deg [16][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 16 dirs.
- const float __SphCovering_17_dirs_deg [17][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 17 dirs.
- const float __SphCovering_18_dirs_deg [18][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 18 dirs.
- const float __SphCovering_19_dirs_deg [19][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 19 dirs.
- const float __SphCovering_20_dirs_deg [20][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 20 dirs.
- const float __SphCovering_21_dirs_deg [21][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 21 dirs.
- const float __SphCovering_22_dirs_deg [22][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 22 dirs.
- const float __SphCovering_23_dirs_deg [23][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 23 dirs.
- const float __SphCovering_24_dirs_deg [24][2]

Directions [azimuth, Elevation] in degrees, for sphere covering: 24 dirs.
- const float __SphCovering_25_dirs_deg [25][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 25 dirs.
- const float __SphCovering_26_dirs_deg [26][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 26 dirs.
- const float __SphCovering_27_dirs_deg [27][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 27 dirs.
- const float __SphCovering_28_dirs_deg [28][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 28 dirs.
- const float __SphCovering_29_dirs_deg [29][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 29 dirs.
- const float __SphCovering_30_dirs_deg [30][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 30 dirs.
- const float __SphCovering_31_dirs_deg [31][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 31 dirs.
- const float __SphCovering_32_dirs_deg [32][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 32 dirs.
- const float __SphCovering_33_dirs_deg [33][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 33 dirs.
- const float __SphCovering_34_dirs_deg [34][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 34 dirs.
- const float __SphCovering_35_dirs_deg [35][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 35 dirs.
- const float __SphCovering_36_dirs_deg [36][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 36 dirs.
- const float __SphCovering_37_dirs_deg [37][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 37 dirs.
- const float __SphCovering_38_dirs_deg [38][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 38 dirs.
- const float __SphCovering_39_dirs_deg [39][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 39 dirs.
- const float __SphCovering_40_dirs_deg [40][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 40 dirs.
- const float __SphCovering_41_dirs_deg [41][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 41 dirs.
- const float __SphCovering_42_dirs_deg [42][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 42 dirs.
- const float __SphCovering_43_dirs_deg [43][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 43 dirs.
- const float __SphCovering_44_dirs_deg [44][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 44 dirs.
- const float __SphCovering_45_dirs_deg [45][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 45 dirs.
- const float __SphCovering_46_dirs_deg [46][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 46 dirs.
- const float __SphCovering_47_dirs_deg [47][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 47 dirs.
- const float __SphCovering_48_dirs_deg [48][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 48 dirs.
- const float __SphCovering_49_dirs_deg [49][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 49 dirs.

- const float __SphCovering_50_dirs_deg [50][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 50 dirs.
- const float __SphCovering_51_dirs_deg [51][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 51 dirs.
- const float __SphCovering_52_dirs_deg [52][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 52 dirs.
- const float __SphCovering_53_dirs_deg [53][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 53 dirs.
- const float __SphCovering_54_dirs_deg [54][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 54 dirs.
- const float __SphCovering_55_dirs_deg [55][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 55 dirs.
- const float __SphCovering_56_dirs_deg [56][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 56 dirs.
- const float __SphCovering_57_dirs_deg [57][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 57 dirs.
- const float __SphCovering_58_dirs_deg [58][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 58 dirs.
- const float __SphCovering_59_dirs_deg [59][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 59 dirs.
- const float __SphCovering_60_dirs_deg [60][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 60 dirs.
- const float __SphCovering_61_dirs_deg [61][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 61 dirs.
- const float __SphCovering_62_dirs_deg [62][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 62 dirs.
- const float __SphCovering_63_dirs_deg [63][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 63 dirs.
- const float __SphCovering_64_dirs_deg [64][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 64 dirs.
- const float ∗ __HANDLES_SphCovering_dirs_deg [64]

  Sphere covering handles (between 4..64 points only)
- const float __geosphere_ico_0_0_dirs_deg [12][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 0.
- const float __geosphere_ico_1_0_dirs_deg [32][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 1.
- const float __geosphere_ico_2_0_dirs_deg [42][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 2.
- const float __geosphere_ico_3_0_dirs_deg [92][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 3.
- const float __geosphere_ico_4_0_dirs_deg [162][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 4.
- const float __geosphere_ico_5_0_dirs_deg [252][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 5.
- const float __geosphere_ico_6_0_dirs_deg [362][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 6.
- const float __geosphere_ico_7_0_dirs_deg [492][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 7.
- const float __geosphere_ico_8_0_dirs_deg [642][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 8.
- const float __geosphere_ico_9_0_dirs_deg [812][2]

Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 9.
- const float __geosphere_ico_10_0_dirs_deg [1002][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 10.
- const float __geosphere_ico_11_0_dirs_deg [1212][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 11.
- const float __geosphere_ico_12_0_dirs_deg [1442][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 12.
- const float __geosphere_ico_13_0_dirs_deg [1692][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 13.
- const float __geosphere_ico_14_0_dirs_deg [1962][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 14.
- const float __geosphere_ico_15_0_dirs_deg [2252][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 15.
- const float __geosphere_ico_16_0_dirs_deg [2562][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 16.
- const float __geosphere_oct_0_0_dirs_deg [6][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 0.
- const float __geosphere_oct_1_0_dirs_deg [14][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 1.
- const float __geosphere_oct_2_0_dirs_deg [18][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 2.
- const float __geosphere_oct_3_0_dirs_deg [38][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 3.
- const float __geosphere_oct_4_0_dirs_deg [66][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 4.
- const float __geosphere_oct_5_0_dirs_deg [102][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 5.
- const float __geosphere_oct_6_0_dirs_deg [146][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 6.
- const float __geosphere_oct_7_0_dirs_deg [198][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 7.
- const float __geosphere_oct_8_0_dirs_deg [258][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 8.
- const float __geosphere_oct_9_0_dirs_deg [326][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 9.
- const float __geosphere_oct_10_0_dirs_deg [402][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 10.
- const float __geosphere_oct_11_0_dirs_deg [486][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 11.
- const float __geosphere_oct_12_0_dirs_deg [578][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 12.
- const float __geosphere_oct_13_0_dirs_deg [678][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 13.
- const float __geosphere_oct_14_0_dirs_deg [786][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 14.
- const float __geosphere_oct_15_0_dirs_deg [902][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 15.
- const float __geosphere_oct_16_0_dirs_deg [1026][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 16.
- const float ∗ __HANDLES_geosphere_ico_dirs_deg [17]

  3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])

- const float * __HANDLES_geosphere_oct_dirs_deg [17]
  3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])
- const int __geosphere_ico_nPoints [17]
  3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])
- const int __geosphere_oct_nPoints [17]
  3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])

6.31.1 Detailed Description

Comprises the directions of loudspeaker arrays and (nearly) uniform spherical grids.

Author

Leo McCormack

Date

11.07.2016

6.31.2 Variable Documentation

6.31.2.1 __10pX_dirs_deg `const float __10pX_dirs_deg[10][2]`

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 110.0f,    0.0f},
  { -110.0f,    0.0f},
  { 30.0f,    45.0f},
  { -30.0f,    45.0f},
  { 0.0f,    90.0f},
  { 110.0f,    45.0f},
  { -110.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 10.x setup.

Definition at line 79 of file saf_loudspeaker_presets.c.

6.31.2.2 __11pX_7_4_dirs_deg `const float __11pX_7_4_dirs_deg[11][2]`

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 110.0f,    0.0f},
  { -110.0f,    0.0f},
  { 135.0f,    0.0f},
  { -135.0f,    0.0f},
  { 30.0f,    45.0f},
  { -30.0f,    45.0f},
  { 110.0f,    45.0f},
  { -110.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 7.4.x setup.

Definition at line 106 of file saf_loudspeaker_presets.c.

6.31.2.3 __11pX_dirs_deg const float __11pX_dirs_deg[11][2]

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 110.0f,    0.0f},
  { -110.0f,    0.0f},
  { 30.0f,    45.0f},
  { -30.0f,    45.0f},
  { 0.0f,    45.0f},
  { 0.0f,    90.0f},
  { 110.0f,    45.0f},
  { -110.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 11.x setup.

Definition at line 92 of file saf_loudspeaker_presets.c.

6.31.2.4 __13pX_dirs_deg const float __13pX_dirs_deg[13][2]

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 135.0f,    0.0f},
  { -135.0f,    0.0f},
  { 30.0f,    45.0f},
  { -30.0f,    45.0f},
  { 0.0f,    45.0f},
  { 0.0f,    90.0f},
  { 90.0f,    45.0f},
  { -90.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 13.x setup.

Definition at line 120 of file saf_loudspeaker_presets.c.

6.31.2.5 __22pX_dirs_deg const float __22pX_dirs_deg[22][2]

Initial value:

```
=
{ { 45.0f,    0.0f},
  { -45.0f,    0.0f},
  { 0.0f,    0.0f},
  { 135.0f,    0.0f},
  { -135.0f,    0.0f},
  { 15.0f,    0.0f},
  { -15.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 180.0f,    0.0f},
  { 45.0f,    45.0f},
  { -45.0f,    45.0f},
  { 0.0f,    45.0f},
  { 135.0f,    45.0f},
  { -135.0f,    45.0f},
  { 90.0f,    45.0f},
  { -90.0f,    45.0f},
  { 180.0f,    45.0f},
  { 0.0f,    90.0f},
  { 45.0f,    -30.0f},
  { -45.0f,    -30.0f},
  { 0.0f,    -30.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 22.x setup.

Definition at line 136 of file saf_loudspeaker_presets.c.

6.31.2.6 __5pX_dirs_deg const float __5pX_dirs_deg[5][2]

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 110.0f,    0.0f},
  { -110.0f,    0.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 5.x setup.

Definition at line 38 of file saf_loudspeaker_presets.c.

6.31.2.7 __7pX_dirs_deg const float __7pX_dirs_deg[7][2]

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 135.0f,    0.0f},
  { -135.0f,    0.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 7.x setup.

Definition at line 46 of file saf_loudspeaker_presets.c.

6.31.2.8 __8pX_dirs_deg const float __8pX_dirs_deg[8][2]

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 110.0f,    0.0f},
  { -110.0f,    0.0f},
  { 30.0f,    45.0f},
  { -30.0f,    45.0f},
  { 110.0f,    45.0f},
  { -110.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 8.x setup.

Definition at line 56 of file saf_loudspeaker_presets.c.

6.31.2.9 __9pX_dirs_deg const float __9pX_dirs_deg[9][2]

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { 0.0f,    0.0f},
  { 110.0f,    0.0f},
  { -110.0f,    0.0f},
  { 30.0f,    45.0f},
  { -30.0f,    45.0f},
  { 110.0f,    45.0f},
  { -110.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a 9.x setup.

Definition at line 67 of file saf_loudspeaker_presets.c.

6.31.2.10 __Aalto_Apaja_dirs_deg const float __Aalto_Apaja_dirs_deg[29][2]

Initial value:

```
=
{ { 18.0f,     0.0f},
  { 54.0f,     0.0f},
  { 90.0f,     0.0f},
  { 126.0f,     0.0f},
  { 166.0f,     0.0f},
  { -166.0f,     0.0f},
  { -126.0f,     0.0f},
  { -90.0f,     0.0f},
  { -54.0f,     0.0f},
  { -18.0f,     0.0f},
  { 18.0f,    -25.0f},
  { 54.0f,    -25.0f},
  { 90.0f,    -25.0f},
  { 144.0f,    -25.0f},
  { -144.0f,    -25.0f},
  { -90.0f,    -25.0f},
  { -54.0f,    -25.0f},
  { -18.0f,    -25.0f},
  { 18.0f,    25.0f},
  { 54.0f,    25.0f},
  { 90.0f,    25.0f},
  { -90.0f,    25.0f},
  { -54.0f,    25.0f},
  { -18.0f,    25.0f},
  { 40.0f,    45.0f},
  { 137.0f,    45.0f},
  { -137.0f,    45.0f},
  { -40.0f,    45.0f},
  { 0.0f,    90.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for the audio-visual listening room (Apaja), at Aalto University.

Definition at line 248 of file saf_loudspeaker_presets.c.

6.31.2.11 __Aalto_LR_dirs_deg const float __Aalto_LR_dirs_deg[13][2]

Initial value:

```
=
{ { 30.0f,     0.0f},
  { -30.0f,     0.0f},
  { 0.0f,     0.0f},
  { 70.0f,     0.0f},
  { 110.0f,     0.0f},
  { -70.0f,     0.0f},
  { -110.0f,     0.0f},
  { 160.0f,     0.0f},
  { -160.0f,     0.0f},
  { 45.0f,    45.0f},
  { 135.0f,    45.0f},
  { -45.0f,    45.0f},
  { -135.0f,    45.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for the ITU standard listening room (LR), at Aalto University.

Definition at line 280 of file saf_loudspeaker_presets.c.

6.31.2.12 __geosphere_ico_0_0_dirs_deg `const float __geosphere_ico_0_0_dirs_deg[12][2]`

Initial value:

```
=
{ { 31.7174744114610f,    0.0f},
  { 148.282525588539f,    0.0f},
  { -148.282525588539f,    0.0f},
  { -31.7174744114610f,    0.0f},
  { 0.0f,    58.2825255885390f},
  { 0.0f,    -58.2825255885390f},
  { 180.0f,    -58.2825255885390f},
  { 180.0f,    58.2825255885390f},
  { 90.0f,    31.7174744114610f},
  { -90.0f,    31.7174744114610f},
  { -90.0f,    -31.7174744114610f},
  { 90.0f,    -31.7174744114610f}}
```

Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 0.

See also

3LD - Library for Loudspeaker Layout Design; release 2, 2006/03/15 Copyright (c) 2006 Florian Hollerweger (floholl_AT_sbox.tugraz.at) and (c) 2002 Darren Weber.

Definition at line 12441 of file saf_loudspeaker_presets.c.

6.31.2.13 __geosphere_ico_nPoints `const int __geosphere_ico_nPoints[17]`

Initial value:

```
=
{   12, 32, 42, 92, 162, 252, 362, 492, 642, 812, 1002, 1212, 1442, 1692, 1962, 2252, 2562   }
```

3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.

```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_ico_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_ico_nPoints[degree];
```

Definition at line 33637 of file saf_loudspeaker_presets.c.

6.31.2.14 __geosphere_oct_0_0_dirs_deg `const float __geosphere_oct_0_0_dirs_deg[6][2]`

Initial value:

```
=
{ { 0.0f,    0.0f},
  { 180.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 0.0f,    90.0f},
  { 0.0f,    -90.0f}}
```

Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 0.

See also

3LD - Library for Loudspeaker Layout Design; release 2, 2006/03/15 Copyright (c) 2006 Florian Hollerweger (floholl_AT_sbox.tugraz.at) and (c) 2002 Darren Weber.

Definition at line 27516 of file saf_loudspeaker_presets.c.

6.31.2.15 __geosphere_oct_1_0_dirs_deg const float __geosphere_oct_1_0_dirs_deg[14][2]

Initial value:

```
=
{ { 0.0f,    0.0f},
  { 180.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 0.0f,    90.0f},
  { 0.0f,    -90.0f},
  { 45.0000000000000f,    35.2643896827547f},
  { 135.0f,    35.2643896827547f},
  { -135.0f,    35.2643896827547f},
  { -45.0000000000000f,    35.2643896827547f},
  { 45.0000000000000f,    -35.2643896827547f},
  { 135.0f,    -35.2643896827547f},
  { -135.0f,    -35.2643896827547f},
  { -45.0000000000000f,    -35.2643896827547f}}
```

Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 1.

Definition at line 27525 of file saf_loudspeaker_presets.c.

6.31.2.16 __geosphere_oct_2_0_dirs_deg const float __geosphere_oct_2_0_dirs_deg[18][2]

Initial value:

```
=
{ { 0.0f,    0.0f},
  { 180.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 0.0f,    90.0f},
  { 0.0f,    -90.0f},
  { 0.0f,    45.0f},
  { 45.0f,    0.0f},
  { 90.0f,    45.0f},
  { 135.0f,    0.0f},
  { 180.0f,    45.0f},
  { -135.0f,    0.0f},
  { -90.0f,    45.0f},
  { -45.0f,    0.0f},
  { 0.0f,    -45.0f},
  { 90.0f,    -45.0f},
  { 180.0f,    -45.0f},
  { -90.0f,    -45.0f}}
```

Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 2.

Definition at line 27542 of file saf_loudspeaker_presets.c.

6.31.2.17 __geosphere_oct_nPoints const int __geosphere_oct_nPoints[17]

Initial value:

```
=
{   6, 14, 18, 38, 66, 102, 146, 198, 258, 326, 402, 486, 578, 678, 786, 902, 1026   }
```

3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.

```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_oct_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_oct_nPoints[degree];
```

Definition at line 33641 of file saf_loudspeaker_presets.c.

6.31.2.18 __HANDLES_geosphere_ico_dirs_deg `const float* __HANDLES_geosphere_ico_dirs_deg[17]`

Initial value:

```
=
{ &__geosphere_ico_0_0_dirs_deg[0][0],
  &__geosphere_ico_1_0_dirs_deg[0][0],
  &__geosphere_ico_2_0_dirs_deg[0][0],
  &__geosphere_ico_3_0_dirs_deg[0][0],
  &__geosphere_ico_4_0_dirs_deg[0][0],
  &__geosphere_ico_5_0_dirs_deg[0][0],
  &__geosphere_ico_6_0_dirs_deg[0][0],
  &__geosphere_ico_7_0_dirs_deg[0][0],
  &__geosphere_ico_8_0_dirs_deg[0][0],
  &__geosphere_ico_9_0_dirs_deg[0][0],
  &__geosphere_ico_10_0_dirs_deg[0][0],
  &__geosphere_ico_11_0_dirs_deg[0][0],
  &__geosphere_ico_12_0_dirs_deg[0][0],
  &__geosphere_ico_13_0_dirs_deg[0][0],
  &__geosphere_ico_14_0_dirs_deg[0][0],
  &__geosphere_ico_15_0_dirs_deg[0][0],
  &__geosphere_ico_16_0_dirs_deg[0][0]}
```

3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.

```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_ico_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_ico_nPoints[degree];
```

Definition at line 33597 of file saf_loudspeaker_presets.c.

6.31.2.19 __HANDLES_geosphere_oct_dirs_deg `const float* __HANDLES_geosphere_oct_dirs_deg[17]`

Initial value:

```
=
{ &__geosphere_oct_0_0_dirs_deg[0][0],
  &__geosphere_oct_1_0_dirs_deg[0][0],
  &__geosphere_oct_2_0_dirs_deg[0][0],
  &__geosphere_oct_3_0_dirs_deg[0][0],
  &__geosphere_oct_4_0_dirs_deg[0][0],
  &__geosphere_oct_5_0_dirs_deg[0][0],
  &__geosphere_oct_6_0_dirs_deg[0][0],
  &__geosphere_oct_7_0_dirs_deg[0][0],
  &__geosphere_oct_8_0_dirs_deg[0][0],
  &__geosphere_oct_9_0_dirs_deg[0][0],
  &__geosphere_oct_10_0_dirs_deg[0][0],
  &__geosphere_oct_11_0_dirs_deg[0][0],
  &__geosphere_oct_12_0_dirs_deg[0][0],
  &__geosphere_oct_13_0_dirs_deg[0][0],
  &__geosphere_oct_14_0_dirs_deg[0][0],
  &__geosphere_oct_15_0_dirs_deg[0][0],
  &__geosphere_oct_16_0_dirs_deg[0][0]}
```

3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.

```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_oct_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_oct_nPoints[degree];
```

Definition at line 33617 of file saf_loudspeaker_presets.c.

6.31.2.20 __HANDLES_SphCovering_dirs_deg `const float* __HANDLES_SphCovering_dirs_deg[64]`

Sphere covering handles (between 4..64 points only)

Access as, e.g.

```
const int numPoints = 44; // between 4..64 points
float* sphCov_dirs_deg = __HANDLES_SphCovering_dirs_deg [numPoints-1];
```

Definition at line 12374 of file saf_loudspeaker_presets.c.

6.31.2.21 __HANDLES_Tdesign_dirs_deg const float∗ __HANDLES_Tdesign_dirs_deg[21]

Initial value:

```
=
{ &__Tdesign_degree_1_dirs_deg[0][0],
  &__Tdesign_degree_2_dirs_deg[0][0],
  &__Tdesign_degree_3_dirs_deg[0][0],
  &__Tdesign_degree_4_dirs_deg[0][0],
  &__Tdesign_degree_5_dirs_deg[0][0],
  &__Tdesign_degree_6_dirs_deg[0][0],
  &__Tdesign_degree_7_dirs_deg[0][0],
  &__Tdesign_degree_8_dirs_deg[0][0],
  &__Tdesign_degree_9_dirs_deg[0][0],
  &__Tdesign_degree_10_dirs_deg[0][0],
  &__Tdesign_degree_11_dirs_deg[0][0],
  &__Tdesign_degree_12_dirs_deg[0][0],
  &__Tdesign_degree_13_dirs_deg[0][0],
  &__Tdesign_degree_14_dirs_deg[0][0],
  &__Tdesign_degree_15_dirs_deg[0][0],
  &__Tdesign_degree_16_dirs_deg[0][0],
  &__Tdesign_degree_17_dirs_deg[0][0],
  &__Tdesign_degree_18_dirs_deg[0][0],
  &__Tdesign_degree_19_dirs_deg[0][0],
  &__Tdesign_degree_20_dirs_deg[0][0],
  &__Tdesign_degree_21_dirs_deg[0][0]}
```

minimum T-design HANDLES (up to degree 21 only).

Access as, e.g.

```
const int tdesign_degree = 7;
float* tdesign_dirs_deg = __HANDLES_Tdesign_dirs_deg [tdesign_degree-1];
int num_Tdesign_dirs = __Tdesign_nPoints_per_degree  [tdesign_degree-1];
```

Definition at line 10093 of file saf_loudspeaker_presets.c.

6.31.2.22 __mono_dirs_deg const float __mono_dirs_deg[1][2]

Initial value:

```
=
{ { 0.0f,    0.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a mono setup.

Definition at line 29 of file saf_loudspeaker_presets.c.

6.31.2.23 __SphCovering_10_dirs_deg const float __SphCovering_10_dirs_deg[10][2]

Initial value:

```
=
{ { 132.152715446924f,    -39.0184258484091f},
  { 62.9623321069262f,    9.21831796585981f},
  { 130.130174430113f,    26.4844011221272f},
  { -146.740220910955f,    51.0642905332395f},
  { 33.2573352183686f,    -51.0642905332395f},
  { 8.69635341449563f,    61.7190136914923f},
  { -158.434289509575f,    -13.6656163716653f},
  { -16.8890769270713f,    -0.849180748163393f},
  { -91.2893654981941f,    -53.5709808869368f},
  { -81.6980520077041f,    11.4454049155333f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 10 dirs.

Definition at line 10174 of file saf_loudspeaker_presets.c.

6.31.2.24 __SphCovering_11_dirs_deg const float __SphCovering_11_dirs_deg[11][2]

Initial value:

```
=
{ { 111.113705209721f,    -75.3725979494598f},
  { 43.5510949656890f,    21.3770553363310f},
  { -124.165683782801f,    -29.8889163407945f},
  { 91.2721767643401f,    67.9871719702235f},
  { 16.3757067426341f,    -30.0705439618510f},
  { -58.3672105899770f,    -25.2989514440015f},
  { -112.895603952577f,    36.2183811036047f},
  { 102.267236852901f,    -12.7070579804114f},
  { 178.768561658768f,    -13.2055312621752f},
  { -34.5997753323602f,    32.6041633319195f},
  { 158.050407786838f,    33.4286495991128f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 11 dirs.

Definition at line 10187 of file saf_loudspeaker_presets.c.

6.31.2.25 __SphCovering_12_dirs_deg const float __SphCovering_12_dirs_deg[12][2]

Initial value:

```
=
{ { 0.0f,    -31.7172246650570f},
  { -58.2812669207074f,    0.0f},
  { -90.0002104591497f,    58.2812669207074f},
  { 0.0f,    31.7172246650570f},
  { -121.719153997592f,    0.0f},
  { 90.0002104591497f,    -58.2812669207074f},
  { 180.0f,    -31.7172246650570f},
  { 121.719153997592f,    0.0f},
  { 90.0002104591497f,    58.2812669207074f},
  { 180.0f,    31.7172246650570f},
  { 58.2812669207074f,    0.0f},
  { -90.0002104591497f,    -58.2812669207074f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 12 dirs.

Definition at line 10201 of file saf_loudspeaker_presets.c.

6.31.2.26 __SphCovering_13_dirs_deg const float __SphCovering_13_dirs_deg[13][2]

Initial value:

```
=
{ { 5.50772869303358f,    23.7794673713146f},
  { 142.546169850598f,    -67.9012283009539f},
  { -42.2682424623911f,    68.0788452174444f},
  { -93.0999121308075f,    -41.3142677334983f},
  { -45.8286022013340f,    4.83301359348752f},
  { 44.9027024044026f,    -2.11455803870982f},
  { -2.49288207083470f,    -45.6252021840626f},
  { -108.564043021388f,    19.9635684557433f},
  { -161.018329165615f,    -16.1694419363870f},
  { -179.788426534101f,    44.3910510933508f},
  { 135.991532674301f,    -4.76592023567770f},
  { 89.3470385727006f,    42.6813450326804f},
  { 85.5884354366424f,    -27.0178248293940f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 13 dirs.

Definition at line 10216 of file saf_loudspeaker_presets.c.

6.31.2.27 __SphCovering_14_dirs_deg const float __SphCovering_14_dirs_deg[14][2]

Initial value:

```
=
{ { -45.9265779843014f,    -54.4951618104829f},
  { 126.881503731721f,    -63.0253574643906f},
  { -177.490865775626f,    -14.0007966818168f},
  { 79.0681757280536f,    -10.6495665380966f},
  { 175.880854371309f,    48.0665753491150f},
  { 131.052636480273f,    -0.752580063904336f},
  { 27.8898665935831f,    -35.8218943093742f},
  { 89.9371851016853f,    50.9783468639699f},
  { -22.7137658723712f,    4.77726480002129f},
  { -53.1200631021689f,    63.0253574643906f},
  { -125.798613498924f,    22.3717100686781f},
  { -75.3840571053624f,    2.88272255464171f},
  { -124.950635962130f,    -39.9712546617116f},
  { 27.5065578286406f,    26.5250811255815f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 14 dirs.

Definition at line 10232 of file saf_loudspeaker_presets.c.

6.31.2.28 __SphCovering_15_dirs_deg const float __SphCovering_15_dirs_deg[15][2]

Initial value:

```
=
{ { 38.2804562082806f,    -30.8503395210240f},
  { -10.9944871307654f,    52.3288083870932f},
  { 107.773361264108f,    -72.5078089738057f},
  { 59.1464331913549f,    28.4645432620993f},
  { -104.931490600259f,    -11.1594989757630f},
  { -142.534710694695f,    -43.2348222627768f},
  { 156.658120344670f,    -21.2555882837633f},
  { 95.6495743191396f,    -14.9478959171680f},
  { 1.54996542738790f,    -1.22343677994285f},
  { -102.771439712616f,    43.3889479096670f},
  { -52.6920636292062f,    5.49340474815531f},
  { 131.132850571592f,    27.4269166951174f},
  { 147.645494227262f,    72.1869526085324f},
  { -161.528261603282f,    16.8449591768462f},
  { -44.6563305524964f,    -50.0312476286186f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 15 dirs.

Definition at line 10249 of file saf_loudspeaker_presets.c.

6.31.2.29 __SphCovering_16_dirs_deg const float __SphCovering_16_dirs_deg[16][2]

Initial value:

```
=
{ { 64.8015266292961f,    -34.0268175372293f},
  { -127.517486884316f,    -50.0524470670385f},
  { 5.17552776341673f,    51.2487829432716f},
  { 170.202842621562f,    27.6291707967986f},
  { -73.0864963468878f,    32.0959497676385f},
  { -136.776484853630f,    66.7725014445461f},
  { 120.011739768102f,    -5.26416433432346f},
  { 178.069553148709f,    -22.2737342857108f},
  { -18.2194849273651f,    -56.5056707135969f},
  { 19.3224286829919f,    -5.99199262147815f},
  { -78.1800911456008f,    -18.1461463295883f},
  { -131.035447746419f,    8.38466437394447f},
  { 65.0421689032511f,    16.4519101293865f},
  { -30.7328831730222f,    1.12740905347892f},
  { 110.076651600534f,    52.6427892588249f},
  { 133.321549348991f,    -62.9623321069262f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 16 dirs.

Definition at line 10267 of file saf_loudspeaker_presets.c.

6.31.2.30 __SphCovering_17_dirs_deg const float __SphCovering_17_dirs_deg[17][2]

Initial value:

```
=
{ { -128.010230588129f,    59.8855487470736f},
  { 80.3516011891467f,    -14.4746327783900f},
  { 153.363613022667f,    38.9754540137743f},
  { -161.763174299285f,    6.07449854397699f},
  { 141.457550039849f,    -11.9203869276968f},
  { -117.020900077519f,    -25.7091892253152f},
  { 29.5955619496875f,    17.4442730305530f},
  { -34.6633736476197f,    37.7447406698332f},
  { -62.7446081447765f,    -9.22462050160626f},
  { 29.6035833588194f,    -36.3616205523874f},
  { 108.844792341002f,    -60.0402473517590f},
  { -54.3095234848605f,    -64.1025181192365f},
  { -14.6734491333004f,    -10.8667175424512f},
  { -102.433394613489f,    22.3470728834875f},
  { -171.343028633873f,    -45.3072106077650f},
  { 93.5640079448634f,    25.1133131183791f},
  { 46.4164568991383f,    68.4398086283768f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 17 dirs.

Definition at line 10286 of file saf_loudspeaker_presets.c.

6.31.2.31 __SphCovering_18_dirs_deg const float __SphCovering_18_dirs_deg[18][2]

Initial value:

```
=
{ { 123.294787934202f,    12.2653075203655f},
  { 123.638562611280f,    -42.2808475338840f},
  { 41.9342080678298f,    -42.1473483676185f},
  { -80.5865138851503f,    22.0485618722243f},
  { 76.5357022735754f,    -8.28095901302579f},
  { 127.815424937784f,    64.7614195836370f},
  { 19.5132236287704f,    6.30711940880010f},
  { -35.4890058304032f,    -84.8149424132158f},
  { -20.6104378064460f,    -38.0696077396724f},
  { -118.785610086522f,    58.5104500387597f},
  { -176.975203760009f,    31.7538939639454f},
  { -131.356304111693f,    1.40747082373887f},
  { -34.2703246001599f,    5.64907738109235f},
  { -13.8844862494052f,    53.6775510368312f},
  { 63.5753969477161f,    37.7773992641557f},
  { 171.148222983528f,    -12.3260410466494f},
  { -83.7893479599316f,    -28.6713810261415f},
  { -149.668035244074f,    -46.7957549595149f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 18 dirs.

Definition at line 10306 of file saf_loudspeaker_presets.c.

6.31.2.32 __SphCovering_19_dirs_deg const float __SphCovering_19_dirs_deg[19][2]

Initial value:

```
=
{ { 179.049310978382f,    67.6090198254371f},
  { -76.3981924027440f,    -13.5000315688725f},
  { 37.0921417411792f,    -5.35864507474054f},
  { -113.766499801176f,    32.0397999037156f},
  { 90.4299288054978f,    -31.9716179260951f},
  { 143.514468524369f,    -9.92305605387073f},
  { -170.483591941176f,    17.3079090753119f},
  { 88.4589539902478f,    8.78859961951170f},
  { -36.4464183060668f,    63.9191716247946f},
  { 131.442247780962f,    36.6280459271233f},
  { -165.733771819542f,    -33.2378546533342f},
  { -125.775695187118f,    -14.4740598205949f},
```

```
  { 26.0208782658663f,    -52.9424461856783f},
  { 138.821944182247f,    -65.8901464400447f},
  { -21.0796902406581f,    -25.1178967807402f},
  { -0.924123627766505f,    19.1803351497994f},
  { -54.3416091213878f,    21.0779713672727f},
  { -84.5341930936017f,    -59.6563656290213f},
  { 57.0717530151862f,    46.9315459569609f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 19 dirs.

Definition at line 10327 of file saf_loudspeaker_presets.c.

6.31.2.33 __SphCovering_20_dirs_deg const float __SphCovering_20_dirs_deg[20][2]

Initial value:

```
=
{ { -82.8611563318196f,    -56.0461585619020f},
  { 134.811239616331f,    23.6568544031566f},
  { -100.943704346148f,    -3.51280424194708f},
  { -143.606141771590f,    -43.0818425314769f},
  { 124.108388003288f,    73.1266033925470f},
  { 36.5730419787907f,    -6.54375097818913f},
  { -61.6846362237844f,    23.3130797260781f},
  { -116.436483126486f,    49.4559980023073f},
  { 179.387356077509f,    30.8812792419611f},
  { 7.71888741600245f,    -50.1349529895373f},
  { -48.4796779194044f,    -24.7540685808321f},
  { 83.6174606213923f,    23.1131174555774f},
  { 125.592348692676f,    -71.6770201708660f},
  { -146.642818085783f,    6.58443098164342f},
  { 115.055654840221f,    -22.2278976621003f},
  { -28.3384925471705f,    54.7570035228576f},
  { -9.34837938535451f,    2.50164832510020f},
  { 32.0180275075007f,    43.0268385831443f},
  { 71.0696849080273f,    -30.0315828317821f},
  { 165.750960553396f,    -22.0193410246727f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 20 dirs.

Definition at line 10349 of file saf_loudspeaker_presets.c.

6.31.2.34 __SphCovering_21_dirs_deg const float __SphCovering_21_dirs_deg[21][2]

Initial value:

```
=
{ { 159.104650129878f,    -5.29229656206439f},
  { -154.377748320049f,    -15.7276914763411f},
  { -114.969711170951f,    60.4756952760584f},
  { 19.0531385192804f,    -14.8636711212838f},
  { 109.772983969114f,    -22.8787777173689f},
  { -175.399569823399f,    34.2617302332330f},
  { -124.240168296168f,    17.3038983707460f},
  { 83.7435113363211f,    23.5915372145116f},
  { -19.0794945778564f,    52.6107036222976f},
  { 32.0237570854520f,    33.8532113253047f},
  { 60.6361234586950f,    -62.9164954833157f},
  { -17.9278494096435f,    -50.9920978510530f},
  { 107.125918955610f,    69.2877861651705f},
  { -68.3022987575454f,    21.5139922493673f},
  { 128.308168641597f,    26.6368078956320f},
  { 59.8912783250249f,    -14.5067184149173f},
  { -118.413187519687f,    -62.8649292817539f},
  { 160.124515005211f,    -53.9371009180254f},
  { -104.358532805128f,    -17.8911801107551f},
  { -17.6740291064005f,    5.12218539269005f},
  { -56.7377186206249f,    -24.9190804258298f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 21 dirs.

Definition at line 10372 of file saf_loudspeaker_presets.c.

6.31.2.35 __SphCovering_22_dirs_deg const float __SphCovering_22_dirs_deg[22][2]

Initial value:

```
=
{ { 71.9978765361392f,    43.7808510415365f},
  { -71.9978765361392f,    -9.65892251031542f},
  { 59.2151881267706f,    90.0f},
  { -120.785232791529f,    -90.0f},
  { 108.002544382160f,    -43.7814239993316f},
  { -107.996814804209f,    9.65949546811055f},
  { 36.0000841836599f,    -43.7819969571267f},
  { -180.0f,    9.65892251031542f},
  { 144.001482650230f,    43.7814239993316f},
  { 0.000325308247341427f,    -9.65892251031542f},
  { 0.000637759321760119f,    43.7814239993316f},
  { -144.001482650230f,    -9.65892251031542f},
  { -36.0000841836599f,    -43.7814239993316f},
  { 108.002544382160f,    9.65892251031542f},
  { -180.0f,    -43.7814239993316f},
  { -35.9995112258648f,    9.65892251031542f},
  { -144.001482650230f,    43.7819969571267f},
  { 71.9978765361392f,    -9.65949546811055f},
  { -71.9978765361392f,    43.7814239993316f},
  { 144.001482650230f,    -9.65892251031542f},
  { -108.002544382160f,    -43.7808510415365f},
  { 36.0006571414550f,    9.65892251031542f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 22 dirs.

Definition at line 10396 of file saf_loudspeaker_presets.c.

6.31.2.36 __SphCovering_23_dirs_deg const float __SphCovering_23_dirs_deg[23][2]

Initial value:

```
=
{ { 68.5028339858412f,    -56.9485670892330f},
  { -53.8041747095551f,    -19.1768974030287f},
  { 126.193954377564f,    19.1768974030287f},
  { -18.5856049584536f,    7.72347107836350f},
  { -116.820364849224f,    37.7751074329752f},
  { 111.950223590612f,    61.9424672315933f},
  { -53.5176958119897f,    31.0778037656910f},
  { 71.6999384826712f,    31.2932358966602f},
  { -94.5838728201963f,    75.3955162612650f},
  { 93.2947177811520f,    -10.4243941246102f},
  { 50.8941220680856f,    -16.3430481483116f},
  { -179.862911047468f,    38.6328252522860f},
  { 166.742177538972f,    0.484664998901163f},
  { 179.817074423858f,    -71.8431779314539f},
  { -108.168702142748f,    -34.7246801316987f},
  { -46.9510265219953f,    -62.8248222360948f},
  { -167.630262121425f,    -33.4200552321858f},
  { -142.964429041043f,    2.40986048632024f},
  { 23.4603298794267f,    11.6602640887074f},
  { -92.6014388490437f,    2.59332157232113f},
  { 130.479678685142f,    -30.9093541739225f},
  { 8.96564357820712f,    52.7625374380073f},
  { 0.526966472915672f,    -35.2139860887404f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 23 dirs.

Definition at line 10421 of file saf_loudspeaker_presets.c.

6.31.2.37 __SphCovering_24_dirs_deg const float __SphCovering_24_dirs_deg[24][2]

Initial value:

```
=
{ { 23.1314521050216f,    -10.3550662313994f},
  { -26.5777932427335f,    66.1651661817075f},
  { -26.5686259180114f,    -5.69256487774278f},
  { 137.005667971682f,    32.8448056058744f},
  { -0.620513292126682f,    28.7229472277033f},
  { 97.0876983849180f,    17.0197113043611f},
  { -11.8550697390519f,    -45.3564849781462f},
  { 91.6102218634673f,    65.7182591015054f},
  { -97.4429322178991f,    -1.18373080474028f},
  { 179.014933510674f,    15.5638255469337f},
  { 138.409414569753f,    -13.1705808366722f},
  { 103.269912994380f,    -30.4120268077490f},
  { 50.2501175063586f,    -56.5543721261830f},
  { -128.445678512428f,    -36.5094436635312f},
  { -175.262059952568f,    -26.5955549343826f},
  { -85.5024917673728f,    -70.8061243222671f},
  { -102.800087602372f,    43.6559462421979f},
  { 60.3668332949835f,    -9.40739403825299f},
  { 154.240238449218f,    -62.2060278173535f},
  { 47.4179871250269f,    34.3917916527277f},
  { -69.7346932453725f,    -26.3921549171111f},
  { -54.6848108406712f,    22.2319083666662f},
  { -141.262744389504f,    9.40166446030168f},
  { -158.422830353673f,    58.4416951033440f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 24 dirs.

Definition at line 10447 of file saf_loudspeaker_presets.c.

6.31.2.38 __SphCovering_25_dirs_deg const float __SphCovering_25_dirs_deg[25][2]

Initial value:

```
=
{ { -101.585417076695f,    -9.81533998838613f},
  { 177.284600969379f,    46.2101920928912f},
  { -117.639694496261f,    36.2911467435863f},
  { -145.749003925379f,    9.78726505642472f},
  { -177.788803829094f,    -26.9886039818423f},
  { -74.1063612222207f,    -54.4092181412132f},
  { 136.060287609717f,    -23.4442870611630f},
  { -118.636641059788f,    81.6865928518015f},
  { -21.8067736826791f,    13.6822321477241f},
  { 14.9324260566995f,    -65.3859435803296f},
  { -70.8175834781698f,    17.2488944224134f},
  { -48.6550030047144f,    50.3102780748473f},
  { 166.335377504429f,    8.57259453074738f},
  { 46.0377317965568f,    -22.8455461652513f},
  { -8.56228129043502f,    -30.4893761100916f},
  { -54.5352688561420f,    -20.2964569347143f},
  { -133.166850744306f,    -35.9811765764206f},
  { 66.7381239768383f,    21.7345810004926f},
  { 100.852031098928f,    55.9143782690219f},
  { 94.0395629148220f,    -49.1053478316872f},
  { 122.836421698097f,    22.4335895105523f},
  { 92.0227514759615f,    -6.76605860269989f},
  { 173.451513319954f,    -68.3309466473020f},
  { 20.9725471329687f,    6.61250591360483f},
  { 19.8518416856928f,    49.2405658713381f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 25 dirs.

Definition at line 10474 of file saf_loudspeaker_presets.c.

6.31.2.39 __SphCovering_26_dirs_deg `const float __SphCovering_26_dirs_deg[26][2]`

Initial value:

```
=
{ { 100.072808497550f,    3.17304026943450f},
  { -165.418645032220f,    -74.0777133324641f},
  { 128.543081337600f,    38.3560866372378f},
  { 171.543563862169f,    -38.9215959810320f},
  { 70.0555496106458f,    37.4731586749412f},
  { 26.1257295423753f,    -15.6623742876962f},
  { 67.8324733655382f,    -11.1142353099477f},
  { -56.7130814354343f,    31.1946871558977f},
  { -1.47737167474483f,    58.2698077648047f},
  { -94.6927348012712f,    3.87531463892635f},
  { 106.392532977843f,    -44.7720680271128f},
  { -9.65319293236411f,    -27.9368491327838f},
  { 165.172273180314f,    10.8019733116014f},
  { -51.7077221371714f,    -11.2007519370125f},
  { -166.959901501122f,    47.0123330060743f},
  { -37.6565051693831f,    -59.5990698495082f},
  { 49.4244853235751f,    -56.2592988616907f},
  { -130.903667453539f,    -30.3071755312400f},
  { -96.1079405552443f,    53.3452355156553f},
  { 134.977397376919f,    -10.8025462693965f},
  { -89.1694216562100f,    -39.7913459140406f},
  { -128.388382732915f,    16.2032464462997f},
  { -12.6199683955515f,    15.1495770610541f},
  { 106.759225966726f,    80.2140913183152f},
  { -161.774633455188f,    -1.90806404934467f},
  { 33.6990856784145f,    21.2154812381041f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 26 dirs.

Definition at line 10502 of file saf_loudspeaker_presets.c.

6.31.2.40 __SphCovering_27_dirs_deg `const float __SphCovering_27_dirs_deg[27][2]`

Initial value:

```
=
{ { 101.642712856208f,    -10.3711090496630f},
  { 13.2857453534935f,    -25.0302342380851f},
  { -112.838308173064f,    -42.1169816044766f},
  { 148.527849231763f,    -18.0372843485134f},
  { 37.3230437326170f,    9.55292531821621f},
  { 100.502526843898f,    55.1414582033904f},
  { -140.821566887254f,    32.1807475213178f},
  { 130.846371674026f,    20.0603983231204f},
  { 79.1827672870798f,    20.6264806247096f},
  { -38.6752241291257f,    -55.1787004600739f},
  { -55.3379827271203f,    15.4228779293315f},
  { -27.7678265832202f,    50.9611581301159f},
  { -25.7584635956964f,    -14.5347933468787f},
  { 178.419057403738f,    6.01147318651260f},
  { 115.479643608617f,    -50.6517609207453f},
  { -73.7110203435804f,    -22.3694182374976f},
  { -94.1484248958969f,    40.3236237057171f},
  { -128.760805299750f,    80.1625251167535f},
  { 172.448837178475f,    43.7052206125792f},
  { -151.919759378938f,    -76.3810036688901f},
  { -142.677950143478f,    -9.55292531821621f},
  { 41.3469263278207f,    -57.3817231823520f},
  { 64.0165744499669f,    -23.0397788578007f},
  { -4.94898025122200f,    17.5136009237639f},
  { -175.181845861249f,    -39.7111318227222f},
  { 34.8157804211245f,    51.2728471706671f},
  { -102.708414355151f,    3.06016758379373f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 27 dirs.

Definition at line 10531 of file saf_loudspeaker_presets.c.

6.31.2.41 __SphCovering_28_dirs_deg const float __SphCovering_28_dirs_deg[28][2]

Initial value:

```
=
{ { 130.771887160659f,    1.46413634967731f},
  { -43.4376493222531f,    -25.4828708962385f},
  { -65.9302534857038f,    62.4867771369676f},
  { -76.2549529539613f,    -7.86441869596568f},
  { 167.366701535665f,    -14.8476283030202f},
  { -138.294823010727f,    39.6463875918725f},
  { 86.5395453765595f,    8.84073877886860f},
  { -15.2773466493683f,    -1.99612766245628f},
  { -2.53917706068127f,    41.0959708135534f},
  { -98.3023689105954f,    -42.5123224831168f},
  { -153.180266528226f,    -35.7342317667192f},
  { 54.2218609422055f,    65.5922083865766f},
  { 11.2660691256574f,    -34.6226936441654f},
  { -35.7141782438896f,    -60.9627094019196f},
  { 143.915538980960f,    -44.7250854879121f},
  { 102.616741107930f,    -29.3772650297427f},
  { 55.1764086288934f,    -17.9186820849214f},
  { -158.428559931624f,    8.61556636538219f},
  { 164.049275901857f,    29.6620250539227f},
  { 24.8119373181403f,    8.12167674597942f},
  { -178.854505328038f,    68.0673860615418f},
  { -121.123277890656f,    -5.29424461856783f},
  { 113.222189895802f,    40.7367262760064f},
  { -47.6930068666897f,    21.9866824303502f},
  { -96.8241377991578f,    28.2748942319110f},
  { 64.5150477317307f,    -59.7365797203396f},
  { -170.386189116004f,    -74.5704570362767f},
  { 55.3660576590817f,    29.7296340737482f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 28 dirs.

Definition at line 10561 of file saf_loudspeaker_presets.c.

6.31.2.42 __SphCovering_29_dirs_deg const float __SphCovering_29_dirs_deg[29][2]

Initial value:

```
=
{ { -174.671913423583f,    -23.7983749785539f},
  { -102.507879126856f,    17.4540133130703f},
  { -18.5202877698087f,    1.43600412193638f},
  { 7.18832849771131f,    35.8356452964573f},
  { -62.9852504187314f,    16.5269676005486f},
  { 48.9151258437038f,    -69.6773974658594f},
  { -60.0631656635642f,    -58.8484951378869f},
  { 97.9585942335169f,    -32.0965227254336f},
  { 106.679011875408f,    4.72208896435019f},
  { 68.8122311952119f,    4.51175615775767f},
  { -50.7210888139561f,    -20.8505071226058f},
  { -36.0293050312116f,    41.4145353476462f},
  { 48.2373167720640f,    32.5514512147675f},
  { -137.234851089735f,    -4.65946467734239f},
  { 51.5031762043097f,    -31.1121812333988f},
  { 94.9448362311287f,    44.1893699494647f},
  { 35.1102807278217f,    77.9509080275485f},
  { -179.238387050775f,    14.0099640065389f},
  { -92.5556022254332f,    -18.9987075287430f},
  { 170.237220089270f,    61.3408615467059f},
  { 143.646248817249f,    -13.0588540666217f},
  { 144.242124924185f,    28.7538869486404f},
  { 136.908265146510f,    -50.5211265434555f},
  { -146.344880032315f,    36.2791146298886f},
  { -161.957979949630f,    -66.8355268020105f},
  { -2.71427296287325f,    -37.7802640531314f},
  { -123.409379493228f,    -40.4221724464796f},
  { 23.1486408388755f,    -3.25995796695584f},
  { -90.9570499770182f,    57.3244274028389f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 29 dirs.

Definition at line 10592 of file saf_loudspeaker_presets.c.

6.31.2.43 __SphCovering_4_dirs_deg const float __SphCovering_4_dirs_deg[4][2]

Initial value:

```
=
{ { -161.035517899469f,    32.3554996488327f},
  { -66.6980169311791f,    -31.3940764686032f},
  { 24.3822189717922f,    37.9922584373298f},
  { 109.291699421205f,    -39.0356145822630f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 4 dirs.

See also

Belger, M. (1989). JH Conway, NJA Sloane. Sphere packings, lattices and groups. Springer Verlag New York–Berlin–Heidelberg–London–Paris Tokyo 1988, 663 pages, 112 illustrations, DM 178.00, ISBN 0–387–96617–X. Crystal Research and Technology, 24(1), 90–90.

Definition at line 10117 of file saf_loudspeaker_presets.c.

6.31.2.44 __SphCovering_5_dirs_deg const float __SphCovering_5_dirs_deg[5][2]

Initial value:

```
=
{ { 0.0f,    -90.0f},
  { -105.888330118127f,    0.0f},
  { 14.1136693674576f,    0.0f},
  { 134.112231106272f,    0.0f},
  { 0.0f,    90.0f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 5 dirs.

Definition at line 10124 of file saf_loudspeaker_presets.c.

6.31.2.45 __SphCovering_6_dirs_deg const float __SphCovering_6_dirs_deg[6][2]

Initial value:

```
=
{ { 0.0f,    90.0f},
  { 0.0f,    -90.0f},
  { -77.7274544874475f,    0.0f},
  { 12.2716100561120f,    0.0f},
  { 102.272966430852f,    0.0f},
  { -167.727664946597f,    0.0f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 6 dirs.

Definition at line 10132 of file saf_loudspeaker_presets.c.

6.31.2.46 __SphCovering_7_dirs_deg `const float __SphCovering_7_dirs_deg[7][2]`

Initial value:

```
=
{ { 148.510660497909f,    30.6406369680062f},
  { -65.2942703331086f,    -58.3442922781717f},
  { -14.5084372883027f,    1.52429691816604f},
  { -105.229428653727f,    25.3018162329772f},
  { 40.3665955403519f,    60.1892163784930f},
  { 74.7709922645724f,    -25.3018162329772f},
  { -176.958015026155f,    -33.5518355250659f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 7 dirs.

Definition at line 10141 of file saf_loudspeaker_presets.c.

6.31.2.47 __SphCovering_8_dirs_deg `const float __SphCovering_8_dirs_deg[8][2]`

Initial value:

```
=
{ { -94.9906728547392f,    43.0342870344810f},
  { 109.916223417897f,    -54.2636868612500f},
  { 0.930999121308075f,    49.8685276148015f},
  { 128.147740458960f,    36.0636824989194f},
  { -105.372668102510f,    -38.6694945511744f},
  { -30.9385750214742f,    -27.3489944349796f},
  { -176.075660021653f,    1.82630297197950f},
  { 55.3849652663210f,    -9.88123013482618f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 8 dirs.

Definition at line 10151 of file saf_loudspeaker_presets.c.

6.31.2.48 __SphCovering_9_dirs_deg `const float __SphCovering_9_dirs_deg[9][2]`

Initial value:

```
=
{ { -116.522426795756f,    85.0212072194629f},
  { 16.9658532716188f,    24.5094156023112f},
  { -91.1518556273627f,    -50.4202859715125f},
  { -132.834535223130f,    8.34455732828531f},
  { 149.163832384359f,    -47.7944203964279f},
  { 86.7057031371475f,    6.33576729855664f},
  { 156.577906253351f,    20.8482152914253f},
  { 22.0130384889262f,    -44.3005237617201f},
  { -54.6979888699592f,    10.8036921849868f}}
```

Directions [azimuth, Elevation] in degrees, for sphere covering: 9 dirs.

Definition at line 10162 of file saf_loudspeaker_presets.c.

6.31.2.49 __stereo_dirs_deg `const float __stereo_dirs_deg[2][2]`

Initial value:

```
=
{ { 30.0f,    0.0f},
  { -30.0f,    0.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for a stereo setup.

Definition at line 33 of file saf_loudspeaker_presets.c.

6.31.2.50 __Tdesign_degree_1_dirs_deg `const float __Tdesign_degree_1_dirs_deg[2][2]`

Initial value:

```
=
{ { 0.0f,    0.0f},
  { 180.0f,    0.0f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 1.

See also

McLaren's Improved Snub Cube and Other New Spherical Designs in Three Dimensions", R. H. Hardin and N. J. A. Sloane, Discrete and Computational Geometry, 15 (1996), pp. 429-441.

Definition at line 455 of file saf_loudspeaker_presets.c.

6.31.2.51 __Tdesign_degree_2_dirs_deg `const float __Tdesign_degree_2_dirs_deg[4][2]`

Initial value:

```
=
{ { 45.0f,    35.2643896827547f},
  { -45.0f,    -35.2643896827547f},
  { 135.0f,    -35.2643896827547f},
  { -135.0f,    35.2643896827547f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 2.

Definition at line 460 of file saf_loudspeaker_presets.c.

6.31.2.52 __Tdesign_degree_30_dirs_deg `const float __Tdesign_degree_30_dirs_deg[480][2]`

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 30.

See also

Gra"f, M., & Potts, D. (2011). On the computation of spherical designs by a new optimization approach based on fast spherical Fourier transforms. Numerische Mathematik, 119(4), 699-724.

Definition at line 2362 of file saf_loudspeaker_presets.c.

6.31.2.53 __Tdesign_degree_3_dirs_deg `const float __Tdesign_degree_3_dirs_deg[6][2]`

Initial value:

```
=
{ { 0.0f,    0.0f},
  { 180.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 0.0f,    90.0f},
  { 0.0f,    -90.0f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 3.

Definition at line 467 of file saf_loudspeaker_presets.c.

6.31.2.54 __Tdesign_degree_4_dirs_deg const float __Tdesign_degree_4_dirs_deg[12][2]

Initial value:

```
=
{ { 0.0f,   -31.7174744114557f},
  { -58.2825255885443f,    0.0f},
  { -90.0f,    58.2825255885443f},
  { 0.0f,    31.7174744114557f},
  { -121.717474411456f,    0.0f},
  { 90.0f,    -58.2825255885443f},
  { 180.0f,    -31.7174744114557f},
  { 121.717474411456f,    0.0f},
  { 90.0f,    58.2825255885443f},
  { 180.0f,    31.7174744114557f},
  { 58.2825255885443f,    0.0f},
  { -90.0f,    -58.2825255885443f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 4.

Definition at line 476 of file saf_loudspeaker_presets.c.

6.31.2.55 __Tdesign_degree_5_dirs_deg const float __Tdesign_degree_5_dirs_deg[12][2]

Initial value:

```
=
{ { 0.0f,    -31.7174744114557f},
  { -58.2825255885443f,    0.0f},
  { -90.0f,    58.2825255885443f},
  { 0.0f,    31.7174744114557f},
  { -121.717474411456f,    0.0f},
  { 90.0f,    -58.2825255885443f},
  { 180.0f,    -31.7174744114557f},
  { 121.717474411456f,    0.0f},
  { 90.0f,    58.2825255885443f},
  { 180.0f,    31.7174744114557f},
  { 58.2825255885443f,    0.0f},
  { -90.0f,    -58.2825255885443f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 5.

Definition at line 491 of file saf_loudspeaker_presets.c.

6.31.2.56 __Tdesign_degree_6_dirs_deg const float __Tdesign_degree_6_dirs_deg[24][2]

Initial value:

```
=
{ { 26.0011675216559f,    15.4641512961471f},
  { -26.0011675216559f,    -15.4641512961471f},
  { 17.1086452559122f,    -24.9937030546433f},
  { -17.1086452559122f,    24.9937030546433f},
  { 153.998832478344f,    -15.4641512961471f},
  { -153.998832478344f,    15.4641512961471f},
  { 162.891354744088f,    24.9937030546433f},
  { -162.891354744088f,    -24.9937030546433f},
  { 72.8913547440879f,    24.9937030546433f},
  { 107.108645255912f,    -24.9937030546433f},
  { 116.001167521656f,    15.4641512961471f},
  { 63.9988324783441f,    -15.4641512961471f},
  { -107.108645255912f,    24.9937030546433f},
  { -72.8913547440879f,    -24.9937030546433f},
  { -63.9988324783441f,    15.4641512961471f},
  { -116.001167521656f,    -15.4641512961471f},
  { 32.2544599366034f,    60.0253819510733f},
  { -147.745540063397f,    60.0253819510733f},
  { -57.7455400633966f,    60.0253819510733f},
  { 122.254459936603f,    60.0253819510733f},
  { -32.2544599366034f,    -60.0253819510733f},
  { 147.745540063397f,    -60.0253819510733f},
  { 57.7455400633966f,    -60.0253819510733f},
  { -122.254459936603f,    -60.0253819510733f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 6.

Definition at line 506 of file saf_loudspeaker_presets.c.

6.31.2.57 __Tdesign_degree_7_dirs_deg `const float __Tdesign_degree_7_dirs_deg[24][2]`

Initial value:

```
=
{ { 26.0011675216559f,    15.4641512961471f},
  { -26.0011675216559f,    -15.4641512961471f},
  { 17.1086452559122f,    -24.9937030546433f},
  { -17.1086452559122f,    24.9937030546433f},
  { 153.998832478344f,    -15.4641512961471f},
  { -153.998832478344f,    15.4641512961471f},
  { 162.891354744088f,    24.9937030546433f},
  { -162.891354744088f,    -24.9937030546433f},
  { 72.8913547440879f,    24.9937030546433f},
  { 107.108645255912f,    -24.9937030546433f},
  { 116.001167521656f,    15.4641512961471f},
  { 63.9988324783441f,    -15.4641512961471f},
  { -107.108645255912f,    24.9937030546433f},
  { -72.8913547440879f,    -24.9937030546433f},
  { -63.9988324783441f,    15.4641512961471f},
  { -116.001167521656f,    -15.4641512961471f},
  { 32.2544599366034f,    60.0253819510733f},
  { -147.745540063397f,    60.0253819510733f},
  { -57.7455400633966f,    60.0253819510733f},
  { 122.254459936603f,    60.0253819510733f},
  { -32.2544599366034f,    -60.0253819510733f},
  { 147.745540063397f,    -60.0253819510733f},
  { 57.7455400633966f,    -60.0253819510733f},
  { -122.254459936603f,    -60.0253819510733f}}
```

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 7.

Definition at line 533 of file saf_loudspeaker_presets.c.

6.31.2.58 __Tdesign_nPoints_per_degree `const int __Tdesign_nPoints_per_degree[21]`

Initial value:

```
=
{   2, 4, 6, 12, 12, 24, 24, 36, 48, 60, 70, 84, 94, 108, 120, 144, 156, 180, 204, 216, 240   }
```

Number of points in each t-design (up to degree 21 only).

Access as, e.g.

```
const int tdesign_degree = 7;
float* tdesign_dirs_deg = __HANDLES_Tdesign_dirs_deg [tdesign_degree-1];
int num_Tdesign_dirs = __Tdesign_nPoints_per_degree  [tdesign_degree-1];
```

Definition at line 10089 of file saf_loudspeaker_presets.c.

6.31.2.59 __Zylia_Lab_dirs_deg `const float __Zylia_Lab_dirs_deg[22][2]`

Initial value:

```
=
{ { 45.0f,    0.0f},
  { -45.0f,    0.0f},
  { 0.0f,    0.0f},
  { 135.0f,    0.0f},
  { -135.0f,    0.0f},
  { 30.0f,    0.0f},
  { -30.0f,    0.0f},
  { -180.0f,    0.0f},
  { 90.0f,    0.0f},
  { -90.0f,    0.0f},
  { 45.0f,    30.0f},
  { -45.0f,    30.0f},
  { 0.0f,    30.0f},
  { 0.0f,    90.0f},
  { 135.0f,    30.0f},
  { -135.0f,    30.0f},
  { 90.0f,    30.0f},
  { -90.0f,    30.0f},
  { 180.0f,    30.0f},
  { 0.0f,    -25.0f},
  { 45.0f,    -25.0f},
  { -45.0f,    -25.0f}}
```

Loudspeaker directions [azimuth, Elevation] in degrees, for the 22.x setup, at Zylia Labs.

Definition at line 363 of file saf_loudspeaker_presets.c.

6.32 framework/modules/saf_utilities/saf_loudspeaker_presets.h File Reference

Comprises the directions of loudspeaker arrays and (nearly) uniform spherical grids.

```
#include <stdio.h>
```

Variables

- const float __mono_dirs_deg [1][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a mono setup.
- const float __stereo_dirs_deg [2][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a stereo setup.
- const float __5pX_dirs_deg [5][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 5.x setup.
- const float __7pX_dirs_deg [7][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 7.x setup.
- const float __8pX_dirs_deg [8][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 8.x setup.
- const float __9pX_dirs_deg [9][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 9.x setup.
- const float __10pX_dirs_deg [10][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 10.x setup.
- const float __11pX_dirs_deg [11][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 11.x setup.
- const float __11pX_7_4_dirs_deg [11][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 7.4.x setup.
- const float __13pX_dirs_deg [13][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 13.x setup.
- const float __22pX_dirs_deg [22][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for a 22.x setup.
- const float __Aalto_MCC_dirs_deg [45][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the multi-channel anechoic chamber (MCC), at Aalto University.
- const float __Aalto_MCCsubset_dirs_deg [37][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the multi-channel anechoic chamber (MCC) sub-set, at Aalto University.
- const float __Aalto_Apaja_dirs_deg [29][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the audio-visual listening room (Apaja), at Aalto University.
- const float __Aalto_LR_dirs_deg [13][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the ITU standard listening room (LR), at Aalto University.
- const float __DTU_AVIL_dirs_deg [64][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the Audio Visual Immersion Lab (AVIL), at the Technical University of Denmark (DTU)
- const float __Zylia_Lab_dirs_deg [22][2]

  Loudspeaker directions [azimuth, Elevation] in degrees, for the 22.x setup, at Zylia Labs.
- const float default_LScoords64_rad [64][2]

  Default Loudspeaker directions [azimuth, Elevation] in degrees.
- const float __Tdesign_degree_1_dirs_deg [2][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 1.
- const float __Tdesign_degree_2_dirs_deg [4][2]

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 2.

- const float __Tdesign_degree_3_dirs_deg [6][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 3.
- const float __Tdesign_degree_4_dirs_deg [12][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 4.
- const float __Tdesign_degree_5_dirs_deg [12][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 5.
- const float __Tdesign_degree_6_dirs_deg [24][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 6.
- const float __Tdesign_degree_7_dirs_deg [24][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 7.
- const float __Tdesign_degree_8_dirs_deg [36][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 8.
- const float __Tdesign_degree_9_dirs_deg [48][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 9.
- const float __Tdesign_degree_10_dirs_deg [60][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 10.
- const float __Tdesign_degree_11_dirs_deg [70][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 11.
- const float __Tdesign_degree_12_dirs_deg [84][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 12.
- const float __Tdesign_degree_13_dirs_deg [94][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 13.
- const float __Tdesign_degree_14_dirs_deg [108][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 14.
- const float __Tdesign_degree_15_dirs_deg [120][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 15.
- const float __Tdesign_degree_16_dirs_deg [144][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 16.
- const float __Tdesign_degree_17_dirs_deg [156][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 17.
- const float __Tdesign_degree_18_dirs_deg [180][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 18.
- const float __Tdesign_degree_19_dirs_deg [204][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 19.
- const float __Tdesign_degree_20_dirs_deg [216][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 20.
- const float __Tdesign_degree_21_dirs_deg [240][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 21.
- const float __Tdesign_degree_30_dirs_deg [480][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 30.
- const float __Tdesign_degree_40_dirs_deg [840][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 40.
- const float __Tdesign_degree_50_dirs_deg [1296][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 50.
- const float __Tdesign_degree_100_dirs_deg [5100][2]

  Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 100.
- const float ∗ __HANDLES_Tdesign_dirs_deg [21]

  minimum T-design HANDLES (up to degree 21 only).
- const int __Tdesign_nPoints_per_degree [21]

  Number of points in each t-design (up to degree 21 only).

- const float __SphCovering_4_dirs_deg [4][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 4 dirs.
- const float __SphCovering_5_dirs_deg [5][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 5 dirs.
- const float __SphCovering_6_dirs_deg [6][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 6 dirs.
- const float __SphCovering_7_dirs_deg [7][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 7 dirs.
- const float __SphCovering_8_dirs_deg [8][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 8 dirs.
- const float __SphCovering_9_dirs_deg [9][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 9 dirs.
- const float __SphCovering_10_dirs_deg [10][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 10 dirs.
- const float __SphCovering_11_dirs_deg [11][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 11 dirs.
- const float __SphCovering_12_dirs_deg [12][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 12 dirs.
- const float __SphCovering_13_dirs_deg [13][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 13 dirs.
- const float __SphCovering_14_dirs_deg [14][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 14 dirs.
- const float __SphCovering_15_dirs_deg [15][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 15 dirs.
- const float __SphCovering_16_dirs_deg [16][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 16 dirs.
- const float __SphCovering_17_dirs_deg [17][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 17 dirs.
- const float __SphCovering_18_dirs_deg [18][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 18 dirs.
- const float __SphCovering_19_dirs_deg [19][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 19 dirs.
- const float __SphCovering_20_dirs_deg [20][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 20 dirs.
- const float __SphCovering_21_dirs_deg [21][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 21 dirs.
- const float __SphCovering_22_dirs_deg [22][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 22 dirs.
- const float __SphCovering_23_dirs_deg [23][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 23 dirs.
- const float __SphCovering_24_dirs_deg [24][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 24 dirs.
- const float __SphCovering_25_dirs_deg [25][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 25 dirs.
- const float __SphCovering_26_dirs_deg [26][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 26 dirs.
- const float __SphCovering_27_dirs_deg [27][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 27 dirs.
- const float __SphCovering_28_dirs_deg [28][2]

    Directions [azimuth, Elevation] in degrees, for sphere covering: 28 dirs.
- const float __SphCovering_29_dirs_deg [29][2]

Directions [azimuth, Elevation] in degrees, for sphere covering: 29 dirs.

- const float __SphCovering_30_dirs_deg [30][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 30 dirs.
- const float __SphCovering_31_dirs_deg [31][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 31 dirs.
- const float __SphCovering_32_dirs_deg [32][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 32 dirs.
- const float __SphCovering_33_dirs_deg [33][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 33 dirs.
- const float __SphCovering_34_dirs_deg [34][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 34 dirs.
- const float __SphCovering_35_dirs_deg [35][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 35 dirs.
- const float __SphCovering_36_dirs_deg [36][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 36 dirs.
- const float __SphCovering_37_dirs_deg [37][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 37 dirs.
- const float __SphCovering_38_dirs_deg [38][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 38 dirs.
- const float __SphCovering_39_dirs_deg [39][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 39 dirs.
- const float __SphCovering_40_dirs_deg [40][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 40 dirs.
- const float __SphCovering_41_dirs_deg [41][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 41 dirs.
- const float __SphCovering_42_dirs_deg [42][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 42 dirs.
- const float __SphCovering_43_dirs_deg [43][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 43 dirs.
- const float __SphCovering_44_dirs_deg [44][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 44 dirs.
- const float __SphCovering_45_dirs_deg [45][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 45 dirs.
- const float __SphCovering_46_dirs_deg [46][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 46 dirs.
- const float __SphCovering_47_dirs_deg [47][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 47 dirs.
- const float __SphCovering_48_dirs_deg [48][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 48 dirs.
- const float __SphCovering_49_dirs_deg [49][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 49 dirs.
- const float __SphCovering_50_dirs_deg [50][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 50 dirs.
- const float __SphCovering_51_dirs_deg [51][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 51 dirs.
- const float __SphCovering_52_dirs_deg [52][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 52 dirs.
- const float __SphCovering_53_dirs_deg [53][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 53 dirs.
- const float __SphCovering_54_dirs_deg [54][2]

  Directions [azimuth, Elevation] in degrees, for sphere covering: 54 dirs.

- const float __SphCovering_55_dirs_deg [55][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 55 dirs.
- const float __SphCovering_56_dirs_deg [56][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 56 dirs.
- const float __SphCovering_57_dirs_deg [57][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 57 dirs.
- const float __SphCovering_58_dirs_deg [58][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 58 dirs.
- const float __SphCovering_59_dirs_deg [59][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 59 dirs.
- const float __SphCovering_60_dirs_deg [60][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 60 dirs.
- const float __SphCovering_61_dirs_deg [61][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 61 dirs.
- const float __SphCovering_62_dirs_deg [62][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 62 dirs.
- const float __SphCovering_63_dirs_deg [63][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 63 dirs.
- const float __SphCovering_64_dirs_deg [64][2]
  Directions [azimuth, Elevation] in degrees, for sphere covering: 64 dirs.
- const float ∗ __HANDLES_SphCovering_dirs_deg [64]
  Sphere covering handles (between 4..64 points only)
- const float __geosphere_ico_0_0_dirs_deg [12][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 0.
- const float __geosphere_ico_1_0_dirs_deg [32][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 1.
- const float __geosphere_ico_2_0_dirs_deg [42][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 2.
- const float __geosphere_ico_3_0_dirs_deg [92][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 3.
- const float __geosphere_ico_4_0_dirs_deg [162][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 4.
- const float __geosphere_ico_5_0_dirs_deg [252][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 5.
- const float __geosphere_ico_6_0_dirs_deg [362][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 6.
- const float __geosphere_ico_7_0_dirs_deg [492][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 7.
- const float __geosphere_ico_8_0_dirs_deg [642][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 8.
- const float __geosphere_ico_9_0_dirs_deg [812][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 9.
- const float __geosphere_ico_10_0_dirs_deg [1002][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 10.
- const float __geosphere_ico_11_0_dirs_deg [1212][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 11.
- const float __geosphere_ico_12_0_dirs_deg [1442][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 12.
- const float __geosphere_ico_13_0_dirs_deg [1692][2]
  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 13.
- const float __geosphere_ico_14_0_dirs_deg [1962][2]

Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 14.
- const float __geosphere_ico_15_0_dirs_deg [2252][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 15.
- const float __geosphere_ico_16_0_dirs_deg [2562][2]

  Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 16.
- const float __geosphere_oct_0_0_dirs_deg [6][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 0.
- const float __geosphere_oct_1_0_dirs_deg [14][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 1.
- const float __geosphere_oct_2_0_dirs_deg [18][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 2.
- const float __geosphere_oct_3_0_dirs_deg [38][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 3.
- const float __geosphere_oct_4_0_dirs_deg [66][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 4.
- const float __geosphere_oct_5_0_dirs_deg [102][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 5.
- const float __geosphere_oct_6_0_dirs_deg [146][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 6.
- const float __geosphere_oct_7_0_dirs_deg [198][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 7.
- const float __geosphere_oct_8_0_dirs_deg [258][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 8.
- const float __geosphere_oct_9_0_dirs_deg [326][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 9.
- const float __geosphere_oct_10_0_dirs_deg [402][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 10.
- const float __geosphere_oct_11_0_dirs_deg [486][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 11.
- const float __geosphere_oct_12_0_dirs_deg [578][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 12.
- const float __geosphere_oct_13_0_dirs_deg [678][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 13.
- const float __geosphere_oct_14_0_dirs_deg [786][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 14.
- const float __geosphere_oct_15_0_dirs_deg [902][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 15.
- const float __geosphere_oct_16_0_dirs_deg [1026][2]

  Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 16.
- const float ∗ __HANDLES_geosphere_ico_dirs_deg [17]

  3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])
- const float ∗ __HANDLES_geosphere_oct_dirs_deg [17]

  3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])
- const int __geosphere_ico_nPoints [17]

  3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])
- const int __geosphere_oct_nPoints [17]

  3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])

6.32.1 Detailed Description

Comprises the directions of loudspeaker arrays and (nearly) uniform spherical grids.

Author

Leo McCormack

Date

11.07.2016

6.32.2 Variable Documentation

6.32.2.1 __geosphere_ico_0_0_dirs_deg `const float __geosphere_ico_0_0_dirs_deg[12][2]`

Directions [azimuth, Elevation] in degrees, for ico geosphere, degree: 0.

See also

3LD - Library for Loudspeaker Layout Design; release 2, 2006/03/15 Copyright (c) 2006 Florian Hollerweger (floholl_AT_sbox.tugraz.at) and (c) 2002 Darren Weber.

Definition at line 12441 of file saf_loudspeaker_presets.c.

6.32.2.2 __geosphere_ico_nPoints `const int __geosphere_ico_nPoints[17]`

3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.

```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_ico_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_ico_nPoints[degree];
```

Definition at line 33637 of file saf_loudspeaker_presets.c.

6.32.2.3 __geosphere_oct_0_0_dirs_deg `const float __geosphere_oct_0_0_dirs_deg[6][2]`

Directions [azimuth, Elevation] in degrees, for oct geosphere, degree: 0.

See also

3LD - Library for Loudspeaker Layout Design; release 2, 2006/03/15 Copyright (c) 2006 Florian Hollerweger (floholl_AT_sbox.tugraz.at) and (c) 2002 Darren Weber.

Definition at line 27516 of file saf_loudspeaker_presets.c.

6.32.2.4 __geosphere_oct_nPoints `const int __geosphere_oct_nPoints[17]`

3LD geosphere number of points (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.
```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_oct_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_oct_nPoints[degree];
```

Definition at line 33641 of file saf_loudspeaker_presets.c.

6.32.2.5 __HANDLES_geosphere_ico_dirs_deg `const float* __HANDLES_geosphere_ico_dirs_deg[17]`

3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.
```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_ico_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_ico_nPoints[degree];
```

Definition at line 33597 of file saf_loudspeaker_presets.c.

6.32.2.6 __HANDLES_geosphere_oct_dirs_deg `const float* __HANDLES_geosphere_oct_dirs_deg[17]`

3LD geosphere HANDLES (freq = [0 0], [1 0],..., [16 0])

Access as, e.g.
```
const int degree = 10; // between 0..16
float* geo_dirs_degree_10 = __HANDLES_geosphere_oct_dirs_deg[degree];
int num_dirs_degree_10 = __geosphere_oct_nPoints[degree];
```

Definition at line 33617 of file saf_loudspeaker_presets.c.

6.32.2.7 __HANDLES_SphCovering_dirs_deg `const float* __HANDLES_SphCovering_dirs_deg[64]`

Sphere covering handles (between 4..64 points only)

Access as, e.g.
```
const int numPoints = 44; // between 4..64 points
float* sphCov_dirs_deg = __HANDLES_SphCovering_dirs_deg [numPoints-1];
```

Definition at line 12374 of file saf_loudspeaker_presets.c.

6.32.2.8 __HANDLES_Tdesign_dirs_deg `const float* __HANDLES_Tdesign_dirs_deg[21]`

minimum T-design HANDLES (up to degree 21 only).

Access as, e.g.
```
const int tdesign_degree = 7;
float* tdesign_dirs_deg = __HANDLES_Tdesign_dirs_deg [tdesign_degree-1];
int num_Tdesign_dirs = __Tdesign_nPoints_per_degree  [tdesign_degree-1];
```

Definition at line 10093 of file saf_loudspeaker_presets.c.

6.32.2.9 __SphCovering_4_dirs_deg `const float __SphCovering_4_dirs_deg[4][2]`

Directions [azimuth, Elevation] in degrees, for sphere covering: 4 dirs.

See also

Belger, M. (1989). JH Conway, NJA Sloane. Sphere packings, lattices and groups. Springer Verlag New York–Berlin–Heidelberg–London–Paris Tokyo 1988, 663 pages, 112 illustrations, DM 178.00, ISBN 0–387–96617–X. Crystal Research and Technology, 24(1), 90–90.

Definition at line 10117 of file saf_loudspeaker_presets.c.

6.32.2.10 __Tdesign_degree_1_dirs_deg `const float __Tdesign_degree_1_dirs_deg[2][2]`

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 1.

See also

McLaren's Improved Snub Cube and Other New Spherical Designs in Three Dimensions", R. H. Hardin and N. J. A. Sloane, Discrete and Computational Geometry, 15 (1996), pp. 429-441.

Definition at line 455 of file saf_loudspeaker_presets.c.

6.32.2.11 __Tdesign_degree_30_dirs_deg `const float __Tdesign_degree_30_dirs_deg[480][2]`

Directions [azimuth, Elevation] in degrees, for minimum Tdesign degree: 30.

See also

Gra"f, M., & Potts, D. (2011). On the computation of spherical designs by a new optimization approach based on fast spherical Fourier transforms. Numerische Mathematik, 119(4), 699-724.

Definition at line 2362 of file saf_loudspeaker_presets.c.

6.32.2.12 __Tdesign_nPoints_per_degree `const int __Tdesign_nPoints_per_degree[21]`

Number of points in each t-design (up to degree 21 only).

Access as, e.g.

```
const int tdesign_degree = 7;
float* tdesign_dirs_deg = __HANDLES_Tdesign_dirs_deg [tdesign_degree-1];
int num_Tdesign_dirs = __Tdesign_nPoints_per_degree  [tdesign_degree-1];
```

Definition at line 10089 of file saf_loudspeaker_presets.c.

6.33 framework/modules/saf_utilities/saf_matrixConv.c File Reference

Matrix convolver functions mostly taken from some Matlab scripts by Archontis Politis.

```
#include "saf_utilities.h"
#include "saf_matrixConv.h"
```

Data Structures

- struct _safMatConv_data
  Data structure for the matrix convolver.
- struct _safMulConv_data
  Data structure for the multi-channel convolver.

Functions

- void saf_matrixConv_create (void ∗∗const phMC, int hopSize, float ∗H, int length_h, int nCHin, int nCHout, int usePartFLAG)
  Creates an instance of matrixConv.
- void saf_matrixConv_destroy (void ∗∗const phMC)
  Destroys an instance of matrixConv.
- void saf_matrixConv_apply (void ∗const hMC, float ∗inputSig, float ∗outputSig)
  Performs the matrix convolution.
- void saf_multiConv_create (void ∗∗const phMC, int hopSize, float ∗H, int length_h, int nCH, int usePartFLAG)
  Creates an instance of multiConv.
- void saf_multiConv_destroy (void ∗∗const phMC)
  Destroys an instance of multiConv.
- void saf_multiConv_apply (void ∗const hMC, float ∗inputSig, float ∗outputSig)
  Performs the multi-channel convolution.

6.33.1 Detailed Description

Matrix convolver functions mostly taken from some Matlab scripts by Archontis Politis.

Author

Leo McCormack

Date

06.04.2019

6.33.2 Function Documentation

6.33.2.1 saf_matrixConv_apply()

```
void saf_matrixConv_apply (
    void *const hMC,
    float * inputSigs,
    float * outputSigs )
```

Performs the matrix convolution.

Note

If the number of input+output channels, the filters, or the hopsize need tochange: simply destroy and re-create the matrixConv instance.

Parameters

in	*hMC*	matrixConv handle
in	*inputSigs*	Input signals; FLAT: nCHin x hopSize
out	*outputSigs*	Output signals; FLAT: nCHout x hopSize

Definition at line 164 of file saf_matrixConv.c.

6.33.2.2 saf_matrixConv_create()

```
void saf_matrixConv_create (
    void **const phMC,
    int hopSize,
    float * H,
    int length_h,
    int nCHin,
    int nCHout,
    int usePartFLAG )
```

Creates an instance of matrixConv.

This is a matrix convolver intended for block-by-block processing.

Parameters

in	*phMC*	(&) address of matrixConv handle
in	*hopSize*	Hop size in samples.
in	*H*	Time-domain filters; FLAT: nCHout x nCHin x length_h
in	*length_h*	Length of the filters
in	*nCHin*	Number of input channels
in	*nCHout*	Number of output channels
in	*usePartFLAG*	'0': normal fft-based convolution, '1': fft-based partitioned convolution

Definition at line 50 of file saf_matrixConv.c.

6.33.2.3 saf_matrixConv_destroy()

```
void saf_matrixConv_destroy (
    void **const phMC )
```

Destroys an instance of matrixConv.

Parameters

in	*phMC*	(&) address of matrixConv handle

Definition at line 133 of file saf_matrixConv.c.

6.33.2.4 saf_multiConv_apply()

```
void saf_multiConv_apply (
        void *const hMC,
        float * inputSigs,
        float * outputSigs )
```

Performs the multi-channel convolution.

Parameters

in	*hMC*	multiConv handle
in	*inputSigs*	Input signals; FLAT: nCH x hopSize
out	*outputSigs*	Output signals; FLAT: nCH x hopSize

Definition at line 353 of file saf_matrixConv.c.

6.33.2.5 saf_multiConv_create()

```
void saf_multiConv_create (
        void **const phMC,
        int hopSize,
        float * H,
        int length_h,
        int nCH,
        int usePartFLAG )
```

Creates an instance of multiConv.

This is a multi-channel convolver intended for block-by-block processing.

Note

nCH can just be 1, in which case this is simply a single-channel convolver.

Parameters

in	*phMC*	(&) address of multiConv handle
in	*hopSize*	Hop size in samples.
in	*H*	Time-domain filters; FLAT: nCH x length_h
in	*length_h*	Length of the filters
in	*nCH*	Number of filters & input/output channels
in	*usePartFLAG*	'0': normal fft-based convolution, '1': fft-based partitioned convolution

Definition at line 251 of file saf_matrixConv.c.

6.33.2.6 saf_multiConv_destroy()

```
void saf_multiConv_destroy (
        void **const phMC )
```

Destroys an instance of multiConv.

Parameters

in	*phMC*	(&) address of multiConv handle

Definition at line 326 of file saf_matrixConv.c.

6.34 framework/modules/saf_utilities/saf_matrixConv.h File Reference

Matrix convolver functions mostly taken from some Matlab scripts by Archontis Politis.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
```

Functions

- void saf_matrixConv_create (void ∗∗const phMC, int hopSize, float ∗H, int length_h, int nCHin, int nCHout, int usePartFLAG)

  Creates an instance of matrixConv.
- void saf_matrixConv_destroy (void ∗∗const phMC)

  Destroys an instance of matrixConv.
- void saf_matrixConv_apply (void ∗const hMC, float ∗inputSigs, float ∗outputSigs)

  Performs the matrix convolution.
- void saf_multiConv_create (void ∗∗const phMC, int hopSize, float ∗H, int length_h, int nCH, int usePartFLAG)

  Creates an instance of multiConv.
- void saf_multiConv_destroy (void ∗∗const phMC)

  Destroys an instance of multiConv.
- void saf_multiConv_apply (void ∗const hMC, float ∗inputSigs, float ∗outputSigs)

  Performs the multi-channel convolution.

6.34.1 Detailed Description

Matrix convolver functions mostly taken from some Matlab scripts by Archontis Politis.

Author

Leo McCormack

Date

06.04.2019

6.34.2 Function Documentation

6.34.2.1 saf_matrixConv_apply()

```
void saf_matrixConv_apply (
        void *const hMC,
        float * inputSigs,
        float * outputSigs )
```

Performs the matrix convolution.

Note

If the number of input+output channels, the filters, or the hopsize need tochange: simply destroy and re-create the matrixConv instance.

Parameters

in	*hMC*	matrixConv handle
in	*inputSigs*	Input signals; FLAT: nCHin x hopSize
out	*outputSigs*	Output signals; FLAT: nCHout x hopSize

Definition at line 164 of file saf_matrixConv.c.

6.34.2.2 saf_matrixConv_create()

```
void saf_matrixConv_create (
        void **const phMC,
        int hopSize,
        float * H,
        int length_h,
        int nCHin,
        int nCHout,
        int usePartFLAG )
```

Creates an instance of matrixConv.

This is a matrix convolver intended for block-by-block processing.

Parameters

in	*phMC*	(&) address of matrixConv handle
in	*hopSize*	Hop size in samples.
in	*H*	Time-domain filters; FLAT: nCHout x nCHin x length_h
in	*length_h*	Length of the filters
in	*nCHin*	Number of input channels
in	*nCHout*	Number of output channels
in	*usePartFLAG*	'0': normal fft-based convolution, '1': fft-based partitioned convolution

Definition at line 50 of file saf_matrixConv.c.

6.34.2.3 saf_matrixConv_destroy()

```
void saf_matrixConv_destroy (
        void **const phMC )
```

Destroys an instance of matrixConv.

Parameters

in	*phMC*	(&) address of matrixConv handle

Definition at line 133 of file saf_matrixConv.c.

6.34.2.4 saf_multiConv_apply()

```
void saf_multiConv_apply (
    void *const hMC,
    float * inputSigs,
    float * outputSigs )
```

Performs the multi-channel convolution.

Parameters

in	*hMC*	multiConv handle
in	*inputSigs*	Input signals; FLAT: nCH x hopSize
out	*outputSigs*	Output signals; FLAT: nCH x hopSize

Definition at line 353 of file saf_matrixConv.c.

6.34.2.5 saf_multiConv_create()

```
void saf_multiConv_create (
    void **const phMC,
    int hopSize,
    float * H,
    int length_h,
    int nCH,
    int usePartFLAG )
```

Creates an instance of multiConv.

This is a multi-channel convolver intended for block-by-block processing.

Note

nCH can just be 1, in which case this is simply a single-channel convolver.

Parameters

in	*phMC*	(&) address of multiConv handle
in	*hopSize*	Hop size in samples.
in	*H*	Time-domain filters; FLAT: nCH x length_h
in	*length_h*	Length of the filters
in	*nCH*	Number of filters & input/output channels
in	*usePartFLAG*	'0': normal fft-based convolution, '1': fft-based partitioned convolution

Definition at line 251 of file saf_matrixConv.c.

6.34.2.6 saf_multiConv_destroy()

```
void saf_multiConv_destroy (
        void **const phMC )
```

Destroys an instance of multiConv.

Parameters

in	*phMC*	(&) address of multiConv handle

Definition at line 326 of file saf_matrixConv.c.

6.35 framework/modules/saf_utilities/saf_misc.c File Reference

Miscellaneous functions.

```
#include "saf_misc.h"
```

Functions

- long double factorial (int n)

    Factorial, accurate up to n<=25.
- float matlab_fmodf (float x, float y)

    C fmodf function, which behaves like 'mod' in Matlab (with the wrap around)
- void cxcorr (float ∗a, float ∗b, float ∗x_ab, size_t la, size_t lb)

    Calculates the cross correlation between two vectors.

6.35.1 Detailed Description

Miscellaneous functions.

Author

Leo McCormack

Date

29.01.2020

6.35.2 Function Documentation

6.35.2.1 cxcorr()

```
void cxcorr (
        float * a,
        float * b,
        float * x_ab,
        size_t la,
        size_t lb )
```

Calculates the cross correlation between two vectors.

Parameters

in	*a*	Vector a; la x 1
in	*b*	Vector b; lb x 1
in	*la*	Length of vector a
in	*lb*	Length of vector b
out	*x_ab*	Cross-correlation between a and b; (la + lb - 1) x 1

Definition at line 43 of file saf_misc.c.

6.35.2.2 factorial() `long double factorial ( int n )`

Factorial, accurate up to n<=25.

Note

The magnitude will still be correct >25, but the precision will be truncated.

Parameters

in	*n*	Order

Returns

factorial(n)

Definition at line 27 of file saf_misc.c.

6.35.2.3 matlab_fmodf() `float matlab_fmodf ( float x, float y )`

C fmodf function, which behaves like 'mod' in Matlab (with the wrap around)

Parameters

in	*x*	Value 'x'
in	*y*	Value 'y'

Returns

fmodf(n)

Definition at line 37 of file saf_misc.c.

6.36 framework/modules/saf_utilities/saf_misc.h File Reference

Miscellaneous functions.

```
#include <stdio.h>
#include <string.h>
#include <math.h>
```

Functions

- long double factorial (int n)

    Factorial, accurate up to n<=25.
- float matlab_fmodf (float x, float y)

    C fmodf function, which behaves like 'mod' in Matlab (with the wrap around)
- void cxcorr (float *a, float *b, float *x_ab, size_t la, size_t lb)

    Calculates the cross correlation between two vectors.

6.36.1 Detailed Description

Miscellaneous functions.

Author

Leo McCormack

Date

29.01.2020

6.36.2 Function Documentation

6.36.2.1 cxcorr()

```
void cxcorr (
    float * a,
    float * b,
    float * x_ab,
    size_t la,
    size_t lb )
```

Calculates the cross correlation between two vectors.

Parameters

in	*a*	Vector a; la x 1
in	*b*	Vector b; lb x 1
in	*la*	Length of vector a
in	*lb*	Length of vector b
out	*x_ab*	Cross-correlation between a and b; (la + lb - 1) x 1

Definition at line 43 of file saf_misc.c.

6.36.2.2 factorial()

```
long double factorial (
    int n )
```

Factorial, accurate up to n<=25.

Note

The magnitude will still be correct >25, but the precision will be truncated.

Parameters

in	*n*	Order

Returns

factorial(n)

Definition at line 27 of file saf_misc.c.

6.36.2.3 matlab_fmodf()

```
float matlab_fmodf (
    float x,
    float y )
```

C fmodf function, which behaves like 'mod' in Matlab (with the wrap around)

Parameters

in	*x*	Value 'x'
in	*y*	Value 'y'

Returns

fmodf(n)

Definition at line 37 of file saf_misc.c.

6.37 framework/modules/saf_utilities/saf_sensorarray_presets.c File Reference

Comprises the directions of microphone array sensors.

```
#include "saf_sensorarray_presets.h"
```

Variables

- const float __Aalto_Hydrophone_coords_rad [4][2]

  Sensor array coordinates for the custom hydrophone array made at Aalto University [1].
- const float __Sennheiser_Ambeo_coords_rad [4][2]

  Sensor array coordinates for the Sennheiser Ambeo.
- const float __Core_Sound_TetraMic_coords_rad [4][2]

  Sensor array coordinates for the Core Sound TetraMic.
- const float __Zoom_H3VR_coords_rad [4][2]

  Sensor array coordinates for the Zoom H3VR.
- const float __Sound_field_SPS200_coords_rad [4][2]

  Sensor array coordinates for the Sound-field SPS200.
- const float __Zylia1D_coords_rad [19][2]

  Sensor array coordinates for the Zylia mic.
- const float __Eigenmike32_coords_rad [32][2]

  Sensor array coordinates for the Eigenmike32.
- const float __DTU_mic_coords_rad [52][2]

  Sensor array coordinates for the custom 52-sensor array built at the Technical University of Denmark (DTU).
- const float __default_coords_rad [(UTIL_DEFAULT_SH_ORDER+1) ∗(UTIL_DEFAULT_SH_ORDER+1)][2]

  Default sensor array coordinates.
- const float __default_SENSORcoords64_rad [64][2]

  Default sensor array coordinates.
- const int __Aalto_Hydrophone_maxOrder = 1

  Max spherical harmonic order for the custom hydrophone array made at Aalto University.
- const int __Sennheiser_Ambeo_maxOrder = 1

  Max spherical harmonic order for the Sennheiser Ambeo.
- const int __Core_Sound_TetraMic_maxOrder = 1

  Max spherical harmonic order for the Core Sound TetraMic.
- const int __Sound_field_SPS200_maxOrder = 1

  Max spherical harmonic order for the Sound-field SPS200.
- const int __Zylia_maxOrder = 3

  Max spherical harmonic order for the Zylia mic.
- const int __Eigenmike32_maxOrder = 4

  Max spherical harmonic order for the Eigenmike32.
- const int __DTU_mic_maxOrder = 6

  Max spherical harmonic order for the custom 52-sensor array built at the Technical University of Denmark (DTU).
- const float __Zylia_freqRange [4]

  Sensor array frequency ranges for each SH order, for the Zylia array (should only be used as a rough estimate).
- const float __Eigenmike32_freqRange [6]

  Sensor array frequency ranges for each SH order, for the Eigenmike32 (should only be used as a rough estimate).
- const float __DTU_mic_freqRange [10]

  Sensor array frequency ranges for each SH order, for the DTU mic (should only be used as a rough estimate).

6.37.1 Detailed Description

Comprises the directions of microphone array sensors.

Author

Leo McCormack

Date

11.07.2016

6.37.2 Variable Documentation

6.37.2.1 __Aalto_Hydrophone_coords_rad `const float __Aalto_Hydrophone_coords_rad[4][2]`

Initial value:

```
=
{ { -2.35619449019235f,    -0.615490213662568f},
  { 0.785398163397448f,    -0.615479708670387f},
  { -0.785398163397448f,    0.615479708670387f},
  { 2.35619449019235f,    0.615479708670387f} }
```

Sensor array coordinates for the custom hydrophone array made at Aalto University [1].

See also

[1] Delikaris-Manias, S., McCormack, L., Huhtakallio, I., & Pulkki, V. (2018, May). Real-time underwater spatial audio: a feasibility study. In Audio Engineering Society Convention 144. Audio Engineering Society.

Definition at line 31 of file saf_sensorarray_presets.c.

6.37.2.2 __Core_Sound_TetraMic_coords_rad `const float __Core_Sound_TetraMic_coords_rad[4][2]`

Initial value:

```
=
{ { 0.785398163397448f,    0.615472907423280f},
  { -0.785398163397448f,    -0.615472907423280f},
  { 2.35619449019235f,    -0.615472907423280f},
  { -2.35619449019235f,    0.615472907423280f}}
```

Sensor array coordinates for the Core Sound TetraMic.

Definition at line 43 of file saf_sensorarray_presets.c.

6.37.2.3 __DTU_mic_freqRange `const float __DTU_mic_freqRange[10]`

Initial value:

```
=
{ 350.0f, 950.0f, 1700.0f, 2600.0f, 3500.0f, 5800.0f, 6600.0f, 7200.0f, 7700.0f, 8300.0f}
```

Sensor array frequency ranges for each SH order, for the DTU mic (should only be used as a rough estimate).

Definition at line 499 of file saf_sensorarray_presets.c.

6.37.2.4 __Eigenmike32_freqRange `const float __Eigenmike32_freqRange[6]`

Initial value:

```
=
{ 460.0f, 1200.0f, 2200.0f, 6500.0f, 7500.0f, 8300.0f}
```

Sensor array frequency ranges for each SH order, for the Eigenmike32 (should only be used as a rough estimate).

Definition at line 496 of file saf_sensorarray_presets.c.

6.37.2.5 __Sennheiser_Ambeo_coords_rad `const float __Sennheiser_Ambeo_coords_rad[4][2]`

Initial value:

```
=
{ { 0.785398163397448f,    0.615472907423280f},
  { -0.785398163397448f,    -0.615472907423280f},
  { 2.35619449019235f,    -0.615472907423280f},
  { -2.35619449019235f,    0.615472907423280f}}
```

Sensor array coordinates for the Sennheiser Ambeo.

Definition at line 37 of file saf_sensorarray_presets.c.

6.37.2.6 __Sound_field_SPS200_coords_rad `const float __Sound_field_SPS200_coords_rad[4][2]`

Initial value:

```
=
{ { 0.785398163397448f,    0.615472907423280f},
  { -0.785398163397448f,    -0.615472907423280f},
  { 2.35619449019235f,    -0.615472907423280f},
  { -2.35619449019235f,    0.615472907423280f}}
```

Sensor array coordinates for the Sound-field SPS200.

Definition at line 55 of file saf_sensorarray_presets.c.

6.37.2.7 __Zoom_H3VR_coords_rad `const float __Zoom_H3VR_coords_rad[4][2]`

Initial value:

```
=
{ { 0.785398163397448f,    0.615472907423280f},
    { -0.785398163397448f,    -0.615472907423280f},
    { 2.35619449019235f,    -0.615472907423280f},
    { -2.35619449019235f,    0.615472907423280f}}
```

Sensor array coordinates for the Zoom H3VR.

Definition at line 49 of file saf_sensorarray_presets.c.

6.37.2.8 __Zylia1D_coords_rad `const float __Zylia1D_coords_rad[19][2]`

Initial value:

```
=
{ { 0.0f,    1.57079632679490f},
  { 0.00305809444245928f,    0.840254037451382f},
  { 2.09600986753364f,    0.840126252832125f},
  { -2.09336058192593f,    0.840886905122138f},
  { -1.43409959239697f,    0.338967177556435f},
  { -0.656487391713457f,    0.339152933310760f},
  { 0.661232814211584f,    0.338858655681573f},
  { 1.43624308141539f,    0.339058915910358f},
  { 2.75545932621978f,    0.339167630604397f},
  { -2.75063229463181f,    0.339281599533891f},
  { -2.48035983937821f,    -0.338858655681573f},
  { -1.70534957217440f,    -0.339058915910358f},
  { -0.386133327370014f,    -0.339167630604397f},
  { 0.390960358957982f,    -0.339281599533891f},
  { 1.70749306119282f,    -0.338967177556435f},
  { 2.48510526187634f,    -0.339152933310760f},
  { -3.13853455914733f,    -0.840254037451382f},
  { -1.04558278605616f,    -0.840126252832125f},
  { 1.04823207166387f,    -0.840886905122138f}}
```

Sensor array coordinates for the Zylia mic.

Definition at line 66 of file saf_sensorarray_presets.c.

6.37.2.9 __Zylia_freqRange `const float __Zylia_freqRange[4]`

Initial value:

```
=
{ 420.0f, 1200.0f, 3500.0f, 3700.0f}
```

Sensor array frequency ranges for each SH order, for the Zylia array (should only be used as a rough estimate).

The upper frequency limits were selected as the point where the spatial correlation went <0.9. The lower frequency limits were selected as the point where the level difference exceeded 6dB (assuming a 15dB maximum amplification with the Tikhonov regularisation method for all mics).

For more information on determining the usable frequency range per spherical harmonic order, for a given microphone array, the reader is directed to [1].

See also

[1] Moreau, S., Daniel, J., & Bertet, S. (2006, May). 3D sound field recording with higher order ambisonics–objective measurements and validation of a 4th order spherical microphone. In 120th Convention of the AES (pp. 20-23).

Definition at line 493 of file saf_sensorarray_presets.c.

6.38 framework/modules/saf_utilities/saf_sensorarray_presets.h File Reference

Comprises the directions of microphone array sensors.

Macros

- #define **UTIL_DEFAULT_SH_ORDER** (7)

Variables

- const float __Aalto_Hydrophone_coords_rad [4][2]
  Sensor array coordinates for the custom hydrophone array made at Aalto University [1].
- const float __Sennheiser_Ambeo_coords_rad [4][2]
  Sensor array coordinates for the Sennheiser Ambeo.
- const float __Core_Sound_TetraMic_coords_rad [4][2]
  Sensor array coordinates for the Core Sound TetraMic.
- const float __Sound_field_SPS200_coords_rad [4][2]
  Sensor array coordinates for the Sound-field SPS200.
- const float __Zoom_H3VR_coords_rad [4][2]
  Sensor array coordinates for the Zoom H3VR.
- const float __Zylia1D_coords_rad [19][2]
  Sensor array coordinates for the Zylia mic.
- const float __Eigenmike32_coords_rad [32][2]
  Sensor array coordinates for the Eigenmike32.
- const float __DTU_mic_coords_rad [52][2]
  Sensor array coordinates for the custom 52-sensor array built at the Technical University of Denmark (DTU).
- const float __default_coords_rad [(UTIL_DEFAULT_SH_ORDER+1) ∗(UTIL_DEFAULT_SH_ORDER+1)][2]
  Default sensor array coordinates.

- const float __default_SENSORcoords64_rad [64][2]
  Default sensor array coordinates.
- const int __Aalto_Hydrophone_maxOrder
  Max spherical harmonic order for the custom hydrophone array made at Aalto University.
- const int __Sennheiser_Ambeo_maxOrder
  Max spherical harmonic order for the Sennheiser Ambeo.
- const int __Core_Sound_TetraMic_maxOrder
  Max spherical harmonic order for the Core Sound TetraMic.
- const int __Sound_field_SPS200_maxOrder
  Max spherical harmonic order for the Sound-field SPS200.
- const int __Zylia_maxOrder
  Max spherical harmonic order for the Zylia mic.
- const int __Eigenmike32_maxOrder
  Max spherical harmonic order for the Eigenmike32.
- const int __DTU_mic_maxOrder
  Max spherical harmonic order for the custom 52-sensor array built at the Technical University of Denmark (DTU).
- const float __Zylia_freqRange [4]
  Sensor array frequency ranges for each SH order, for the Zylia array (should only be used as a rough estimate).
- const float __Eigenmike32_freqRange [6]
  Sensor array frequency ranges for each SH order, for the Eigenmike32 (should only be used as a rough estimate).
- const float __DTU_mic_freqRange [10]
  Sensor array frequency ranges for each SH order, for the DTU mic (should only be used as a rough estimate).

6.38.1 Detailed Description

Comprises the directions of microphone array sensors.

Author

Leo McCormack

Date

11.07.2016

6.38.2 Variable Documentation

6.38.2.1 __Aalto_Hydrophone_coords_rad `const float __Aalto_Hydrophone_coords_rad[4][2]`

Sensor array coordinates for the custom hydrophone array made at Aalto University [1].

See also

[1] Delikaris-Manias, S., McCormack, L., Huhtakallio, I., & Pulkki, V. (2018, May). Real-time underwater spatial audio: a feasibility study. In Audio Engineering Society Convention 144. Audio Engineering Society.

Definition at line 31 of file saf_sensorarray_presets.c.

6.38.2.2 __Zylia_freqRange `const float __Zylia_freqRange[4]`

Sensor array frequency ranges for each SH order, for the Zylia array (should only be used as a rough estimate).

The upper frequency limits were selected as the point where the spatial correlation went <0.9. The lower frequency limits were selected as the point where the level difference exceeded 6dB (assuming a 15dB maximum amplification with the Tikhonov regularisation method for all mics).

For more information on determining the usable frequency range per spherical harmonic order, for a given microphone array, the reader is directed to [1].

See also

[1] Moreau, S., Daniel, J., & Bertet, S. (2006, May). 3D sound field recording with higher order ambisonics–objective measurements and validation of a 4th order spherical microphone. In 120th Convention of the AES (pp. 20-23).

Definition at line 493 of file saf_sensorarray_presets.c.

6.39 framework/modules/saf_utilities/saf_sort.c File Reference

Contains some useful sorting functions.

```
#include "saf_utilities.h"
```

Data Structures

- struct saf_sort_int

  Helper struct for sorting a vector of integers using 'qsort'.
- struct saf_sort_float

  Helper struct for sorting a vector of floats using 'qsort'.
- struct saf_sort_double

  Helper struct for sorting a vector of doubles using 'qsort'.

Functions

- static int cmp_asc_int (const void *a, const void *b)

  Helper function for sorting a vector of integers using 'qsort' in ascending order.
- static int cmp_desc_int (const void *a, const void *b)

  Helper function for a sorting vector of integers using 'qsort' in decending order.
- static int cmp_asc_float (const void *a, const void *b)

  Helper function for a sorting vector of floats using 'qsort' in ascending order.
- static int cmp_desc_float (const void *a, const void *b)

  Helper function for a sorting vector of floats using 'qsort' in decending order.
- static int cmp_asc_double (const void *a, const void *b)

  Helper function for a sorting vector of doubles using 'qsort' in ascending order.
- static int cmp_desc_double (const void *a, const void *b)

  Helper function for a sorting vector of doubles using 'qsort' in decending order.
- void sorti (int *in_vec, int *out_vec, int *new_idices, int len, int descendFLAG)

  Sort a vector of integer values into ascending/decending order (optionally returning the new indices as well).
- void sortf (float *in_vec, float *out_vec, int *new_idices, int len, int descendFLAG)

  Sort a vector of floating-point values into ascending/decending order (optionally returning the new indices as well).
- void sortd (double *in_vec, double *out_vec, int *new_idices, int len, int descendFLAG)

  Sort a vector of double floating-point values into ascending/decending order (optionally returning the new indices as well).
- void findClosestGridPoints (float *grid_dirs, int nGrid, float *target_dirs, int nTarget, int degFLAG, int *idx_↩ closest, float *dirs_closest, float *angle_diff)

  Finds indicies into "grid_dirs" that are the closest to "target dirs".

6.39.1 Detailed Description

Contains some useful sorting functions.

Author

Leo McCormack

Date

30.07.2018

6.39.2 Function Documentation

6.39.2.1 findClosestGridPoints()

```
void findClosestGridPoints (
            float * grid_dirs,
            int nGrid,
            float * target_dirs,
            int nTarget,
            int degFLAG,
            int * idx_closest,
            float * dirs_closest,
            float * angle_diff )
```

Finds indicies into "grid_dirs" that are the closest to "target dirs".

- grid_dirs[idx_closest[0]] will be the closest direction in "grid_dirs" to target_dirs[0].

Parameters

in	*grid_dirs*	Spherical coordinates of grid directions; FLAT: nGrid x 2
in	*nGrid*	Number of directions in grid
in	*target_dirs*	Spherical coordinates of target directions; FLAT: nTarget x 2
in	*nTarget*	Number of target directions to find
in	*degFLAG*	'0' coordinates are in RADIANS, '1' coords are in DEGREES
out	*idx_closest*	Resulting indices (set to NULL to ignore); nTarget x 1
out	*dirs_closest*	grid_dirs(idx_closest); (set to NULL to ignore); nTarget x 1
out	*angle_diff*	Angle diff between target a d grid dir, in degrees (set to NULL to ignore); nTarget x 1

Definition at line 220 of file saf_sort.c.

6.39.2.2 sortd()

```
void sortd (
            double * in_vec,
```

```
double * out_vec,
int * new_idices,
int len,
int descendFLAG )
```

Sort a vector of double floating-point values into ascending/decending order (optionally returning the new indices as well).

Parameters

in,out	*in_vec*	Vector to be sorted; len x 1
out	*out_vec*	Output vector. If NULL, then 'in_vec' is sorted "in-place"
out	*new_idices*	Indices used to sort 'in_vec' (set to NULL if you don't want them)
in	*len*	Number of elements in vectors
in	*descendFLAG*	'0' ascending, '1' descending

Definition at line 188 of file saf_sort.c.

6.39.2.3 sortf()

```
void sortf (
float * in_vec,
float * out_vec,
int * new_idices,
int len,
int descendFLAG )
```

Sort a vector of floating-point values into ascending/decending order (optionally returning the new indices as well).

Parameters

in,out	*in_vec*	Vector to be sorted; len x 1
out	*out_vec*	Output vector. If NULL, then 'in_vec' is sorted "in-place"
out	*new_idices*	Indices used to sort 'in_vec' (set to NULL if you don't want them)
in	*len*	Number of elements in vectors
in	*descendFLAG*	'0' ascending, '1' descending

Definition at line 156 of file saf_sort.c.

6.39.2.4 sorti()

```
void sorti (
int * in_vec,
int * out_vec,
int * new_idices,
int len,
int descendFLAG )
```

Sort a vector of integer values into ascending/decending order (optionally returning the new indices as well).

Parameters

in,out	*in_vec*	Vector to be sorted; len x 1
out	*out_vec*	Output vector. If NULL, then 'in_vec' is sorted "in-place"
out	*new_idices*	Indices used to sort 'in_vec' (set to NULL if you don't want them)
in	*len*	Number of elements in vectors
in	*descendFLAG*	'0' ascending, '1' descending

Definition at line 124 of file saf_sort.c.

6.40 framework/modules/saf_utilities/saf_sort.h File Reference

Contains some useful sorting functions.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <float.h>
#include <math.h>
```

Functions

- void sorti (int ∗in_vec, int ∗out_vec, int ∗new_idices, int len, int descendFLAG)

  Sort a vector of integer values into ascending/decending order (optionally returning the new indices as well).
- void sortf (float ∗in_vec, float ∗out_vec, int ∗new_idices, int len, int descendFLAG)

  Sort a vector of floating-point values into ascending/decending order (optionally returning the new indices as well).
- void sortd (double ∗in_vec, double ∗out_vec, int ∗new_idices, int len, int descendFLAG)

  Sort a vector of double floating-point values into ascending/decending order (optionally returning the new indices as well).
- void findClosestGridPoints (float ∗grid_dirs, int nGrid, float ∗target_dirs, int nTarget, int degFLAG, int ∗idx_↩ closest, float ∗dirs_closest, float ∗angle_diff)

  Finds indicies into "grid_dirs" that are the closest to "target dirs".

6.40.1 Detailed Description

Contains some useful sorting functions.

Author

Leo McCormack

Date

30.07.2018

6.40.2 Function Documentation

6.40.2.1 findClosestGridPoints()

```
void findClosestGridPoints (
    float * grid_dirs,
    int nGrid,
    float * target_dirs,
    int nTarget,
    int degFLAG,
    int * idx_closest,
    float * dirs_closest,
    float * angle_diff )
```

Finds indicies into "grid_dirs" that are the closest to "target dirs".

- grid_dirs[idx_closest[0]] will be the closest direction in "grid_dirs" to target_dirs[0].

Parameters

in	*grid_dirs*	Spherical coordinates of grid directions; FLAT: nGrid x 2
in	*nGrid*	Number of directions in grid
in	*target_dirs*	Spherical coordinates of target directions; FLAT: nTarget x 2
in	*nTarget*	Number of target directions to find
in	*degFLAG*	'0' coordinates are in RADIANS, '1' coords are in DEGREES
out	*idx_closest*	Resulting indices (set to NULL to ignore); nTarget x 1
out	*dirs_closest*	grid_dirs(idx_closest); (set to NULL to ignore); nTarget x 1
out	*angle_diff*	Angle diff between target a d grid dir, in degrees (set to NULL to ignore); nTarget x 1

Definition at line 220 of file saf_sort.c.

6.40.2.2 sortd()

```
void sortd (
    double * in_vec,
    double * out_vec,
    int * new_idices,
    int len,
    int descendFLAG )
```

Sort a vector of double floating-point values into ascending/decending order (optionally returning the new indices as well).

Parameters

in,out	*in_vec*	Vector to be sorted; len x 1
out	*out_vec*	Output vector. If NULL, then 'in_vec' is sorted "in-place"
out	*new_idices*	Indices used to sort 'in_vec' (set to NULL if you don't want them)
in	*len*	Number of elements in vectors
in	*descendFLAG*	'0' ascending, '1' descending

Definition at line 188 of file saf_sort.c.

6.40.2.3 sortf()

```
void sortf (
    float * in_vec,
    float * out_vec,
    int * new_idices,
    int len,
    int descendFLAG )
```

Sort a vector of floating-point values into ascending/decending order (optionally returning the new indices as well).

Parameters

in,out	*in_vec*	Vector to be sorted; len x 1
out	*out_vec*	Output vector. If NULL, then 'in_vec' is sorted "in-place"
out	*new_idices*	Indices used to sort 'in_vec' (set to NULL if you don't want them)
in	*len*	Number of elements in vectors
in	*descendFLAG*	'0' ascending, '1' descending

Definition at line 156 of file saf_sort.c.

6.40.2.4 sorti()

```
void sorti (
    int * in_vec,
    int * out_vec,
    int * new_idices,
    int len,
    int descendFLAG )
```

Sort a vector of integer values into ascending/decending order (optionally returning the new indices as well).

Parameters

in,out	*in_vec*	Vector to be sorted; len x 1
out	*out_vec*	Output vector. If NULL, then 'in_vec' is sorted "in-place"
out	*new_idices*	Indices used to sort 'in_vec' (set to NULL if you don't want them)
in	*len*	Number of elements in vectors
in	*descendFLAG*	'0' ascending, '1' descending

Definition at line 124 of file saf_sort.c.

6.41 framework/modules/saf_utilities/saf_utilities.h File Reference

Contains a collection of useful memory allocation functions, cross- platform complex number wrappers, and optimised linear algebra routines utilising BLAS and LAPACK.

```
#include "../saf_utilities/saf_error.h"
#include "../resources/md_malloc/md_malloc.h"
#include "../resources/kissFFT/kiss_fftr.h"
#include "../resources/convhull_3d/convhull_3d.h"
#include "../saf_utilities/saf_complex.h"
#include "../saf_utilities/saf_sort.h"
#include "../saf_utilities/saf_filters.h"
#include "../saf_utilities/saf_veclib.h"
#include "../saf_utilities/saf_fft.h"
#include "../saf_utilities/saf_matrixConv.h"
#include "../saf_utilities/saf_decor.h"
#include "../saf_utilities/saf_erb.h"
#include "../saf_utilities/saf_misc.h"
#include "../saf_utilities/saf_loudspeaker_presets.h"
#include "../saf_utilities/saf_sensorarray_presets.h"
```

Macros

- #define **MIN**(a, b) (((a) < (b)) ? (a) : (b))
- #define **MAX**(a, b) (((a) > (b)) ? (a) : (b))
- #define **CLAMP**(a, min, max) (MAX(min, MIN(max, a)))
- #define **PI** (3.14159265358979323846264338327950288f)
- #define **M_PI** (3.14159265358979323846264338327950288f)
- #define **SAF_ISPOW2**(x) (((x & ∼(x-1))==x) ? x : 0);

6.41.1 Detailed Description

Contains a collection of useful memory allocation functions, cross- platform complex number wrappers, and optimised linear algebra routines utilising BLAS and LAPACK.

6.41.1.1 Dependencies A performance library comprising CBLAS and LAPACK routines is required by the module and, thus, also by the SAF framework as a whole. Add one of the following FLAGS to your project's preprocessor definitions list, in order to enable one of these suitable performance libraries, which must also be linked correctly to your project.

- SAF_USE_INTEL_MKL: to enable Intel's Math Kernal Library with Fortran LAPACK interface
- SAF_USE_ATLAS: to enable ATLAS BLAS routines and ATLAS's CLAPACK interface
- SAF_USE_OPENBLAS_WITH_LAPACKE: to enable OpenBLAS with LAPACKE interface

See also

More information can be found here: https://github.com/leomccormack/Spatial_Audio_Framework

Note

MacOSX users only: saf_utilities will employ Apple's Accelerate library by default, if none of the above FLAGS are defined.

Author

Leo McCormack

Date

11.07.2016

6.42 framework/modules/saf_utilities/saf_veclib.c File Reference

Contains wrappers for optimised linear algebra routines, utilising CBLAS and LAPACK.

```
#include "saf_utilities.h"
#include <float.h>
```

Typedefs

- typedef int **veclib_int**
- typedef float **veclib_float**
- typedef float_complex **veclib_float_complex**
- typedef double_complex **veclib_double_complex**

Functions

- void utility_siminv (const float ∗a, const int len, int ∗index)
  Single-precision, index of minimum absolute value in a vector, i.e.
- void utility_ciminv (const float_complex ∗a, const int len, int ∗index)
  Single-precision, complex, index of maximum absolute value in a vector, i.e.
- void utility_simaxv (const float ∗a, const int len, int ∗index)
  Single-precision, index of maximum absolute value in a vector, i.e.
- void utility_cimaxv (const float_complex ∗a, const int len, int ∗index)
  Single-precision, complex, index of maximum absolute value in a vector, i.e.
- void utility_svabs (const float ∗a, const int len, float ∗c)
  Single-precision, absolute values of vector elements, i.e.
- void utility_cvabs (const float_complex ∗a, const int len, float ∗c)
  Single-precision, complex, absolute values of vector elements, i.e.
- void utility_svvcopy (const float ∗a, const int len, float ∗c)
  Single-precision, vector-vector copy, i.e.
- void utility_cvvcopy (const float_complex ∗a, const int len, float_complex ∗c)
  Single-precision, complex, vector-vector copy, i.e.
- void utility_svvadd (float ∗a, const float ∗b, const int len, float ∗c)
  Single-precision, vector-vector addition, i.e.
- void utility_cvvadd (float_complex ∗a, const float_complex ∗b, const int len, float_complex ∗c)
  Single-precision, complex, vector-vector addition, i.e.

- void utility_svvsub (float *a, const float *b, const int len, float *c)

  Single-precision, vector-vector subtraction, i.e.
- void utility_cvvsub (float_complex *a, const float_complex *b, const int len, float_complex *c)

  Single-precision, complex, vector-vector subtraction, i.e.
- void utility_svvmul (float *a, const float *b, const int len, float *c)

  Single-precision, element-wise vector-vector multiplication i.e.
- void utility_cvvmul (float_complex *a, const float_complex *b, const int len, float_complex *c)

  Single-precision, complex, element-wise vector-vector multiplication i.e.
- void utility_svvdot (const float *a, const float *b, const int len, float *c)

  Single-precision, vector-vector dot product, i.e.
- void utility_cvvdot (const float_complex *a, const float_complex *b, const int len, CONJ_FLAG flag, float_↩ complex *c)

  Single-precision, complex, vector-vector dot product, i.e.
- void utility_svsmul (float *a, const float *s, const int len, float *c)

  Single-precision, multiplies each element in vector 'a' with a scalar 's', i.e.
- void utility_cvsmul (float_complex *a, const float_complex *s, const int len, float_complex *c)

  Single-precision, complex, multiplies each element in vector 'a' with a scalar 's', i.e.
- void utility_svsdiv (float *a, const float *s, const int len, float *c)

  Single-precision, divides each element in vector 'a' with a scalar 's', i.e.
- void utility_svsadd (float *a, const float *s, const int len, float *c)

  Single-precision, adds each element in vector 'a' with a scalar 's', i.e.
- void utility_svssub (float *a, const float *s, const int len, float *c)

  Single-precision, subtracts each element in vector 'a' with a scalar 's', i.e.
- void utility_ssvd (const float *A, const int dim1, const int dim2, float *U, float *S, float *V, float *sing)

  Row-major, singular value decomposition: single precision, i.e.
- void utility_csvd (const float_complex *A, const int dim1, const int dim2, float_complex *U, float_complex *S, float_complex *V, float *sing)

  Row-major, singular value decomposition: single precision complex, i.e.
- void utility_sseig (const float *A, const int dim, int sortDecFLAG, float *V, float *D, float *eig)

  Row-major, eigenvalue decomposition of a SYMMETRIC matrix: single precision, i.e.
- void utility_cseig (const float_complex *A, const int dim, int sortDecFLAG, float_complex *V, float_complex *D, float *eig)

  Row-major, eigenvalue decomposition of a SYMMETRIC/HERMITION matrix: single precision complex, i.e.
- void utility_ceigmp (const float_complex *A, const float_complex *B, const int dim, float_complex *VL, float↩ _complex *VR, float_complex *D)

  Row-major, finds eigenvalues of a matrix pair using the QZ method, single precision complex, i.e.
- void utility_zeigmp (const double_complex *A, const double_complex *B, const int dim, double_complex *VL, double_complex *VR, double_complex *D)

  Row-major, finds eigenvalues of a matrix pair using the QZ method, double precision complex, i.e.
- void utility_ceig (const float_complex *A, const int dim, int sortDecFLAG, float_complex *VL, float_complex *VR, float_complex *D, float *eig)

  Row-major, eigenvalue decomposition of a NON-SYMMETRIC matrix: single precision complex, i.e.
- void utility_sglslv (const float *A, const int dim, float *B, int nCol, float *X)

  Row-major, general linear solver: single precision, i.e.
- void utility_cglslv (const float_complex *A, const int dim, float_complex *B, int nCol, float_complex *X)

  Row-major, general linear solver: single precision complex, i.e.
- void utility_dglslv (const double *A, const int dim, double *B, int nCol, double *X)

  Row-major, general linear solver: double precision, i.e.
- void utility_zglslv (const double_complex *A, const int dim, double_complex *B, int nCol, double_complex *X)

  Row-major, general linear solver: double precision complex, i.e.
- void utility_sslslv (const float *A, const int dim, float *B, int nCol, float *X)

Row-major, linear solver for SYMMETRIC positive-definate 'A': single precision, i.e.

- void utility_cslslv (const float_complex ∗A, const int dim, float_complex ∗B, int nCol, float_complex ∗X)

  Row-major, linear solver for HERMITIAN positive-definate 'A': single precision complex, i.e.
- void utility_spinv (const float ∗inM, const int dim1, const int dim2, float ∗outM)

  Row-major, general matrix pseudo-inverse (the svd way): single precision, i.e.
- void utility_cpinv (const float_complex ∗inM, const int dim1, const int dim2, float_complex ∗outM)

  Row-major, general matrix pseudo-inverse (the svd way): single precision complex, i.e.
- void utility_dpinv (const double ∗inM, const int dim1, const int dim2, double ∗outM)

  Row-major, general matrix pseudo-inverse (the svd way): double precision, i.e.
- void utility_zpinv (const double_complex ∗inM, const int dim1, const int dim2, double_complex ∗outM)

  Row-major, general matrix pseudo-inverse (the svd way): double precision complex, i.e.
- void utility_schol (const float ∗A, const int dim, float ∗X)

  Row-major, Cholesky factorisation of a symmetric matrix positive-definate matrix: single precision, i.e.
- void utility_cchol (const float_complex ∗A, const int dim, float_complex ∗X)

  Row-major, Cholesky factorisation of a hermitian matrix positive-definate matrix: single precision complex, i.e.
- void **utility_sinv** (float ∗A, const int N)
- void **utility_dinv** (double ∗A, const int N)
- void **utility_cinv** (float_complex ∗A, const int N)

6.42.1 Detailed Description

Contains wrappers for optimised linear algebra routines, utilising CBLAS and LAPACK.

6.42.1.1 Dependencies

A performance library comprising CBLAS and LAPACK routines is required by the module and, thus, also by the SAF framework as a whole. Add one of the following FLAGS to your project's preprocessor definitions list, in order to enable one of these suitable performance libraries, which must also be linked correctly to your project.

- SAF_USE_INTEL_MKL: to enable Intel's Math Kernal Library with Fortran LAPACK interface
- SAF_USE_ATLAS: to enable ATLAS BLAS routines and ATLAS's CLAPACK interface
- SAF_USE_OPENBLAS_WITH_LAPACKE: to enable OpenBLAS with LAPACKE interface

See also

More information can be found here: https://github.com/leomccormack/Spatial_↩
Audio_Framework

Note

MacOSX users only: saf_utilities will employ Apple's Accelerate library by default, if none of the above FLAGS are defined.

Author

Leo McCormack

Date

11.07.2016

6.42.2 Function Documentation

6.42.2.1 utility_cchol()

```
void utility_cchol (
        const float_complex * A,
        const int dim,
        float_complex * X )
```

Row-major, Cholesky factorisation of a hermitian matrix positive-definate matrix: single precision complex, i.e.
`X = chol(A); where A = X.'*X`

Parameters

in	*A*	Input square symmetric positive-definate matrix; FLAT: dim x dim
in	*dim*	Number of rows/colums in 'A'
out	*X*	The solution; FLAT: dim x dim

Definition at line 1954 of file saf_veclib.c.

6.42.2.2 utility_ceig()

```
void utility_ceig (
        const float_complex * A,
        const int dim,
        int sortDecFLAG,
        float_complex * VL,
        float_complex * VR,
        float_complex * D,
        float * eig )
```

Row-major, eigenvalue decomposition of a NON-SYMMETRIC matrix: single precision complex, i.e.
`[VL,VR,D] = eig(A); where A*VR = VR*D, and  A*VR = VR*diag(eig)`

Note

'D' contains the eigen values along the diagonal, while 'eig' are the eigen values as a vector

Parameters

in	*A*	Input NON-SYMMETRIC square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*sortDecFLAG*	'1' sort eigen values and vectors in decending order. '0' ascending
out	*VL*	Left Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*VR*	Right Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim
out	*eig*	Eigen values not diagonalised (set to NULL if not needed); dim x 1

Definition at line 1127 of file saf_veclib.c.

6.42.2.3 utility_ceigmp()

```
void utility_ceigmp (
    const float_complex * A,
    const float_complex * B,
    const int dim,
    float_complex * VL,
    float_complex * VR,
    float_complex * D )
```

Row-major, finds eigenvalues of a matrix pair using the QZ method, single precision complex, i.e.
`[VL,VR,D] = eig(A,B,'qz'); where A*VL = B*VL*VR`

Parameters

in	*A*	Input left square matrix; FLAT: dim x dim
in	*B*	Input right square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrices 'A' and 'B'
out	*VL*	Left Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*VR*	Right Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim

Definition at line 946 of file saf_veclib.c.

6.42.2.4 utility_cglslv()

```
void utility_cglslv (
    const float_complex * A,
    const int dim,
    float_complex * B,
    int nCol,
    float_complex * X )
```

Row-major, general linear solver: single precision complex, i.e.
`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1286 of file saf_veclib.c.

6.42.2.5 utility_cimaxv()

```
void utility_cimaxv (
    const float_complex * a,
    const int len,
    int * index )
```

Single-precision, complex, index of maximum absolute value in a vector, i.e.
`[~,ind] = max(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) index of maximum value; 1 x 1

Definition at line 146 of file saf_veclib.c.

6.42.2.6 utility_ciminv()

```
void utility_ciminv (
        const float_complex * a,
        const int len,
        int * index )
```

Single-precision, complex, index of maximum absolute value in a vector, i.e. `[~,ind] = min(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) index of minimum value; 1 x 1

Definition at line 98 of file saf_veclib.c.

6.42.2.7 utility_cpinv()

```
void utility_cpinv (
        const float_complex * A,
        const int dim1,
        const int dim2,
        float_complex * B )
```

Row-major, general matrix pseudo-inverse (the svd way): single precision complex, i.e. `B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1638 of file saf_veclib.c.

6.42.2.8 utility_cseig()

```
void utility_cseig (
        const float_complex * A,
```

```
const int dim,
int sortDecFLAG,
float_complex * V,
float_complex * D,
float * eig )
```

Row-major, eigenvalue decomposition of a SYMMETRIC/HERMITION matrix: single precision complex, i.e.

```
[V,D] = eig(A); where A*V = V*D, and  A*V = V*diag(eig)
```

Note

'D' contains the eigen values along the diagonal, while 'eig' are the eigen values as a vector

Parameters

in	*A*	Input SYMMETRIC square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*sortDecFLAG*	'1' sort eigen values and vectors in decending order. '0' ascending
out	*V*	Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim
out	*eig*	Eigen values not diagonalised (set to NULL if not needed); dim x 1

Definition at line 853 of file saf_veclib.c.

6.42.2.9 utility_cslslv()

```
void utility_cslslv (
const float_complex * A,
const int dim,
float_complex * B,
int nCol,
float_complex * X )
```

Row-major, linear solver for HERMITIAN positive-definate 'A': single precision complex, i.e.

```
opts.LT=true
X = linsolve(A,B, opts); where, AX = B
```

Parameters

in	*A*	Input square SYMMETRIC positive-definate matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1500 of file saf_veclib.c.

6.42.2.10 utility_csvd()

```
void utility_csvd (
const float_complex * A,
```

```
const int dim1,
const int dim2,
float_complex * U,
float_complex * S,
float_complex * V,
float * sing )
```

Row-major, singular value decomposition: single precision complex, i.e.
`[U,S,V] = svd(A); such that A = U*S*V' = U*diag(sing)*V'`

Note

'S' contains the singular values along the diagonal, whereas 'sing' are the singular values as a vector. Also, V is returned untransposed! (like in Matlab)

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	First dimension of matrix 'A'
in	*dim2*	Second dimension of matrix 'A'
out	*U*	Left matrix (set to NULL if not needed); FLAT: dim1 x dim1
out	*S*	Singular values along the diagonal min(dim1, dim2), (set to NULL if not needed); FLAT: dim1 x dim2
out	*V*	Right matrix (UNTRANSPOSED!) (set to NULL if not needed); FLAT: dim2 x dim2
out	*sing*	Singular values as a vector, (set to NULL if not needed); min(dim1, dim2) x 1

Definition at line 672 of file saf_veclib.c.

6.42.2.11 utility_cvabs()

```
void utility_cvabs (
const float_complex * a,
const int len,
float * c )
```

Single-precision, complex, absolute values of vector elements, i.e.
`c = abs(a)`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 177 of file saf_veclib.c.

6.42.2.12 utility_cvsmul()

```
void utility_cvsmul (
float_complex * a,
const float_complex * s,
```

```
const int len,
float_complex * c )
```

Single-precision, complex, multiplies each element in vector 'a' with a scalar 's', i.e.

```
c = a.*s, OR: a = a.*s (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output);

Definition at line 489 of file saf_veclib.c.

6.42.2.13 utility_cvvadd()

```
void utility_cvvadd (
float_complex * a,
const float_complex * b,
const int len,
float_complex * c )
```

Single-precision, complex, vector-vector addition, i.e.

```
c = a+b, OR: a = a+b (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 258 of file saf_veclib.c.

6.42.2.14 utility_cvvcopy()

```
void utility_cvvcopy (
const float_complex * a,
const int len,
float_complex * c )
```

Single-precision, complex, vector-vector copy, i.e.

```
c = a
```

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 208 of file saf_veclib.c.

6.42.2.15 utility_cvvdot()

```
void utility_cvvdot (
        const float_complex * a,
        const float_complex * b,
        const int len,
        CONJ_FLAG flag,
        float_complex * c )
```

Single-precision, complex, vector-vector dot product, i.e.

```
c = a*b^T, (where size(c) = [1  1])
```

Parameters

in	*a*	Input vector a; len x 1
in	*b*	Input vector b; len x 1
in	*flag*	'0' do not take the conjugate of 'b', '1', take the conjugate of 'b'.
in	*len*	Vector length
out	*c*	(&) output vector c; 1 x 1

Definition at line 441 of file saf_veclib.c.

6.42.2.16 utility_cvvmul()

```
void utility_cvvmul (
        float_complex * a,
        const float_complex * b,
        const int len,
        float_complex * c )
```

Single-precision, complex, element-wise vector-vector multiplication i.e.

```
c = a.*b, OR: a = a.*b (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 394 of file saf_veclib.c.

6.42.2.17 utility_cvvsub()

```
void utility_cvvsub (
        float_complex * a,
        const float_complex * b,
        const int len,
        float_complex * c )
```

Single-precision, complex, vector-vector subtraction, i.e.

```
c = a-b, OR: a = a-b (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 323 of file saf_veclib.c.

6.42.2.18 utility_dglslv()

```
void utility_dglslv (
    const double * A,
    const int dim,
    double * B,
    int nCol,
    double * X )
```

Row-major, general linear solver: double precision, i.e.
`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1339 of file saf_veclib.c.

6.42.2.19 utility_dpinv()

```
void utility_dpinv (
    const double * A,
    const int dim1,
    const int dim2,
    double * B )
```

Row-major, general matrix pseudo-inverse (the svd way): double precision, i.e.
`B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1730 of file saf_veclib.c.

6.42.2.20 utility_schol()

```
void utility_schol (
    const float * A,
    const int dim,
    float * X )
```

Row-major, Cholesky factorisation of a symmetric matrix positive-definate matrix: single precision, i.e.

`X = chol(A); where A = X.'*X`

Parameters

in	*A*	Input square symmetric positive-definate matrix; FLAT: dim x dim
in	*dim*	Number of rows/colums in 'A'
out	*X*	The solution; FLAT: dim x dim

Definition at line 1909 of file saf_veclib.c.

6.42.2.21 utility_sglslv()

```
void utility_sglslv (
    const float * A,
    const int dim,
    float * B,
    int nCol,
    float * X )
```

Row-major, general linear solver: single precision, i.e.

`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1233 of file saf_veclib.c.

6.42.2.22 utility_simaxv()

```
void utility_simaxv (
    const float * a,
    const int len,
    int * index )
```

Single-precision, index of maximum absolute value in a vector, i.e.

`[~,ind] = max(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) index of maximum value; 1 x 1

Definition at line 136 of file saf_veclib.c.

6.42.2.23 utility_siminv()

```
void utility_siminv (
    const float * a,
    const int len,
    int * index )
```

Single-precision, index of minimum absolute value in a vector, i.e.
`[~,ind] = min(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) Index of minimum value; 1 x 1

Definition at line 71 of file saf_veclib.c.

6.42.2.24 utility_spinv()

```
void utility_spinv (
    const float * A,
    const int dim1,
    const int dim2,
    float * B )
```

Row-major, general matrix pseudo-inverse (the svd way): single precision, i.e.
`B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1555 of file saf_veclib.c.

6.42.2.25 utility_sseig()

```
void utility_sseig (
    const float * A,
```

```
const int dim,
int sortDecFLAG,
float * V,
float * D,
float * eig )
```

Row-major, eigenvalue decomposition of a SYMMETRIC matrix: single precision, i.e.

```
[V,D] = eig(A); where A*V = V*D, and  A*V = V*diag(eig)
```

Note

'D' contains the eigen values along the diagonal, while 'eig' are the eigen values as a vector

Parameters

in	*A*	Input SYMMETRIC square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*sortDecFLAG*	'1' sort eigen values and vectors in decending order. '0' ascending
out	*V*	Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim
out	*eig*	Eigen values not diagonalised (set to NULL if not needed); dim x 1

Definition at line 775 of file saf_veclib.c.

6.42.2.26 utility_sslslv()

```
void utility_sslslv (
const float * A,
const int dim,
float * B,
int nCol,
float * X )
```

Row-major, linear solver for SYMMETRIC positive-definate 'A': single precision, i.e.

```
opts.LT=true
X = linsolve(A,B, opts); where, AX = B
```

Parameters

in	*A*	Input square SYMMETRIC positive-definate matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1450 of file saf_veclib.c.

6.42.2.27 utility_ssvd()

```
void utility_ssvd (
const float * A,
```

```
const int dim1,
const int dim2,
float * U,
float * S,
float * V,
float * sing )
```

Row-major, singular value decomposition: single precision, i.e.

```
[U,S,V] = svd(A); such that A = U*S*V.' = U*diag(sing)*V.'
```

Note

'S' contains the singular values along the diagonal, whereas 'sing' are the singular values as a vector. Also, V is returned untransposed! (like in Matlab)

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	First dimension of matrix 'A'
in	*dim2*	Second dimension of matrix 'A'
out	*U*	Left matrix (set to NULL if not needed); FLAT: dim1 x dim1
out	*S*	Singular values along the diagonal min(dim1, dim2), (set to NULL if not needed); FLAT: dim1 x dim2
out	*V*	Right matrix (UNTRANSPOSED!) (set to NULL if not needed); FLAT: dim2 x dim2
out	*sing*	Singular values as a vector, (set to NULL if not needed); min(dim1, dim2) x 1

Definition at line 577 of file saf_veclib.c.

6.42.2.28 utility_svabs()

```
void utility_svabs (
const float * a,
const int len,
float * c )
```

Single-precision, absolute values of vector elements, i.e.

```
c = abs(a)
```

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 161 of file saf_veclib.c.

6.42.2.29 utility_svsadd()

```
void utility_svsadd (
float * a,
const float * s,
```

```
const int len,
float * c )
```

Single-precision, adds each element in vector 'a' with a scalar 's', i.e.
`c = a+s, OR: a = a+s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 537 of file saf_veclib.c.

6.42.2.30 utility_svsdiv()

```
void utility_svsdiv (
float * a,
const float * s,
const int len,
float * c )
```

Single-precision, divides each element in vector 'a' with a scalar 's', i.e.
`c = a./s, OR: a = a./s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 511 of file saf_veclib.c.

6.42.2.31 utility_svsmul()

```
void utility_svsmul (
float * a,
const float * s,
const int len,
float * c )
```

Single-precision, multiplies each element in vector 'a' with a scalar 's', i.e.
`c = a.*s, OR: a = a.*s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 466 of file saf_veclib.c.

6.42.2.32 utility_svssub()

```
void utility_svssub (
    float * a,
    const float * s,
    const int len,
    float * c )
```

Single-precision, subtracts each element in vector 'a' with a scalar 's', i.e.
`c = a-s, OR: a = a-s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 559 of file saf_veclib.c.

6.42.2.33 utility_svvadd()

```
void utility_svvadd (
    float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, vector-vector addition, i.e.
`c = a+b, OR: a = a+b (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 223 of file saf_veclib.c.

6.42.2.34 utility_svvcopy()

```
void utility_svvcopy (
    const float * a,
    const int len,
    float * c )
```

Single-precision, vector-vector copy, i.e.
`c = a`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 198 of file saf_veclib.c.

6.42.2.35 utility_svvdot()

```
void utility_svvdot (
    const float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, vector-vector dot product, i.e.
`c = a*b^T, (where size(c) = [1  1])`

Parameters

in	*a*	Input vector a; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	(&) output vector c; 1 x 1

Definition at line 430 of file saf_veclib.c.

6.42.2.36 utility_svvmul()

```
void utility_svvmul (
    float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, element-wise vector-vector multiplication i.e.
`c = a.*b, OR: a = a.*b (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 359 of file saf_veclib.c.

6.42.2.37 utility_svvsub()

```
void utility_svvsub (
    float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, vector-vector subtraction, i.e.
`c = a-b, OR: a = a-b (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 288 of file saf_veclib.c.

6.42.2.38 utility_zeigmp()

```
void utility_zeigmp (
    const double_complex * A,
    const double_complex * B,
    const int dim,
    double_complex * VL,
    double_complex * VR,
    double_complex * D )
```

Row-major, finds eigenvalues of a matrix pair using the QZ method, double precision complex, i.e.
`[VL,VR,D] = eig(A,B,'qz'); where A*VL = B*VL*VR`

Parameters

in	*A*	Input left square matrix; FLAT: dim x dim
in	*B*	Input right square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrices 'A' and 'B'
out	*VL*	Left Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*VR*	Right Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim

Definition at line 1034 of file saf_veclib.c.

6.42.2.39 utility_zglslv()

```
void utility_zglslv (
    const double_complex * A,
    const int dim,
    double_complex * B,
    int nCol,
    double_complex * X )
```

Row-major, general linear solver: double precision complex, i.e.
`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1392 of file saf_veclib.c.

6.42.2.40 utility_zpinv()

```
void utility_zpinv (
    const double_complex * A,
    const int dim1,
    const int dim2,
    double_complex * B )
```

Row-major, general matrix pseudo-inverse (the svd way): double precision complex, i.e. `B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1812 of file saf_veclib.c.

6.43 framework/modules/saf_utilities/saf_veclib.h File Reference

Contains wrappers for optimised linear algebra routines, utilising CBLAS and LAPACK.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "saf_complex.h"
#include "saf_error.h"
```

Enumerations

- enum _TRANS_FLAG { NO_TRANSPOSE = 1, TRANSPOSE = 2, CONJ_TRANSPOSE = 3 }
- enum _CONJ_FLAG { NO_CONJ = 1, CONJ = 2 }

Functions

- void utility_siminv (const float ∗a, const int len, int ∗index)

    Single-precision, index of minimum absolute value in a vector, i.e.
- void utility_ciminv (const float_complex ∗a, const int len, int ∗index)

    Single-precision, complex, index of maximum absolute value in a vector, i.e.
- void utility_simaxv (const float ∗a, const int len, int ∗index)

    Single-precision, index of maximum absolute value in a vector, i.e.
- void utility_cimaxv (const float_complex ∗a, const int len, int ∗index)

    Single-precision, complex, index of maximum absolute value in a vector, i.e.
- void utility_svabs (const float ∗a, const int len, float ∗c)

    Single-precision, absolute values of vector elements, i.e.
- void utility_cvabs (const float_complex ∗a, const int len, float ∗c)

    Single-precision, complex, absolute values of vector elements, i.e.
- void utility_svvcopy (const float ∗a, const int len, float ∗c)

    Single-precision, vector-vector copy, i.e.
- void utility_cvvcopy (const float_complex ∗a, const int len, float_complex ∗c)

    Single-precision, complex, vector-vector copy, i.e.
- void utility_svvadd (float ∗a, const float ∗b, const int len, float ∗c)

    Single-precision, vector-vector addition, i.e.
- void utility_cvvadd (float_complex ∗a, const float_complex ∗b, const int len, float_complex ∗c)

    Single-precision, complex, vector-vector addition, i.e.
- void utility_svvsub (float ∗a, const float ∗b, const int len, float ∗c)

    Single-precision, vector-vector subtraction, i.e.
- void utility_cvvsub (float_complex ∗a, const float_complex ∗b, const int len, float_complex ∗c)

    Single-precision, complex, vector-vector subtraction, i.e.
- void utility_svvmul (float ∗a, const float ∗b, const int len, float ∗c)

    Single-precision, element-wise vector-vector multiplication i.e.
- void utility_cvvmul (float_complex ∗a, const float_complex ∗b, const int len, float_complex ∗c)

    Single-precision, complex, element-wise vector-vector multiplication i.e.
- void utility_svvdot (const float ∗a, const float ∗b, const int len, float ∗c)

    Single-precision, vector-vector dot product, i.e.
- void utility_cvvdot (const float_complex ∗a, const float_complex ∗b, const int len, CONJ_FLAG flag, float_↩ complex ∗c)

    Single-precision, complex, vector-vector dot product, i.e.
- void utility_svsmul (float ∗a, const float ∗s, const int len, float ∗c)

    Single-precision, multiplies each element in vector 'a' with a scalar 's', i.e.
- void utility_cvsmul (float_complex ∗a, const float_complex ∗s, const int len, float_complex ∗c)

    Single-precision, complex, multiplies each element in vector 'a' with a scalar 's', i.e.
- void utility_svsdiv (float ∗a, const float ∗s, const int len, float ∗c)

    Single-precision, divides each element in vector 'a' with a scalar 's', i.e.
- void utility_svsadd (float ∗a, const float ∗s, const int len, float ∗c)

    Single-precision, adds each element in vector 'a' with a scalar 's', i.e.
- void utility_svssub (float ∗a, const float ∗s, const int len, float ∗c)

    Single-precision, subtracts each element in vector 'a' with a scalar 's', i.e.
- void utility_ssvd (const float ∗A, const int dim1, const int dim2, float ∗U, float ∗S, float ∗V, float ∗sing)

    Row-major, singular value decomposition: single precision, i.e.
- void utility_csvd (const float_complex ∗A, const int dim1, const int dim2, float_complex ∗U, float_complex ∗S, float_complex ∗V, float ∗sing)

    Row-major, singular value decomposition: single precision complex, i.e.
- void utility_sseig (const float ∗A, const int dim, int sortDecFLAG, float ∗V, float ∗D, float ∗eig)

Row-major, eigenvalue decomposition of a SYMMETRIC matrix: single precision, i.e.

- void utility_cseig (const float_complex *A, const int dim, int sortDecFLAG, float_complex *V, float_complex *D, float *eig)

  Row-major, eigenvalue decomposition of a SYMMETRIC/HERMITION matrix: single precision complex, i.e.

- void utility_ceigmp (const float_complex *A, const float_complex *B, const int dim, float_complex *VL, float↩ _complex *VR, float_complex *D)

  Row-major, finds eigenvalues of a matrix pair using the QZ method, single precision complex, i.e.

- void utility_zeigmp (const double_complex *A, const double_complex *B, const int dim, double_complex *VL, double_complex *VR, double_complex *D)

  Row-major, finds eigenvalues of a matrix pair using the QZ method, double precision complex, i.e.

- void utility_ceig (const float_complex *A, const int dim, int sortDecFLAG, float_complex *VL, float_complex *VR, float_complex *D, float *eig)

  Row-major, eigenvalue decomposition of a NON-SYMMETRIC matrix: single precision complex, i.e.

- void utility_sglslv (const float *A, const int dim, float *B, int nCol, float *X)

  Row-major, general linear solver: single precision, i.e.

- void utility_cglslv (const float_complex *A, const int dim, float_complex *B, int nCol, float_complex *X)

  Row-major, general linear solver: single precision complex, i.e.

- void utility_dglslv (const double *A, const int dim, double *B, int nCol, double *X)

  Row-major, general linear solver: double precision, i.e.

- void utility_zglslv (const double_complex *A, const int dim, double_complex *B, int nCol, double_complex *X)

  Row-major, general linear solver: double precision complex, i.e.

- void utility_sslslv (const float *A, const int dim, float *B, int nCol, float *X)

  Row-major, linear solver for SYMMETRIC positive-definate 'A': single precision, i.e.

- void utility_cslslv (const float_complex *A, const int dim, float_complex *B, int nCol, float_complex *X)

  Row-major, linear solver for HERMITIAN positive-definate 'A': single precision complex, i.e.

- void utility_spinv (const float *A, const int dim1, const int dim2, float *B)

  Row-major, general matrix pseudo-inverse (the svd way): single precision, i.e.

- void utility_cpinv (const float_complex *A, const int dim1, const int dim2, float_complex *B)

  Row-major, general matrix pseudo-inverse (the svd way): single precision complex, i.e.

- void utility_dpinv (const double *A, const int dim1, const int dim2, double *B)

  Row-major, general matrix pseudo-inverse (the svd way): double precision, i.e.

- void utility_zpinv (const double_complex *A, const int dim1, const int dim2, double_complex *B)

  Row-major, general matrix pseudo-inverse (the svd way): double precision complex, i.e.

- void utility_schol (const float *A, const int dim, float *X)

  Row-major, Cholesky factorisation of a symmetric matrix positive-definate matrix: single precision, i.e.

- void utility_cchol (const float_complex *A, const int dim, float_complex *X)

  Row-major, Cholesky factorisation of a hermitian matrix positive-definate matrix: single precision complex, i.e.

- void **utility_sinv** (float *A, const int N)
- void **utility_dinv** (double *A, const int N)
- void **utility_cinv** (float_complex *A, const int N)

6.43.1 Detailed Description

Contains wrappers for optimised linear algebra routines, utilising CBLAS and LAPACK.

6.43.1.1 Dependencies

A performance library comprising CBLAS and LAPACK routines is required by the module and, thus, also by the SAF framework as a whole. Add one of the following FLAGS to your project's preprocessor definitions list, in order to enable one of these suitable performance libraries, which must also be linked correctly to your project.

- SAF_USE_INTEL_MKL: to enable Intel's Math Kernal Library with Fortran LAPACK interface
- SAF_USE_ATLAS: to enable ATLAS BLAS routines and ATLAS's CLAPACK interface
- SAF_USE_OPENBLAS_WITH_LAPACKE: to enable OpenBLAS with LAPACKE interface

See also

More information can be found here: `https://github.com/leomccormack/Spatial_Audio_Framework`

Note

MacOSX users only: saf_utilities will employ Apple's Accelerate library by default, if none of the above FLAGS are defined.

Author

Leo McCormack

Date

11.07.2016

6.43.2 Enumeration Type Documentation

6.43.2.1 _CONJ_FLAG

`enum _CONJ_FLAG`

Enumerator

NO_CONJ	Do not take the conjugate.
CONJ	Take the conjugate.

Definition at line 69 of file saf_veclib.h.

6.43.2.2 _TRANS_FLAG

`enum _TRANS_FLAG`

Enumerator

NO_TRANSPOSE	Do not transpose.
TRANSPOSE	Transpose.
CONJ_TRANSPOSE	Conjugate transpose / Hermition.

Definition at line 63 of file saf_veclib.h.

6.43.3 Function Documentation

6.43.3.1 utility_cchol()

```
void utility_cchol (
    const float_complex * A,
    const int dim,
    float_complex * X )
```

Row-major, Cholesky factorisation of a hermitian matrix positive-definate matrix: single precision complex, i.e. `X = chol(A); where A = X.'*X`

Parameters

in	*A*	Input square symmetric positive-definate matrix; FLAT: dim x dim
in	*dim*	Number of rows/colums in 'A'
out	*X*	The solution; FLAT: dim x dim

Definition at line 1954 of file saf_veclib.c.

6.43.3.2 utility_ceig()

```
void utility_ceig (
    const float_complex * A,
    const int dim,
    int sortDecFLAG,
    float_complex * VL,
    float_complex * VR,
    float_complex * D,
    float * eig )
```

Row-major, eigenvalue decomposition of a NON-SYMMETRIC matrix: single precision complex, i.e. `[VL,VR,D] = eig(A); where A*VR = VR*D, and  A*VR = VR*diag(eig)`

Note

'D' contains the eigen values along the diagonal, while 'eig' are the eigen values as a vector

Parameters

in	*A*	Input NON-SYMMETRIC square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*sortDecFLAG*	'1' sort eigen values and vectors in decending order. '0' ascending
out	*VL*	Left Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*VR*	Right Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim
out	*eig*	Eigen values not diagonalised (set to NULL if not needed); dim x 1

Definition at line 1127 of file saf_veclib.c.

6.43.3.3 utility_ceigmp()

```
void utility_ceigmp (
        const float_complex * A,
        const float_complex * B,
        const int dim,
        float_complex * VL,
        float_complex * VR,
        float_complex * D )
```

Row-major, finds eigenvalues of a matrix pair using the QZ method, single precision complex, i.e.
`[VL,VR,D] = eig(A,B,'qz'); where A*VL = B*VL*VR`

Parameters

in	*A*	Input left square matrix; FLAT: dim x dim
in	*B*	Input right square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrices 'A' and 'B'
out	*VL*	Left Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*VR*	Right Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim

Definition at line 946 of file saf_veclib.c.

6.43.3.4 utility_cglslv()

```
void utility_cglslv (
        const float_complex * A,
        const int dim,
        float_complex * B,
        int nCol,
        float_complex * X )
```

Row-major, general linear solver: single precision complex, i.e.
`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1286 of file saf_veclib.c.

6.43.3.5 utility_cimaxv()

```
void utility_cimaxv (
    const float_complex * a,
    const int len,
    int * index )
```

Single-precision, complex, index of maximum absolute value in a vector, i.e.
`[~,ind] = max(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) index of maximum value; 1 x 1

Definition at line 146 of file saf_veclib.c.

6.43.3.6 utility_ciminv()

```
void utility_ciminv (
    const float_complex * a,
    const int len,
    int * index )
```

Single-precision, complex, index of maximum absolute value in a vector, i.e.
`[~,ind] = min(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) index of minimum value; 1 x 1

Definition at line 98 of file saf_veclib.c.

6.43.3.7 utility_cpinv()

```
void utility_cpinv (
    const float_complex * A,
    const int dim1,
    const int dim2,
    float_complex * B )
```

Row-major, general matrix pseudo-inverse (the svd way): single precision complex, i.e.
`B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1638 of file saf_veclib.c.

6.43.3.8 utility_cseig()

```
void utility_cseig (
    const float_complex * A,
    const int dim,
    int sortDecFLAG,
    float_complex * V,
    float_complex * D,
    float * eig )
```

Row-major, eigenvalue decomposition of a SYMMETRIC/HERMITION matrix: single precision complex, i.e.

```
[V,D] = eig(A); where A*V = V*D, and  A*V = V*diag(eig)
```

Note

'D' contains the eigen values along the diagonal, while 'eig' are the eigen values as a vector

Parameters

in	*A*	Input SYMMETRIC square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*sortDecFLAG*	'1' sort eigen values and vectors in decending order. '0' ascending
out	*V*	Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim
out	*eig*	Eigen values not diagonalised (set to NULL if not needed); dim x 1

Definition at line 853 of file saf_veclib.c.

6.43.3.9 utility_cslslv()

```
void utility_cslslv (
    const float_complex * A,
    const int dim,
    float_complex * B,
    int nCol,
    float_complex * X )
```

Row-major, linear solver for HERMITIAN positive-definate 'A': single precision complex, i.e.

```
opts.LT=true
X = linsolve(A,B, opts); where, AX = B
```

Parameters

in	*A*	Input square SYMMETRIC positive-definate matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1500 of file saf_veclib.c.

6.43.3.10 utility_csvd()

```
void utility_csvd (
    const float_complex * A,
    const int dim1,
    const int dim2,
    float_complex * U,
    float_complex * S,
    float_complex * V,
    float * sing )
```

Row-major, singular value decomposition: single precision complex, i.e.

```
[U,S,V] = svd(A); such that A = U*S*V' = U*diag(sing)*V'
```

Note

'S' contains the singular values along the diagonal, whereas 'sing' are the singular values as a vector. Also, V is returned untransposed! (like in Matlab)

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	First dimension of matrix 'A'
in	*dim2*	Second dimension of matrix 'A'
out	*U*	Left matrix (set to NULL if not needed); FLAT: dim1 x dim1
out	*S*	Singular values along the diagonal min(dim1, dim2), (set to NULL if not needed); FLAT: dim1 x dim2
out	*V*	Right matrix (UNTRANSPOSED!) (set to NULL if not needed); FLAT: dim2 x dim2
out	*sing*	Singular values as a vector, (set to NULL if not needed); min(dim1, dim2) x 1

Definition at line 672 of file saf_veclib.c.

6.43.3.11 utility_cvabs()

```
void utility_cvabs (
    const float_complex * a,
    const int len,
    float * c )
```

Single-precision, complex, absolute values of vector elements, i.e.

```
c = abs(a)
```

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 177 of file saf_veclib.c.

6.43.3.12 utility_cvsmul()

```
void utility_cvsmul (
    float_complex * a,
    const float_complex * s,
    const int len,
    float_complex * c )
```

Single-precision, complex, multiplies each element in vector 'a' with a scalar 's', i.e.

```
c = a.*s, OR: a = a.*s (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output);

Definition at line 489 of file saf_veclib.c.

6.43.3.13 utility_cvvadd()

```
void utility_cvvadd (
    float_complex * a,
    const float_complex * b,
    const int len,
    float_complex * c )
```

Single-precision, complex, vector-vector addition, i.e.

```
c = a+b, OR: a = a+b (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 258 of file saf_veclib.c.

6.43.3.14 utility_cvvcopy()

```
void utility_cvvcopy (
    const float_complex * a,
    const int len,
    float_complex * c )
```

Single-precision, complex, vector-vector copy, i.e.

```
c = a
```

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 208 of file saf_veclib.c.

6.43.3.15 utility_cvvdot()

```
void utility_cvvdot (
    const float_complex * a,
    const float_complex * b,
    const int len,
    CONJ_FLAG flag,
    float_complex * c )
```

Single-precision, complex, vector-vector dot product, i.e.

```
c = a*b^T, (where size(c) = [1  1])
```

Parameters

in	*a*	Input vector a; len x 1
in	*b*	Input vector b; len x 1
in	*flag*	'0' do not take the conjugate of 'b', '1', take the conjugate of 'b'.
in	*len*	Vector length
out	*c*	(&) output vector c; 1 x 1

Definition at line 441 of file saf_veclib.c.

6.43.3.16 utility_cvvmul()

```
void utility_cvvmul (
    float_complex * a,
    const float_complex * b,
    const int len,
    float_complex * c )
```

Single-precision, complex, element-wise vector-vector multiplication i.e.

```
c = a.*b, OR: a = a.*b (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 394 of file saf_veclib.c.

6.43.3.17 utility_cvvsub()

```
void utility_cvvsub (
    float_complex * a,
    const float_complex * b,
    const int len,
    float_complex * c )
```

Single-precision, complex, vector-vector subtraction, i.e.
`c = a-b, OR: a = a-b (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 323 of file saf_veclib.c.

6.43.3.18 utility_dglslv()

```
void utility_dglslv (
    const double * A,
    const int dim,
    double * B,
    int nCol,
    double * X )
```

Row-major, general linear solver: double precision, i.e.

`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1339 of file saf_veclib.c.

6.43.3.19 utility_dpinv()

```
void utility_dpinv (
    const double * A,
    const int dim1,
    const int dim2,
    double * B )
```

Row-major, general matrix pseudo-inverse (the svd way): double precision, i.e.

`B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1730 of file saf_veclib.c.

6.43.3.20 utility_schol()

```
void utility_schol (
    const float * A,
    const int dim,
    float * X )
```

Row-major, Cholesky factorisation of a symmetric matrix positive-definate matrix: single precision, i.e.
`X = chol(A); where A = X.'*X`

Parameters

in	*A*	Input square symmetric positive-definate matrix; FLAT: dim x dim
in	*dim*	Number of rows/colums in 'A'
out	*X*	The solution; FLAT: dim x dim

Definition at line 1909 of file saf_veclib.c.

6.43.3.21 utility_sglslv()

```
void utility_sglslv (
    const float * A,
    const int dim,
    float * B,
    int nCol,
    float * X )
```

Row-major, general linear solver: single precision, i.e.
`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1233 of file saf_veclib.c.

6.43.3.22 utility_simaxv()

```
void utility_simaxv (
    const float * a,
    const int len,
    int * index )
```

Single-precision, index of maximum absolute value in a vector, i.e.
`[~,ind] = max(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) index of maximum value; 1 x 1

Definition at line 136 of file saf_veclib.c.

6.43.3.23 utility_siminv()

```
void utility_siminv (
    const float * a,
    const int len,
    int * index )
```

Single-precision, index of minimum absolute value in a vector, i.e.
`[~,ind] = min(abs(a))`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*index*	(&) Index of minimum value; 1 x 1

Definition at line 71 of file saf_veclib.c.

6.43.3.24 utility_spinv()

```
void utility_spinv (
    const float * A,
    const int dim1,
    const int dim2,
    float * B )
```

Row-major, general matrix pseudo-inverse (the svd way): single precision, i.e.
`B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1555 of file saf_veclib.c.

6.43.3.25 utility_sseig()

```
void utility_sseig (
    const float * A,
```

```
const int dim,
int sortDecFLAG,
float * V,
float * D,
float * eig )
```

Row-major, eigenvalue decomposition of a SYMMETRIC matrix: single precision, i.e.

```
[V,D] = eig(A); where A*V = V*D, and  A*V = V*diag(eig)
```

Note

'D' contains the eigen values along the diagonal, while 'eig' are the eigen values as a vector

Parameters

in	*A*	Input SYMMETRIC square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*sortDecFLAG*	'1' sort eigen values and vectors in decending order. '0' ascending
out	*V*	Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim
out	*eig*	Eigen values not diagonalised (set to NULL if not needed); dim x 1

Definition at line 775 of file saf_veclib.c.

6.43.3.26 utility_sslslv() `void utility_sslslv (`

```
const float * A,
const int dim,
float * B,
int nCol,
float * X )
```

Row-major, linear solver for SYMMETRIC positive-definate 'A': single precision, i.e.

```
opts.LT=true
X = linsolve(A,B, opts); where, AX = B
```

Parameters

in	*A*	Input square SYMMETRIC positive-definate matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1450 of file saf_veclib.c.

6.43.3.27 utility_ssvd() `void utility_ssvd (`

```
const float * A,
```

```
const int dim1,
const int dim2,
float * U,
float * S,
float * V,
float * sing )
```

Row-major, singular value decomposition: single precision, i.e.
`[U,S,V] = svd(A); such that A = U*S*V.' = U*diag(sing)*V.'`

Note

'S' contains the singular values along the diagonal, whereas 'sing' are the singular values as a vector. Also, V is returned untransposed! (like in Matlab)

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	First dimension of matrix 'A'
in	*dim2*	Second dimension of matrix 'A'
out	*U*	Left matrix (set to NULL if not needed); FLAT: dim1 x dim1
out	*S*	Singular values along the diagonal min(dim1, dim2), (set to NULL if not needed); FLAT: dim1 x dim2
out	*V*	Right matrix (UNTRANSPOSED!) (set to NULL if not needed); FLAT: dim2 x dim2
out	*sing*	Singular values as a vector, (set to NULL if not needed); min(dim1, dim2) x 1

Definition at line 577 of file saf_veclib.c.

6.43.3.28 utility_svabs()

```
void utility_svabs (
const float * a,
const int len,
float * c )
```

Single-precision, absolute values of vector elements, i.e.
`c = abs(a)`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 161 of file saf_veclib.c.

6.43.3.29 utility_svsadd()

```
void utility_svsadd (
float * a,
const float * s,
```

```
const int len,
float * c )
```

Single-precision, adds each element in vector 'a' with a scalar 's', i.e.
`c = a+s, OR: a = a+s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 537 of file saf_veclib.c.

6.43.3.30 utility_svsdiv()

```
void utility_svsdiv (
float * a,
const float * s,
const int len,
float * c )
```

Single-precision, divides each element in vector 'a' with a scalar 's', i.e.
`c = a./s, OR: a = a./s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 511 of file saf_veclib.c.

6.43.3.31 utility_svsmul()

```
void utility_svsmul (
float * a,
const float * s,
const int len,
float * c )
```

Single-precision, multiplies each element in vector 'a' with a scalar 's', i.e.
`c = a.*s, OR: a = a.*s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 466 of file saf_veclib.c.

6.43.3.32 utility_svssub()

```
void utility_svssub (
    float * a,
    const float * s,
    const int len,
    float * c )
```

Single-precision, subtracts each element in vector 'a' with a scalar 's', i.e.
`c = a-s, OR: a = a-s (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*s*	(&) input scalar s; 1 x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 559 of file saf_veclib.c.

6.43.3.33 utility_svvadd()

```
void utility_svvadd (
    float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, vector-vector addition, i.e.
`c = a+b, OR: a = a+b (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 223 of file saf_veclib.c.

6.43.3.34 utility_svvcopy()

```
void utility_svvcopy (
    const float * a,
    const int len,
    float * c )
```

Single-precision, vector-vector copy, i.e.
`c = a`

Parameters

in	*a*	Input vector a; len x 1
in	*len*	Vector length
out	*c*	Output vector c; len x 1

Definition at line 198 of file saf_veclib.c.

6.43.3.35 utility_svvdot()

```
void utility_svvdot (
    const float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, vector-vector dot product, i.e.

```
c = a*b^T, (where size(c) = [1  1])
```

Parameters

in	*a*	Input vector a; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	(&) output vector c; 1 x 1

Definition at line 430 of file saf_veclib.c.

6.43.3.36 utility_svvmul()

```
void utility_svvmul (
    float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, element-wise vector-vector multiplication i.e.

```
c = a.*b, OR: a = a.*b (if c==NULL)
```

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 359 of file saf_veclib.c.

6.43.3.37 utility_svvsub()

```
void utility_svvsub (
    float * a,
    const float * b,
    const int len,
    float * c )
```

Single-precision, vector-vector subtraction, i.e.

`c = a-b, OR: a = a-b (if c==NULL)`

Parameters

in	*a*	Input vector a, and output if c==NULL; len x 1
in	*b*	Input vector b; len x 1
in	*len*	Vector length
out	*c*	Output vector c (set to NULL if you want 'a' as output); len x 1

Definition at line 288 of file saf_veclib.c.

6.43.3.38 utility_zeigmp()

```
void utility_zeigmp (
    const double_complex * A,
    const double_complex * B,
    const int dim,
    double_complex * VL,
    double_complex * VR,
    double_complex * D )
```

Row-major, finds eigenvalues of a matrix pair using the QZ method, double precision complex, i.e.

`[VL,VR,D] = eig(A,B,'qz'); where A*VL = B*VL*VR`

Parameters

in	*A*	Input left square matrix; FLAT: dim x dim
in	*B*	Input right square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrices 'A' and 'B'
out	*VL*	Left Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*VR*	Right Eigen vectors (set to NULL if not needed); FLAT: dim x dim
out	*D*	Eigen values along the diagonal (set to NULL if not needed); FLAT: dim x dim

Definition at line 1034 of file saf_veclib.c.

6.43.3.39 utility_zglslv()

```
void utility_zglslv (
    const double_complex * A,
    const int dim,
    double_complex * B,
    int nCol,
    double_complex * X )
```

Row-major, general linear solver: double precision complex, i.e.

`X = linsolve(A,B) = A\B; where, AX = B`

Parameters

in	*A*	Input square matrix; FLAT: dim x dim
in	*dim*	Dimensions for square matrix 'A'
in	*B*	Right hand side matrix; FLAT: dim x nCol
in	*nCol*	Number of columns in right hand side matrix
out	*X*	The solution; FLAT: dim x nCol

Definition at line 1392 of file saf_veclib.c.

6.43.3.40 utility_zpinv()

```
void utility_zpinv (
    const double_complex * A,
    const int dim1,
    const int dim2,
    double_complex * B )
```

Row-major, general matrix pseudo-inverse (the svd way): double precision complex, i.e. `B = pinv(A)`

Parameters

in	*A*	Input matrix; FLAT: dim1 x dim2
in	*dim1*	Number of rows in 'A' / columns in 'B'
in	*dim2*	Number of columns in 'A' / rows in 'B'
out	*B*	The solution; FLAT: dim2 x dim1

Definition at line 1812 of file saf_veclib.c.

6.44 framework/modules/saf_vbap/saf_vbap.c File Reference

Public part of the VBAP/MDAP module (saf_vbap)

```
#include "saf_vbap.h"
#include "saf_vbap_internal.h"
```

Macros

- #define **ENABLE_VBAP_DEBUGGING_CODE** 0

Functions

- void generateVBAPgainTable3D_srcs (float ∗src_dirs_deg, int S, float ∗ls_dirs_deg, int L, int omitLargeTriangles, int enableDummies, float spread, float ∗∗gtable, int ∗N_gtable, int ∗nTriangles)

  Generates a 3-D VBAP [1] gain table based on specified source and loudspeaker directions, with optional spreading [2].

- void generateVBAPgainTable3D (float *ls_dirs_deg, int L, int az_res_deg, int el_res_deg, int omitLargeTriangles, int enableDummies, float spread, float **gtable, int *N_gtable, int *nTriangles)

  Generates a 3-D VBAP gain table based on specified loudspeaker directions, with optional spreading [2].
- void compressVBAPgainTable3D (float *vbap_gtable, int nTable, int nDirs, float *vbap_gtableComp, int *vbap_gtableIdx)

  Compresses a VBAP gain table to use less memory and CPU (by removing the elements that are zero)
- void VBAPgainTable2InterpTable (float *vbap_gtable, int nTable, int nDirs)

  Renormalises a vbap gain table in-place, so it may be utilised for interpolation of data (for example, powermaps or HRTFs)
- void generateVBAPgainTable2D_srcs (float *src_dirs_deg, int S, float *ls_dirs_deg, int L, float **gtable, int *N_gtable, int *nPairs)

  Generates a 2-D VBAP gain table based on specified source and loudspeaker directions.
- void generateVBAPgainTable2D (float *ls_dirs_deg, int L, int az_res_deg, float **gtable, int *N_gtable, int *nPairs)

  Generates a 2-D VBAP gain table based on specified loudspeaker directions.
- void getPvalues (float DTT, float *freq, int nFreq, float *pValues)

  Calculates the frequency dependent pValues, which can be applied to ENERGY normalised VBAP gains, to compensate for the room effect on the perceived loudness fluctuations of sources when panning between loudspeakers.
- void findLsTriplets (float *ls_dirs_deg, int L, int omitLargeTriangles, float **out_vertices, int *numOutVertices, int **out_faces, int *numOutFaces)

  Computes the 3D convex-hull of a spherical grid of loudspeaker directions.
- void invertLsMtx3D (float *U_spkr, int *ls_groups, int N_group, float **layoutInvMtx)

  Inverts a loudspeaker matrix.
- void getSpreadSrcDirs3D (float src_azi_rad, float src_elev_rad, float spread, int num_src, int num_rings_3d, float *U_spread)

  Computes a set of points that surround the source direction with a specific degree of spread.
- void vbap3D (float *src_dirs, int src_num, int ls_num, int *ls_groups, int nFaces, float spread, float *layoutInvMtx, float **GainMtx)

  Calculates 3D VBAP gains for pre-calculated loudspeaker triangles and predefined source directions.
- void findLsPairs (float *ls_dirs_deg, int L, int **out_pairs, int *numOutPairs)

  Calculates loudspeaker pairs for a circular grid of loudspeaker directions.
- void invertLsMtx2D (float *U_spkr, int *ls_pairs, int N_pairs, float **layoutInvMtx)

  Inverts the loudspeaker matrix.
- void vbap2D (float *src_dirs, int src_num, int ls_num, int *ls_pairs, int N_pairs, float *layoutInvMtx, float **GainMtx)

  Calculates 2D VBAP gains for pre-calculated loudspeaker pairs and predefined source positions.

6.44.1 Detailed Description

Public part of the VBAP/MDAP module (saf_vbap)

VBAP functions largely derived from the MATLAB library by Archontis Politis, found in [1].

See also

[1] https://github.com/polarch/Vector-Base-Amplitude-Panning

Author

Leo McCormack

Date

02.10.2017

6.44.2 Function Documentation

6.44.2.1 compressVBAPgainTable3D()

```
void compressVBAPgainTable3D (
        float * vbap_gtable,
        int nTable,
        int nDirs,
        float * vbap_gtableComp,
        int * vbap_gtableIdx )
```

Compresses a VBAP gain table to use less memory and CPU (by removing the elements that are zero)

Handy for large grid sizes for interpolation purposes. Therefore, the gains are also re-normalised to have the A↩ MPLITUDE-preserving property. If 'vbap_gtable' is generated by generateVBAPgainTable3D, then the compressed tables should be accessed as:

```
N_azi = (int)(360.0f / aziRes + 0.5f) + 1;
aziIndex = (int)(matlab_fmodf(AZI + 180.0f, 360.0f)/az_res_deg + 0.5f);
elevIndex = (int)((ELEVATION + 90.0f) / el_res_deg + 0.5f);
idx3d = elevIndex * N_azi + aziIndex;
for (i = 0; i < 3; i++){
    gains[i] =  vbap_gtableComp[idx3d*3+i];
    idx[i]   =  vbap_gtableIdx[idx3d*3+i];
}
```

where 'gains' are then the gains for loudspeakers('idx') to pan the source to [AZI ELEV], using the nearest grid point

Note

The VBAP gains are AMPLITUDE normalised; i.e. sum(gains) = 1

Parameters

in	*vbap_gtable*	The 3D VBAP gain table; FLAT: nTable x nDirs
in	*nTable*	number of points in the gain table
in	*nDirs*	number of loudspeakers
out	*vbap_gtableComp*	The compressed 3D VBAP gain table AMPLITUDE- NORMALISED; FLAT: nTable x 3
out	*vbap_gtableIdx*	The indices for the compressed 3D VBAP gain table; FLAT: nTable x 3

Definition at line 300 of file saf_vbap.c.

6.44.2.2 findLsPairs()

```
void findLsPairs (
        float * ls_dirs_deg,
        int L,
        int ** out_pairs,
        int * numOutPairs )
```

Calculates loudspeaker pairs for a circular grid of loudspeaker directions.

Parameters

in	*ls_dirs_deg*	Loudspeaker/source directions; FLAT: L x 1

Parameters

in	*L*	Number of loudspeakers
out	*out_pairs*	(&) loudspeaker pair indices; FLAT: numOutPairs x 2
out	*numOutPairs*	(&) number of loudspeaker pairs

Definition at line 901 of file saf_vbap.c.

6.44.2.3 findLsTriplets()

```
void findLsTriplets (
        float * ls_dirs_deg,
        int L,
        int omitLargeTriangles,
        float ** out_vertices,
        int * numOutVertices,
        int ** out_faces,
        int * numOutFaces )
```

Computes the 3D convex-hull of a spherical grid of loudspeaker directions.

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES; FLAT: L x 2
in	*L*	Number of loudspeakers
in	*omitLargeTriangles*	'0' normal triangulation, '1' remove large triangles
out	*out_vertices*	(&) loudspeaker directions in cartesian coordinates; FLAT: L x 3
out	*numOutVertices*	(&) number of loudspeakers
out	*out_faces*	(&) true loudspeaker triangle indices; FLAT: numOutFaces x 3
out	*numOutFaces*	(&) number of true loudspeaker triangles

Definition at line 483 of file saf_vbap.c.

6.44.2.4 generateVBAPgainTable2D()

```
void generateVBAPgainTable2D (
        float * ls_dirs_deg,
        int L,
        int az_res_deg,
        float ** gtable,
        int * N_gtable,
        int * nPairs )
```

Generates a 2-D VBAP gain table based on specified loudspeaker directions.

This function generates the VBAP gains for a grid: -180:az_res_deg:180 azimuths, which should be accessed as:

```
aziIndex = (int)(matlab_fmodf(AZI + 180.0f, 360.0f)/az_res_deg + 0.5f);
idx2d = aziIndex;
for (ls = 0; ls < L; ls++){
    gains2D[ls] =  gtable[idx2d*L+ls];}
```

'gains2D' are then the loudspeaker gains to pan the source to [AZI 0], using the nearest grid point.

Note

The VBAP gains are ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES (elev assumed to be 0 for all); FLAT: L x 2
in	*L*	Number of loudspeakers
in	*az_res_deg*	Azimuthal resolution in DEGREES
out	*gtable*	(&) The 2D VBAP gain table ENERGY NORMALISED; FLAT: S x L
out	*N_gtable*	(&) number of points in the gain table, N_gtable=S
out	*nPairs*	(&) number of loudspeaker pairs

Definition at line 416 of file saf_vbap.c.

6.44.2.5 generateVBAPgainTable2D_srcs()

```
void generateVBAPgainTable2D_srcs (
    float * src_dirs_deg,
    int S,
    float * ls_dirs_deg,
    int L,
    float ** gtable,
    int * N_gtable,
    int * nPairs )
```

Generates a 2-D VBAP gain table based on specified source and loudspeaker directions.

Note

source and loudspeaker directions are required to be inter-leaved with zeros, i.e. [src_az1, 0; src_az2, 0; src_az3, 0;]. The VBAP gains are also ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*src_dirs_deg*	Source directions in DEGREES (elev assumed to be 0 for all); FLAT: S x 2
in	*S*	Number of Sources
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES (elev assumed to be 0 for all); FLAT: L x 2
in	*L*	Number of loudspeakers
out	*gtable*	(&) The 2D VBAP gain table ENERGY NORMALISED; FLAT: S x L
out	*N_gtable*	(&) number of points in the gain table, N_gtable=S
out	*nPairs*	(&) number of loudspeaker pairs

Definition at line 378 of file saf_vbap.c.

6.44.2.6 generateVBAPgainTable3D()

```
void generateVBAPgainTable3D (
    float * ls_dirs_deg,
    int L,
    int az_res_deg,
    int el_res_deg,
    int omitLargeTriangles,
    int enableDummies,
```

```
float spread,
float ** gtable,
int * N_gtable,
int * nTriangles )
```

Generates a 3-D VBAP gain table based on specified loudspeaker directions, with optional spreading [2].

This function generates the VBAP gains for a grid: -180:az_res_deg:180 azimuths and -90:el_res_deg:90 elevations, which should be accessed as:

```
N_azi = (int)(360.0f / aziRes + 0.5f) + 1;
aziIndex = (int)(matlab_fmodf(AZI + 180.0f, 360.0f)/az_res_deg + 0.5f);
elevIndex = (int)((ELEV + 90.0f) / el_res_deg + 0.5f);
idx3d = elevIndex * N_azi + aziIndex;
for (ls = 0; ls < L; ls++)
    gains3D[ls] =  gtable[idx3d*L+ls];}
```

where 'gains3D' are the loudspeaker gains to pan the source to [AZI ELEV], using the nearest grid point

Note

'gtable' is returned as NULL if the triangulation failed. The VBAP gains are also ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES; FLAT: L x 2
in	*L*	Number of loudspeakers
in	*az_res_deg*	Azimuthal resolution in DEGREES
in	*el_res_deg*	Elevation resolution in DEGREES
in	*omitLargeTriangles*	'0' normal triangulation, '1' remove large triangles
in	*enableDummies*	'0' disabled, '1' enabled. Dummies are placed at +/-90 elevation if required
in	*spread*	Spreading factor in DEGREES, 0: VBAP, >0: MDAP
out	*gtable*	(&) The 3D VBAP gain table ENERGY NORMALISED; FLAT: N_gtable x L
out	*N_gtable*	(&) number of points in the gain table
out	*nTriangles*	(&) number of loudspeaker triangles

See also

[1] Pulkki, V. (1997). Virtual sound source positioning using vector base amplitude panning. Journal of the audio engineering society, 45(6), 456-466.

[2] Pulkki, V. (1999). Uniform spreading of amplitude panned virtual sources. In Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. WASPAA'99 (Cat. No. 99TH8452) (pp. 187-190). IEEE.

Definition at line 160 of file saf_vbap.c.

6.44.2.7 generateVBAPgainTable3D_srcs()

```
void generateVBAPgainTable3D_srcs (
float * src_dirs_deg,
int S,
float * ls_dirs_deg,
int L,
int omitLargeTriangles,
```

```
int enableDummies,
float spread,
float ** gtable,
int * N_gtable,
int * nTriangles )
```

Generates a 3-D VBAP [1] gain table based on specified source and loudspeaker directions, with optional spreading [2].

Note

gtable is returned as NULL if the triangulation failed. The VBAP gains are also ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*src_dirs_deg*	Source directions in DEGREES; FLAT: S x 2
in	*S*	Number of Sources
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES; FLAT: L x 2
in	*L*	Number of loudspeakers
in	*omitLargeTriangles*	'0' normal triangulation, '1' remove large triangles
in	*enableDummies*	'0' disabled, '1' enabled, and dummies are placed at +/-90 elevation if required
in	*spread*	Spreading factor in DEGREES, 0: VBAP, >0: MDAP
out	*gtable*	(&) The 3D VBAP gain table ENERGY NORMALISED; FLAT: N_gtable x L
out	*N_gtable*	(&) number of points in the gain table
out	*nTriangles*	(&) number of loudspeaker triangles

See also

[1] Pulkki, V. (1997). Virtual sound source positioning using vector base amplitude panning. Journal of the audio engineering society, 45(6), 456-466.

[2] Pulkki, V. (1999). Uniform spreading of amplitude panned virtual sources. In Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. WASPAA'99 (Cat. No. 99TH8452) (pp. 187-190). IEEE.

Definition at line 41 of file saf_vbap.c.

6.44.2.8 getPvalues()

```
void getPvalues (
float DTT,
float * freq,
int nFreq,
float * pValues )
```

Calculates the frequency dependent pValues, which can be applied to ENERGY normalised VBAP gains, to compensate for the room effect on the perceived loudness fluctuations of sources when panning between loudspeakers.

This should be applied as:

```
if(pValues[band] != 2.0f){
   gains3D_sum_pvf = 0.0f;
   for (i = 0; i < nLoudspeakers; i++){
       gains3D_sum_pvf += powf(MAX(gains[i], 0.0f), pValues[band]);}
   gains3D_sum_pvf = powf(gains3D_sum_pvf, 1.0f/(pValues[band]+2.23e-13f));
```

```
    for (i = 0; i < nLoudspeakers; i++){
        gains_p[i] = gains[i] / (gains3D_sum_pvf+2.23e-13f);}
}
```

Where "gains" are the original energy normalised VBAP gains and "gains_p" have amplitude normalisation for the low frequencies, and energy normalisation at the high frequencies [1].

Parameters

in	*DTT*	0..1 '0': for normal room, '1' for anechoic room, '0.5' for listening room
in	*freq*	Frequency vector in Hz; nFreq x 1
in	*nFreq*	Number of frequencies in the frequency vector
out	*pValues*	pValues for each frequency; nFreq x 1

See also

[1] Laitinen, M., Vilkamo, J., Jussila, K., Politis, A., Pulkki, V. (2014). Gain normalisation in amplitude panning as a function of frequency and room reverberance. 55th International Conference of the AES. Helsinki, Finland.

Definition at line 463 of file saf_vbap.c.

6.44.2.9 getSpreadSrcDirs3D()

```
void getSpreadSrcDirs3D (
    float src_azi_rad,
    float src_elev_rad,
    float spread,
    int num_src,
    int num_rings_3d,
    float * U_spread )
```

Computes a set of points that surround the source direction with a specific degree of spread.

Parameters

in	*src_azi_rad*	Source azimuth, in RADIANS
in	*src_elev_rad*	Source elevation, in RADIANS
in	*spread*	Spread in DEGREES
in	*num_src*	Number of auxiliary sources to use for spreading
in	*num_rings_3d*	Number of concentric rings of num_src each to generate inside the spreading surface
out	*U_spread*	Spread directions Cartesian coordinates; FLAT: (num_src*num_rings_3d+1) x 3

Definition at line 710 of file saf_vbap.c.

6.44.2.10 invertLsMtx2D()

```
void invertLsMtx2D (
    float * U_spkr,
    int * ls_pairs,
    int N_pairs,
    float ** layoutInvMtx )
```

Inverts the loudspeaker matrix.

Parameters

in	*U_spkr*	Loudspeaker directions in cartesian (xy) coordinates; FLAT: L x 2

Parameters

in	*ls_pairs*	Loudspeaker pair indices; FLAT: N_pairs x 3
in	*N_pairs*	Number of loudspeaker pairs
out	*layoutInvMtx*	(&) inverted 2x2 loudspeaker matrix flattened; FLAT: N_group x 4

Definition at line 933 of file saf_vbap.c.

6.44.2.11 invertLsMtx3D()

```
void invertLsMtx3D (
    float * U_spkr,
    int * ls_groups,
    int N_group,
    float ** layoutInvMtx )
```

Inverts a loudspeaker matrix.

Parameters

in	*U_spkr*	Loudspeaker directions as cartesian coordinates (unit length); FLAT: L x 3
in	*ls_groups*	True loudspeaker triangle indices; FLAT: N_group x 3
in	*N_group*	Number of true loudspeaker triangles
out	*layoutInvMtx*	(&) inverted 3x3 loudspeaker matrices per group; FLAT: N_group x 9

Definition at line 670 of file saf_vbap.c.

6.44.2.12 vbap2D()

```
void vbap2D (
    float * src_dirs,
    int src_num,
    int ls_num,
    int * ls_pairs,
    int N_pairs,
    float * layoutInvMtx,
    float ** GainMtx )
```

Calculates 2D VBAP gains for pre-calculated loudspeaker pairs and predefined source positions.

Parameters

in	*src_dirs*	Source directions in DEGREES; FLAT: src_num x 1
in	*src_num*	Number of sources
in	*ls_num*	Number of loudspeakers
in	*ls_pairs*	Loudspeaker pair indices; FLAT: N_pairs x 2
in	*N_pairs*	Number of loudspeaker pairs
in	*layoutInvMtx*	Inverted 2x2 loudspeaker matrix flattened; FLAT: N_pairs x 4
out	*GainMtx*	(&) Loudspeaker VBAP gain table; FLAT: src_num x ls_num

Definition at line 974 of file saf_vbap.c.

6.44.2.13 vbap3D()

```
void vbap3D (
    float * src_dirs,
    int src_num,
    int ls_num,
    int * ls_groups,
    int nFaces,
    float spread,
    float * layoutInvMtx,
    float ** GainMtx )
```

Calculates 3D VBAP gains for pre-calculated loudspeaker triangles and predefined source directions.

Parameters

in	*src_dirs*	Source directions; FLAT: src_num x 2
in	*src_num*	Number of sources
in	*ls_num*	Number of loudspeakers
in	*ls_groups*	True loudspeaker triangle indices; FLAT: nFaces x 3
in	*nFaces*	Number of true loudspeaker triangles
in	*spread*	Spreading in degrees, 0: VBAP, $>$0: MDAP
in	*layoutInvMtx*	Inverted 3x3 loudspeaker matrix flattened; FLAT: nFaces x 9
out	*GainMtx*	(&) Loudspeaker VBAP gain table; FLAT: src_num x ls_num

Definition at line 789 of file saf_vbap.c.

6.44.2.14 VBAPgainTable2InterpTable()

```
void VBAPgainTable2InterpTable (
    float * vbap_gtable,
    int nTable,
    int nDirs )
```

Renormalises a vbap gain table in-place, so it may be utilised for interpolation of data (for example, powermaps or HRTFs)

Note

The VBAP gains are AMPLITUDE normalised; i.e. sum(gains) = 1.

Parameters

in,out	*vbap_gtable*	The 3D VBAP gain table; FLAT: nTable x nDirs
in	*nTable*	Number of points in the gain table
in	*nDirs*	Number of loudspeakers

Definition at line 357 of file saf_vbap.c.

6.45 framework/modules/saf_vbap/saf_vbap.h File Reference

Public part of the VBAP/MDAP module (saf_vbap)

Functions

- void generateVBAPgainTable3D_srcs (float ∗src_dirs_deg, int S, float ∗ls_dirs_deg, int L, int omitLarge↩ Triangles, int enableDummies, float spread, float ∗∗gtable, int ∗N_gtable, int ∗nTriangles)

  Generates a 3-D VBAP [1] gain table based on specified source and loudspeaker directions, with optional spreading [2].
- void generateVBAPgainTable3D (float ∗ls_dirs_deg, int L, int az_res_deg, int el_res_deg, int omitLarge↩ Triangles, int enableDummies, float spread, float ∗∗gtable, int ∗N_gtable, int ∗nTriangles)

  Generates a 3-D VBAP gain table based on specified loudspeaker directions, with optional spreading [2].
- void compressVBAPgainTable3D (float ∗vbap_gtable, int nTable, int nDirs, float ∗vbap_gtableComp, int ∗vbap_gtableIdx)

  Compresses a VBAP gain table to use less memory and CPU (by removing the elements that are zero)
- void VBAPgainTable2InterpTable (float ∗vbap_gtable, int nTable, int nDirs)

  Renormalises a vbap gain table in-place, so it may be utilised for interpolation of data (for example, powermaps or HRTFs)
- void generateVBAPgainTable2D_srcs (float ∗src_dirs_deg, int S, float ∗ls_dirs_deg, int L, float ∗∗gtable, int ∗N_gtable, int ∗nPairs)

  Generates a 2-D VBAP gain table based on specified source and loudspeaker directions.
- void generateVBAPgainTable2D (float ∗ls_dirs_deg, int L, int az_res_deg, float ∗∗gtable, int ∗N_gtable, int ∗nPairs)

  Generates a 2-D VBAP gain table based on specified loudspeaker directions.
- void getPvalues (float DTT, float ∗freq, int nFreq, float ∗pValues)

  Calculates the frequency dependent pValues, which can be applied to ENERGY normalised VBAP gains, to compensate for the room effect on the perceived loudness fluctuations of sources when panning between loudspeakers.
- void findLsTriplets (float ∗ls_dirs_deg, int L, int omitLargeTriangles, float ∗∗out_vertices, int ∗numOutVertices, int ∗∗out_faces, int ∗numOutFaces)

  Computes the 3D convex-hull of a spherical grid of loudspeaker directions.
- void invertLsMtx3D (float ∗U_spkr, int ∗ls_groups, int N_group, float ∗∗layoutInvMtx)

  Inverts a loudspeaker matrix.
- void getSpreadSrcDirs3D (float src_azi_rad, float src_elev_rad, float spread, int num_src, int num_rings_3d, float ∗U_spread)

  Computes a set of points that surround the source direction with a specific degree of spread.
- void vbap3D (float ∗src_dirs, int src_num, int ls_num, int ∗ls_groups, int nFaces, float spread, float ∗layout↩ InvMtx, float ∗∗GainMtx)

  Calculates 3D VBAP gains for pre-calculated loudspeaker triangles and predefined source directions.
- void findLsPairs (float ∗ls_dirs_deg, int L, int ∗∗out_pairs, int ∗numOutPairs)

  Calculates loudspeaker pairs for a circular grid of loudspeaker directions.
- void invertLsMtx2D (float ∗U_spkr, int ∗ls_pairs, int N_pairs, float ∗∗layoutInvMtx)

  Inverts the loudspeaker matrix.
- void vbap2D (float ∗src_dirs, int src_num, int ls_num, int ∗ls_pairs, int N_pairs, float ∗layoutInvMtx, float ∗∗GainMtx)

  Calculates 2D VBAP gains for pre-calculated loudspeaker pairs and predefined source positions.

6.45.1 Detailed Description

Public part of the VBAP/MDAP module (saf_vbap)

VBAP functions largely derived from the MATLAB library by Archontis Politis, found in [1].

See also

[1] https://github.com/polarch/Vector-Base-Amplitude-Panning

Author

Leo McCormack

Date

02.10.2017

6.45.2 Function Documentation

6.45.2.1 compressVBAPgainTable3D()

```
void compressVBAPgainTable3D (
        float * vbap_gtable,
        int nTable,
        int nDirs,
        float * vbap_gtableComp,
        int * vbap_gtableIdx )
```

Compresses a VBAP gain table to use less memory and CPU (by removing the elements that are zero)

Handy for large grid sizes for interpolation purposes. Therefore, the gains are also re-normalised to have the AMPLITUDE-preserving property. If 'vbap_gtable' is generated by generateVBAPgainTable3D, then the compressed tables should be accessed as:

```
N_azi = (int)(360.0f / aziRes + 0.5f) + 1;
aziIndex = (int)(matlab_fmodf(AZI + 180.0f, 360.0f)/az_res_deg + 0.5f);
elevIndex = (int)((ELEVATION + 90.0f) / el_res_deg + 0.5f);
idx3d = elevIndex * N_azi + aziIndex;
for (i = 0; i < 3; i++){
    gains[i] =  vbap_gtableComp[idx3d*3+i];
    idx[i]   =  vbap_gtableIdx[idx3d*3+i];
}
```

where 'gains' are then the gains for loudspeakers('idx') to pan the source to [AZI ELEV], using the nearest grid point

Note

The VBAP gains are AMPLITUDE normalised; i.e. sum(gains) = 1

Parameters

in	*vbap_gtable*	The 3D VBAP gain table; FLAT: nTable x nDirs
in	*nTable*	number of points in the gain table
in	*nDirs*	number of loudspeakers
out	*vbap_gtableComp*	The compressed 3D VBAP gain table AMPLITUDE- NORMALISED; FLAT: nTable x 3
out	*vbap_gtableIdx*	The indices for the compressed 3D VBAP gain table; FLAT: nTable x 3

Definition at line 300 of file saf_vbap.c.

6.45.2.2 findLsPairs()

```
void findLsPairs (
    float * ls_dirs_deg,
    int L,
    int ** out_pairs,
    int * numOutPairs )
```

Calculates loudspeaker pairs for a circular grid of loudspeaker directions.

Parameters

in	*ls_dirs_deg*	Loudspeaker/source directions; FLAT: L x 1
in	*L*	Number of loudspeakers
out	*out_pairs*	(&) loudspeaker pair indices; FLAT: numOutPairs x 2
out	*numOutPairs*	(&) number of loudspeaker pairs

Definition at line 901 of file saf_vbap.c.

6.45.2.3 findLsTriplets()

```
void findLsTriplets (
    float * ls_dirs_deg,
    int L,
    int omitLargeTriangles,
    float ** out_vertices,
    int * numOutVertices,
    int ** out_faces,
    int * numOutFaces )
```

Computes the 3D convex-hull of a spherical grid of loudspeaker directions.

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES; FLAT: L x 2
in	*L*	Number of loudspeakers
in	*omitLargeTriangles*	'0' normal triangulation, '1' remove large triangles
out	*out_vertices*	(&) loudspeaker directions in cartesian coordinates; FLAT: L x 3
out	*numOutVertices*	(&) number of loudspeakers
out	*out_faces*	(&) true loudspeaker triangle indices; FLAT: numOutFaces x 3
out	*numOutFaces*	(&) number of true loudspeaker triangles

Definition at line 483 of file saf_vbap.c.

6.45.2.4 generateVBAPgainTable2D()

```
void generateVBAPgainTable2D (
    float * ls_dirs_deg,
```

```
int L,
int az_res_deg,
float ** gtable,
int * N_gtable,
int * nPairs )
```

Generates a 2-D VBAP gain table based on specified loudspeaker directions.

This function generates the VBAP gains for a grid: -180:az_res_deg:180 azimuths, which should be accessed as:

```
aziIndex = (int)(matlab_fmodf(AZI + 180.0f, 360.0f)/az_res_deg + 0.5f);
idx2d = aziIndex;
for (ls = 0; ls < L; ls++){
    gains2D[ls] =  gtable[idx2d*L+ls];}
```

'gains2D' are then the loudspeaker gains to pan the source to [AZI 0], using the nearest grid point.

Note

The VBAP gains are ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES (elev assumed to be 0 for all); FLAT: L x 2
in	*L*	Number of loudspeakers
in	*az_res_deg*	Azimuthal resolution in DEGREES
out	*gtable*	(&) The 2D VBAP gain table ENERGY NORMALISED; FLAT: S x L
out	*N_gtable*	(&) number of points in the gain table, N_gtable=S
out	*nPairs*	(&) number of loudspeaker pairs

Definition at line 416 of file saf_vbap.c.

6.45.2.5 generateVBAPgainTable2D_srcs()

```
void generateVBAPgainTable2D_srcs (
float * src_dirs_deg,
int S,
float * ls_dirs_deg,
int L,
float ** gtable,
int * N_gtable,
int * nPairs )
```

Generates a 2-D VBAP gain table based on specified source and loudspeaker directions.

Note

source and loudspeaker directions are required to be inter-leaved with zeros, i.e. [src_az1, 0; src_az2, 0; src_az3, 0;]. The VBAP gains are also ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*src_dirs_deg*	Source directions in DEGREES (elev assumed to be 0 for all); FLAT: S x 2
in	*S*	Number of Sources
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES (elev assumed to be 0 for all); FLAT: L x 2

Parameters

in	*L*	Number of loudspeakers
out	*gtable*	(&) The 2D VBAP gain table ENERGY NORMALISED; FLAT: S x L
out	*N_gtable*	(&) number of points in the gain table, N_gtable=S
out	*nPairs*	(&) number of loudspeaker pairs

Definition at line 378 of file saf_vbap.c.

6.45.2.6 generateVBAPgainTable3D()

```
void generateVBAPgainTable3D (
    float * ls_dirs_deg,
    int L,
    int az_res_deg,
    int el_res_deg,
    int omitLargeTriangles,
    int enableDummies,
    float spread,
    float ** gtable,
    int * N_gtable,
    int * nTriangles )
```

Generates a 3-D VBAP gain table based on specified loudspeaker directions, with optional spreading [2].

This function generates the VBAP gains for a grid: -180:az_res_deg:180 azimuths and -90:el_res_deg:90 elevations, which should be accessed as:

```
N_azi = (int)(360.0f / aziRes + 0.5f) + 1;
aziIndex = (int)(matlab_fmodf(AZI + 180.0f, 360.0f)/az_res_deg + 0.5f);
elevIndex = (int)((ELEV + 90.0f) / el_res_deg + 0.5f);
idx3d = elevIndex * N_azi + aziIndex;
for (ls = 0; ls < L; ls++)
    gains3D[ls] =  gtable[idx3d*L+ls];}
```

where 'gains3D' are the loudspeaker gains to pan the source to [AZI ELEV], using the nearest grid point

Note

'gtable' is returned as NULL if the triangulation failed. The VBAP gains are also ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*ls_dirs_deg*	Loudspeaker directions in DEGREES; FLAT: L x 2
in	*L*	Number of loudspeakers
in	*az_res_deg*	Azimuthal resolution in DEGREES
in	*el_res_deg*	Elevation resolution in DEGREES
in	*omitLargeTriangles*	'0' normal triangulation, '1' remove large triangles
in	*enableDummies*	'0' disabled, '1' enabled. Dummies are placed at +/-90 elevation if required
in	*spread*	Spreading factor in DEGREES, 0: VBAP, >0: MDAP
out	*gtable*	(&) The 3D VBAP gain table ENERGY NORMALISED; FLAT: N_gtable x L
out	*N_gtable*	(&) number of points in the gain table
out	*nTriangles*	(&) number of loudspeaker triangles

See also

[1] Pulkki, V. (1997). Virtual sound source positioning using vector base amplitude panning. Journal of the audio engineering society, 45(6), 456-466.

[2] Pulkki, V. (1999). Uniform spreading of amplitude panned virtual sources. In Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. WASPAA'99 (Cat. No. 99TH8452) (pp. 187-190). IEEE.

Definition at line 160 of file saf_vbap.c.

6.45.2.7 generateVBAPgainTable3D_srcs()

```
void generateVBAPgainTable3D_srcs (
    float * src_dirs_deg,
    int S,
    float * ls_dirs_deg,
    int L,
    int omitLargeTriangles,
    int enableDummies,
    float spread,
    float ** gtable,
    int * N_gtable,
    int * nTriangles )
```

Generates a 3-D VBAP [1] gain table based on specified source and loudspeaker directions, with optional spreading [2].

Note

gtable is returned as NULL if the triangulation failed. The VBAP gains are also ENERGY normalised; i.e. sum(gains^2) = 1

Parameters

in	*src_dirs_deg*	Source directions in DEGREES; FLAT: S x 2
in	*S*	Number of Sources
in	*ls_dirs_deg*	Loudspeaker directions in DEGREES; FLAT: L x 2
in	*L*	Number of loudspeakers
in	*omitLargeTriangles*	'0' normal triangulation, '1' remove large triangles
in	*enableDummies*	'0' disabled, '1' enabled, and dummies are placed at +/-90 elevation if required
in	*spread*	Spreading factor in DEGREES, 0: VBAP, >0: MDAP
out	*gtable*	(&) The 3D VBAP gain table ENERGY NORMALISED; FLAT: N_gtable x L
out	*N_gtable*	(&) number of points in the gain table
out	*nTriangles*	(&) number of loudspeaker triangles

See also

[1] Pulkki, V. (1997). Virtual sound source positioning using vector base amplitude panning. Journal of the audio engineering society, 45(6), 456-466.

[2] Pulkki, V. (1999). Uniform spreading of amplitude panned virtual sources. In Proceedings of the 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. WASPAA'99 (Cat. No. 99TH8452) (pp. 187-190). IEEE.

Definition at line 41 of file saf_vbap.c.

6.45.2.8 getPvalues()

```
void getPvalues (
    float DTT,
    float * freq,
    int nFreq,
    float * pValues )
```

Calculates the frequency dependent pValues, which can be applied to ENERGY normalised VBAP gains, to compensate for the room effect on the perceived loudness fluctuations of sources when panning between loudspeakers.

This should be applied as:

```
if(pValues[band] != 2.0f){
   gains3D_sum_pvf = 0.0f;
   for (i = 0; i < nLoudspeakers; i++){
       gains3D_sum_pvf += powf(MAX(gains[i], 0.0f), pValues[band]);}
   gains3D_sum_pvf = powf(gains3D_sum_pvf, 1.0f/(pValues[band]+2.23e-13f));
   for (i = 0; i < nLoudspeakers; i++){
       gains_p[i] = gains[i] / (gains3D_sum_pvf+2.23e-13f);}
}
```

Where "gains" are the original energy normalised VBAP gains and "gains_p" have amplitude normalisation for the low frequencies, and energy normalisation at the high frequencies [1].

Parameters

in	*DTT*	0..1 '0': for normal room, '1' for anechoic room, '0.5' for listening room
in	*freq*	Frequency vector in Hz; nFreq x 1
in	*nFreq*	Number of frequencies in the frequency vector
out	*pValues*	pValues for each frequency; nFreq x 1

See also

[1] Laitinen, M., Vilkamo, J., Jussila, K., Politis, A., Pulkki, V. (2014). Gain normalisation in amplitude panning as a function of frequency and room reverberance. 55th International Conference of the AES. Helsinki, Finland.

Definition at line 463 of file saf_vbap.c.

6.45.2.9 getSpreadSrcDirs3D()

```
void getSpreadSrcDirs3D (
    float src_azi_rad,
    float src_elev_rad,
    float spread,
    int num_src,
    int num_rings_3d,
    float * U_spread )
```

Computes a set of points that surround the source direction with a specific degree of spread.

Parameters

in	*src_azi_rad*	Source azimuth, in RADIANS

Parameters

in	*src_elev_rad*	Source elevation, in RADIANS
in	*spread*	Spread in DEGREES
in	*num_src*	Number of auxiliary sources to use for spreading
in	*num_rings_3d*	Number of concentric rings of num_src each to generate inside the spreading surface
out	*U_spread*	Spread directions Cartesian coordinates; FLAT: (num_src∗num_rings_3d+1) x 3

Definition at line 710 of file saf_vbap.c.

6.45.2.10 invertLsMtx2D()

```
void invertLsMtx2D (
    float * U_spkr,
    int * ls_pairs,
    int N_pairs,
    float ** layoutInvMtx )
```

Inverts the loudspeaker matrix.

Parameters

in	*U_spkr*	Loudspeaker directions in cartesian (xy) coordinates; FLAT: L x 2
in	*ls_pairs*	Loudspeaker pair indices; FLAT: N_pairs x 3
in	*N_pairs*	Number of loudspeaker pairs
out	*layoutInvMtx*	(&) inverted 2x2 loudspeaker matrix flattened; FLAT: N_group x 4

Definition at line 933 of file saf_vbap.c.

6.45.2.11 invertLsMtx3D()

```
void invertLsMtx3D (
    float * U_spkr,
    int * ls_groups,
    int N_group,
    float ** layoutInvMtx )
```

Inverts a loudspeaker matrix.

Parameters

in	*U_spkr*	Loudspeaker directions as cartesian coordinates (unit length); FLAT: L x 3
in	*ls_groups*	True loudspeaker triangle indices; FLAT: N_group x 3
in	*N_group*	Number of true loudspeaker triangles
out	*layoutInvMtx*	(&) inverted 3x3 loudspeaker matrices per group; FLAT: N_group x 9

Definition at line 670 of file saf_vbap.c.

6.45.2.12 vbap2D()

```
void vbap2D (
    float * src_dirs,
    int src_num,
    int ls_num,
    int * ls_pairs,
    int N_pairs,
    float * layoutInvMtx,
    float ** GainMtx )
```

Calculates 2D VBAP gains for pre-calculated loudspeaker pairs and predefined source positions.

Parameters

in	*src_dirs*	Source directions in DEGREES; FLAT: src_num x 1
in	*src_num*	Number of sources
in	*ls_num*	Number of loudspeakers
in	*ls_pairs*	Loudspeaker pair indices; FLAT: N_pairs x 2
in	*N_pairs*	Number of loudspeaker pairs
in	*layoutInvMtx*	Inverted 2x2 loudspeaker matrix flattened; FLAT: N_pairs x 4
out	*GainMtx*	(&) Loudspeaker VBAP gain table; FLAT: src_num x ls_num

Definition at line 974 of file saf_vbap.c.

6.45.2.13 vbap3D()

```
void vbap3D (
    float * src_dirs,
    int src_num,
    int ls_num,
    int * ls_groups,
    int nFaces,
    float spread,
    float * layoutInvMtx,
    float ** GainMtx )
```

Calculates 3D VBAP gains for pre-calculated loudspeaker triangles and predefined source directions.

Parameters

in	*src_dirs*	Source directions; FLAT: src_num x 2
in	*src_num*	Number of sources
in	*ls_num*	Number of loudspeakers
in	*ls_groups*	True loudspeaker triangle indices; FLAT: nFaces x 3
in	*nFaces*	Number of true loudspeaker triangles
in	*spread*	Spreading in degrees, 0: VBAP, $>$0: MDAP
in	*layoutInvMtx*	Inverted 3x3 loudspeaker matrix flattened; FLAT: nFaces x 9
out	*GainMtx*	(&) Loudspeaker VBAP gain table; FLAT: src_num x ls_num

Definition at line 789 of file saf_vbap.c.

6.45.2.14 VBAPgainTable2InterpTable()

```
void VBAPgainTable2InterpTable (
    float * vbap_gtable,
    int nTable,
    int nDirs )
```

Renormalises a vbap gain table in-place, so it may be utilised for interpolation of data (for example, powermaps or HRTFs)

Note

The VBAP gains are AMPLITUDE normalised; i.e. sum(gains) = 1.

Parameters

in,out	*vbap_gtable*	The 3D VBAP gain table; FLAT: nTable x nDirs
in	*nTable*	Number of points in the gain table
in	*nDirs*	Number of loudspeakers

Definition at line 357 of file saf_vbap.c.

6.46 framework/modules/saf_vbap/saf_vbap_internal.c File Reference

Internal part of the VBAP/MDAP module (saf_vbap)

```
#include "saf_vbap.h"
#include "saf_vbap_internal.h"
```

Functions

- void ccross (float a[3], float b[3], float c[3])

    Cross product between two 3-element floating point vectors.

6.46.1 Detailed Description

Internal part of the VBAP/MDAP module (saf_vbap)

VBAP functions largely derived from the MATLAB library by Archontis Politis, found in [1].

See also

[1] https://github.com/polarch/Vector-Base-Amplitude-Panning

Author

Leo McCormack

Date

02.10.2017

6.47 framework/modules/saf_vbap/saf_vbap_internal.h File Reference

Internal part of the VBAP/MDAP module (saf_vbap)

```
#include <stdio.h>
#include <math.h>
#include <complex.h>
#include <string.h>
#include "saf_vbap.h"
#include "../saf_utilities/saf_utilities.h"
```

Macros

- #define ADD_DUMMY_LIMIT (60.0f)

  In degrees, if no ls_dirs have elevation +/- this value, dummies are placed at +/- 90 elevation.
- #define APERTURE_LIMIT_DEG (180.0f)

  if omitLargeTriangles==1, triangles with an aperture larger than this are discarded

Functions

- void ccross (float a[3], float b[3], float c[3])

  Cross product between two 3-element floating point vectors.

6.47.1 Detailed Description

Internal part of the VBAP/MDAP module (saf_vbap)

VBAP functions largely derived from the MATLAB library by Archontis Politis, found in [1].

See also

[1] https://github.com/polarch/Vector-Base-Amplitude-Panning

Author

Leo McCormack

Date

02.10.2017

6.47.2 Macro Definition Documentation

6.47.2.1 ADD_DUMMY_LIMIT `#define ADD_DUMMY_LIMIT ( 60.0f )`

In degrees, if no ls_dirs have elevation +/- this value, dummies are placed at +/- 90 elevation.

Definition at line 46 of file saf_vbap_internal.h.

6.48 framework/resources/afSTFT/afSTFT_protoFilter.h File Reference

Prototype filter used by afSTFTlib.

Variables

- const float **protoFilter1024** [10240]
- const float **protoFilter1024LD** [10240]

6.48.1 Detailed Description

Prototype filter used by afSTFTlib.

6.49 framework/resources/afSTFT/afSTFTlib.c File Reference

Slightly modified version of afSTFTlib.

```
#include "afSTFTlib.h"
#include "afSTFT_protoFilter.h"
#include "../../modules/saf_utilities/saf_utilities.h"
```

Data Structures

- struct afSTFT
  Main data structure for afSTFTlib.
- struct afHybrid
  Data structure for the hybrid filtering employed by afSTFTlib.

Macros

- #define **COEFF1** 0.031273141818515176604f
- #define **COEFF2** 0.28127313041521179171f
- #define **COEFF3** 0.5f

Functions

- void **afHybridInit** (void **handle, int hopSize, int inChannels, int outChannels)
- void **afHybridForward** (void *handle, complexVector *FD)
- void **afHybridInverse** (void *handle, complexVector *FD)
- void **afHybridFree** (void *handle)
- void afSTFTinit (void **handle, int hopSize, int inChannels, int outChannels, int LDmode, int hybridMode)

  Initialises an instance of afSTFTlib [1].
- void afSTFTchannelChange (void *handle, int new_inChannels, int new_outChannels)

  Re-allocates memory to support a change in the number of input/output channels.
- void afSTFTclearBuffers (void *handle)

  Flushes time-domain buffers with zeros.
- void afSTFTforward (void *handle, float **inTD, complexVector *outFD)

  Applies the forward afSTFT transform.
- void afSTFTinverse (void *handle, complexVector *inFD, float **outTD)

  Applies the backward afSTFT transform.
- void afSTFTfree (void *handle)

  Destroys an instance of afSTFTlib.

Variables

- const double __afCenterFreq48e3 [133]

  afSTFT centre frequencies for 128 hop size and hybrid-mode enabled (48kHz)
- const double __afCenterFreq44100 [133]

  afSTFT centre frequencies for 128 hop size and hybrid-mode enabled (44.1kHz)

6.49.1 Detailed Description

Slightly modified version of afSTFTlib.

The original afSTFT code, written by Juha Vilkamo, can be found here: `https://github.com/jvilkamo/afSTFT` This version is slightly modified. It adds a function to change the number of channels on the fly and includes vectors for the hybrid mode centre frequencies @44.1kHz/48kHz with 128 hop size for convenience. It also supports the use of SAF utilities (for the vectorisation and FFT).

6.49.2 Function Documentation

6.49.2.1 afSTFTchannelChange()

```
void afSTFTchannelChange (
    void * handle,
    int new_inChannels,
    int new_outChannels )
```

Re-allocates memory to support a change in the number of input/output channels.

Note

Not thread safe. So do not call in the middle of a real-time loop.

Parameters

in	*handle*	afSTFTlib handle
in	*new_inChannels*	New number of input channels
in	*new_outChannels*	New number of output channels

Definition at line 209 of file afSTFTlib.c.

6.49.2.2 afSTFTclearBuffers()

```
void afSTFTclearBuffers (
    void * handle )
```

Flushes time-domain buffers with zeros.

Parameters

in	*handle*	afSTFTlib handle

Definition at line 259 of file afSTFTlib.c.

6.49.2.3 afSTFTforward()

```
void afSTFTforward (
    void * handle,
    float ** inTD,
    complexVector * outFD )
```

Applies the forward afSTFT transform.

Parameters

in	*handle*	afSTFTlib handle
in	*inTD*	input time-domain signals; inChannels x hopSize
in	*outFD*	input time-frequency domain signals; inChannels x nBands

Definition at line 279 of file afSTFTlib.c.

6.49.2.4 afSTFTfree()

```
void afSTFTfree (
    void * handle )
```

Destroys an instance of afSTFTlib.

Parameters

in	*handle*	(&) afSTFTlib handle

Definition at line 493 of file afSTFTlib.c.

6.49.2.5 afSTFTinit()

```
void afSTFTinit (
    void ** handle,
    int hopSize,
    int inChannels,
    int outChannels,
    int LDmode,
    int hybridMode )
```

Initialises an instance of afSTFTlib [1].

Parameters

in	*handle*	(&) afSTFTlib handle
in	*hopSize*	Hop size, in samples
in	*inChannels*	Number of input channels
in	*outChannels*	Number of output channels
in	*LDmode*	'0' disable low-delay mode, '1' enable
in	*hybridMode*	'0' disable hybrid-mode, '1' enable

See also

[1] Vilkamo, J., & Backstrom, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time-Frequency Domain Spatial Audio. John Wiley & Sons.

Definition at line 115 of file afSTFTlib.c.

6.49.2.6 afSTFTinverse()

```
void afSTFTinverse (
    void * handle,
    complexVector * inFD,
    float ** outTD )
```

Applies the backward afSTFT transform.

Parameters

in	*handle*	afSTFTlib handle
in	*inFD*	output time-domain signals; outChannels x hopSize
in	*outTD*	output time-frequency domain signals; outChannels x nBands

Definition at line 375 of file afSTFTlib.c.

6.49.3 Variable Documentation

6.49.3.1 __afCenterFreq44100 `const double __afCenterFreq44100[133]`

Initial value:

```
=
{    0.000000000, 129.216965656, 215.314095512, 301.482605287, 387.579738729, 473.748225285, 559.845379541,
      646.013853751, 732.111030944, 861.328154418, 1033.593765929, 1205.859407569, 1378.125069596,
      1550.390654200, 1722.656272269, 1894.921852124, 2067.187549340, 2239.453165674, 2411.718752127,
      2583.984393174, 2756.250038304, 2928.515610236, 3100.781245532, 3273.046869646, 3445.312517049,
      3617.578144885, 3789.843759058, 3962.109385592, 4134.375009576, 4306.640638272, 4478.906262484,
      4651.171887467, 4823.437506959, 4995.703134452, 5167.968753839, 5340.234378143, 5512.500004739,
      5684.765628127, 5857.031253205, 6029.296881607, 6201.562505487, 6373.828132809, 6546.093756373,
      6718.359382855, 6890.625004623, 7062.890629479, 7235.156254481, 7407.421881970, 7579.687505713,
      7751.953124821, 7924.218750103, 8096.484373148, 8268.750008140, 8441.015629043, 8613.281251405,
      8785.546881031, 8957.812505821, 9130.078124593, 9302.343752690, 9474.609377190, 9646.875004048,
      9819.140627591, 9991.406251289, 10163.671877038, 10335.937501008, 10508.203126187, 10680.468750748,
      10852.734375129, 11025.000000000, 11197.265624618, 11369.531249516, 11541.796874351, 11714.062498897,
      11886.328122716, 12058.593748662, 12230.859372797, 12403.124996237, 12575.390622207, 12747.656246830,
      12919.921875455, 13092.187494451, 13264.453118973, 13436.718748035, 13608.984370830, 13781.249991857,
      13953.515626614, 14125.781249475, 14298.046875918, 14470.312494312, 14642.578118001, 14814.843746034,
      14987.109370627, 15159.374994823, 15331.640617057, 15503.906242865, 15676.171867063, 15848.437494762,
      16020.703118027, 16192.968746288, 16365.234371274, 16537.499995256, 16709.765622164, 16882.031246440,
      17054.296865897, 17226.562492526, 17398.828113024, 17571.093736919, 17743.359361459, 17915.624990597,
      18087.890614243, 18260.156240676, 18432.421855285, 18604.687483097, 18776.953130401, 18949.218754459,
      19121.484389530, 19293.749961692, 19466.015606863, 19638.281247918, 19810.546834386, 19982.812450661,
      20155.078147972, 20327.343727455, 20499.609346121, 20671.874930324, 20844.140592387, 21016.406233899,
      21188.671845644, 21360.937478510, 21533.203108994, 21705.468678356, 21877.734276834, 22050.000000000
        }
```

afSTFT centre frequencies for 128 hop size and hybrid-mode enabled (44.1kHz)

Definition at line 64 of file afSTFTlib.c.

6.49.3.2 __afCenterFreq48e3 `const double __afCenterFreq48e3[133]`

Initial value:

```
=
{    0.000000000, 140.644316361, 234.355478108, 328.144332285, 421.855497937, 515.644326841, 609.355515147,
      703.144330614, 796.855543885, 937.500032020, 1125.000017338, 1312.500035449, 1500.000075751,
      1687.500031782, 1875.000024239, 2062.499975101, 2250.000053703, 2437.500044271, 2625.000002315,
      2812.500019782, 3000.000041692, 3187.499983930, 3374.999995137, 3562.499994173, 3750.000018557,
      3937.500021643, 4125.000009859, 4312.500011528, 4500.000010423, 4687.500014446, 4875.000013588,
      5062.500013570, 5250.000007575, 5437.500010288, 5625.000004178, 5812.500003421, 6000.000005158,
      6187.500003404, 6375.000003488, 6562.500007191, 6750.000005972, 6937.500008499, 7125.000006936,
      7312.500008549, 7500.000005032, 7687.500004875, 7875.000004878, 8062.500007586, 8250.000006218,
      8437.499999805, 8625.000000113, 8812.499997984, 9000.000008860, 9187.500004401, 9375.000001529,
      9562.500006565, 9750.000006335, 9937.499999557, 10125.000002928, 10312.500002384, 10500.000004406,
      10687.500002820, 10875.000001403, 11062.500002219, 11250.000001097, 11437.500001292, 11625.000000815,
      11812.500000140, 12000.000000000, 12187.499999584, 12374.999999473, 12562.499999294, 12749.999998799,
      12937.499997514, 13124.999998543, 13312.499997602, 13499.999995904, 13687.499996961, 13874.999996550,
      14062.500000495, 14249.999993960, 14437.499993440, 14624.999997861, 14812.499995461, 14999.999991137,
      15187.500001756, 15374.999999428, 15562.500000999, 15749.999993809, 15937.499992382, 16124.999995683,
      16312.499995240, 16499.999994365, 16687.499991354, 16874.999992234, 17062.499991361, 17249.999994298,
      17437.499992410, 17624.999995960, 17812.499995945, 17999.999994836, 18187.499996913, 18374.999996125,
      18562.499990092, 18749.999991865, 18937.499986965, 19124.999985762, 19312.499985261, 19499.999989766,
      19687.499988292, 19874.999989851, 20062.499978542, 20249.999981602, 20437.500005879, 20625.000004853,
      20812.500015815, 20999.999958305, 21187.499980259, 21374.999997733, 21562.499955794, 21749.999946298,
      21937.500025004, 22124.999975461, 22312.499968567, 22499.999924162, 22687.499964503, 22874.999982475,
      23062.499968048, 23249.999976609, 23437.499982579, 23624.999922020, 23812.499893152, 24000.000000000
        }
```

afSTFT centre frequencies for 128 hop size and hybrid-mode enabled (48kHz)

Definition at line 61 of file afSTFTlib.c.

6.50 framework/resources/afSTFT/afSTFTlib.h File Reference

Slightly modified version of afSTFTlib.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct complexVector

  Complex data type used by afSTFTlib.

Macros

- #define AFSTFT_USE_SAF_UTILITIES

  Remove the "AFSTFT_USE_SAF_UTILITIES" definition, and add the vecTools.h/c and fft4g.h/c to your project, if you want to use the original afSTFT vector code.

Functions

- void afSTFTinit (void ∗∗handle, int hopSize, int inChannels, int outChannels, int LDmode, int hybridMode)

  Initialises an instance of afSTFTlib [1].
- void afSTFTchannelChange (void ∗handle, int new_inChannels, int new_outChannels)

  Re-allocates memory to support a change in the number of input/output channels.
- void afSTFTclearBuffers (void ∗handle)

  Flushes time-domain buffers with zeros.
- void afSTFTforward (void ∗handle, float ∗∗inTD, complexVector ∗outFD)

  Applies the forward afSTFT transform.
- void afSTFTinverse (void ∗handle, complexVector ∗inFD, float ∗∗outTD)

  Applies the backward afSTFT transform.
- void afSTFTfree (void ∗handle)

  Destroys an instance of afSTFTlib.

Variables

- const double __afCenterFreq48e3 [133]

  afSTFT centre frequencies for 128 hop size and hybrid-mode enabled (48kHz)
- const double __afCenterFreq44100 [133]

  afSTFT centre frequencies for 128 hop size and hybrid-mode enabled (44.1kHz)

6.50.1 Detailed Description

Slightly modified version of afSTFTlib.

The original afSTFT code, written by Juha Vilkamo, can be found here: `https://github.com/jvilkamo/afSTFT` This version is slightly modified. It adds a function to change the number of channels on the fly and includes vectors for the hybrid mode centre frequencies @44.1kHz/48kHz with 128 hop size for convenience. It also supports the use of SAF utilities (for the vectorisation and FFT).

6.50.2 Macro Definition Documentation

6.50.2.1 AFSTFT_USE_SAF_UTILITIES `#define AFSTFT_USE_SAF_UTILITIES`

Remove the "AFSTFT_USE_SAF_UTILITIES" definition, and add the vecTools.h/c and fft4g.h/c to your project, if you want to use the original afSTFT vector code.

Note the vecTools.h/c and fft4g.h/c files, may be found here: https://github.com/jvilkamo/afSTFT

Definition at line 48 of file afSTFTlib.h.

6.50.3 Function Documentation

6.50.3.1 afSTFTchannelChange()

```
void afSTFTchannelChange (
        void * handle,
        int new_inChannels,
        int new_outChannels )
```

Re-allocates memory to support a change in the number of input/output channels.

Note

Not thread safe. So do not call in the middle of a real-time loop.

Parameters

in	*handle*	afSTFTlib handle
in	*new_inChannels*	New number of input channels
in	*new_outChannels*	New number of output channels

Definition at line 209 of file afSTFTlib.c.

6.50.3.2 afSTFTclearBuffers()

```
void afSTFTclearBuffers (
        void * handle )
```

Flushes time-domain buffers with zeros.

Parameters

in	*handle*	afSTFTlib handle

Definition at line 259 of file afSTFTlib.c.

6.50.3.3 afSTFTforward()

```
void afSTFTforward (
        void * handle,
```

```
float ** inTD,
complexVector * outFD )
```

Applies the forward afSTFT transform.

Parameters

in	*handle*	afSTFTlib handle
in	*inTD*	input time-domain signals; inChannels x hopSize
in	*outFD*	input time-frequency domain signals; inChannels x nBands

Definition at line 279 of file afSTFTlib.c.

6.50.3.4 afSTFTfree()

```
void afSTFTfree (
    void * handle )
```

Destroys an instance of afSTFTlib.

Parameters

in	*handle*	(&) afSTFTlib handle

Definition at line 493 of file afSTFTlib.c.

6.50.3.5 afSTFTinit()

```
void afSTFTinit (
    void ** handle,
    int hopSize,
    int inChannels,
    int outChannels,
    int LDmode,
    int hybridMode )
```

Initialises an instance of afSTFTlib [1].

Parameters

in	*handle*	(&) afSTFTlib handle
in	*hopSize*	Hop size, in samples
in	*inChannels*	Number of input channels
in	*outChannels*	Number of output channels
in	*LDmode*	'0' disable low-delay mode, '1' enable
in	*hybridMode*	'0' disable hybrid-mode, '1' enable

See also

[1] Vilkamo, J., & Backstrom, T. (2018). Time–Frequency Processing: Methods and Tools. In Parametric Time-Frequency Domain Spatial Audio. John Wiley & Sons.

Definition at line 115 of file afSTFTlib.c.

6.50.3.6 afSTFTinverse()

```
void afSTFTinverse (
        void * handle,
        complexVector * inFD,
        float ** outTD )
```

Applies the backward afSTFT transform.

Parameters

in	*handle*	afSTFTlib handle
in	*inFD*	output time-domain signals; outChannels x hopSize
in	*outTD*	output time-frequency domain signals; outChannels x nBands

Definition at line 375 of file afSTFTlib.c.

6.51 framework/resources/convhull_3d/convhull_3d.c File Reference

An implementation of the 3-D quickhull algorithm [1].

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <float.h>
#include <ctype.h>
#include "convhull_3d.h"
#include "../../modules/saf_utilities/saf_utilities.h"
```

Data Structures

- struct float_w_idx

  Helper struct for qsort in convhull_3d_build()
- struct int_w_idx

  Helper struct for qsort in convhull_3d_build()

Macros

- #define **CONVHULL_3D_USE_CBLAS**
- #define **CV_STRNCPY**(a, b, c) strncpy(a,b,c);
- #define **CV_STRCAT**(a, b) strcat(a,b);
- #define **CH_FLT_MIN** DBL_MIN
- #define **CH_FLT_MAX** DBL_MAX
- #define **CH_NOISE_VAL** 0.0000001
- #define **CH_MAX_NUM_FACES** 50000

Functions

- static int **cmp_asc_float** (const void ∗, const void ∗)
- static int **cmp_desc_float** (const void ∗, const void ∗)
- static int **cmp_asc_int** (const void ∗, const void ∗)
- static int **cmp_desc_int** (const void ∗, const void ∗)
- static void **sort_float** (CH_FLOAT ∗, CH_FLOAT ∗, int ∗, int, int)
- static void **sort_int** (int ∗, int ∗, int ∗, int, int)
- static ch_vec3 **cross** (ch_vec3 ∗, ch_vec3 ∗)
- static CH_FLOAT **det_4x4** (CH_FLOAT ∗)
- static void **plane_3d** (CH_FLOAT ∗, CH_FLOAT ∗, CH_FLOAT ∗)
- static void **ismember** (int ∗, int ∗, int ∗, int, int)
- void convhull_3d_build (ch_vertex ∗const in_vertices, const int nVert, int ∗∗out_faces, int ∗nOut_faces)

  Builds the 3-D convexhull using the quickhull algorithm [1].
- void convhull_3d_export_obj (ch_vertex ∗const vertices, const int nVert, int ∗const faces, const int nFaces, const int keepOnlyUsedVerticesFLAG, char ∗const obj_filename)

  Exports the vertices, face indices, and face normals, as an '.obj' file, ready for the GPU.
- void convhull_3d_export_m (ch_vertex ∗const vertices, const int nVert, int ∗const faces, const int nFaces, char ∗const m_filename)

  Exports the vertices, face indices, and face normals, as an '.m' file, for Matlab verification.
- void extractVerticesFromObjFile (char ∗const obj_filename, ch_vertex ∗∗out_vertices, int ∗out_nVert)

  Reads an '.obj' file and extracts only the vertices.

6.51.1 Detailed Description

An implementation of the 3-D quickhull algorithm [1].

The code is largely derived from the "computational-geometry-toolbox" by George Papazafeiropoulos (c) 2014, originally distributed under the BSD (2-clause) license. Taken from: https://github.com/leomccormack/convhull_3d

6.51.1.1 Dependencies

CBLAS (optional) for speed ups, especially for very large meshes

See also

[1] C. Bradford, Barber, David P. Dobkin and Hannu Huhdanpaa, "The Quickhull Algorithm for Convex Hull". Geometry Center Technical Report GCG53, July 30, 1993

Author

Leo McCormack

Date

02.10.2017

6.51.2 Function Documentation

6.51.2.1 convhull_3d_build()

```
void convhull_3d_build (
        ch_vertex *const in_vertices,
        const int nVert,
        int ** out_faces,
        int * nOut_faces )
```

Builds the 3-D convexhull using the quickhull algorithm [1].

Parameters

in	*in_vertices*	Vector of input vertices; nVert x 1
in	*nVert*	Number of vertices
out	*out_faces*	(&) output face indices; FLAT: nOut_faces x 3
out	*nOut_faces*	(&) number of output face indices

See also

[1] C. Bradford, Barber, David P. Dobkin and Hannu Huhdanpaa, "The Quickhull Algorithm for Convex Hull". Geometry Center Technical Report GCG53, July 30, 1993

Definition at line 300 of file convhull_3d.c.

6.51.2.2 convhull_3d_export_m()

```
void convhull_3d_export_m (
    ch_vertex *const vertices,
    const int nVert,
    int *const faces,
    const int nFaces,
    char *const m_filename )
```

Exports the vertices, face indices, and face normals, as an '.m' file, for Matlab verification.

Parameters

in	*vertices*	Vector of input vertices; nVert x 1
in	*nVert*	Number of vertices
in	*faces*	Face indices; flat: nFaces x 3
in	*nFaces*	Number of faces in hull
in	*m_filename*	*.m filename, WITHOUT extension

Definition at line 832 of file convhull_3d.c.

6.51.2.3 convhull_3d_export_obj()

```
void convhull_3d_export_obj (
    ch_vertex *const vertices,
    const int nVert,
    int *const faces,
    const int nFaces,
    const int keepOnlyUsedVerticesFLAG,
    char *const obj_filename )
```

Exports the vertices, face indices, and face normals, as an '.obj' file, ready for the GPU.

Parameters

in	*vertices*	Vector of input vertices; nVert x 1
in	*nVert*	Number of vertices

Parameters

in	*faces*	Face indices; flat: nFaces x 3
in	*nFaces*	Number of faces in hull
in	*keepOnlyUsedVerticesFLAG*	'0' exports in_vertices, '1': exports Only used vertices
in	*obj_filename*	∗.obj filename, WITHOUT extension

Definition at line 752 of file convhull_3d.c.

6.51.2.4 extractVerticesFromObjFile()

```
void extractVerticesFromObjFile (
        char *const obj_filename,
        ch_vertex ** out_vertices,
        int * out_nVert )
```

Reads an '.obj' file and extracts only the vertices.

Parameters

in	*obj_filename*	∗.obj filename, WITHOUT extension
out	*out_vertices*	(&) output vertices; out_nVert x 1
out	*out_nVert*	(&) number of vertices

Definition at line 867 of file convhull_3d.c.

6.52 framework/resources/convhull_3d/convhull_3d.h File Reference

An implementation of the 3-D quickhull algorithm [1].

Data Structures

- struct _ch_vertex

  vertex structure, used by convhull_3d

Typedefs

- typedef double **CH_FLOAT**
- typedef ch_vertex **ch_vec3**

Functions

- void convhull_3d_build (ch_vertex ∗const in_vertices, const int nVert, int ∗∗out_faces, int ∗nOut_faces)

  Builds the 3-D convexhull using the quickhull algorithm [1].
- void convhull_3d_export_obj (ch_vertex ∗const vertices, const int nVert, int ∗const faces, const int nFaces, const int keepOnlyUsedVerticesFLAG, char ∗const obj_filename)

  Exports the vertices, face indices, and face normals, as an '.obj' file, ready for the GPU.
- void convhull_3d_export_m (ch_vertex ∗const vertices, const int nVert, int ∗const faces, const int nFaces, char ∗const m_filename)

  Exports the vertices, face indices, and face normals, as an '.m' file, for Matlab verification.
- void extractVerticesFromObjFile (char ∗const obj_filename, ch_vertex ∗∗out_vertices, int ∗out_nVert)

  Reads an '.obj' file and extracts only the vertices.

6.52.1 Detailed Description

An implementation of the 3-D quickhull algorithm [1].

The code is largely derived from the "computational-geometry-toolbox" by George Papazafeiropoulos (c) 2014, originally distributed under the BSD (2-clause) license. Taken from: https://github.com/leomccormack/convhull_3d

6.52.1.1 Dependencies

CBLAS (optional) for speed ups, especially for very large meshes

See also

[1] C. Bradford, Barber, David P. Dobkin and Hannu Huhdanpaa, "The Quickhull Algorithm for Convex Hull". Geometry Center Technical Report GCG53, July 30, 1993

Author

Leo McCormack

Date

02.10.2017

6.52.2 Function Documentation

6.52.2.1 convhull_3d_build()

```
void convhull_3d_build (
        ch_vertex *const in_vertices,
        const int nVert,
        int ** out_faces,
        int * nOut_faces )
```

Builds the 3-D convexhull using the quickhull algorithm [1].

Parameters

in	*in_vertices*	Vector of input vertices; nVert x 1
in	*nVert*	Number of vertices
out	*out_faces*	(&) output face indices; FLAT: nOut_faces x 3
out	*nOut_faces*	(&) number of output face indices

See also

[1] C. Bradford, Barber, David P. Dobkin and Hannu Huhdanpaa, "The Quickhull Algorithm for Convex Hull". Geometry Center Technical Report GCG53, July 30, 1993

Definition at line 300 of file convhull_3d.c.

6.52.2.2 convhull_3d_export_m()

```
void convhull_3d_export_m (
        ch_vertex *const vertices,
        const int nVert,
        int *const faces,
        const int nFaces,
        char *const m_filename )
```

Exports the vertices, face indices, and face normals, as an '.m' file, for Matlab verification.

Parameters

in	*vertices*	Vector of input vertices; nVert x 1
in	*nVert*	Number of vertices
in	*faces*	Face indices; flat: nFaces x 3
in	*nFaces*	Number of faces in hull
in	*m_filename*	*.m filename, WITHOUT extension

Definition at line 832 of file convhull_3d.c.

6.52.2.3 convhull_3d_export_obj()

```
void convhull_3d_export_obj (
        ch_vertex *const vertices,
        const int nVert,
        int *const faces,
        const int nFaces,
        const int keepOnlyUsedVerticesFLAG,
        char *const obj_filename )
```

Exports the vertices, face indices, and face normals, as an '.obj' file, ready for the GPU.

Parameters

in	*vertices*	Vector of input vertices; nVert x 1
in	*nVert*	Number of vertices
in	*faces*	Face indices; flat: nFaces x 3
in	*nFaces*	Number of faces in hull
in	*keepOnlyUsedVerticesFLAG*	'0' exports in_vertices, '1': exports Only used vertices
in	*obj_filename*	*.obj filename, WITHOUT extension

Definition at line 752 of file convhull_3d.c.

6.52.2.4 extractVerticesFromObjFile()

```
void extractVerticesFromObjFile (
        char *const obj_filename,
        ch_vertex ** out_vertices,
        int * out_nVert )
```

Reads an '.obj' file and extracts only the vertices.

Parameters

in	*obj_filename*	∗.obj filename, WITHOUT extension
out	*out_vertices*	(&) output vertices; out_nVert x 1
out	*out_nVert*	(&) number of vertices

Definition at line 867 of file convhull_3d.c.

6.53 framework/resources/md_malloc/md_malloc.c File Reference

Implementations of dynamic memory allocation functions for contiguous multidimensional "arrays".

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "md_malloc.h"
```

Functions

- void ∗ malloc1d (size_t dim1_data_size)

  1-D malloc
- void ∗ calloc1d (size_t dim1, size_t data_size)

  1-D calloc
- void ∗ realloc1d (void ∗ptr, size_t dim1_data_size)

  1-D realloc
- void free1d (void ∗∗ptr)

  1-D free
- void ∗∗ malloc2d (size_t dim1, size_t dim2, size_t data_size)

  2-D malloc
- void ∗∗ calloc2d (size_t dim1, size_t dim2, size_t data_size)

  2-D calloc
- void ∗∗ realloc2d (void ∗∗ptr, size_t dim1, size_t dim2, size_t data_size)

  2-D realloc
- void free2d (void ∗∗∗ptr)

  2-D free
- void ∗∗∗ malloc3d (size_t dim1, size_t dim2, size_t dim3, size_t data_size)

  3-D malloc
- void ∗∗∗ calloc3d (size_t dim1, size_t dim2, size_t dim3, size_t data_size)

  3-D calloc
- void ∗∗∗ realloc3d (void ∗∗∗ptr, size_t dim1, size_t dim2, size_t dim3, size_t data_size)

  3-D realloc
- void free3d (void ∗∗∗∗ptr)

  3-D free

6.53.1 Detailed Description

Implementations of dynamic memory allocation functions for contiguous multidimensional "arrays".

Taken from: https://github.com/leomccormack/md_malloc

An example of allocating, indexing and freeing a 3-D "array":

```
float*** example3D = (float***)malloc3d(10, 20, 5, sizeof(float);
// Due to the contiguous nature of the allocation, this is possible:
memset(ADR3D(example3D), 0, 10*20*5*sizeof(float));
// And my still be indexed normally as:
example3D[3][19][2] = 22.0f;
// To free, simply call:
free(example3D);
```

Author

Leo McCormack

Date

11.06.2019

6.54 framework/resources/md_malloc/md_malloc.h File Reference

Implementations of dynamic memory allocation functions for contiguous multidimensional "arrays".

Macros

- #define **ADR1D**(A) (&A[0])
- #define **ADR2D**(A) (&A[0][0])
- #define **ADR3D**(A) (&A[0][0][0])

Functions

- void * malloc1d (size_t dim1_data_size)

  1-D malloc
- void * calloc1d (size_t dim1, size_t data_size)

  1-D calloc
- void * realloc1d (void *ptr, size_t dim1_data_size)

  1-D realloc
- void free1d (void **ptr)

  1-D free
- void ** malloc2d (size_t dim1, size_t dim2, size_t data_size)

  2-D malloc
- void ** calloc2d (size_t dim1, size_t dim2, size_t data_size)

  2-D calloc
- void ** realloc2d (void **ptr, size_t dim1, size_t dim2, size_t data_size)

  2-D realloc
- void free2d (void ***ptr)

  2-D free
- void *** malloc3d (size_t dim1, size_t dim2, size_t dim3, size_t data_size)

  3-D malloc
- void *** calloc3d (size_t dim1, size_t dim2, size_t dim3, size_t data_size)

  3-D calloc
- void *** realloc3d (void ***ptr, size_t dim1, size_t dim2, size_t dim3, size_t data_size)

  3-D realloc
- void free3d (void ****ptr)

  3-D free

6.54.1 Detailed Description

Implementations of dynamic memory allocation functions for contiguous multidimensional "arrays".

Taken from: https://github.com/leomccormack/md_malloc

An example of allocating, indexing and freeing a 3-D "array":

```
float*** example3D = (float***)malloc3d(10, 20, 5, sizeof(float);
// Due to the contiguous nature of the allocation, this is possible:
memset(ADR3D(example3D), 0, 10*20*5*sizeof(float));
// And my still be indexed normally as:
example3D[3][19][2] = 22.0f;
// To free, simply call:
free(example3D);
```

Author

Leo McCormack

Date

11.06.2019

Index