HARK version 1.0.0 Document
(document version 0.1.0)

HARK Documentation Team

February 4, 2011

[ARIEL sings]
Come unto these yellow sands,
And then tale hands:
Curt'sied when you have, and kiss'd,
(The wild waves whist;)
Foot it featly hear and there;
And sweet sprites, the burden bear.
[Burden dispersedly.]
HARK, hark! bowgh-wowgh: the watch-dogs bark,
Bowgh-wowgh.
Ariel. HARK, hark! I hear
The strain of strutting chanticleer
Cry cock-a-doodle-doo.

Ariel's Song, The Tempest, Act I, Scene II, William Shakespeare

Contents

1 Introduction **1**
- 1.1 Robot audition software is an integrated system . . . 1
- 1.2 Design philosophy of HARK . . . 1
- 1.3 Application of HARK . . . 7
  - 1.3.1 Three-speaker simultaneous speech recognition . . . 8
  - 1.3.2 Judge in rock-paper-scissors game orally played . . . 9
  - 1.3.3 CASA 3D Visualizer . . . 9
  - 1.3.4 Application to telepresence robot . . . 10
- 1.4 Conclusion . . . 11

2 Robot audition and its problem **14**
- 2.1 Technology to distinguish sounds is the base for robot audition . . . 14
- 2.2 Robot audition based on sound environment understanding . . . 14
- 2.3 Distinguish sounds with two microphones like human being . . . 15
- 2.4 Function of suppress self-generated sound . . . 16
- 2.5 Elimination of ambiguity by integration of audio visual information . . . 17
- 2.6 Killer application that robot audition opens up . . . 18
- 2.7 Conclusion . . . 19

3 Instructions for First-Time HARK Users **20**
- 3.1 Acquisition of the Software . . . 20
- 3.2 Installation of the Software . . . 20
  - 3.2.1 Installation from the package . . . 20
- 3.3 FlowDesigner . . . 21
  - 3.3.1 Basic operation of FlowDesigner . . . 21
  - 3.3.2 Module operation / terminal connection / property setting . . . 22
  - 3.3.3 Creation of network for beginner . . . 24

4 Data Types **30**
- 4.1 Basic type . . . 32
- 4.2 FlowDesigner object type . . . 33
  - 4.2.1 Vector . . . 33
  - 4.2.2 Matrix . . . 33
- 4.3 Type peculiar to FlowDesigner . . . 35
  - 4.3.1 any . . . 35
  - 4.3.2 ObjectRef . . . 35
  - 4.3.3 Object . . . 35
  - 4.3.4 subnet_param . . . 35
- 4.4 HARK-specific type . . . 36
  - 4.4.1 Map . . . 36
  - 4.4.2 Source . . . 36
- 4.5 HARK standard coordinate system . . . 37

5 File Formats **38**
5.1 HGTF file format . . . 38
5.1.1 Source localization transfer function for LocalizeMUSIC . . . 40
5.1.2 Sound source separation transfer function for GHDSS . . . 40
5.1.3 Separation matrix for GHDSS . . . 40
5.2 HARK text format . . . 41
5.2.1 Microphone position text format . . . 41
5.2.2 Noise position text format . . . 42
5.3 Other file formats . . . 43
5.3.1 Source position list information (srcinf) format . . . 43
5.3.2 PCM binary format . . . 43
5.3.3 `float`binary . . . 44
5.3.4 localization result text . . . 44
5.3.5 Map text . . . 44

6 Module References **45**
6.1 AudioIO category . . . 47
6.1.1 AudioStreamFromMic . . . 47
6.1.2 AudioStreamFromWave . . . 51
6.1.3 SaveRawPCM . . . 53
6.2 Localization category . . . 55
6.2.1 ConstantLocalization . . . 55
6.2.2 DisplayLocalization . . . 57
6.2.3 LocalizeMUSIC . . . 58
6.2.4 LoadSourceLocation . . . 62
6.2.5 SaveSourceLocation . . . 64
6.2.6 SourceIntervalExtender . . . 65
6.2.7 SourceTracker . . . 67
6.3 Separation category . . . 70
6.3.1 BGNEstimator . . . 70
6.3.2 CalcSpecSubGain . . . 74
6.3.3 CalcSpecAddPower . . . 76
6.3.4 EstimateLeak . . . 77
6.3.5 GHDSS . . . 78
6.3.6 HRLE . . . 89
6.3.7 PostFilter . . . 92
6.3.8 SpectralGainFilter . . . 104
6.4 FeatureExtraction category . . . 106
6.4.1 Delta . . . 106
6.4.2 FeatureRemover . . . 108
6.4.3 MelFilterBank . . . 110
6.4.4 MFCCExtraction . . . 114
6.4.5 MSLSExtraction . . . 116
6.4.6 PreEmphasis . . . 119
6.4.7 SaveFeatures . . . 121
6.4.8 SaveHTKFeatures . . . 123
6.4.9 SpectralMeanNormalization . . . 125
6.5 MFM category . . . 127
6.5.1 DeltaMask . . . 127
6.5.2 DeltaPowerMask . . . 129
6.5.3 MFMGeneration . . . 130
6.6 ASRIF category . . . 133
6.6.1 SpeechRecognitionClient . . . 133
6.6.2 SpeechRecognitionSMNClient . . . 135

6.7 MISC category . . . 137
6.7.1 ChannelSelector . . . 137
6.7.2 DataLogger . . . 139
6.7.3 MatrixToMap . . . 141
6.7.4 MultiDownSampler . . . 143
6.7.5 MultiFFT . . . 147
6.7.6 MultiGain . . . 151
6.7.7 PowerCalcForMap . . . 153
6.7.8 PowerCalcForMatrix . . . 154
6.7.9 SegmentAudioStreamByID . . . 155
6.7.10 SourceSelectorByDirection . . . 157
6.7.11 SourceSelectorByID . . . 159
6.7.12 Synthesize . . . 161
6.7.13 WhiteNoiseAdder . . . 163
6.8 Modules independent of FlowDesigner . . . 165
6.8.1 JuliusMFT . . . 165

7 Support Tools **172**
7.1 harktool . . . 172
7.1.1 Outline . . . 172
7.1.2 Installation method . . . 172
7.1.3 Usage . . . 172

8 HARK-compatible Multi-Channel AD Devices and Their Installation **173**
8.1 System In Fronteir, Inc. the RASP series, Radio RASP . . . 173
8.1.1 Connection to a PC of the radio RASP . . . 173
8.1.2 Installation and setting of software for the radio RASP . . . 174
8.1.3 Recording test in HARK with the radio RASP . . . 174
8.2 RME Hammerfall DSP series Multiface AE . . . 175
8.2.1 Connection of Multiface to PC . . . 176
8.2.2 Sound recording test with Multiface in HARK . . . 178
8.3 Tokyo Electron Device TD-BD-16ADUSB . . . 181
8.3.1 Connection to PC of 16ADUSB . . . 182
8.3.2 Installation and setting of software for 16ADUSB . . . 182
8.3.3 Sound recording test in HARK with TD-BD-16ADUSB . . . 182

Chapter 1

Introduction

This document is the collected studies of information on the robot audition software HARK (HRI-JP Audition for Robots with Kyoto Univ., "hark" means "listen" in medieval English). Chapter 1 describes outlines of the design philosophy, design strategy, individual technology and applications of HARK while it overviews the new horizon that the robot audition software and robot audition functions commencing with HARK open up.

1.1 Robot audition software is an integrated system

A person "recognizes" sounds in various environments where various sounds are heard, process them to communicate with people and enjoy TV, music or movies. A robot audition function that provides such recognizing processing needs to include a function to process the sounds that are heard in various environments at various levels and therefore it cannot be defined at a word similar to the robot vision function.

Indeed, OpenCV, the open source image processing software, is an aggregate of a huge number of processing modules and therefore it is essential also for the robot audition software to consist of aggregates that include minimum required functions. The robot audition software HARK is the system that aims at being "auditory OpenCV". HARK, like OpenCV, contains signal processing algorithms, measurement tools and GUI from a device level for the module necessary to "recognize" and also is published as an open source. The three major tasks in Computational Auditory Scene Analysis, which is for understanding the environment from sound information, are sound source localization, sound source separation and automatic speech recognition. The first edition HARK has been developed as an achievement obtained from these researches. HARK is currently opened charge-free[1] as an open source for research use.

Chapter 2 describes the design philosophy of HARK and provides an outline about FlowDesigner, which HARK presently uses as middleware. Chapter 3 gives an outline about the module group of HARK. Chapter 4 describes a future development schedule of HARK.

1.2 Design philosophy of HARK

Design philosophy of the robot audition software HARK is summarized as follows.

1. **Provision of total functions, from the input to sound source localization / source separation / speech recognition**: Guarantee of the total performance such as inputs from microphones installed in a robot, sound source localization, source separation, noise suppression and automatic speech recognition,

2. **Correspondence to robot shape**: Corresponds to the microphone layout required by a user and incorporation to signal processing,

3. **Correspondence to multichannel A/D systems**: Supports various multichannel A/D systems depending on price range / function,

[1] http://winnie.kuis.kyoto-u.ac.jp/HARK/

4. **Provision of optimal sound processing module and advice** For the signal processing algorithms, each algorithm is based on effective premises, plural algorithms have been developed for the same function and optimal modules are provided through the usage experience,

5. **Real-time processing**: Essential for performing interactions and behaviors through sounds.

HARK 0.1.7, with such a design philosophy, was published as an open source in April, 2008 through improvement by developers themselves, reflection of feedbacks from a users, bug-fixes and substantiality of documents. HARK 1.0.0 prerelease was published in November, 2009 and HARK 1.0.0 of the final version is going to be published in October, 2010. "HARK 1.0.0" stands for "The final version HARK 1.0.0" below. The main new features of HARK 1.0.0 are as follows:

1. Performance improvement and robustization of source separation by a new implement of source separation,
2. Correspondence to comparatively complicated robot shapes at signal processing levels,
3. A simple function of self-generated sound suppression that suppresses background noise of a robot,
4. Sound source localization / source separation methods developed for moving sound sources,
5. Reduction of parameters for sound source localization / source separation i and detail set-up functions of them
6. Speech feature quantity newly developed and a function for improving speech recognition,
7. Visualization of setting data and provision of creation tools,
8. Improved operationality of the middleware FlowDesigner.

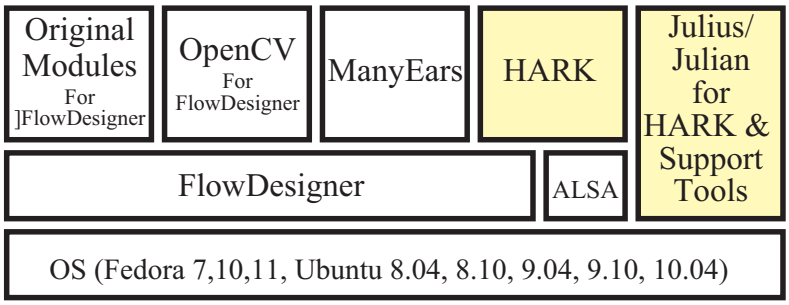


Figure 1.1: Relation between the robot audition software HARK and the middleware FlowDesigner and OS

As shown in Figure 1.1, HARK uses FlowDesigner [**?**] as middleware excluding the speech recognition part (Julius) and support tools. As understood from Figure 1.1, only the Linux OS is supported. One of the reasons is that API called ALSA (Advanced Linux Sound Architecture) is used so as to support plural multichannel A/D systems. HARK for PortAudio is also being developed since PortAudio comes to be used in Windows systems recently.

Middleware FlowDesigner

In most robot auditions, sound sources are separated based on sound source localization data and speech recognition is performed for separated speeches. Each processing would become flexible by consisting it of plural modules so that an algorithm can be partially substituted. Therefore, it is essential to introduce the middleware that allows efficient integrations between the modules. However, as the number of modules to be integrated increases, the total overhead of the module connections increases and a real time performance is lost. It is difficult to respond to such a problem with a common frame such as CORBA (Common Object Request Broker Architecture), which requires data serialization at the time of module connection. Indeed, in each module of HARK, processing is performed with the same acoustic data in the same time frame, If each module used the acoustic data by memory-copying each time, the processing would be inefficient in terms of speed and memory efficiency. We have employed FlowDesigner [**?**] as a middleware that can respond to such a problem, which is a data flow-oriented GUI support environment. The processing in FlowDesigner is faster and lighter than those in the frames that can universally be used for integrations such as the CORBA frame. FlowDesigner is free

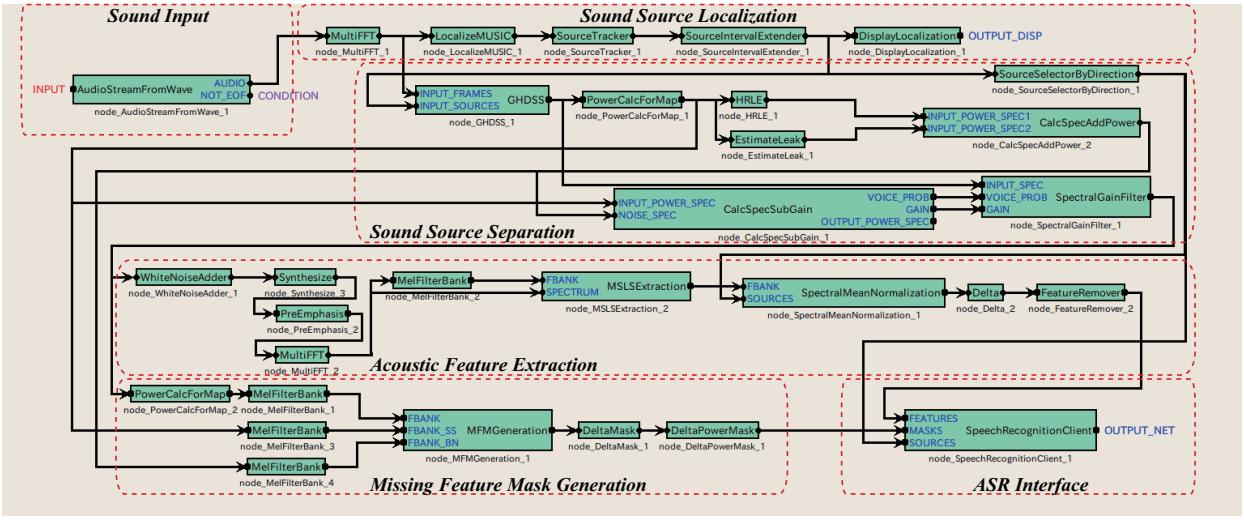


Figure 1.2: Module construction on FlowDesigner for typical robot audition with HARK

(LGPL/GPL) middleware equipped with a data flow-oriented GUI development environment that realizes high-speed and lightweight module integration, premised on use in a single computer. In FlowDesigner, each module is realized as a class of C++. Since these classes have inherited common superclasses, the interface between modules is naturally commonized. Since module connections are realized by calling a specific method of each class (function call), the overhead is small. Since data are transferred by pass-by-reference and pointers, processing is performed at high speed with few resources for the above-mentioned acoustic data. In other words, both data transmission rate between modules and module reuse can be maintained by using FlowDesigner. We are publishing FlowDesigner for which bugs such as memory leak have been removed and its operationality (mainly the attribute setting) has been improved based on the past use experience.
[2]

1.2 shows a network of FlowDesigner for the typical robot audition with HARK. Multichannel acoustic signals are acquired by input files and sound source localization / source separation are performed. Missing feature masks (MFM) are generated by extracting acoustic features from the separated sound and sent to speech recognition (ASR). Attribute of each module can be set on the attribute setting screen (Figure1.3 shows an example of the attribute setting screen of GHDSS). Table 1.1 shows HARK modules and external tools that are currently provided for HARK. In the following section, outlines of each module are described with the design strategy.

Input device

Plural microphones (microphone array) are mounted as ears of a robot in HARK for processing. Figure 4 shows an installment example of ears of a robot. Each of these example is equipped with a microphone array with eight channels though microphone arrays with the arbitrary number of channels can be used in HARK. The followings three types are the multichannel A/D conversion devices supported by HARK.

- JEOL System Technology Co., Ltd., The RASP series,
- okyo Electron Device Ltd., TD-BD-16ADUSB (USB interface),
- A/D conversion device of ALSA base, (e.g. RME, Hammerfall DSP series, Multiface AE)

These A/D systems have 16 channel inputs. The sixteen channel inputs can be used in HARK by changing internal parameters in HARK. However, the processing speed might fall under such a condition. Moreover, even when sampling is performed with 24 bits for signal expression, no changes are required. Further, the sampling rate assumed for HARK is 16kHz and therefore a downsampling module can be used for 48KHz sampling data. Low-priced pin microphones are enough though it will be better if a preamplifier is used for resolving lack of gain. OctaMic II is available from RME.

[2]The original version of FlowDesigner and function-improved version of FlowDesigner 0.9.0 are available at http://flowdesigner.sourceforge.net/ and http://winnie.kuis.kyotou.ac.jp/HARK/, respectively.

Table 1.1: Modules and Tools provided by HARK 1.0.0

Function	Category name	Module name	Description
Voice input output	AudioIO	AudioStreamFromMic	Acquire sound from microphone
		AudioStreamFromWave	Acquire sound from file
		SaveRawPCM	Save sound in file
Sound source Localization / tracking	Localization	ConstantLocalization	Output constant localized value
		DisplayLocalization	Display localization result
		LocalizeMUSIC	Localize sound source
		LoadSourceLocation	Load localization information from file
		SaveSourceLocation	Save source location information in file
		SourceIntervalExtender	Extend forward the tracking result
		SourceTracker	Source tracking
Sound source separation	Separation	BGNEstimator	Estimate background noise
		CalcSpecSubGain	Subtract noise spectrum subtraction and estimate optimum gain
		CalcSpecAddPower	Add power spectrum
		EstimateLeak	Estimate inter-channel leak noise
		GHDSS	Separate sound source by GHDSS
		HRLE	Estimate noise spectrum
		PostFilter	Perform post-filtering after sound source separation
		SpectralGainFilter	Estimate voice spectrum
Feature extract	FeatureExtraction	Delta	Calculate Δ term
		FeatureRemover	Remove term
		MelFilterBank	Perform mel-scale filter bank processing
		MFCCExtraction	Extract MFCC
		MSLSExtraction	Extract MSLS
		PreEmphasis	Perform pre-emphasis
		SaveFeatures	Save features
		SaveHTKFeatures	Save features in the HTK form
		SpectralMeanNormalization	Normalize spectrum mean
Missing Feature Mask	MFM	DeltaMask	Calculate Δ mask term
		DeltaPowerMask	Calculate Δ power mask term
		MFMGeneration	Generate MFM
Communication with ASR	ASRIF	SpeechRecognitionClient	Send feature to ASR
		SpeechRecognitionSMNClient	Same as above, with the feature SMN
Others	MISC	ChannelSelector	Select channel
		DataLogger	Generate log output of data
		MatrixToMap	Convert Matrix $\rightarrow$ to Map
		MultiGain	Calculate gain of multiple-channel
		MultiDownSampler	Perform downsampling
		MultiFFT	Perform multichannel FFT
		PowerCalcForMap	Calculate power of Map input
		PowerCalcForMatrix	Calculate power of matrix input
		SegmentAudioStreamByID	Select audio stream segment by ID
		SourceSelectorByDirection	Select sound source by direction
		SourceSelectorByID	Select sound source by ID
		Synthesize	Convert waveform
		WhiteNoiseAdder	Add white noise
A function	Category	Tool name	Description
Data generation	External tool	hark-tool	Visualize data / Generate setting file

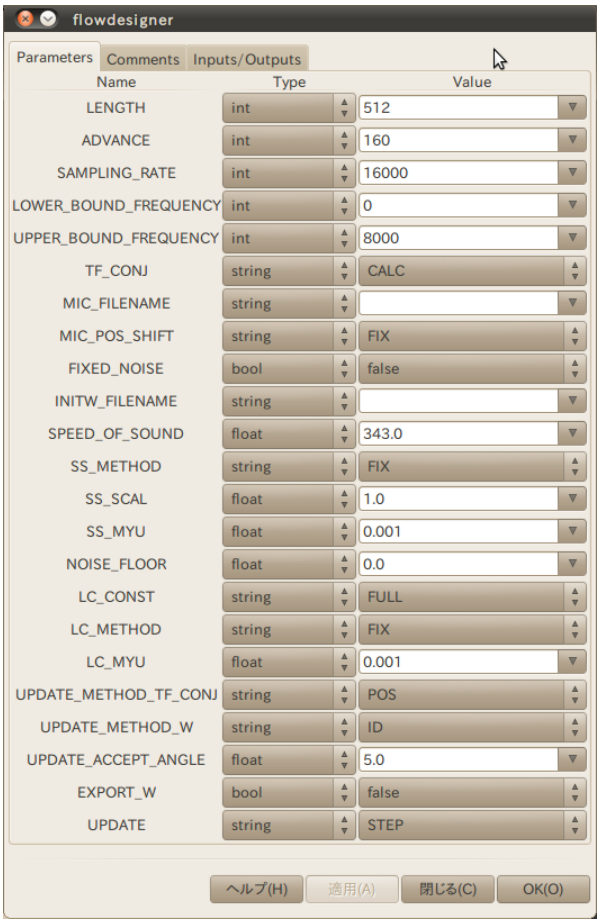


Figure 1.3: Attribute setting screen of GHDSS

SIG2 (whole body)

Robovie-R2

HRP-2 (8ch)

Figure 1.4: Three types of ear for robot (microphone layout).

Microphone array

MUltiple SIgnal Classification (MUSIC) method, which has shown the best performance in the past experience, is employed for microphone arrays. The MUSIC method is the method that localizes sound sources based on source positions and impulse responses (transfer function) between each microphone. Impulse responses can be obtained by actual measurements or calculation with geometric positions of microphones. In HARK 0.1.7, the beamformer of ManyEars [**?**] was available as a microphone array. This module is a 2D polar coordinate space (called "2D" in the semantics that direction information can be recognized in a 3D polar coordinate space). It has been reported that the error due to incorrect orientation is about 1.4° when it is within 5 m, from a microphone array and the sound source interval leaves more than 20°. However, the entire module of ManyEars is originally designed for 48 kHz sampling under the assumption that the sampling frequency is not 16 kHz, which is used in HARK, and microphones are arranged in free space when impulse responses are simulated from the microphone layout. For the above reason, impacts of the robot body cannot be considered and sound source localization accuracy of adaptive beamformers such as MUSIC is higher than that of common beamformers and therefore HARK 1.0.0 supports only the MUSIC method.

Sound source separation

For sound source separation, Geometric-Constrained High-order Source Separation (GHDSS) [**?**], which is known to have the highest total performance in various acoustic environments from the past usage experience, PostFilterand the noise estimation method Histogram-based Recursive Level Estimation HRLEare employed for HARK 1.0.0. Presently, the best performance and stability in various acoustic environments are obtained by the combination of GHDSSand HRLE. Until now, various methods such as adoptive beamformer (delayed union type, adoptive type), Independent Component Analysis (ICA) and Geometric Source Separation (GSS) have been developed and tested for evaluation. Sound source separation

methods employed for HARK are summarized as follows:

1. Delayed union type beamformer employed for HARK 0.1.7,
2. Combination of ManyEars Geometric Source Separation (GSS) and PostFilter[**?**], which was supported as an external module with HARK 0.1.7,
3. Combination of GSSand PostFilteras an original design [**?**] employed for in 1.0.0 HARK prerelease,
4. Combination of GHDSSand HRLEemployed for HARK 1.0.0 [**?**, **?**].

GSSof ManyEars used for HARK 0.1.7 is the method that uses transfer functions from a sound source to a microphone as a geometric constraint and separates the signal coming from a given sound source direction. A geometrical constraint is supposed to be a transfer function from the sound source to each microphone and transfer functions are obtained from the relation between microphone positions and sound source positions. This way of obtaining transfer functions was a cause of performance degradation under the condition that a transfer function changes as shape of a robot changes though the microphone layout is the same. GSSwas redesigned for the HARK 1.0.0 prerelease. It was extended so that transfer functions of actual measurements can be used as a geometrical constraint. Further, modifications such as adaptive change of stepsize were made so as to accelerate convergence of a separation matrix. Furthermore, it has become possible to constitute a delayed union type beamformer by changing attribute setting of GSS. In accordance with the above change, the delayed union type beamformer DSBeamformer, which had been employed for HARK 0.1.7, has been abolished. Most of sound source separation methods except ICA require direction information of the sound source to be separated as a parameter, which is common in sound source separation. If localization information is not provided, separation itself cannot be executed. On the other hand, robot's steady noise has a comparatively strong property as a directional sound source and therefore the steady noise can be removed if sound source is localized. However, in fact, sound sources are not localized successfully for such noise in many cases and there was an actual case that separation performance of steady noise was degraded as a result. A function that continuously specifies noise sources in specific directions is added in GSS and GHDSSof HARK 1.0.0 prerelease, which enables to separate continuously the sound sources that cannot be localized. Generally, there is a limit for separation performance of the sound source separation based on linear processing, such as GSSand GHDSSand therefore it is essential to perform nonlinear processing called post-filter to improve the quality of separated sounds. The post-filter of ManyEars was redesigned and the post-filter for which parameter quantity was considerably reduced is employed for HARK 1.0.0 prerelease and the final version. The post-filter can be a "good knife" if it is used in a proper way though it is difficult to make full use of it and users may suffer its adverse effect if it is used in a wrong way. There are at least some parameters that should be set in PostFilterand it is difficult to set them properly. Furthermore, the post-filter performs nonlinear processing based on a probabilistic model. Therefore, a non-linear distortion spectrum occurs for separated sounds and the performance of speech recognition ratios for separated sounds does not easily improve. The steady noise estimation method called HRLE(Histogram-based Recursive Level Estimation), which is suited for GHDSS, is employed for HARK 1.0.0. The separated sounds with improved quality are obtained when using EstimateLeak, which has been developed by fully examining the GHDSSseparation algorithm and estimates inter-channel leak energy, in combination with HRLE.

MFT-ASR: Speech recognition based on MFT

The spectral distortion caused by various factors such as sound mixture or separation is beyond those that are assumed in the conventional speech recognition community. In order to deal with it, it is necessary to connect more closely the sound source separation and speech recognition. In HARK, it is dealt with the speech recognition based on the missing feature theory (MFT-ASR) [**?**]. The concept of MFT-ASR is shown in Figure 1.5. The black and red lines in the Figure indicates time variation of acoustic features in a separated sound and that of an acoustic model for corresponding speech, which is held by the ASR system, respectively. Acoustic features of a separated sound greatly differ at some points from those of the system by distortion (Figure 1.5(A)). In MTF-ASR, influences of the distortion are ignored by masking the distorted points with Missing Feature Mask (MFM) (Figure 1.5(B)). MFM is a time reliability map that corresponds to acoustic features of a separated sound and a binary mask (also called Hard Mask) is usually used. Masks with continuous values from 0 to 1 are called Soft Mask. In HARK, MFM is provided from the steady noise obtained from the post-filter and inter-channel energy. MTF-ASR, same as common speech recognition, is based on Hidden Markov Model (HMM). Parts related to acoustic scores calculated from HMM (mainly the output probability calculation) are modified so that MFM can be used. In HARK, the multiband Julius developed by Tokyo Institute of Technology Kobi （古井）Laboratory is used,

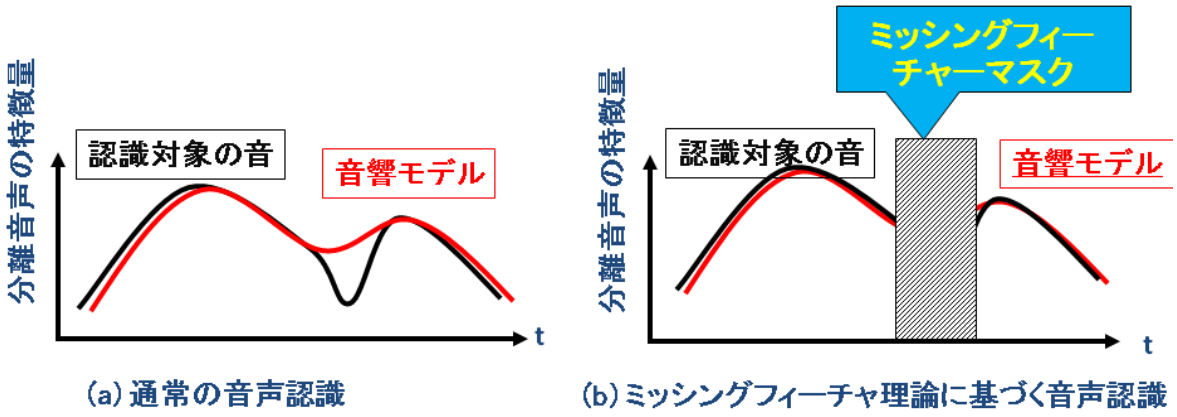


Figure 1.5: Schematic diagram of speech recognition by the missing feature theory

reinterpreted as MFT-ASR [**?**]. HARK 1.0.0 uses plug-in features of the Julius 4 type and the main part of MFT-ASR serves as a Julius plug in. Using MFT-ASR serving as a plug in allows Julius' new functions obtained by version up to be used without being modified. Moreover, MFT-ASR works as a server / daemon independent from FlowDesigner and outputs results to the acoustic features transmitted via socket communication by a speech recognition client of HARK and to their MFM.

Acoustic feature extraction and noise application to noise adaptation of acoustic model

In order to improve the effectiveness of MFT with trapping the spectral distortion only for specific acoustic features, Mel Scale Log Spectrum (MSLS) [**?**] is used for acoustic features. Mel-Frequency Cepstrum Coefficient (MFCC), which is generally used for speech recognition, is also employed for HARK. However, distortion spreads in all features in MFCC and therefore it does not get along with MFT. When simultaneous speech is little, better performance is achieved by speech recognition with MFCC in some cases. HARK 1.0.0 provides a new module to use Δ the power term with MSLS features [**?**]. The effectiveness of Δ power term for MFCC features has already been reported so far. It has already been confirmed that the 13-dimensional MSLS and Δ MSLS, and Δ power, which is the 27-dimensional MSLS feature, have better performance than the 24-dimensional MSLS and Δ MSLS (48 dimensions in total)used for HARK 0.1.7. In HARK, influences of distortion by the aforementioned non-linear separation are reduced by adding a small amount of white noise. An acoustic model is constructed by multi-condition training with clean speech and that with white noise while speech recognition is performed with the same amount of white noise added to recognized speech after separation. In this way, highly precise recognition is realized even when S/N is around -3 dB [**?**] for one speaker's speech.

1.3 Application of HARK

We have developed an auditory function robot with binaural hearing with two microphones and used three-speaker simultaneous speech recognition as a kind of benchmark. For auditory functions on SIG and SIG2, the upper body humanoid robots, the simultaneous speech recognition on the place 1m away from three speakers who stand at an interval of 30 degrees has come to be possible at a certain level of accuracy [**?**]. However, this system requires a large amount of prior knowledge and processing and therefore we had to judge that it would be difficult to equip this system as an easy auditory function that can be used in sound environments. In order to overcome this performance limit, we started to develop an auditory function robot with increased number of microphones and invented HARK. Therefore, it was natural that HARK was applied to the system that recognizes orders for dishes from three persons at the same time, which had been used as a benchmark. It presently works on robots such as Robovie-R2 and HRP -2. As deformation of three-speaker simultaneous speech recognition, the umpire robot that judges the winner of the rock-paper-scissors game orally played by three persons has been developed on Robovie-R2 [**?**]. Moreover, as a non-robot application, we have developed a system that visualizes the sounds that HARK localizes and separates from based on real time or archived data. In presentation of sounds, the situation that "the sound is not detected" is often seen in some environments. We captured this problem as lack of **auditory awareness** (sound recognition). In order to improve the auditory awareness, we designed a three-dimensional sound

environment visualization system that supports comprehension of sound environment and implemented in HARK [**?**, **?**].

1.3.1 Three-speaker simultaneous speech recognition

a) Robovie takes orders.

b) Three people order for dishes simultaneously.

c) Robovie repeats the orders after 1.9 seconds and reply a total amount.

Figure 1.6: Robovie-R2 recognizing three orders for dishes from three persons

The three-speaker simultaneous speech recognition returns speech recognition results of each speaker though a series of processing of microphone inputs, source localization, sound source separation, missing feature mask generation and ASR. The module network in this FlowDesigner is shown in Figure 1.2. The dialogue management module performs the followings

1. Listen to speech of a user. When judging it is an order request, perform the following processing.
2. Perform a series of processing of robot audition – sound source localization / sound source separation / post-filter processing / extraction of acoustic features / missing feature mask generation –
3. Transmit the acoustic features and Missing Feature Mask for the number of speakers to the speech recognition engine and receive speech recognition results.
4. Analyze the speech recognition results. When they are about orders for dishes, repeat the orders and reply a total amount for the dishes.
5. Take next orders.

The acoustic model in speech recognition is intended for unspecified speakers. The language model is described in the context-free grammar and therefore it will become possible to recognize "large portion of ramen", "large portion of spicy ramen" or "large portion of ramen and rice" by devising the grammar. In the conventional processing, which needed to go though plural files, it took 7.9 seconds from the completion of three speakers' speech to the completion of recognition. However, the response has been shortened to around 1.9 seconds by HARK. [3]. It seems that since the response is fast, the robot immediately repeats each order after taking orders from all the speakers and reply a total amount. Further, in the case of the file input, speech complete time is clear though it depends on module setup and therefore latency from the completion of recognition after utterance to starting of the response by the robot is around 0.4 seconds. Moreover, the robot can turn round the speaker's face at the time of repeating. HRP-2 performs responses with gestures. However, when giving such gestures to the robot, responses are delayed for preparation for the gestures, which leads to clumsy motions and therefore we need to make such setting carefully.

[3] Demonstration is available at http://winnie.kuis.kyoto-u.ac.jp/SIG/

1.3.2 Judge in rock-paper-scissors game orally played

It was pointed out that the simultaneous ordering from three speakers was unnatural as a demonstration and therefore we have performed a game for which simultaneous speech is essential, that is, “Oral rock-paper-scissors game”, which is a rock-paper-scissors game orally performed. Fun of “oral rock-paper-scissors game” is that players can play the game without showing their faces or they can play it in darkness. However, a problem in such cases is that players cannot figure out the winner instantly. Here, we attempt to make a robot with an auditory function judge for the oral rock-paper-scissors game [**?**]. Only the above-mentioned three-speaker simultaneous speech recognition and dialogue strategy have been modified for the program of the oral rock-paper-scissors game judge. The robot judges if the players uttered properly, if any of the players uttered late in other words, so as to judge the winner or the game was a draw and tells the result. When the game is not finished, the robot orders the players to play the game again. The details of this system are described in the paper of ICRA-2008 [**?**]. Please read it if you are interested in it.

1.3.3 CASA 3D Visualizer

Generally, speech plays a fundamental role as a communication medium between people who share the temporal and spatial space and we human being exchange information through speech in various environments. However, the robot often fails to detect various sounds. Moreover, even in the case recorded sounds are played highly faithfully, it is difficult to avoid such failure. This is a life log that attempts to record all in the life and will be a major challenge in terms of playing sounds. One of the causes of such a problem is that sound recognition (awareness) cannot be obtained from a recorded sound, lack of auditory awareness in other words, we presume. A highly-faithful replaying technique would not improve the auditory awareness to the level beyond the real world. Things that cannot be recognized in the actual world would not be solved simply by a highly-faithful replaying, we suppose. Indeed, it has been reported that it is difficult for a person to recognize more than two different sounds simultaneously from the viewpoint of psychophysics [**?**], In the case that plural sounds occur at the same time, such as a case of multi-speakers, it is essential to recognize and provide each speech. In order to

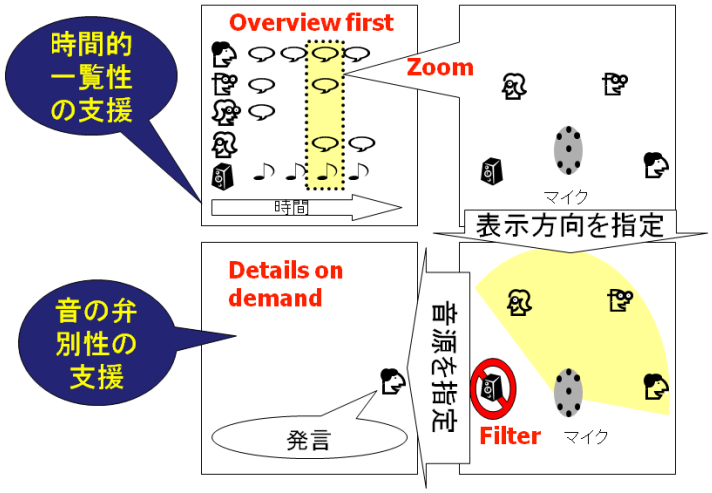


Figure 1.7: CASA 3D Visualizer: CASA 3D Visualizer: Visual Information-Seeking Matra Visualization of outputs from HARK in accordance with “Overview first, zoom and filter, then details on demand”

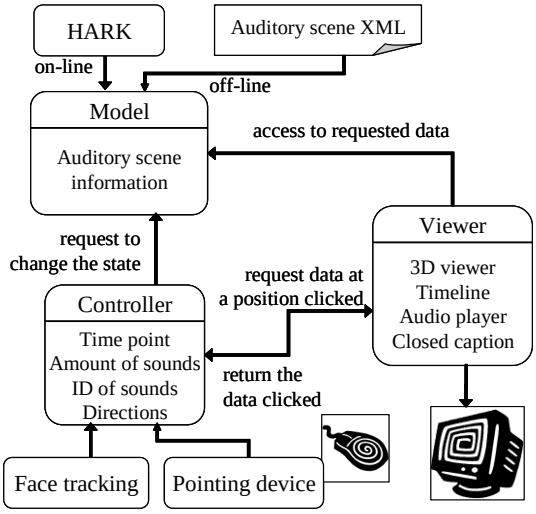


Figure 1.8: Implementation with the MVC (Model-View-Control) model of CASA 3D Visualizer

improve **auditory awareness** (sound recognition), we modified HARK, and designed and implemented a three-dimensional sound environment visualization system that supports sound environment comprehension [**?**, **?**]. For GUI, the principle of information visualization that Schneiderman proposed “ overview first, zoom and filter, then details on demand ” (Figure 1.7) was reinterpreted to sound information presentation and the following functions were designed.

1. Overview first: Show overview first.
2. Zoom: Show specific time zones in detail.

3. Filter: Extract only the sound from a specific direction and let the robot to listened to it.
4. Details on Demand: Let the robot listen to only a specific sound.

We attempted with the above GUI to support temporal viewing and distinctiveness of a sound, which had been a problem in dealing with sound information. Moreover, we designed based on the model Model-View-Control (MVC) for implementation (Figure 1.8). Information to be provided from HARK is converted into AuditoryScene XML first. Next, the 3D visualization system displays the AuditoryScene XML. Figure 1.9 shows display screen. Scaling and rotation can be

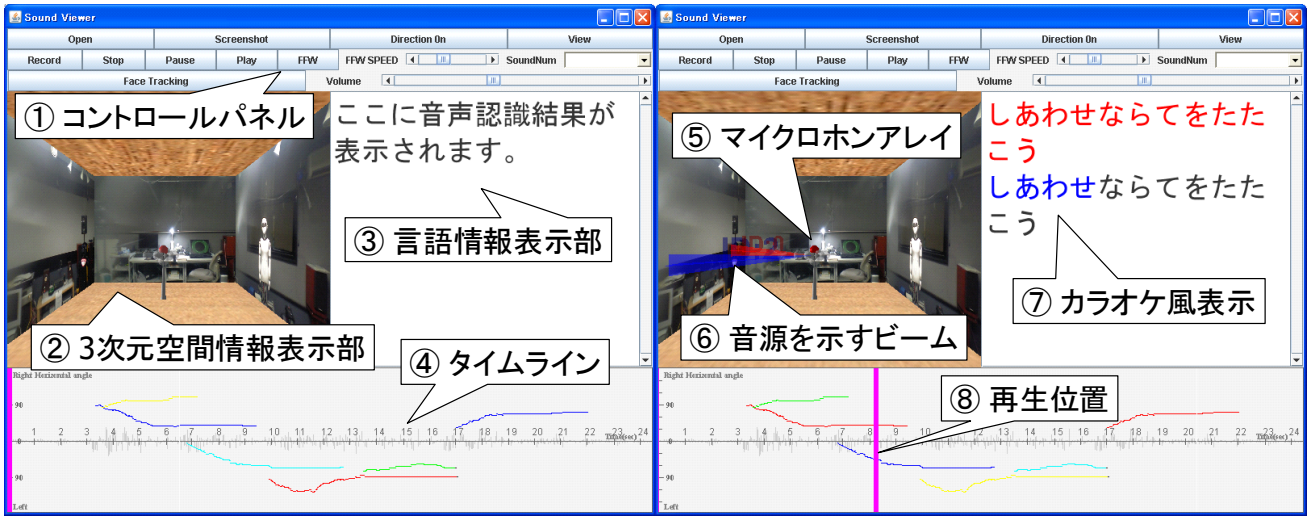


Figure 1.9: GUI of CASA 3D Visualizer

performed in the three-dimensional space information display. At the time of playing a sound, a beam that indicates a sound source direction is displayed with ID. Moreover, size of the arrows corresponds to sound volume. Speech recognition results are displayed in the language information window. At the time of playing a sound, corresponding subtitles are displayed in a karaoke style. Overview information of changes in sound source localization is displayed on the time line and playing position is displayed at the time of playing a sound. What are displayed and acoustic data are corresponded and therefore when clicking on the beam or sound source on the timeline with a mouse, a corresponding separated sound is played. Moreover, the rapid traverse mode is available when playing a function. In this way, we have attempted to improve auditory awareness by visually showing sound information. The following system is produced experimentally as a further application of visualization of the HARK outputs.

1. Alter GUI displays or sound playing in accordance with facial movement of a user [**?**],
2. Display results of Visualizer on Head Mount Display (HMD) [**?**].

The GUI explained above is a mode of an external observer that gives bird's eye views of the 3D sound environment. While on the other hand, the first application is provision of an immersion mode that puts the observer in the middle of the 3D sound environment. Taking analogy of Google Map, these two visualizing methods correspond to the bird's eye view mode and street view mode. In the immersion mode, sound volume rises when the face closes and all sounds are heard when keeping away the face. Moreover, when moving the face from side to side and up and down, the sound from the relevant direction is heard as one of the functions provided. The second application is to display sound source directions in real time by displaying CASA 3D Visualizer in HMD, showing subtitles on the lower part. Subtitles are created not by speech recognition but by the subtitles creation software "iptalk". When a hearing-impaired person gets lectures relying on subtitles, their visual line wanders back and forth between the blackboard and subtitles. This is extremely large load for them and they may in many cases miss important information without noticing that the lecture is going on. Since sound source directions are displayed in this system, auditory awareness for a switch of topic is expected to be strengthened.

1.3.4 Application to telepresence robot

In March 2010, HARK and a system that visualizes sound environment was transplanted to the telepresence robot Texai made by Willow Garage, the US, and we realized a function that enables remote users to display sound source directions

in pictures and listen to the sound from the sound source in a specific direction [4]. The design of acoustic information presentation in a telepresence robot is based on the past experience "Auditory awareness is the key technology" described above. Details on transplant of HARK and development of an related modules of HARK for Texai are separated to the

Figure 1.10: A remote operator interacts with two speakers and one Texai through Texai (center). This demonstration was performed in California (CA), Texai on the left was operated from Indiana (IN) by remote control.

following two processes.

1. Installment microphones in Texai, measurement of impulse response and transplant of HARK,
2. Implementation of HARK interfaces and modules to ROS (Robot Operating System), which Texai control programs runs on.

Figure 1.11 shows the microphones firstly embedded. This robot was placed in the lecture room and dining hall where it was going to be used, and impulse responses were measured every five degrees and performance of source localization was estimated in each place. Next, in order to improve its visual impression and further to reduce the cross-talks between microphones, we discussed to put a head on Texai. Concretely, it was a bamboo-made salad bowl available in general shops. MEMS microphones were embedded on the points where the diameter became same as that of the head firstly set (Figurefig:newTexai). Impulse responses were measured every five degrees and performance of source localization was estimated in the same way as above. The result revealed that there was not much difference in their performance. As for GUI, the overview and filter of Visual Information-seeking matra were implemented. It is shown in Figure 1.13 The arrows from the center in the all-round view, obliquely down Texai itself, are the sound source direction of the speaker. Length of the arrow indicates sound volume. The figure shows three speakers speaking. An image from another camera on Texai is shown in the lower-right corner and that from the remote operator is shown in the lower left corner. The circular arc in the figure indicates the range for filtering. The sound that received from directions within this arc is sent to the remote operator. As shown in Figure 1.14, data are transmitted through the Internet. Since the control command groups for remote operators and GUI are all implemented as RS modules, HARK was transplanted in the way shown Figure 1.15. The brown part in the figure is the HARK system. The modules developed here are available at the website of ROS. These series of works including processing of the head, measurement of impulse response, pretests and design of GUI and control command groups were successfully completed in one week. We presume that the high modularity of HARK and ROS contributed to the improvement of productivity.

1.4 Conclusion

Outlines of HARK 1.0.0 have been reported as above. Sound source localization, sound source separation and recognition of separated sounds, which are the fundamental functions for understanding sound environment, have been realized with

[4] http://www.willowgarage.com/blog/2010/03/25/hark-texai

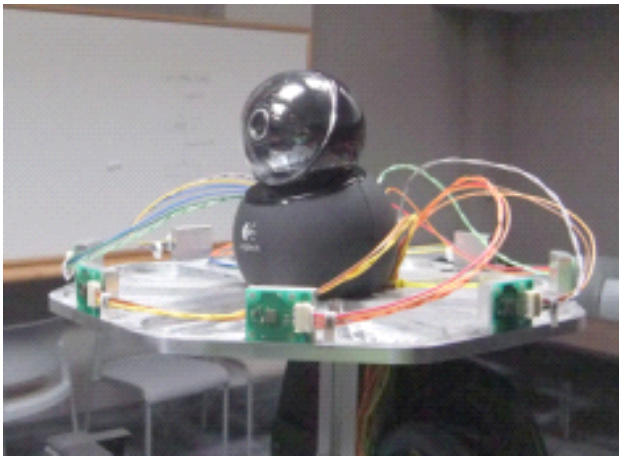


Figure 1.11: Closeup of the first head of Texai: Eight MEMS microphones are embedded on a disk

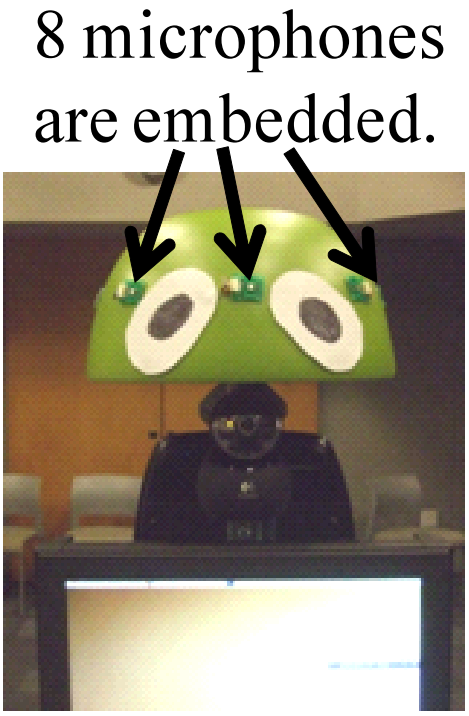


Figure 1.12: Closeup of the head of Texai: Eight MEMS microphones are embedded in the shape of circle

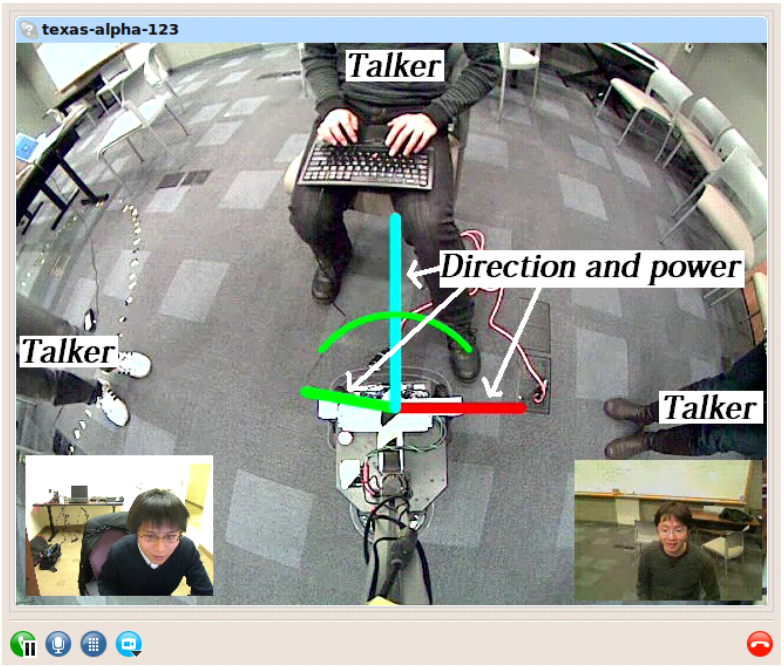


Figure 1.13: Image seen by the remote operator through Texai

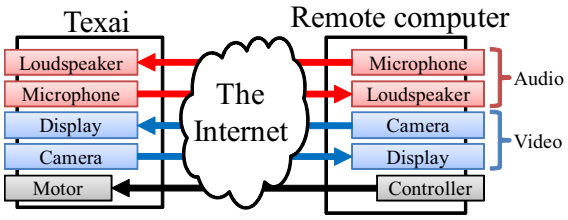


Figure 1.14: Teleoperation of Texai

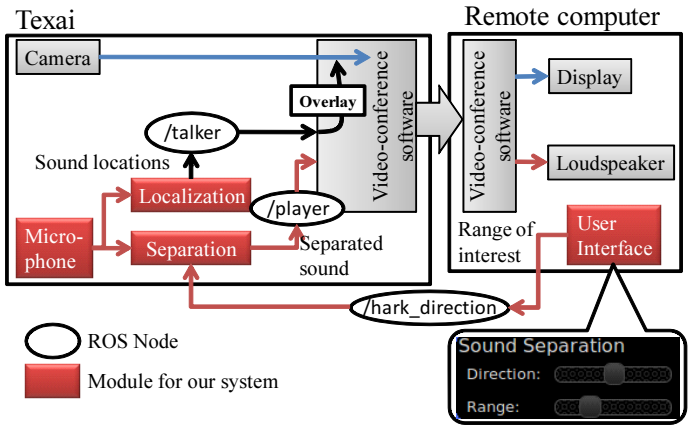


Figure 1.15: Built-in of HARK in Texai

the middleware FlowDesigner as modules and its application to ears of a robot has been reviewed. HARK 1.0.0 provides functions to develop furthermore the researches on robot audition. For example, functions for sound source processing, detailed set-up functions for various parameters of sound source separation and setting data visualization / creation tool are provided. Moreover, support for Windows and interface for OpenRTM are also in progress. Although recognition is realized at some level only by downloading and installing HARK, if performing the tuning for shape and usage environment of an individual robot, even better performance of source localization, sound source separation and recognition of the separated sound would be obtained. For exposing such know-how, it will be important to form HARK community. We hope this paper provides users with an opportunity to exceed the critical mass of robot audition R&D researchers.

Chapter 2

Robot audition and its problem

This chapter describes the robot audition research that had led to development of HARKand problems on it.

2.1 Technology to distinguish sounds is the base for robot audition

According to Astro Boy Encyclopedia (Written by Mitsumasa Oki, Shobunsha), the Astro Boy is equipped with a sound locator that "rises his audibility by 1,000 times by pressing a switch, and enables him to hear distant voices and further to hear ultrasonic waves of 20,000,000Hz" [1]. The sound locator must be a super device that realizes "Cocktail party effect" discovered by Cherry in 1953, which distinguishes speeches selectively. Hearing-impaired persons or elderly adults with weak hearing may gave a question "Isn't a function that distinguishes simultaneous utterances enough, though it's not a super device?". Chronicle of Japan Suiko-ki introduces an anecdote of "Prince Shotoku" who understood utterances from ten speakers at the same time and judged. An old tale of "Ear Hood" with which people can hear the talk of animals, trees and plants fires the imagination of kids. If we could give such a separation function to a robot, mutualism with human beings would advances greatly.

It is needless to say that the most important communication tool in daily life is the speech, including speaking or singing voices. Speech communication includes words acquisition and back channels by non-voice and it has extremely various functions. Indeed, the importance of researches on Automatic Speech Recognition (ASR) has been highly recognized and huge amounts of fund and efforts have been spent over the past 20 years. On the other hand, there have been only few studies on systems that distinguish sounds with microphones attached to robots themselves and recognize speech, except those by Aso et al. The study stance of the authors has been to develop a processing method for sounds of minimal prior knowledge. Therefore, we considered that it would be important to study on sound environment understanding for analyzing sound environment not only through speeches but also through music, environmental sounds, and furthermore the sound mixture of those. From this position, it is understood that the present ASR, which assumes single speech input, can no longer play an important role in robotics.

2.2 Robot audition based on sound environment understanding

We have been studying on sound environment understanding (Computational Auditory Scene Analysis) [**?**], focusing on the importance of dealing with general sounds including music, environmental sounds and the sound mixture of those in addition to speeches. An important task in a study on sound environment understanding is processing of sound mixture. Sound environment understanding is not to avoid the problem of sound mixture with a close-talking microphone set on the lips of a speaker, but to tackle the processing of sound mixture with a straight pitch regarding sound mixture as an input. The three major problems in sound environment understanding are **sound source localization**, **sound source separation** and **automatic speech recognition** in doa recognition. Various technologies have been researched and developed for each of those problems so far. However, all those technologies require some specific conditions to draw their maximum performance. In order to draw the maximum performance in combining these technologies for robot audition, it is essential to systematize interfaces of the individual techniques, required conditions in other words. In order to achieve it, the middleware that can effectively provide a combination with good balance is also important, like "Liebig's Law of the Minimum".

[1] http://www31.ocn.ne.jp/ goodold60net/atmgum3.htm

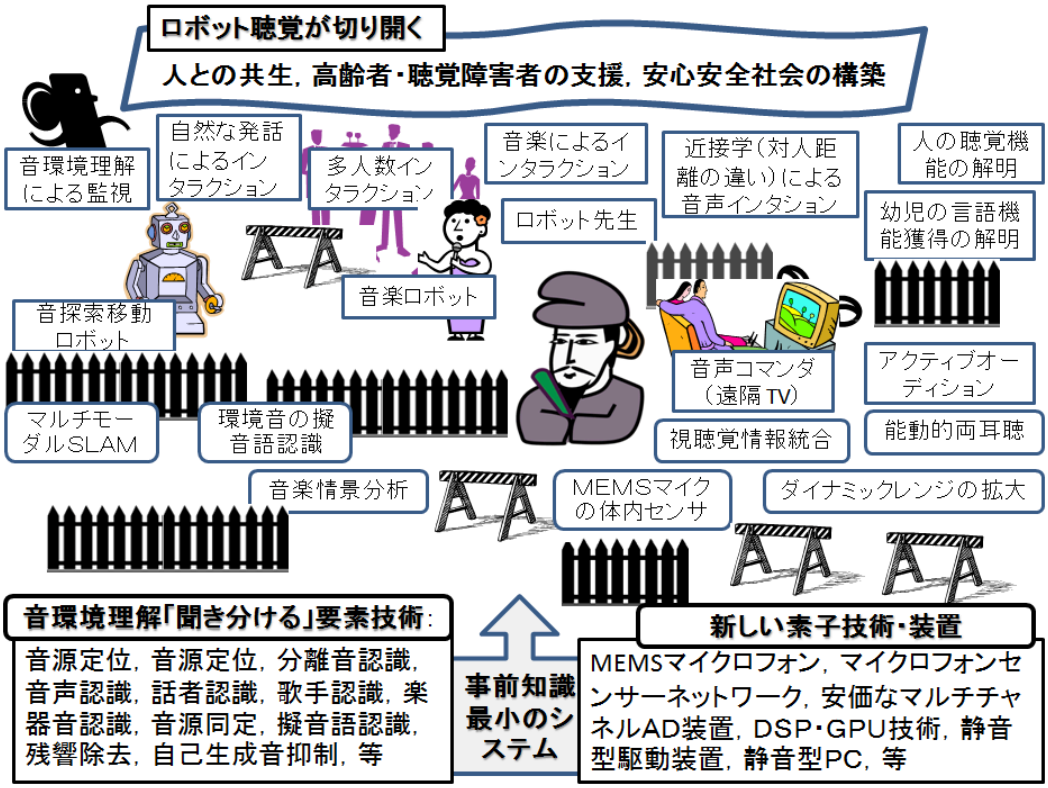


Figure 2.1: Development of robot audition based on sound environment understanding

The robot audition software **HARK** is constructed on the middleware called FlowDesigner and provides a function of sound environment understanding on the premise of using eight microphones. HARK is designed based on the principle of reducing as much prior knowledge as possible and is a system that aims at "OpenCV of acoustic processing". Indeed, a robot that recognizes orders for dishes from three different persons and that judge orally-played rock, paper, scissors have been realized. Although images and video pictures are generally the environmental sensors, they do not accept appearance and disappearance and dark places. Therefore they are not always almighty. It is necessary to remove ambiguity of images and video pictures by sound information and adversely to remove ambiguity of acoustic information by image information. For example, it is extremely difficult to judge if the sound source is in front or back by sound source localization with two microphones.

2.3 Distinguish sounds with two microphones like human being

Human beings and mammals distinguish sounds with two ears. However, it has been experimentally reported that they can distinguish only two sounds under the condition that their heads are fixed. The Jeffress model, which harmonizes by delay-filtering the inputs from both ears, and a model with an interaural cross-correlation function are known well as models for the sound source localizing function of human beings. Nakadai and the authors, getting a clue from stereovision, extracted a harmonic structure in both ears and localizes sound sources by obtaining interaural phase difference and interaural intensity difference for the sound with the same fundamental blade frequency[**?**, **?**]. For acquiring pair of reference points, the epipolar geometry is used in stereovision and harmonic structures are used in our method. In sound source localization from sound mixture by two microphones, localization becomes unstable and wobbles in many cases and it is difficult to distinguish especially the sound sources in front and back. Nakadai has realized stable source localization by audio visual information integration while he has realized a robot named SIG, which turns around when it is called [**?**, **?**, **?**]. "Seeing is believing" is for solving ambiguity resolution in the front and back problem. Kim and Okuno have realized a system that cancels the ambiguity in source localization by moving the head for a robot named SIG2. The ambiguity is cleared no only by rotating the head by 10 degrees in right and left but also by making the robot nod downward by 10 degrees when

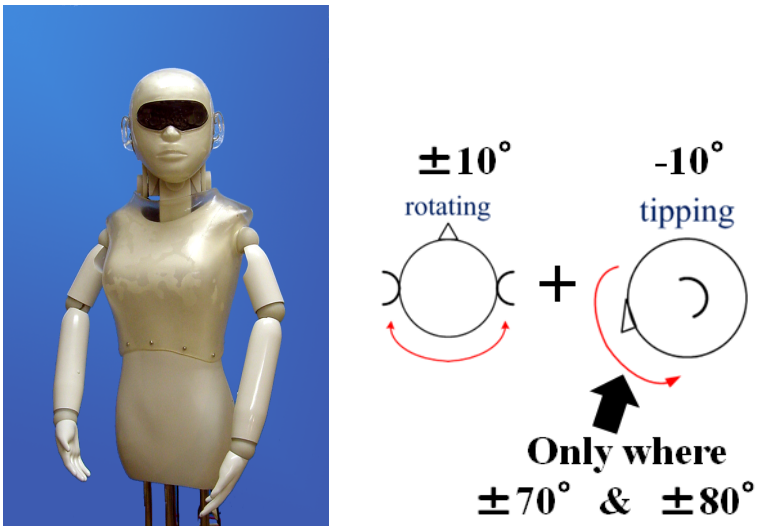


Figure 2.2: Active audition of SIG2: For the surrounding sound, ambiguity of front and back is removed by moving the neck right, left and down.

the sound source is located at 70 - 80 degrees. Indeed, the performance in source identification for the sound in front is 97.6%, which means the performance increase is only 1.1%. The performance in source identification for the sound in back is 75.6%, which means the performance increase is significant as 10% (Figure2.2). This corresponds to the head movement that aimed at solving the front and back problem of humans that Blauert reported in "Spatial Hearing". A method that uses motions for solving such ambiguity is one of the forms of **active audition**. Kumon's group and Nakajima's group work on performance enhancement of source localization by moving the head and auriculars themselves, using various auricles [**?**]. Rabbit's ears usually hang down and listen to a broad range of sounds. The ears stand when they catch abnormal sounds and raise their directivity to listen to the sound from a specific direction. Their basic study on realization method of the active audition is such as above. If this is applied not only to a robot but also to constructive clarification of various animal audition functions, we can expect that it will lead to design development of aural functions of a new robot. In particular, since a stereo input device can be used for binaural hearing without modification, realization of high-performance binaural hearing function will greatly contribute to engineering fields, we presume.

2.4 Function of suppress self-generated sound

In active audition, a sound occurs by creak of a robot itself in some cases as well as the sound of a motor itself occurred by driving the motor. Although the sound that occurs with robot's movement is small, its sound volume is greater than those of external sound sources according to the inverse-square law since the sound source is near a microphone.

Self-generated sound suppression based on model

Nakadai et al. have attempted to suppress this self-generated sound by setting two microphones in the head of the robot SIG. Having simple templates for motor sounds and machine sounds, when the sound that matches the template occurs during operation of a motor, the subbands that are easily destroyed are broken off with heuristics. The reason why they have used this method is that right and left ears are separately processed in the active noise canceller based on the FIR filter and therefore correct interaural phase differences are not obtained, and furthermore that the FIR filter does not have so much effect on burst noise suppression. Further, in SIG2, microphones are embedded in the model ear canal and a motor used for the robot is silent type, which does not require noise suppression processing. As for QRIO made by Sony, one microphone is embedded in its body and self-generated noise is suppressed by using six microphones aiming at outside. Ince et al. have developed a method to predict self-generated noise caused by the movement of the robot itself by joint angle information and reduce it by the spectral subtraction [**?**]. Nakadai et al. incorporated a function to reject motor noise from specific directions into HARK [**?**]. Even et al. have developed a method that estimates directions of the sound radiated from the body surface with three vibration sensors set in the body and adjusts angles of a linear microphone array so that

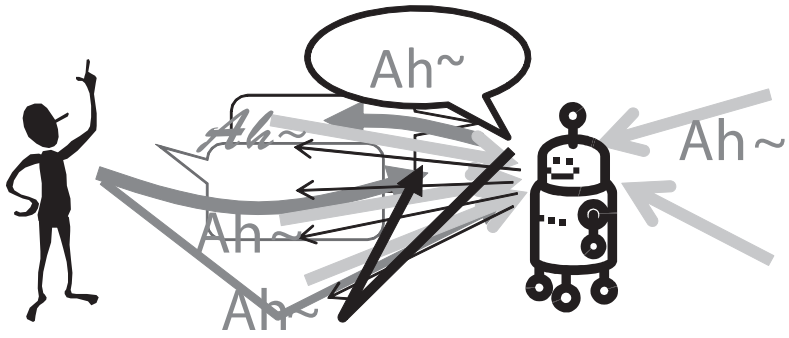


Figure 2.3: Own voice enters the own ears accompanied with reverberant and the further interrupting utterance (barge in) of a partner is heard

speaker's direction do not corresponds to the radiated sound's direction, so as to suppress self-generated sound [**?**]. For interactions between a robot and human, it is essential to develop a "strategy for better hearing" such as moving to the position where the robot can hear the sound best or turning the body in consideration for influences of the self-generated sound and environment on the sound.

Self-generated sound suppression by semi-blind separation

In robot audition, it is possible to perform self-generated sound suppression that utilizes the point that self-generated signals are known to the robot itself. Takeda et al., from a semi-blind separation technique based on ICA, have developed a function of self-generated sound suppression that estimates reverberation with self-generated utterances as already-known, suppresses the input sound mixture in self-generated utterances and extracts utterances of a partner in the situation shown in Figure2.3 [**?**].

Barge-in-capable utterance recognition and a music robot (described later) have been developed as a prototype of the application of this technique. Barge-in-capable utterance is a function that allows humans to utter freely even during the utterance of the robot. When a user barges in and the user utters "that", "the second one" or "Atom" while a robot provides information enumerating items, the robot can judge which item has been designated from the utterance content and timing with higher accuracy, based on this technology. For symbosis of human beings and robots, it is essential to have mixed-initiative interactions, which allow them to speak freely at any time, not alternating speaking. This self-generated sound suppression method realizes such a function. In the semi-blind separation technique, a self-generated sound enters to ears though it is deleted when separated and therefore it cannot be used for higher-order processing. According to "Brain that speaks and hears words" written by Honjo, in adult, the own voice enters to the primary auditory cortex of a temporal lobe though it is not transmitted to the associative auditory cortex of a cerebral cortex and such voice is ignored. The above self-generated sound suppression by the semi-blind separation can be understood as an engineering case of the processing that is terminated at a primary auditory cortex.

2.5 Elimination of ambiguity by integration of audio visual information

Robot audition is **not an elemental technology but is a process consisting of plural systems**. There are a number of elemental technologies for component parts and in addition, component parts vary considerably in their performance and therefore all those need to function with a good interaction between each other in processes. Moreover, the better interaction enables better functioning of the processes. Since the ambiguity cannot be eliminated only by acoustic treatment, integration of audiovisual information is the important key for the interaction. There are various levels of information integration, such as temporal, spatial, intermedia and intersystem integration and furthermore hierarchical information integration is required between those levels and within each level. Nakadai et al. have proposed the following audiovisual information integration. At the undermost level, a speaker is detected from audio signals and lap movement. At the levels above, phoneme recognition and viseme recognition are integrated. At even higher levels, a speaker position and 3D position of the face are integrated. At the topmost level, speaker identification / verification and face identification / verification are integrated. Of course, not only information integration at the same level but also interactions such as bottom-up or

top-down processing is possible. Generally sound mixture processing is an ill-posed problem. In order to obtain more complete solutions, it would be necessary to have some assumptions such as the assumption of sparseness. Sparseness in a time domain, sparseness in a frequency domain, sparseness in a 3D space and furthermore sparseness in a feature space are possible. Note that the success or failure of such information integration depends not only on design of the sparseness but also on performance of individual elemental technologies.

2.6 Killer application that robot audition opens up

Even though satisfactory robot audition functions are obtained, they are integrations of individual signal processing modules and we cannot see what applications will be possible. Indeed, the position of speech recognition is extremely low in the IT industry. Considering such present conditions, in order to find out really essential applications, we need to construct usable systems first and accumulate experience.

Interaction by proxemics

Proxemics based on inter-personal distance is known well as a basic principle of interactions. Quality of interactions are different in each of intimate distance (- 0.5 m), personal distance (0.5 m - 1.2 m), social distance (1.2 m - 3.6 m), and public distance (3.6 m -). The problem on the robot audition in terms of proxemics is the expansion of a dynamic range of a microphone. In an interaction for plural speakers, if each speaker talks with the same sound volume, the voice of a distant speaker reduces in accordance with the inverse-square law. The conventional 16-bit inputs would not be sufficient and it would be essential to employ 24-bit inputs. It would be difficult to use 24- bit inputs for the entire system from the viewpoint of consistency with computational resources and existing software. Arai et al. have proposed a method of downsampling to 16 bit, which has less information deficits [**?**]. Moreover, it will be necessary to accept new systems such as multichannel A/D systems or MEMS microphones for cellular phones.

Music robot

Since humans move their bodies naturally when they listen to music and their interaction becomes smooth, expectation to the music interaction is large. In order to make a robot deal with music, a function to “distinguish sounds” is the key. The followings are the flow of the music robot processing developed as a test bed.

1. Suppress self-generated sound or separate it from the input sound (sound mixture)
2. Recognize tempo from the beat tracking of a separated sound and estimate the next tempo,
3. Carry out motions (singing and moving) with the tempo.

The robot begins to step with the tempo immediately as the music starts from a speaker and stops stepping as the music stops. The robot uses a function to suppress self-generated sounds so as to separate the own singing voice from the input sound mixture including the influence of reverberant. Errors cannot be avoided in beat tracking and tempo estimation. It is important for a music robot to recover quickly from wandering at the time of the score tracking caused by tempo estimation errors and join back to an ensemble or a chorus smoothly, which is an essential function for interactions with humans.

Audio visual integration type SLAM

Sasaki / Kagami (Advanced Industrial Science and Technology) et al. have developed a mobile robot equipped with a 32-channel microphone array and are working on research and development for understanding indoor sound environment. It is an acoustic version of SLAM (Simultaneous Localization And Mapping), which performs localization and map creation at the same time while following several landmarks with a map given beforehand [**?**]. Although the conventional SLAM uses image sensors, laser range sensors and ultrasonic sensors, microphones, acoustic signals from the audio band in other words, have not been used. The study of Sasaki et al. aims at incorporating the acoustic signals that were not treated in the conventional SLAM into SLAM, and it is a pioneering study. When a sound is heard though the sound source is not seen, this system enables SLAM or source searching, which has opened up the way to true scene analyses and environmental analyses, we presume.

2.7 Conclusion

The authors have described our idea for studies on robot audition based on the creation of "**the robot that hears a sound with its own ears**" and expectation for future development. Since the robot audition study started from almost nothing, we have attempted to promote not only our own study but also the studies concerned We garnered the cooperation of the academia of Asano (AIST), Kobayashi (Waseda University) and Saruwatari et al.(NAIST), enterprises developing robot audition such as NEC, Hitachi, Toshiba and HRI-JP, and furthermore of overseas research institutes such as University of Sherbrooke in Canada, KIST in Korea, LAAS in France and HRI-EU in Germany and have organized the sessions for robot audition in IEEE/RSJ IROS for the past six years and the special sessions in academic lectures of the Robotics Society of Japan for the past five years. Furthermore, in 2009, we organized the robot audition special session in ICASSP-2009, the International Conference on Acoustics, Speech and Signal Processing organized by IEEE Signal Processing Society. Raising such a study community, researchers have been increasing slowly in the world and above all the high level of Japan's robot audition studies is outstanding. We expect that Prince Shotoku robot will support hearing-impaired persons and elderly people and furthermore contribute to the construction of a peaceful society through more and more future development of our study.

Learned to listen to what others say at sixty ("The Analects of Confucius / Government") It is said that a person learns to listen to what others say at sixty. However, the sensitivity of auditory organs for high frequencies drops when they overwork or age and the person come not to be able to hear conversations so he/she cannot depend on ears even if they wish to.

Chapter 3

Instructions for First-Time HARK Users

This chapter describes how to acquire the software and to install it, and further describes the basic operations for first-time HARK users.

3.1 Acquisition of the Software

Access the Web site of HARK, download the software and install it. Two ways are available to acquire the software: download a package file and install it, or download a source code and install it. The source code is available at the following URL:

http://winnie.kuis.kyoto-u.ac.jp/HARK/.

It is strongly recommended for a user who installs HARK for the first time to download the package file and install it, not the source code. Installation with the source code is for advanced users and therefore this document does not deal with it.

3.2 Installation of the Software

Instructions in this section show operation examples for all the processes till the installation completion. > on the line head indicates a command prompt. The bold and the italic letters in the operation example indicate inputs from a user and a message from the system, respectively. In the following example, > in the first line indicates a command prompt.

> **echo Hello World!**
> *Hello World!*

Note that the prompt is displayed in different ways (e.g. %, $) according to operating environments. The bold letters that come after the prompt in the first line are the letters entered by the user. In the above example, the seventeen letters of echo " Hello World! " are the letters entered by the user. Press the enter key at the end of line. The italic letters in the second line are the output from the system, which are displayed as pressing the enter key at the end of the first line. Some parts of inputs from the user and message from the system contain version and release numbers. Those contained in the actual messages are of the actual software that a user installs. Moreover, messages from the system shown in the operation examples differ depending on the presence of libraries, which are available as options. Even if the message texts are not completely identical to those shown in this manual, the user may proceed with the operation unless an error message appears.

3.2.1 Installation from the package

Ubuntu 8.04, 8.10, 9.04, 9.10, and 10.04 users may install from the package. Add a distribution server of the package to the setup file before installing the package. Use the root authority for setting the distribution server and create a file with the following content, with the file name " harklist " in the directory under /etc/apt/sources.list.d.

```
deb http://winnie.kuis.kyoto-u.ac.jp/hark jaunty/
```

```
> sudo echo deb http://winnie.kuis.kyoto-u.ac.jp/hark jaunty/ > /etc/apt/sources.list.d/harklist
```

Next, install the package. The followings are a detail example when apt-get is used.

```
> sudo apt-get update
> sudo apt-get install flowdesignerhri harkfd harktool libharkio julius-4.1.5-hark
```

The user may use the package manager for the installation. The root authority is required for an execution of Synaptic package manager. Designate flowdesignerhri in the search window, select flowdesignerhri package and install by right-clicking "Application". Install harkfd, harktool, libharkio and julius-4.1.5-hark in the same way.

3.3 FlowDesigner

FlowDesigner is one of the open source middleware. Presently, the only middleware the HARK is based on is FlowDesigner. Future implementations on other middleware are also planned. The feature of FlowDesigner that a system can be constructed though GUI. Since the graphical programming styles used in simulink and LabView are employed, even users with little programming experience can easily create programs. The programming in FlowDesigner is achieved by laying out and connecting modules, and setting property values of the modules. An overview of FlowDesigner is shown in Figure3.1 FlowDesigner is started by the following command.

```
> flowdesigner
```

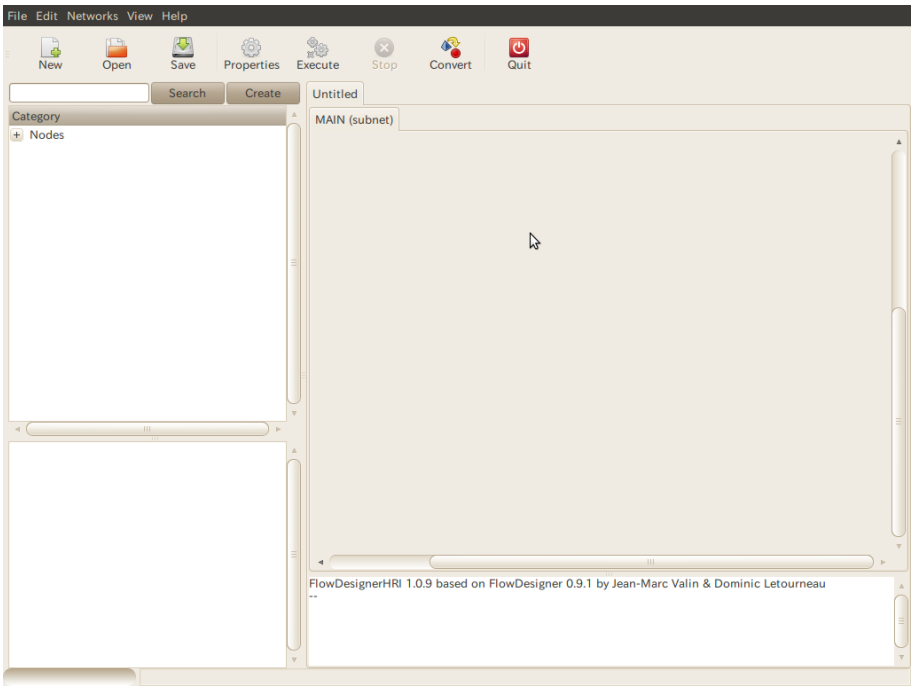


Figure 3.1: Overview of FlowDesigner

3.3.1 Basic operation of FlowDesigner

The module is processing units that process data. First, an example of a module is shown in Figure 3.2. The green part enclosed with a rectangle is the module. A module name is displayed in the center of this module. In this example, PostFilteris displayed. The name nodePostfilter1 displayed below the module is an instance name of the module. The same modules plurally applied are named in the order of generation of the modules, such as nodePostfilter2, nodePostfilter3. It

is useful to distinguish each module. Each module may have plural terminals for inputs and outputs of data processed. The black points on the right and left of the module are the terminals. Those on the left are input terminals and on the right are output terminals. Terminals are named or unnamed depending on the cases. All terminals are named in this example. A module has at least one output terminal though there are no input terminals in some cases. For example, the **constant** module included in FlowDesigner by default does not have inputs but one output. Data processing flows are defined by

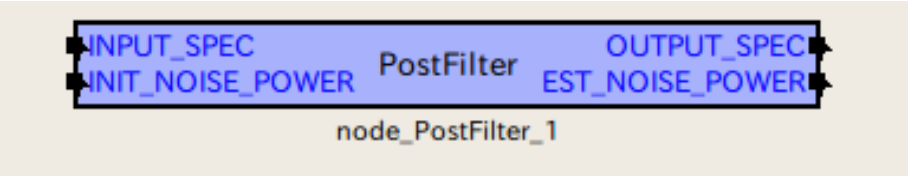


Figure 3.2: Overview of module

positioning plural modules and connecting the terminals of the modules. A series of modules connected is called network. Creating a network itself is programming. Networks can be save by the file save menu. Once a network is saved, it can be loaded by FlowDesigner anytime. When opening a saved file by a text editor, an XML-like description is seen. The user may become able to edit such a file directly by a text editor once they get used to. Since some modules have properties, the user can designate processing of the modules in detail by setting their property values. For setting property values, move the mouse cursor on a module and left-double click on it. The setting DIALOG window opens and the user may input property values by keyboard or select values. Since there are data types for property values, the user is required to set data types and values as a pair. For details of the data types, see the chapter of Data Type. Select data types from the pull-down menu. For values, the user may enter them with the keyboard or select values from the pull-down menu depending on the case.

3.3.2 Module operation / terminal connection / property setting

The following describes details of module operations, terminal connections and property setting.

- **Creation of module**
  Right-clicking on the network construction tab, a category list of a module is displayed. Move the mouse cursor on the object category name, a module is displayed. Moving the mouse cursor on the objective module name and left-clicking on, a module is created. When cancel the module layout from the status that category lists or module lists are displayed, click the background part of the network construction tab (either of right or left).

- **Relocation of module**
  To relocate a module positioned once to an easy-to-see position, move the mouse cursor onto the module and drag with the right mouse button.

- **Deleition of module**
  Moving the mouse cursor onto the module and clicking on the right mouse button, an operation menu for the module is displayed. The module is deleted by moving the mouse cursor onto ”Delete” and right-clicking on it.

- **Copy of module**
  When clicking on the left mouse button on a module with pressing the Shift key, the module is copied. Since attribute values are also copied as well as the module itself, this function is useful in constructing a network that executes the same processes in parallel / concurrently.

- **Connection of module**
  Modules are connected by connecting the terminals with arrows. Pressing down the left button on a terminal on the connection source, dragging and releasing it on the connection point, the terminals are connected with an arrow and the module connection is completed. The user can connect input and output terminals though they cannot be connected between input terminals and between output terminals. Plural connections can be drawn from one output terminal. A processing result of a module is sent to plural modules. On the other hand, plural connections cannot be established in one input terminal. Even a connection between input and output terminals cannot be established in some cases. Connections cannot be established when a data type that the terminal can process is not available and such a connection is indicated with a red arrow. In some cases, although the arrow is black, an error may occur in a

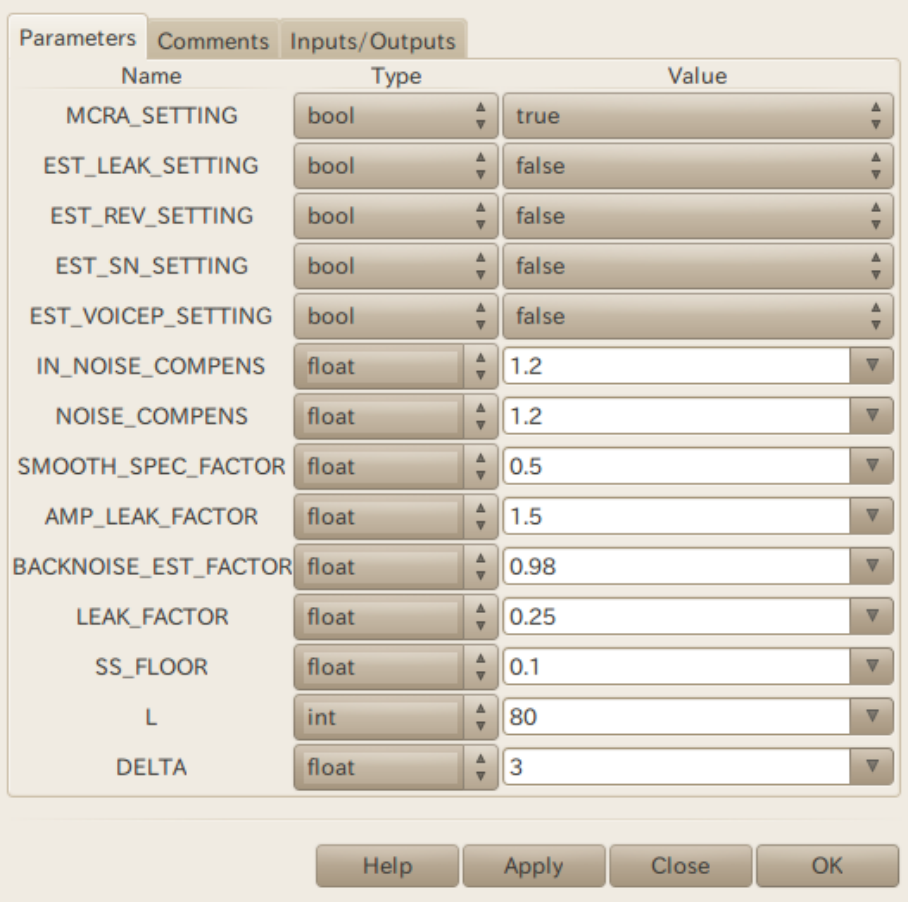


Figure 3.3: Overview of property

program execution when fail to judge the data type, which the terminal can process. Since the automatic data type judgment function is only a supplementary tool, it is recommended for the user to confirm appropriate data types in the module reference for accurate connections.

- **Disconnection of module**
  A connection is cut by right-clicking on the starting point or end point of the arrow of the connection that the user wish to disconnect, with pressing Shift key.

- **Relocation of module connection line**
  When moving a module, the module moves with the connection maintained. When positioning a large number of modules, modules and their connection lines overlap, which makes them difficult-to-see network. Therefore, it is recommended to curve the lines so that network can be easily seen with fewer overlapping. A connection line curves on the point where moving the mouse cursor on and drags with the right button. Dragging with the left button on the curved point, the point is moved.

- **Property value setting**
  Moving the mouse cursor on a module and right-clicking on it, an operation menu for the module is displayed. Moving the mouse cursor to ”Properties” and right-clicking on it, a setting dialog of property values is opened. Enter desired values and data types referring to the module reference. The setting is reflected by clicking on ”Apply”. The setting is reflected and the dialog is closed by clicking on ”OK”.

Description of the basic module operation is as above. Next, we describe the basic operation necessary for a network construction described, building a sound source localizing network.

3.3.3 Creation of network for beginner

This section show example of building a sound source localizing network for users who create such a network for the first time. As well as a main network tab, the subnetwork tab for Iteration is needed. First, build a sound source waveform reading part on the main tab followed by creating a subnetwork and building a localizing part and result display part in the sub network. First, modular arrangement of the sound source waveform reading part, connections and property setting are described. Starting FlowDesigner, the MAIN (subnet) tab appears on the right side in the window. Proceed with a work in this tab first. Arrange the Constant module and the InputStream module as shown in Figure3.4. Right-clicking on the ground part in the tab, a pull-down menu appears. Moving the mouse cursor on New Node inside, the module list registered in FlowDesigner is displayed. Here, select General category. Holding it, move the mouse cursor onto General. Here, the pull-down menu is spread by one step and the module list registered in General is displayed. The constant module is included in this. Holding it, moving the mouse cursor onto Constant, the Constant module is arranged in the MAIN (subnet) tab. In the following descriptions, an operation to select a module out of a pull-down menu is indicated as "New Node → General → Constant". In the same way, arrange the InputStream module as New Node → IO → InputStream. Next, connect the Constant module and an InputStream module as shown in Figure3.5. Next, set properties of the module.

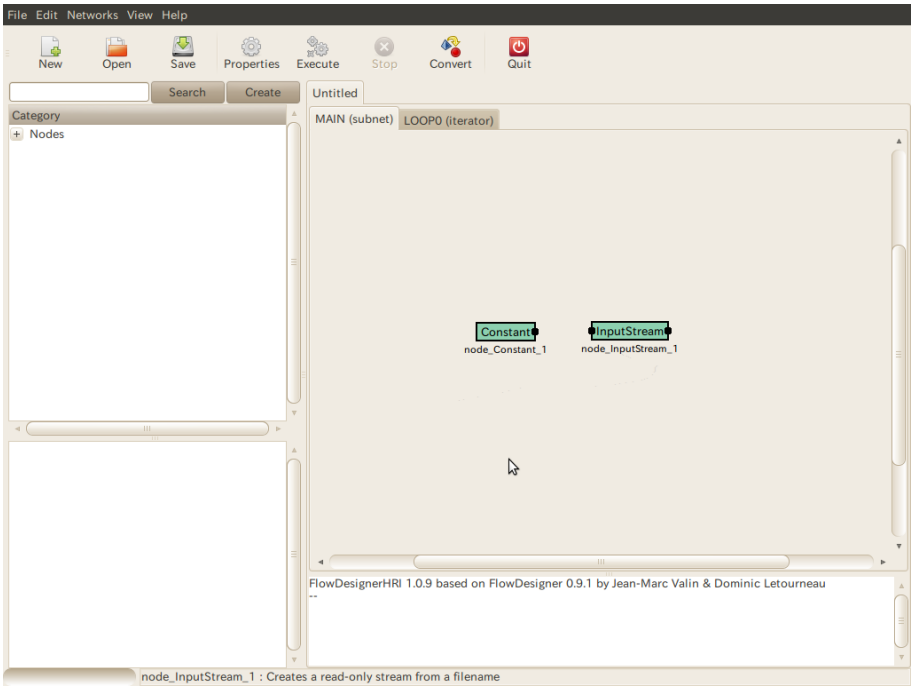


Figure 3.4: Arrangement of two modules

Here, set only the Constant module. The InputStream module does not have settable property values. Right-clicking on the Constant module, a pull-down menu appears. Moving the mouse cursor onto Properties and left-clicking on it, a property setting dialog opens. There are Parameters, Comments and Inputs/Outputs tabs in the dialog. The Parameters tab is opened by default. The Comments tab and Inputs/Outputs tab are not used. The Parameters tab contains three items, which are NAME, Type and Value. NAME is for a property name, Type indicates a data type of values to be set, Value indicates the value to be designated for attributes. The Constant module has only one settable property, which has the attribute name VALUE. Since a file name is to be indicated by this module, give tutorial1.wav for the file name to the property value. The data type is `string`. Finally, the setting is reflected by pressing "Apply" in the "Properties" window. The setting content is cancelled by pressing "Close". The same processing is performed by pressing "OK", and pressing "Application" followed by "Close". Since the user cannot operate other windows of FlowDesigner while the "Properties" window is displayed, it is necessary to finish the property setting before backing such operations. Here, the property setting is completed. A network of a sound source waveform reading part is completed now. Next, we describe subnetwork creation. A subnetwork is created by selecting "add Iterator" in "Networks " included in the FlowDesigner's window menu. Selecting "add Iterator", a dialog window starts requesting the user to enter a name for the Iterator tab added. If using the default name "LOOP0", press "OK". The subnetwork creation is cancelled by pressing the cancel button. Clicking on "OK" for LOOP0, a tab named

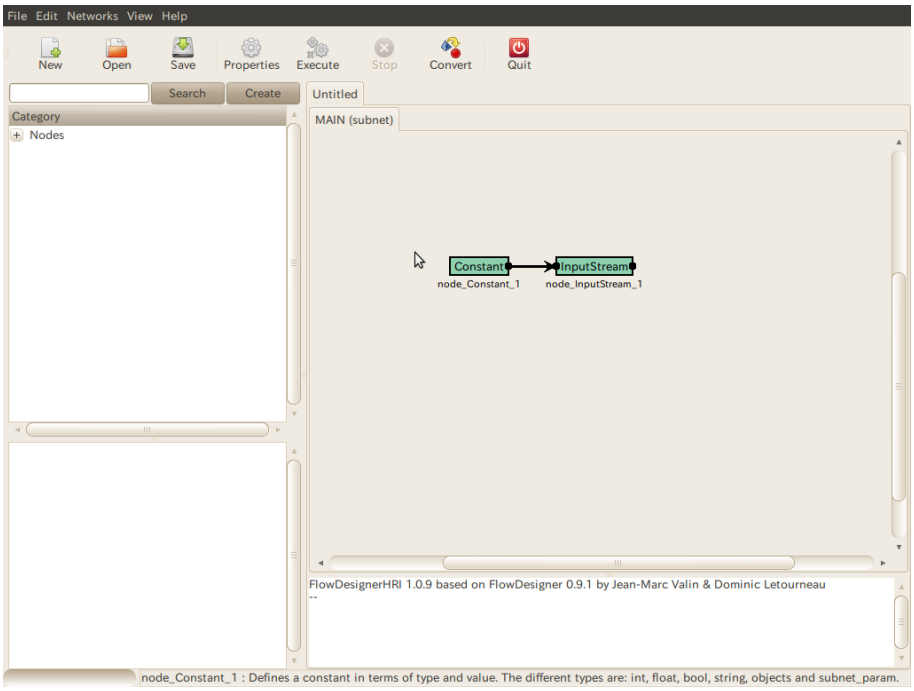


Figure 3.5: Connections of two modules

”LOOP0” appears next to the MAIN (subnet) tab and the LOOP0 tab becomes active. Build a localizing part and result display part in this tab. Sound source localizing part is built with modules for file reading, FFT, sound source localizing by the MUSIC method, sound source tracking and localization result displaying. Figure 3.6 shows modules arranged by the following procedures.
New Node → HARK → AudioIO → AudioStreamFromMic
New Node → HARK → MISC → MultiFFT
New Node → HARK → Localization → LocalizeMUSIC
New Node → HARK → Localization → SourceTracker
New Node → HARK → Localization → DisplayLocalization
Connect these modules as shown in Figure 3.7. Completing the connection, set module properties. Make setting assuming the audio file to be processed is 16,000 [Hz]. In the AudioStreamFromWavemodule, there are three setting points for property values, which are LENGTH, ADVANCE and USEWAIT. LENGTH and ADVANCE indicate the analysis frame length and frame shift length of speech by unit samples, respectively. A data type is `int`. Default values 512 and 160 are set, which correspond to 32 msec and 10 msec, respectively. The user does not need to change these values. The default value `false`is set in USEWAIT of the data type `bool`. The user does not need to change these values. In the MultiFFTmodule, there are three setting points, which are LENGTH, WINDOW and WINDOWLENGTH. Analysis frame length, window function type and window length of the sound are expressed in unit samples. The default values 512, CONJ and 512 are set. LENGTH and WINDOWLENGTH correspond to 32 msec and its data type is `int`. CONJ designates the user to use the CONJ window as an analysis window. Its data type is `string`. The user does not need to change these values. LocalizeMUSICmodules have twelve property setting points, which are NBCHANNELS, LENGTH, SAMPLINGRATE, AMATRIX, ELEVATION, PERIOD, NUMSOURCE, MINDEG, MAXDEG, LOWERBOUNDFREQUENCY and DEBUG UPPERBOUNDFREQUENCY.

- **MBCHANNELS**
  For NBCHANNELS, the user sets the number of microphones or channels of an audio file. The data type is `int`. In this example, the set value is 8. Analysis frame length of the voice is 512 samples, which is the default value. It must be same as LENGTH of MultiFFT.

- **SAMPLINGRATE**
  For SAMPLINGRATE, designate 16,000 [Hz] as assumed above. Its data type is `int`.

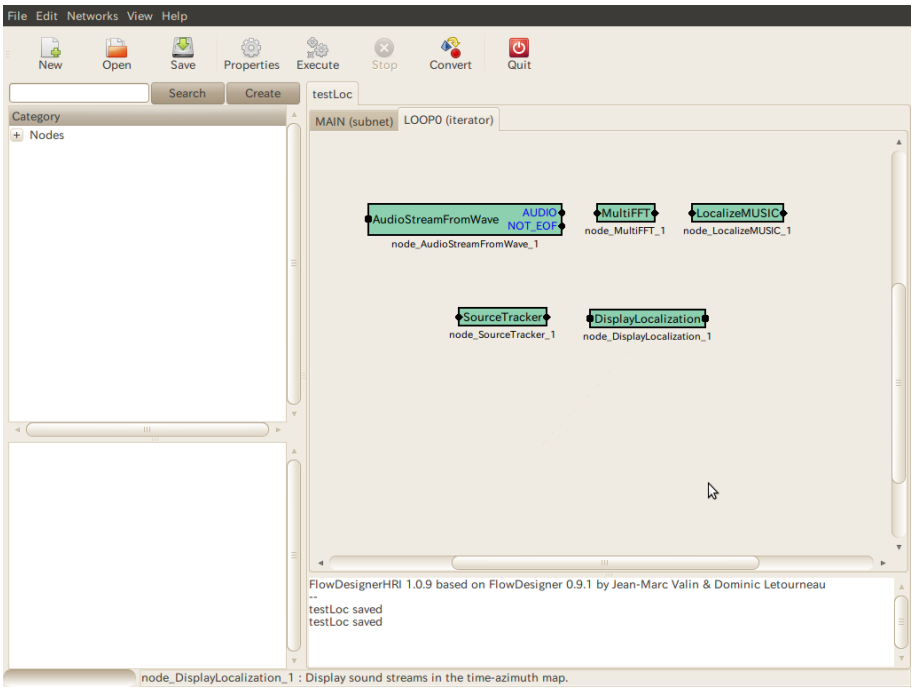


Figure 3.6: Arrangement of module for sound source localizing part

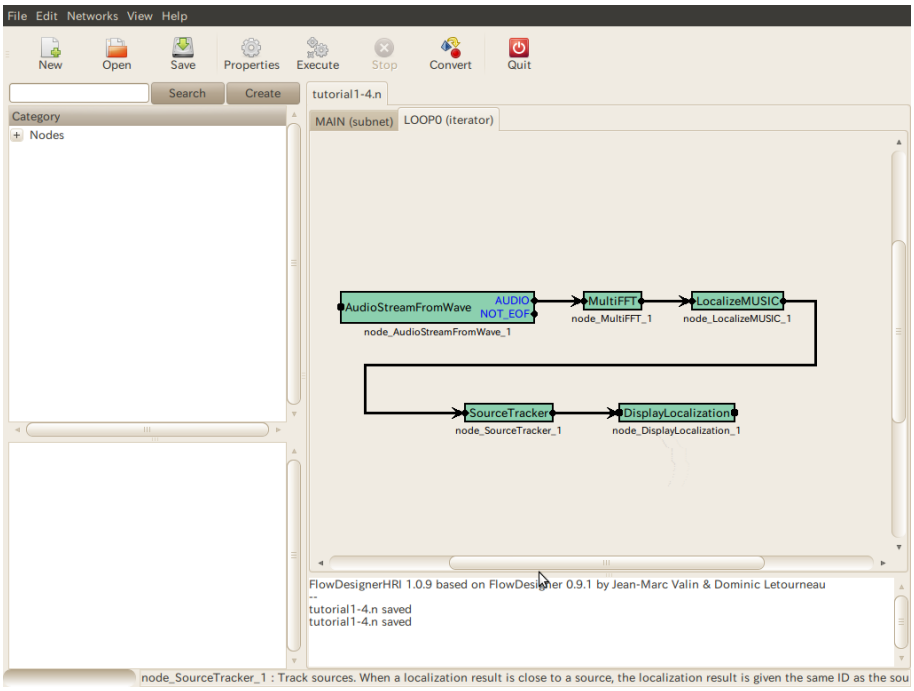


Figure 3.7: Connection of module for sound source localizing part

- **AMATRIX**
  AMATRIX, the user sets a file name of a transfer function. The data type is `string`. For how to create a transfer function file, see description of harktool. Here, sampletf.dat is set for the file name.
- **ELEVATION**

ELEVATION indicates elevation angles of a sound source. Its data type is `float`. The angle 16.7 [deg] is set as a default value. The angle 16.7 [deg] is the value based on the assumption that a sound source is 1[m] away from a microphone and a relative position between a microphone and a sound source in a height direction is 0.3 [m] in an elevation angle direction.

- **PERIOD**
  PERIOD is analysis frame frequency used for computation of correlation functions. Its data type is `int`. The default value is 50. The user does not need to change these values.

- **NUMSOURCE**
  For NUMSOURCE, the user sets the number of sound sources to be assumed. Its data type is `int`. Its default value is 2. The user does not need to change these values.

- **MINDEG, MAXDEG**
  For MINDEG and MAXDEG, the user designates a measurement range for source localization processing. Its data type is `int`. The default values are -180 and 180, intended for source localization processing for all azimuth directions. The user does not need to change these values.

- **LOWERBOUNDFREQUENCY, UPPERBOUNDFREQUENCY**
  For LOWERBOUNDFREQUENCY and UPPERBOUNDFREQUENCY, the user restricts frequency bands used for localization processing. Its data type is `int`. The default values are 500 and 2800, which are the bands where the sound energy is large. The user does not need to change these values.

- **DEBUG**
  For DEBUG, the user switches display / non-display of debug-like information. `bool`type value is used here and `false`, which means non-display, is set as a default value. The user does not need to change these values.

For the SourceTrackermodule, there are four setting points for property values of THRESH, PAUSELENGTH, MINSOURCEINTERVAL and DEBUG.

- **THRESH**
  A sound source is detected while the power of the sound source is larger than THRESH. Its data type is `int`. No default value is set. In this example, 25 is set. For how to tune values, see the module reference.

- **PAUSELENGTH**
  In a pose section during utterance, the power of a sound source temporarily fall below THRESH in some cases. If the sound source detections were cut off every time it happened, it would be difficult to execute processing after detections. If such a cut off could be assumed as a pose, it would be convenient to keep tracking sound sources even when the power of a sound source temporarily fall below THRESH.Its data type is `float`. The default value is 800. The user does not need to change these values. The longer the set value of PAUSELENGTH, the more easily the source detection continues though the utterance finish time is extended by PAUSELENGTH. The problem is that an utterance overlaps with the start of the next utterance and the two utterances are seen as one utterance. If an utterance is too short, the utterance that should be assumed as one utterance is detected temporally uncontinuously.

- **MINSOURCEINTERVAL**
  When tracking a sound source, it is rare that the same sound source continues being detected from the same direction. Since a detection direction usually sways, the sound source should be assumed as identical when such swaying is within in a certain width. The user designates the swaying width to be permitted by angle. The data type is `float`. The default value is 20. The user does not need to change these values.

- **DEBUG**
  For debugs

For the DisplayLocalizationmodule, there is one property setting point for the value of LOGISPROVIDED. When a user wishes to have a log of source location, set `true`. Its data type is `bool`. The default value is `false`. The user does not need to change these values. Property setting for source localization is completed here. Next, set Iterator. Iterator is used like a subroutine from MAIN (subnet). The user needs to set the followings for a relation with MAIN (subnet). What are required for Iterator are INPUT, OUTPUT and CONDITION in a network. INPUT and OUTPUT are input and output parts of data,

respectively. The user may easily understand them by corresponding them to those of a subroutine. Iterator has a structure that repeatedly performs processing while it is a subroutine. Therefore, the user needs to describe CONDITION, which is a halt condition for repetition processing. Set it as shown in Figure 3.8. For setting INPUT, when left-clicking on the input terminal with pressing Shift, a dialog appears and the user may change names on it. The setting is completed by pressing ”OK” and ”INPUT” is displayed on the input part in red letters. For setting OUTPUT, when left-clicking on the input terminal with pressing Shift, a dialog appears and the user may change names on it. The setting is completed by pressing ”OK” and ”OUTPUT” is displayed on the input part in blue letters. CONDITION is set by left-clicking on the output terminal with pressing Control. ”CONDITION” is displayed in purple. When setting INPUT, OUTPUT or CONDITION in an unintended location, the user may cancel the setting by left-clicking on those letters with pressing Shift. Finally, integrate

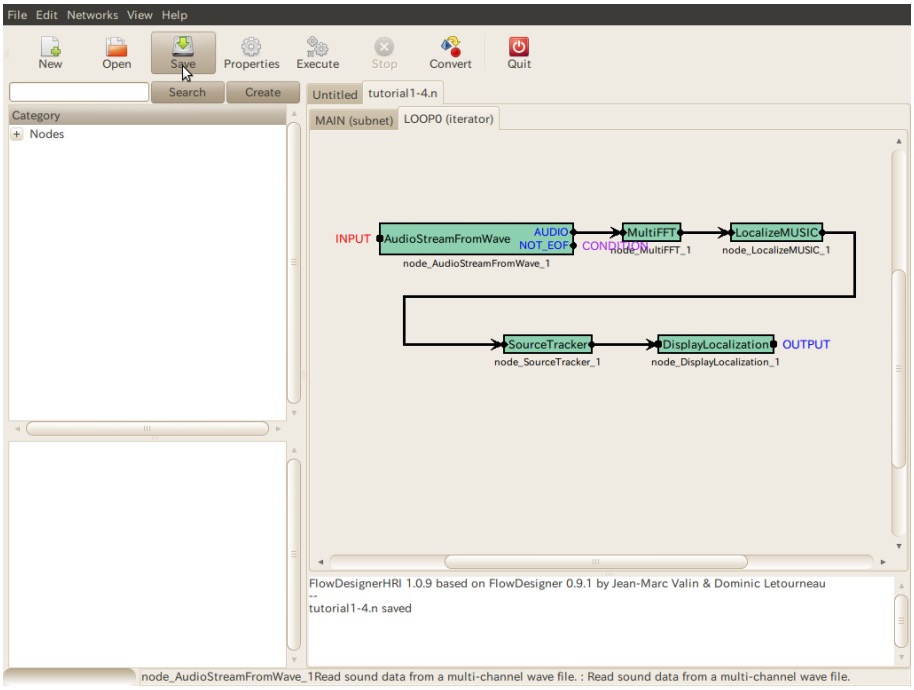


Figure 3.8: Setting of INPUT, OUTPUT and CONDITION on sound source localization module

Iterator and Subnet. Back to the network display of MAIN (subnet) by pressing the MAIN (subnet) tub. Left-clicking on the ground and moving the mouse cursor onto New Node, a new item named subnet appears on the module category name. Moving the mouse cursor onto the subnet category, the item named LOOP0 appears. Left-clicking on LOOP0, same as when arranging a module, a module named LOOP0, which has one input and one output, is arranged. In other words, the subnetwork described in LOOP0 (Iterator) is made a virtual module by the above operation. The source localization network is completed as connecting InputStream and LOOP0 and setting OUTPUT on LOOP0 as shown in Figure 3.9.

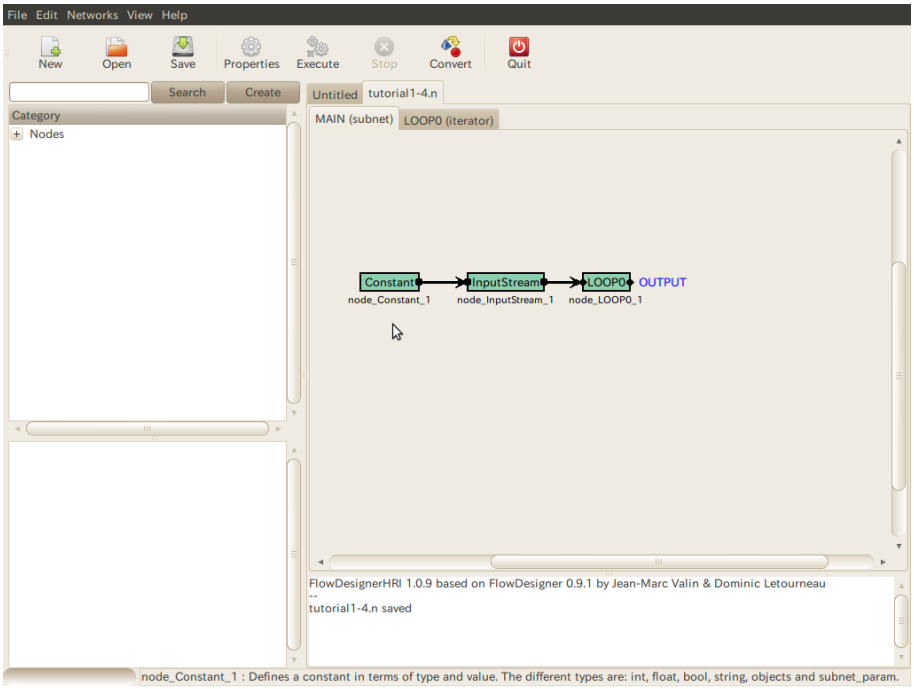


Figure 3.9: Source localization network completed

Chapter 4

Data Types

This chapter describes a data type used in module groups of FlowDesignerand HARK. Users need to be aware of data types in HARKin the following two cases.

- Property setting of module
- Connection between modules (inter-module communication)

Data types used by property setting of modules

The data types that can be designated presently as a property of a module are five types as follows.

Type	Meaning	Data type level
int	Integral type	Primitive type
float	Single-precision floating point type	Primitive type
string	String type	Primitive type
bool	Logical type	Primitive type
Object	Object type	FlowDesigner-specific type
subnet_param	Subnet parameter type	FlowDesigner-specific type

Since the basic data type of C++ is used for int, float, stringand bool,their specifications conform to C++. Data types peculiar to FlowDesignerare used for Objectand subnet_param. Objectis a generic name of a data type that is defined as a class succeeding to Objecttype in FlowDesigner. In HARK, Objectthat can be designated as a property is Vectoror Matrix. Since all types except the primitive type succeed to Objectas described later, the classes with description in a text format can be designated. Even a primitive type can be designated as Objectby utilizing an object that succeeds to Object(e.g. <Int >, for example). subnet_paramis a special data type that is used for sharing one parameter with a label between plural modules. **Data type used for connections between modules** Connection (inter-module communication) between modules is achieved by connecting with lines the terminals of subject modules (indicated as a black point on the right and left of a module) on GUI of FlowDesigner. Data types used in such a case are as follows.

Type	Meaning	Data type level
any	Any type	FlowDesigner-specific type
int	integral type	Primitive type
float	Single-precision floating point real type	Primitive type
double	Double-precision floating point real type	Primitive type
complex<float>	single-precision floating point complex type	Primitive type
complex<double>	Double-precision floating point complex type	primitive type
char	Character type	Primitive type
& String type	Primitive type	
bool	Logical type	Primitive type
Vector	Array type	FlowDesignerobject type
Matrix	Matrix type	FlowDesignerobject type
Int	Integral type	FlowDesignerobject type
Float	Single-precision floating point real type	FlowDesignerobject type
String	String type	FlowDesignerobject type
Complex	Complex type	FlowDesignerobject type
TrueObject	Logical type (true)	FlowDesignerobject type
FalseObject	Logical type (false)	FlowDesignerobject type
nilObject	object type (nil)	FlowDesignerobject type
ObjectRef	Object reference type	FlowDesigner-specific type
Map	Map type	HARK-specific type
Source	Sound source information type	HARK-specific type

anyis a discrete data type containing various data types and defined as peculiar to FlowDesigner. Since the primitive data type of C++ is used for int, float, double, complex<float>, complex<double>, char, string, bool, their specifications conform to C++. When a user attempts to use the primitive type in contexts of Object, it is converted into GenericType<T> automatically and can be treated as a class with the first initial of an upper case character, succeeding to Object, such as Intand Float. Stringand Complex, however, are not GenericType and they are defined for std:string and std:complex, respectively. Likewise, they are used when and complex are used as an Objecttype. In this way, a data type defined in a form of succeeding to Objectof FlowDesignerfor the primitive type is to be called "FlowDesignerobject type". TrueObject, FalseObjectand nilObjectare also defined as Objectthat correspond to true, false and nil, respectively. The types used most often for the FlowDesignerobject type best are Vectorand Matrix. These are the FlowDesignerobject types that succeed to STL of C++ and their specifications basically conform to those of the data types corresponding to STL of C++. ObjectRefis a FlowDesigner-specific data type that is realized as a smart pointer for object types and often used as elements of Vector, Matrixand Map. Mapalso succeeds to STL of C++ and is a HARK-specific data type, not to FlowDesigner. Sourceis a HARK-specific data type, defined as a sound source information type.

A terminal type of a module is displayed on the bottom of the screen by focusing the cursor on a terminal of the module that the user wishes to know a type of on FlowDesigner. Figure 4.1 shows an example of focusing on an AUDIO terminal of a AudioStreamFromMicmodule. The message "AUDIO (Matrix<float>Windowed wave form). A row index is a channel, and a column index is time" is displayed on the bottom of FlowDesigner, describing that the AUDIO port supports Matrix<float>, Windowed wave forms are output, and a row of a matrix indicates a channel and a column of that indicates time. Terminals can be successfully connected and the connection is displayed with black arrows when data types of the terminals to be connected are same or the recipient terminal includes the data type of the transmitting terminal. When a user connects terminals without meeting this condition, such a connection is displayed with red arrows as warning. The following section describes details of the above for each of the primitive type, FlowDesignerobject type, FlowDesigner-specific type and HARK-specific type.

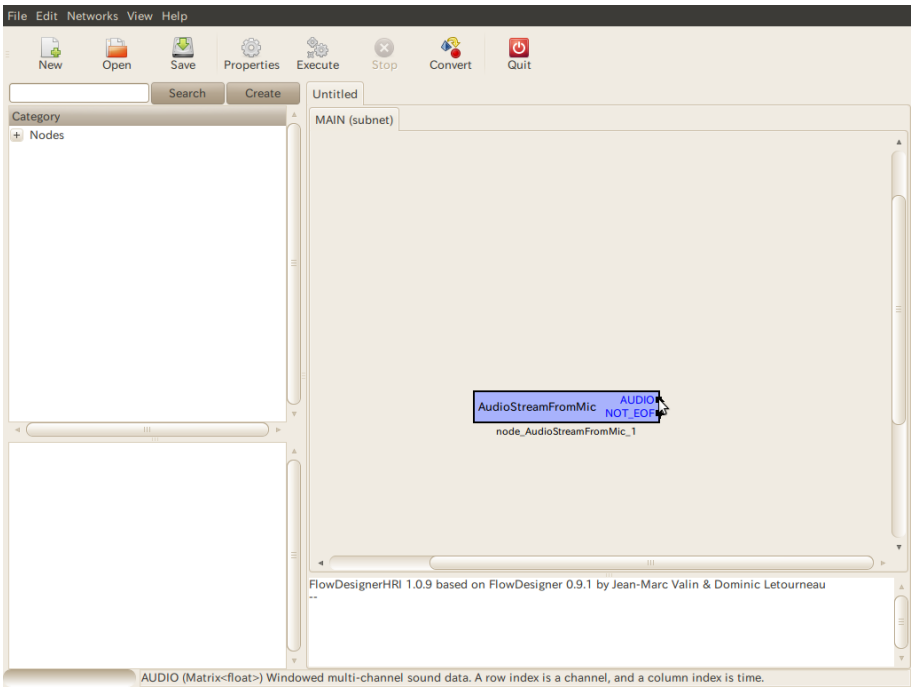


Figure 4.1: Display example of message bar

4.1 Basic type

As described above, `int`, `float`, `double`, `bool`, `char`, `string`and `complex`(`complex<float>`, `complex<double>`) are a primitive type that succeed to the C++ data type. In HARK, `int`is used for numbers that are always integer (e.g. number of sound sources or window length of FFT) and `float`is used for other values (angles). When only two values of ”truth” or ”false” are needed such as a flag, `bool`is used. When character strings such as file names are needed, `string`is used. HARKoften performs processing for spectrum units and each time block (frame) for specific length. The primitive type is rarely used as a data type of a terminal of a module though it is usually used as an element of `Matrix`, `Vector`or `Map`. `complex<float>` is also rarely used solely and expresses a spectrum and therefore it is often used as an element of `Vector`or `Matrix`. Floating point (`double`type) with double precision is supported by FlowDesignerthough is used only for `Source`in HARK.

Module converted into this type: The To* module of Conversion category is converted into each type. `int`uses ToInt, `float`uses ToFloat, `bool`uses ToBool and `string`uses ToString.

4.2 FlowDesigner object type

Int, Float, Stringand Complexare Objecttypes of int, float, stringand complex, respectively. TrueObjectand FalseObjectis the Objectthat corresponds to trueand falseof booltype, nilObjectis the Objectthat corresponds to nil. Descriptions for these are abbreviated. The types that are redefined as the Objecttype in FlowDesignerin the form of succeeding to the standard template library (STL) of C++ include Vectorand Matrix. These types are described below.

4.2.1 Vector

Vector is the type that stores an array of data. Any data may be stored in Vector. Vector< Obj >, which has ObjectRefas an element, and Vector< int >and Vector< float >, which have values (int, float) as elements are typically used. Since plural values are used in pairs, it is used to designate a group of angles in ConstantLocalizationand to present a group of localization results, which are the output of LocalizeMUSIC. The followings are definitions of Vectortype. BaseVectoris the type in which a method for FlowDesigneris implemented. As shown below, the Vectortype succeeds to vector type of STL.

```
template<class T> class Vector :
public BaseVector, public std::
vector<T>
```

The ToVect module in Conversion category accepts inputs such as intand floatand outputs Vector, which has only one input value in an element. Moreover, when the user wishes to use Vectoras a parameter of a module, Objectis to be used as a type of a parameter, which can be entered in text as shown below. For example, when the user wishes to enter Vectorof floattype that has two elements of 1.5 and 20.4 to the parameter, character strings are to be entered as shown in Figure4.2. However, for character strings, note that Vectormust come next to the first character < without spacing.

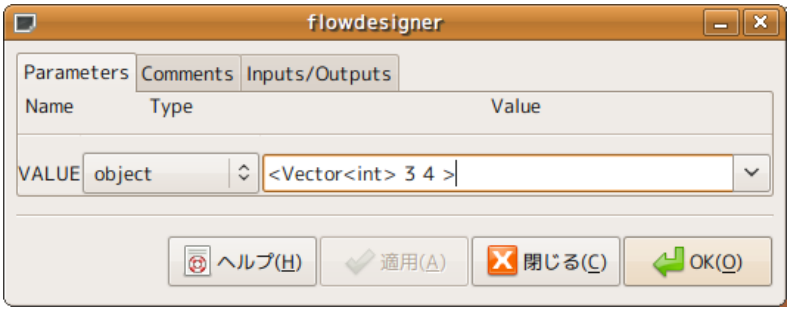


Figure 4.2: Input example of Vector

4.2.2 Matrix

Matrix indicates a matrix. The representative types are Matrix<complex<float> >type and Matrix<float>type. They are a matrix that has complexes and a matrix that has real numbers in their elements, respectively. Matrixtype is defined as follows. However, BaseMatrixis the class in which a method for FlowDesigneris implemented.

```
template<class T> class Matrix :
public BaseMatrix

protected members:
  int rows;
  int cols;
  T *data;
```

The modules that use Matrixfor inter-module communication include MultiFFT(frequency analysis) and LocalizeMUSIC(sound source localization). Further, in a robot audition system having functions of typical sound source location / source tracking / separation and speech recognition for which HARKis used, ID is given to a sound source in sound source

tracking (SourceTracker), and therefore `Matrix`is used for inter-module communication in the processing before ID is given and `Map`is often used in the processing after that.

4.3 Type peculiar to FlowDesigner

Types peculiar to FlowDesigner include any, ObjectRef, Objectand subnet_param.

4.3.1 any

anyis a generic name of various data types. When a terminal of a module is anytype, the terminal can be connected to any types of terminals without warning (with black line). However, since availability of actual communication depends on implementations in the module, it is desired not to use it as much as possible when users attempt to implement themselves. In HARK, its use is limited to the modules such as MultiFFT, DataLogger, SaveRawPCMand MatrixToMap, which are universally used.

4.3.2 ObjectRef

This data type presents reference of the Objecttype defined in FlowDesignerto a data type to be succeeded to. Concretely, it is defined as a smart pointer for the Objecttype. Since all of the FlowDesignerobject type, FlowDesigner-specific type and HARK-specific type are the data types that succeed to Object, these data types can be referred. The primitive data type can also be used since it is, as described above, also finally converted in NetCType<T> when substituting it for ObjectRefand becomes a subclass of Object.

4.3.3 Object

labelsubsec:Object

This is a data type mainly used in properties of a module. In HARK, it is used in ChannelSelector. Basically, this is a data type used when setting data types other than int, float, string, boland subnet_param, which are prepared beforehand, as properties. As described in the Section 4.3.2, it can be used as Objecttype including the primitive data types and therefore all data types can be designated as Objecttheoretically. However, what can be actually entered are limited to the data types in which the input-output by text is implemented. Vectorand Matrixare also implemented though Mapis not and therefore they cannot be entered as Objectpresently. Examples for availability of input are shown below for reference.

	< Vector< float > 0.0 1.0>	Common input method
for Vector	Complex<float(0,0)>	complexcan be entered as Complex
	Vector<complex<float> >(0.0, 1.0)>	NG since an input of complexis not supported
	Vector< ObjectRef > Complex<float> (0.0, 1.0)>>	No problem if entered as Complex
	<Int 1>	intcan be entered as Int

4.3.4 subnet_param

This is a data type used in a property of a module. When setting the same parameter as a property in plural modules of subnet, values in all the points concerned can be revised by rewriting the value in this label of MAIN, if designating subnet_param and describing a common label. For example, in the case of creating an Iterator network (the name is to be LOOP0) and arranging modules for which sampling frequency needs to be designated, such as LocalizeMUSICand GHDSS, SAMPLINGRATE appears in the property by arranging the virtual network LOOP0 on MAIN (subnet) if setting a type of SAMPLINGRATE, which is the property of these, as subnet_param and designating a label "SAMPLINGRATE". SAMPLINGRATE of LocalizeMUSICand GHDSSis guaranteed to be same continuously by setting this property to the inttype and entering 16000. Moreover, as another usage, setting the property of a module on MAIN (subnet) to the subnet_param type enables the parameter to be designated as an argument in batch execution. The user can confirm if the subject parameter can be designated as an argument by clicking the property of FlowDesigner. A value described in a dialog of this property is used as a default value in batch execution.

4.4 HARK-specific type

The data types that HARKdefines originally are Maptype and Sourcetype.

4.4.1 Map

Maptype is the data type consisting of a key and ObjectReftype as a set. ObjectRefdesignates a pointer to the data types that succeed to Objectsuch Matrix, Vectorand Source. Since HARKprovides a speech recognition function, processing is performed for each utterance. In such a case, since processing for each utterance is realized, Map<int, ObjectRef>with an utterance ID (sound source ID) as a key is used.For example, the output of GHDSS(sound source separation) is Map<int, ObjectRef>with an utterance ID as a key, a pointer to Vector< complex > that indicates a spectrum of separated utterance is stored in ObjectRef. MatrixToMapis for connecting such a module and the module that usually performs communication based on Matrix, which is used before a sound source tracking processing.

4.4.2 Source

This is a type that indicates source localization information, In HARK, it is used as the information that ObjectRefof Map<int, ObjectRef>indicates in a series of flows from the sound source localization to the sound source separation, namely, LocalizeMUSIC(output), SourceTracker(input-output), GHDSS(input). Sourcetype has the following information.

1. **ID:** inttype. ID of sound source
2. **Power:** floattype. Power of the direction localized.
3. **Coordinate:** An array that is 3 in length of floattype. The Cartesian coordinate on a unit ball corresponding to a source localizing direction.
4. **Duration:** doubletype. An index that assumes how long the localized sound source lasts from then on.

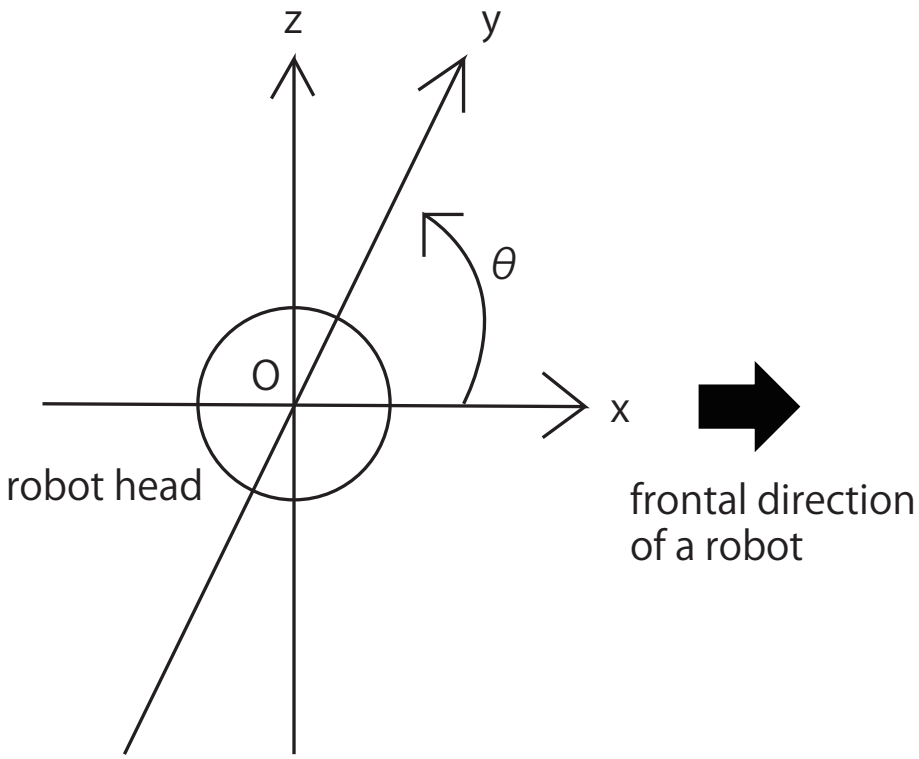


Figure 4.3: HARK standard coordinate system

4.5 HARK standard coordinate system

The center of a coordinate system used in HARKis the origin specified by a user (usually the center of a microphone array), and the positive direction of the x axis indicates the front side, that of the y axis indicates the left side and the of the z axis indicates the upper side. The unit for the coordinate system is meter. Moreover, for an angle, the counter clockwise is assumed to be positive rotation with the front side as 0[deg] (the left direction is 90[deg] for example).

Chapter 5

File Formats

This chapter describes file types used in HARKand their formats. In HARK, a file can be designated by an input-output of a module and property setting. Table 5.1 shows a list of them. Among these files, Juliusformat basically conforms to

Table 5.1: HARK module list that designates file input-output

Module name	Where to use	File type	File format
SaveRawPCM	Output	PCM raw file	PCM binary
LocalizeMUSIC	Property setting	Source localization transfer function file	HGTF binary
SaveSourceLocation	Output	Source localization result file	Localization result text
LoadSourceLocation	Property	Source location result file	Localization result text
GHDSS	Property setting	Sound source separation transfer function file	HGTF binary
	Property setting	Microphone position file	HARK text
	Property setting	Stationary noise position file	HARK text
	Property setting	Initial separation matrix file	HGTF binary
	Output	Separation matrix file	HGTF binary
SaveFeatures	Output	Feature file	`float`binary
DataLogger	Output	`Map`data file	`Map`text
JuliusMFT	Startup argument	Configuration file	jconf text
	In configuration file	Acoustic model / phoneme list	julius format
	In configuration file	Language model / dictionary	julius format
harktool	harktool	Source position list file	srcinf text
	harktool	Inpulse response file	`float`binary

the file formats of Julius. For differences from the original Julius, see the description of JuliusMFT. The file formats except Juliusformat are described below. Further, for implementation, input-output of HGTF (HARK General Transfer Function), binary format and HARK text format are collectively supported by the input-output library harkio of HARKand therefore their generic name is HARK file format. For other srcinf text formats except it, PCM binary, `float`binary, localization result text and `Map`text formats are planned to be modified so that they can be supported in harkio.

5.1 HGTF file format

HGTF(HARK General Transfer Function) file format is a RIFF (Resource Interchange File Format)-like binary file format and is used to present source localization transfer function files of LocalizeMUSIC, sound source separation transferfunction files of GHDSSand separation matrix files. Its data structure is shown in Table 5.2. Versions of the form type "HGTF" are 3 types as follows.

- M2PG 1.0
- GTF 1.0
- SM 1.0

Table 5.2: Data structure of HGTF (generic transfer function)

Identifier ”HARK”		4 bytes
Size after size form type		4 bytes
d	Form type ”HGTF”	4 bytes
a	fmt sub chunk (required)	
t	(Format ID, number of microphones)	
a	base data such as sampling frequency)	
c	M2PG/GTK/SM sub chunk (exclusive)	
h		
u		
n	SPEC sub chunk (exists only when M2PG is available)	
k	(additional information)	

M2PG (MUSIC 2D Polar Grid) is used for source localization transfer function files of LocalizeMUSIC. GTF (GHDSSTransfer Function) and SM (Separation `Matrix`) are used to present sound source separation transfer function files of GHDSSand separation matrix files of GHDSS, respectively. fmt is the essential sub chunk used to enter fundamental data. Its details are shown in Table5.3. Further, the size of chunks is the total size with 4 bytes of chunk names and 4 bytes of chunk sizes excluded. The M2PG/GTF/SM sub chunk is a chunk that stores data of individual files (source localization transfer function

Table 5.3: Definitions of fmt sub chunk

Type	Value
char[4]	”fmt”
`int`	Size of chunk, 24+12*$Nmic$ [bytes]
short[2]	fmt version (major, minor)
`int`	Format ID (0: undefined, 1: M2PG, 2: GTF, 3: SM)
`int`	Number of microphones (Nmic)
`int`	FFT length [sample] (Nfft)
`int`	Impulse response length [sample]
`int`	Sampling frequency [Hz]
Vector3[nmic]	Coordinateof microphone position[m]

file, sound source separation transfer function file and separation matrix file). It is further described in the following section. SPEC is used to enter additional information such as file generation conditions. Table 5.4 shows formats of SPEC. The null terminator is added the each item, the character code is ASCII (only single byte characters are used such as alphanumerics basically) and linefeed code is LF.

Table 5.4: Definitions of SPEC sub chunk

Type	Value
char[4]	”SPEC”
`int`	Size of chunk, 1,551 [bytes]
short[2]	SPEC version (major, minor)
char[11]	Creation date (Example: 2009/05/19)
char[128]	Location (Example: Smart room)
A char[128]	robot (Example: HRP-2, SIG2, $\cdots$)
Type of char[128]	microphone array (Example: 8ch circular, 8ch cubic, $\cdots$)
char[128]	Creator (Example: Kazuhiro Nakadai)
char[1024]	Memorandum (Parameters of TF creation of MUSIC)

5.1.1 Source localization transfer function for LocalizeMUSIC

The source localization transfer function indicates a transfer function from a sound source to a microphone array and is expressed as a file of HGTF format with M2PG chunk. The source localization transfer function file is designated in a property AMATRIX of LocalizeMUSIC. For details of usage, see LocalizeMUSIC. Table 5.5 shows definitions of M2PG. Further, the rows of memories of a multi-dimensional array are the row major (same as the case of C language). This file

Table 5.5: Definitions of M2PG sub chunk

Type	Value
char[4]	”M2PG”
int	Size of chunk, 12+12*Nd*Nr+8*Nd*Nr*$nmic$*($Nfft$/2+1) [bytes]
short[2]	M2PG version (major, minor)
int	Number of elements in horizontal direction (Nd)
int	Number of elements of distance (Nr)
Vector3[Nd][Nr]	Coordinate of source position
`complex<float>`[Nd][Nr][nmic][Nfft/2+1]	Transfer function

is created from a file in the impulse response formatby harktool, which is a support tool. Details of the creation method are described in Section7.1.

5.1.2 Sound source separation transfer function for GHDSS

The sound source separation transfer function, similar to the source localization transfer function, also indicates a transfer function from a sound source to a microphone array and is expressed as a file of HGTF format with GTF sub chunk. A sound source separation transfer function file is designated in TFCONJFILENAME when the property TFCONJ of GHDSSis `true`. For details of usage, see GHDSS. Table 5.1.2 shows definitions of GTF sub chunk. The rows of memories of a multi-dimensional array are the row major (same as the case of C language). This file is created from a file in the

Type	Value
char[4]	”GTF ”
int	Size of chunk, 8+12*Ns+4+4*Nf+8*Ns*$nmic$*Nf [bytes]
short[2]	GTF version (major, minor)
int	Number of source positions (Ns)
Vector3[Ns]	Coordinate of source positions [m]
int	Number of frequency lines (Nf)
int[Nf]	Maps from an index to a frequency bin
`complex<float>`[Ns][nmic][Nf]	Transfer function

impulse response format or from a microphone position and source location list file by harktool, which is a support tool. Details of the creation method are described in Section7.1.

5.1.3 Separation matrix for GHDSS

The separation matrix is expressed as a file of HGTF format with SM sub chunk. The sound source separation transfer function file is designated in the property INITWFILENAME of GHDSS. Moreover, a separation matrix file is output by designating EXPORTWFILENAME, which appears when EXPORTW is set to `true`. For details of usage, see GHDSS. Table5.6 shows definitions of SM sub chunk. The row of memories of a multi-dimensional array is the row major (same as the case of C language).

Table 5.6: Definition of SM sub chunk

Type	Value
char[4]	”SM ”
int	Size of chunk, 8+12*Ns+4+4*Nf+8*Ns*$nmic$*Nf [bytes]
short[2]	SM version (major, minor)
int	Source location quantity (Ns)
Vector3[Ns]	Coordinate of source position[m]
int	Number of frequency lines(Nf)
int[Nf]	Map from index to frequency bin
complex<float>[Ns][nmic][Nf]	Separation matrix

5.2 HARK text format

The HARKtext format is a text file that supports microphone position information and noise position information. Headers that have file identifiers and version numbers are to be described in each of them.

```
MICARY-LocationFile
Version= version number


Noise-LocationFile
Version= version number
```

A line with a sharp on its line head is a comment line where file creation conditions are to be described.

5.2.1 Microphone position text format

The microphone position text format is a file that describes each microphone position of a microphone array used in GHDSSand harktool. Selecting CALC in the TFCONJ property of GHDSSmodule, the MICFILENMAE property appears. Giving coordinate of each microphone in the microphone array saved in the microphone position format to this column, the impulse response is generated and sound source separation is performed in simulation. The followings are its formats.

```
MICARY-LocationFile
Version=1.0
# date 2010/07/13
# place office
# robot HRP-2
#array_type8ch circular
# author HARK team
# memo nothing special
# MUSIC_DIR=32
NumOfPosition=N
Position 0,x,y,z
Position 1,x,y,z
...
Position N-1,x,y,z
```

The first line indicates a header and the second line indicates a file version. The third to ninth lines indicate comments, which describe file creation conditions. The tenth line indicates the number of microphones. Position of each microphone is to be designated on the lines after the tenth line. Write a serial number (from zero) of microphones next to Position and designate the coordinate in the HARK standard coordinate system (in [m]) separating by commas. The following is an example of a microphone position file of a microphone array with eight channels.

```
MICARY-LocationFile
Version=1.0
# date 2010.5.14
# place office
# robot HRP-2
# array_type 8ch circular
# author HARK team
# memo test
NumOfPosition=8
Position 0,0.0715,0,0.0958
Position 1,0.0479,-0.0571,0.0958
Position 2,0.0,-0.0842,0.0808
Position 3,-0.0557,-0.07815,0.0694
Position 4,-0.0916,0.0,0.0728
Position 5,-0.0557,0.07815,0.0699
Position 6,0.0,0.0842,0.0808
Position 7,0.0479,0.0571,0.0958
```

5.2.2 Noise position text format

The noise source file is a file to designate a stationary noise direction from a fixed direction in GHDSS. Selecting `true`in the FIXEDNOISE property of GHDSSmodule, the FIXEDNOISEFILENAME property appears. Giving this column a sound source coordinate in the noise source format, process a noise of the direction as known noise source.

```
Noise-LocationFile
Version=1.0
# date 2010/07/13
# place office
# robot HRP-2
# array_type 8ch circular
# author HARK team
# memo
# MUSIC_DIR=32
NumOfPosition=1
Position -1,x,y,z
```

The first line indicates a header and the second line indicates a file version. The third to the ninth lines indicate comments, which describe file creation conditions. The tenth line indicates the number of noise sources and users designate 1 here. The eleventh line indicates a noise source direction. Write a serial number of sound source next to Positions and designate a noise source direction from the origin by three-dimensional vectors (size 1). Since -1 is allocated to the sound source ID in sound source separation processing, it is desired to set -1 to Position. Further, the number of the noise sources that users can designate is 1. The following noise source files are description examples of the case that stationary noise is located behind a robot.

```
Noise-LocationFile
Version=1.0
#
date
2010.5.14
# place HRI smart room
# robot HRP-2
# author HARK team
# memo test
NumOfPosition=1
Position -1,-0.985,0.0,-0.174
```

5.3 Other file formats

Other file formats include srcinf text format, PCM binary, `float`binary, localization result text and`Map`text formats. These are desctibed here.

5.3.1 Source position list information (srcinf) format

This is a file used in harktool to create a source position transfer function file (Set in AMATRIX of LocalizeMUSIC) and a sound source separation transfer function file (set in TFCONJFILE of GHDSS). It is a text file that enumerates measurement directions and the impulse response files corresponding to them in the form of "**measurement coordinate** %% **file name**, **offset**" (half size space is needed before "%%") for the number of impulse response measurement points on the basis of one record per line.

Measurement coordinate Measurement coordinate represents directional vectors from the center (an origin) of a microphone array to a measurement point (x, y, z). Users are to describe so that the magnitude of the vector $\sqrt{x^2 + y^2 + z^2}$ becomes 1.0. Further, each value must be separated by commas.

File name Users describe file names of the measured impulse responses. File names are to be named as example-1.flt, example-2.flt, ..., example-N.flt (the "example" parts are arbitrary parts to be determined by users. N indicates channel number) In Srcinfo.txt, these are collectively designated as example-*.flt. Note that the channel number begins with 1.

Offset Users designate from which line in an impulse response file the data reading starts. For example, if the user designates 1, reading starts from the first sample. Data reading usually is performed without problems with 1. Offset values must be positive integer.

A concrete example is shown below. This example is of HARK standard coordinate system and a four-channel microphone array are employed.

```
0.97014250,0.00000000,0.24253563 %%000-*.flt,1
0.00000000,0.97014250,0.24253563 %%090-*.flt,1
-0.97014250,0.00000000,0.24253563 %%180-*.flt,1
-0.00000000,-0.97014250,0.24253563 %%270-*.flt,1
```

However, this setting file example assumes that the following impulse response file groups are in the current directory.

```
000-1.flt
000-2.flt
000-3.flt
000-4.flt
090-1.flt
090-2.flt
090-3.flt
090-4.flt
180-1.flt
180-2.flt
180-3.flt
180-4.flt
270-1.flt
270-2.flt
270-3.flt
270-4.flt
```

5.3.2 PCM binary format

This is a file format that represents audio waveforms in the PCM format. It is used when a temporal waveform of a separation sound is saved in a file. Each sampling value is indicated in 16-bit or 24 bit signed integer by little endian and additional

information such as headers is not used. In HARK, SaveRawPCMoutputs in this format and the standard extension is .sw. Acoustic waveforms with plural channels are expressed being multiplexed in channels. Therefore, when the user wishes to read or write a file in this format in other programs, the file must be converted into wav files with sox or appropriate sampling frequency and track numbers and quantifying bit numbers must be designated.

5.3.3 floatbinary

This is a file that stores real or complex numbers in 32 bit floating-point numbers of IEEE 758 by little endian. In HARK, this is a file format mainly used for feature vectors and impulse response data.

Feature vector

Feature vectors are the vector values with real number or complex numbers in their elements. They are output in SaveFeaturesand their suffix is .spec. Dimensional elements of the vectors are written from a low dimensional direction to a high-dimensional direction. The meaning of vector values differs depending on users. Features usually used in HARKare power spectra, complex spectra, MFCC, MSLS and Missing Feature Mask. The complex spectrum is the vector value with a complex number in its element and other are the vector values with real numbers in their elements. Note that since they do not contain header information, the number of dimensions and vectors are not recorded.

Inpulse response format

This format represents impulse responses between a source position and microphone. Its standard suffix is .flt, and it is used to create a matrix of transfer function in harktool. To convert a file of this format into text, users create a program like the following example.

```
#include <stdio.h>
#include <fcntl.h>
int main(int argc, char *argv[])
{int fd=0; float data=0; fd = open(argv[1], O_RDONLY);while(1){int ret = read(fd, &data, 4); /* 4 [byte] =
```

5.3.4 localization result text

Source localization information format

This is a file format for recording source localization results. It is used in the LoadSourceLocationand SaveSourceLocationmodules. Designating a file saved in this format in FILENAME of the property enables to read and write. A source localization result format consists of the following information. Information on N sound sources is displayed in one row with a frame number on the top. For example, when N sound sources are localized for M frames, a text file in the following format is saved.

```
0 id0 x0 y0 z0 id1 x1 y1 z1 ... idN xN yN zN
1 id0 x0 y0 z0 id1 x1 y1 z1 ... idN xN yN zN...
M id0 x0 y0 z0 id1 x1 y1 z1 ... idN xN yN zN
```

The first row indicates serial numbers of input frames and from the second, columns increase according to the number of sound sources detected simultaneously, with four rows as one set (information for one sound source). The first line in one set of sound source information indicates IDs of the sound sources and the second to fourth lines indicate Cartesian coordinates system on a unit ball for a direction of a sound source.

5.3.5 Map text

This is a file format output in DataLogger. It supports `Map<int, float>`, `Map<int, int>`or `Map<int, ObjectRef>`(`ObjectRef`of `Map<int, ObjectRef>`is only `Vector< float >` or `Vector< complex >`). Output format is as follows.
Label frame count key 1 value 1 key two values Two ... A label specified in a parameter is output first, a frame count is output next and all keys and values of `Map`type are output separated by spaces in the form of one format for one frame as shown above. When the value is `Vector`, all elements are output separated by spaces.

Chapter 6

Module References

This chapter describes detail information of each module. How to read module references are described first.

How to read data type

1. **Outline:** Types of the data that the data type means are described.
2. **Module converted into this type:** When the data in the type is needed, which module should be used is described.

How to read file format

1. **Outline:** For what the format is used is described.
2. **Modules that use files in this format:** For what modules the file format is needed is described.
3. **Creation method:** How a file in the format is created is described.
4. **Format:** File formats are described in detail.

How to read module reference

1. **Outline of the module:** What function the module provides is described. The user may read them when wishing to know roughly about the functions.
2. **Necessary file:** The file required to use the module is described. This file links to the description of Section 5 season. Details of the contents of the file are described in Section 5.
3. **Usage:** Concrete connection examples are described for the cases the modules should be used. When the user somehow wishes to use the module concerned, it is recommended to just to try the example.
4. **Input-output and property of module:** Types and meaning of the input and output terminals of modules are described. Moreover, parameters to be set are shown in tables. For parameters for which detailed descriptions are required, descriptions are added for each of such parameters after the table.
5. **Details of the module:** Detail descriptions including theoretical background and implementation methods of modules are described. When the user wishes to know about the module concerned, it is recommended for them to read this.

Definition of symbols

Symbols used in this document are defined as Table 6.1.Moreover, the following notations are used implicitly.

- Lower case characters and upper case characters indicate time domain and frequency domain, respectively.
- Vectors and matrixes are transcribed in bold face.
- Transpose of a matrix is expressed with ^{T} and Hermitian transpose is expressed with in ^{H}. (X^T, X^H)
- A hat is added to the estimated value (e.g. An estimated value of x is $\hat{x}$)
- x is used for inputs and y is used for outputs.

- $\boldsymbol{W}$ is used for separation matrixes and $\boldsymbol{H}$ is used for transfer function matrixes.
- Channel numbers are represented by subscript. (e.g. The signal source of the third channel is s_3)
- Time and frequency are represented in (). (e.g. $X(t, f)$)

Table 6.1: Symbol list

Variable name	Description
m	Index of microphone
M	Number of microphone
$m_1, \cdots, m_M$	Symbol indicating each microphone
n	Index of sound source
N	Number of sound source
$s_1, \cdots, s_N$	Symbol indicating each sound source
i	Index of frequency bin
K	Number of frequency bin
$k_0 \cdots k_{K-1}$	Symbol indicating frequency bin
ω	Frequency
$NFFT$	Number of FFT point
$SHIFT$	Shift length
$WINLEN$	Window length
π	Circular constant
j	Imaginary unit
x	Scalar variable
$\boldsymbol{x}$	Vector variable
$[x(1), \ldots, x(D)]$	D-dimensional row vector
$[x(1), \ldots, x(D)]^T$	D dimensional column vector

6.1 AudioIO category

6.1.1 AudioStreamFromMic

Outline of the module

This module takes in multichannel speech waveform data from a microphone array. The audio interface devices supported by this module are For introduction of various devices, see Section 8.

Necessary file

No files are required.

How to use

When to use

This module is used when speech waveform data obtained from a microphone array is used as an input to the HARK system.

Typical connection

Figure 6.1 shows an use example of a AudioStreamFromMicmodule.

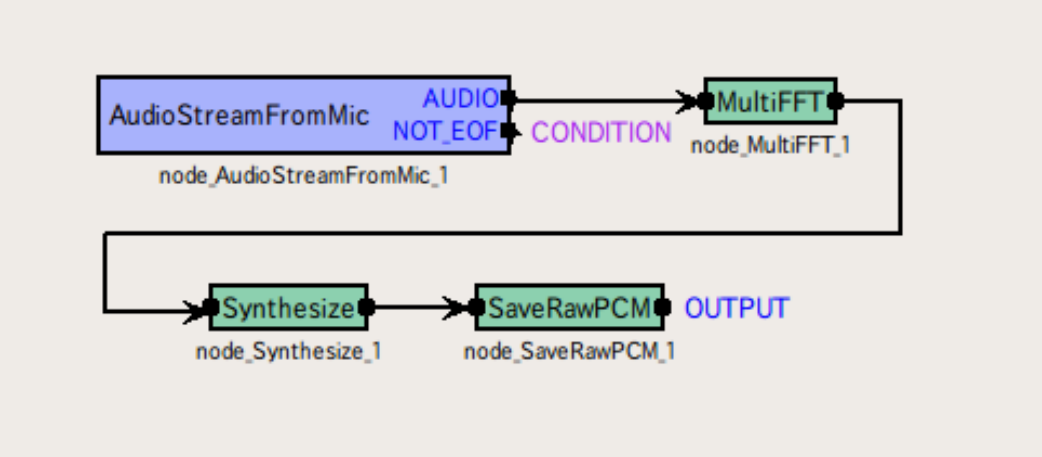


Figure 6.1: Connection example of AudioStreamFromMic

Overview of device

Among the devices that the AudioStreamFromMicmodule supports, the followings are introduced with photos.

1. Radio RASP
2. RME Hammerfall DSP series Multiface (Device corresponding to ALSA).

1. Radio RASP Figure 6.2 shows appearance of the radio RASP. Connection with the HARK system is established through Ethernet with a wireless LAN. Power source supply to radio RASP is performed with an attached AC adapter. Since the radio RASP responds to plug in power, a microphone of the plug in power supply can be connected to the terminal without change. Sound recording can easily performed without a microphone preamplifier as an advantage.

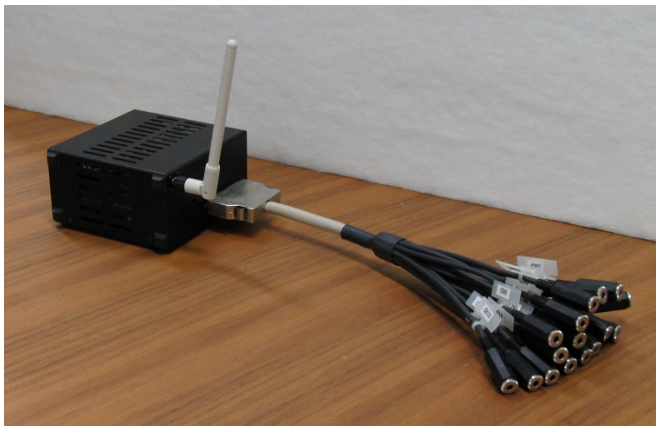


Figure 6.2: Radio RASP

2. RME Hammerfall DSP Multiface series Figures 6.3 and 6.4 show appearance of RME Hammerfall DSP series Multiface. The device communicates with a host PC through 32bit CardBus. Although a microphone can be connected to the device through a 6.3 mm TRS terminal, a microphone amplifier is used to secure the input level (Figure 6.4).) For example, the user may connect a microphone to RME OctaMic II and connect OctaMic II and Multiface. OctaMic II supports phantom power supply and a condenser microphone that requires phantom power (e.g. DPA 4060-BM) can be connected directly. However, since it does not have a plug in power supplying function, a battery box for plug in power is required to connect plug in power supply type microphones. For example, such battery boxes are attached to Sony EMC-C115 and audio-technica AT9903.

Figure 6.3: Front view of RME Hammerfall DSP Multiface

Figure 6.4: Back view of RME Hammerfall DSP Multiface

Input-output and property of module

Table 6.2: Parameter of AudioStreamFromMic

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	Frame length as a fundamental unit for processing.
ADVANCE	int	160	[pt]	Frame shift length.
CHANNELCOUNT	int		[ch]	Microphone input channel number of a device to use.
SAMPLINGRATE	int	16000	[Hz]	Sampling frequency of audio waveform data loaded.
DEVICETYPE	string	WS		Type of device to be used.
DEVICE	string	127.0.0.1		Character string necessary to access to device. Device name such as "plughw:0,1" or IP address when RASP is used.

Input Not required.

Output

AUDIO : Matrix<float>type. Multichannel audio waveform data with and an index for which rows are channels and columns are samples. Size of the column is equal to the parameter LENGTH.

NOTEOF : booltype. This type indicates whether there still are inputs of wave. Used as an ending flag of repetition processing for the waveforms recorded. When it is `true`, waveforms are loaded, and when it is `false`, reading is ended. `true`is output continuously.

Parameter

LENGTH : inttype. The default value is 512. The frame length, which is a base unit of processing, is designated in sample numbers. Frequency resolution rises as the value increases while temporal resolution falls. It is known that length corresponding to 20 $\sim$ 40 [ms] is appropriate for an analysis of audio waveforms. The sampling frequency 16,000 [Hz] correspond to the default value of 32 [ms].

ADVANCE : inttype. The default value is 160. Designate frame shift length in samples. The sampling frequency 16,000 [Hz] correspond to the default value of 10 [ms].

CHANNELCOUNT : inttype. The number of channels of the device to be used.

SAMPLINGRATE : inttype. The default value is 16000. Sample numbers of the waveforms loaded are designated. In processing, when frequency to ω [Hz] is needed, designate sampling frequency over 2ω [Hz]. When sampling frequency is high, data generally increase and it makes it difficult to perform real-time processing.

DEVICETYPE : stringtype. Select from ALSA, SINICH, RASP and WS. When a device supporting ALSA-based drivers is used, select ALSA. When TD-BD-16ADUSB is used, select SINICH. When RASP2 is used, select RASP. When radio RASP is used, select WS.

DEVICE : stringtype. Since input contents are different in each DEVICETYPE, see the following description.

Details of the module

The audio devices that HARK supports are three types as follows,

1. JEOL System Technology Co., Ltd. The RASP series,
2. Tokyo Electron Device LTD. TD-BD-16ADUSB,
3. ALSA-based devices (e.g. RME Hammerfall DSP series Multiface)

The following are notices for the use of each device. **RASP series:** Here, prameter setting for the use of **RASP-2** and **Radio RASP** are described. **TD-BD-16ADUSB:** Device corresponding to ALSA:

RASP-2	
CHANNELCOUNT	8
DEVICETYPE	WS
DEVICE	IP address of **RASP-2**
Radio RASP	
CHANNELCOUNT	16
DEVICETYPE	WS
DEVICE	IP address of **Radio RASP**
Remarks	Some models of the RASP series have both microphone inputs and line inputs among the 16 channels. When such a model is used, ChannelSelectormodule needs to be connected to the AUDIO output of AudioStreamFrom-Micmodule and only the microphone input channel has to be selected.
CHANNELCOUNT	16
DEVICETYPE	SINICH
DEVICE	SINICH
CHANNELCOUNT	8
DEVICETYPE	ALSA
DEVICE	plughw:0,1
Remarks	Designate plughw:a,b. Enter positive integers to a and b. Enter the card number indicated in arecord -l to a. When plural audio input devices are connected, plural card numbers are indicated. Enter card number to be used. Enter the subdevice number indicated in arecord -l to b. For the device that has plural subdevices, enter the number of the subdevice to be used. Devices that have analog input and digital inputs are one of the examples of plural subdevices.

6.1.2 AudioStreamFromWave

Outline of the module

This module reads speech waveform data from a WAVE file. The read waveform data is used in `Matrix<float>`type. Multichannel audio waveform data with an index for which rows are channels and columns are samples.

Necessary file

Audio files in RIFF WAVE format. There are no limits for the number of channels and sampling frequency. For quantifying bit numbers, 16-bit or 24-bit signed integers linear PCM format are assumed.

Usage

When to use

This module is used as an input to the HARK system when a user wishes to make WAVE files read.

Typical connection

Figures 6.5 and **??** show an usage example of the AudioStreamFromWavemodule. Figure 6.5 shows an example of AudioStreamFromWaveconverting `Matrix<float>`type multichannel waveforms read from a file into frequency domain with the MultiFFTmodule. To read a file in AudioStreamFromWave, designate a file name in Constantnode (Normal node FlowDesigner) and generate file descriptor in the InputStreammodule as shown in Figure **??**. Further, connect the output of the InputStreammodule to the iterator subnetwork(LOADWAVE in Figure **??**), which contains networks of various modules of HARK such as AudioStreamFromWave.

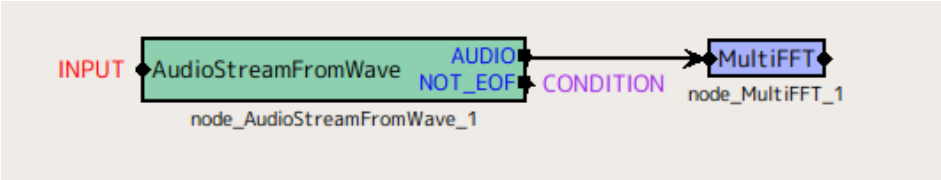


Figure 6.5: Connection example of AudioStreamFromWave1

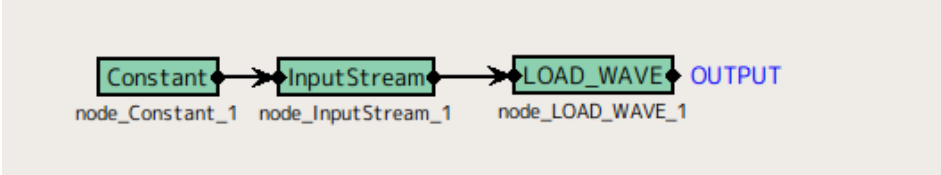


Figure 6.6: Connection example of AudioStreamFromWave2

Input-output and property of module

Input

INPUT : `Stream`type. Receive inputs from the InputStreammodule in IO category of FlowDesigner standard module.

Output

Table 6.3: Parameter list of AudioStreamFromWave

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	Frame length as a fundamental unit for processing.
ADVANCE	int	160	[pt]	Frame shift length.
USEWAIT	bool	false		Designate if processing is performed in real time

AUDIO : Matrix<float>type. Multichannel audio waveform data with an index for which rows are channels and columns are samples. Size of the column is equal to the parameter LENGTH.

NOTEOF : booltype. Indicate if files can be read yet. Used as an ending flag of repetition processing for files. When reaching the end of file, its outputs falseand outputs trueein other cases.

Parameter

LENGTH : inttype. The default value is 512. The frame length, which is a base unit of processing, is designated in sample numbers. Frequency resolution rises as the value increases while temporal resolution falls. It is known that length corresponding to 20 $\sim$ 40 [ms] is appropriate for an analysis of audio waveforms. The sampling frequency 16000 [Hz] correspond to the default value of 32 [ms].

ADVANCE : inttype. The default value is 160. Designate frame shift length in samples. The sampling frequency 16000 [Hz] correspond to the default value of 10 [ms].

USEWAIT : booltype. The default value is false. Usually, acoustic processing of the HARK system proceeds faster than real time. "Waiting" is added to the processing. When wishing to process for input files in real time, set to true. However, it is not effective when the processing speed is lower than that in real time.

Details of the module

Applicable file format: RIFF WAVE files can be read. The number of channels and quantifying bit numbers are read from headers of files though the format IDs that indicate sampling frequency and quantization method are ignored. The number of channels and sampling frequency correspond to arbitrary formats. When sampling frequency is required for the processing, there are as parameters required by modules (e.g. GHDSS, MelFilterBank). The linear PCM by 16- or 24-bit signed integers are assumed for the quantization method and bit counts.

Rough indication of parameters: In the case of an analysis whose purpose of the processing is speech analysis (speech recognition), about 20 $\sim$ 40 [ms] would be appropriate for LENGTH and $1/3 \sim 1/2$ of LENGTH would be appropriate for ADVANCE. In the case that sampling frequency is 16000 [Hz], the default values of LENGTH and ADVANCE are 32 and 10 [ms], respectively.

6.1.3 SaveRawPCM

Outline of the module

This module saves speech waveform data of a time domain in files. The binary files output are Raw PCM sound data in which sample points are recorded in 16 [bit] or 24 [bit] integer numbers. Multichannel audio data or monaural audio data for each separated sound are output depending on input data types.

Necessary file

No files a required.

Usage

When to use

This module is used when wishing to convert separated sound into waveforms in the Synthesizemodule to confirm actual sound or wishing to record the sound from a microphone array by connecting it with the AudioStreamFromMicmodule.

Typical connection

Figures 6.7 and 6.8 show an usage example of SaveRawPCM. Figure 6.7 shows an example of saving multichannel acoustic signals from AudioStreamFromMicin a file by the SaveRawPCMmodule. As shown in this example, select a channel to save a file to by the ChannelSelectormodule. Moreover, since SaveRawPCMaccepts `Map<int, ObjectRef>`type inputs, the MatrixToMapmodule converts from the `Matrix<float>`type into the `Map<int, ObjectRef>`type. Figure 6.8 shows an example of saving separated sound by the SaveRawPCMmodule. Since separated sound output from the GHDSSmodule or the PostFiltermodule, which suppresses noise after separation is in a frequency domain, it is converted into a waveform of a time domain by the Synthesizemodule before it is input into the SaveRawPCMmodule. The WhiteNoiseAddermodule is usually used for improving speech recognition rate of separation sound and is not essential for the use of SaveRawPCM.

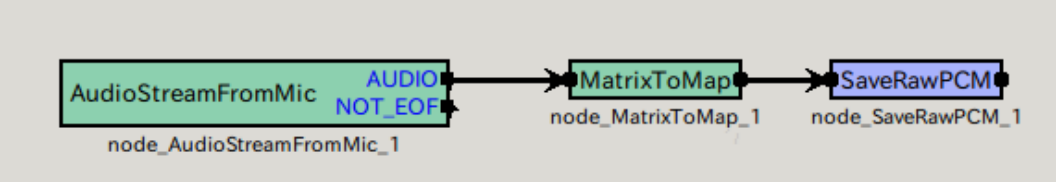


Figure 6.7: Connection example of SaveRawPCM1

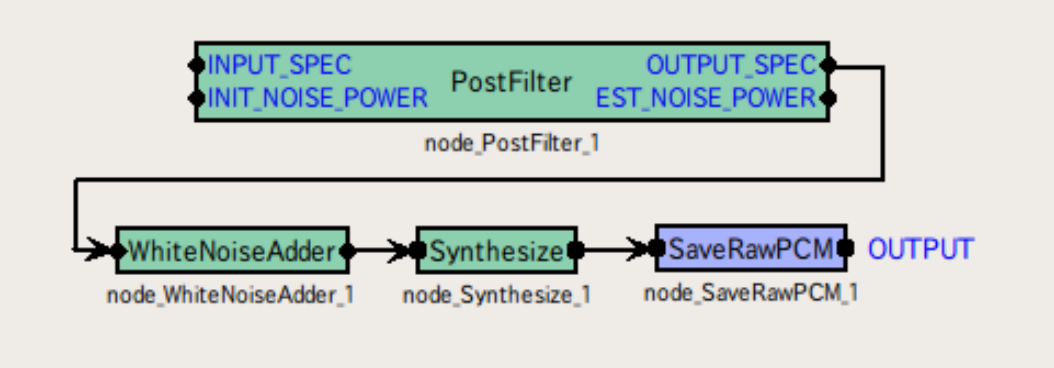


Figure 6.8: Connection example of SaveRawPCM2

Input-output and property of the module

Table 6.4: Parameter list of SaveRawPCM

Parameter name	Type	Default value	Unit	Description
BASENAME	string	sep		Prefix of name of the file to be saved.
ADVANCE	int	160	[pt]	Shift length of analysis frame of speech waveform to be saved in a file.
BITS	int	16	[bit]	Quantifying bit number of speech waveform to be saved in a file. Choose 16 or 24.

Input

INPUT : Map<int, ObjectRef>or Matrix<float>type.The former is a structure of sound source ID and waveform data such as separation sound and the latter is a waveform data matrix of multichannels.

Output

OUTPUT : Map<int, ObjectRef>or Matrix<float>type. Data same as the input are output.

Parameter

BASENAME : stringtype. For the default, designate a prefix of a file name sep_. . The file name output is "BASENAMEID. sw" when a sound source ID is attached. The file name of separated sound as a result of separating three sound mixtures is `sep0.sw`, `sep1.sw`, `sep2.sw` when BASENAME is sep..

ADVANCE : inttype. It is necessary to make even with values of ADVANCE of other modules.

BITS : inttype. Quantifying bit number of speech waveform to be saved in a file. Select 16 or 24.

Details of the module

Format of the files saved: The files saved are recorded as the Raw PCM sound data that do not have header information. Therefore, when reading the files, users need to designate 16 [bit] or 24 [bit] as appropriate sampling frequency, track quantity and quantifying bit number. Moreover, the written files vary depending on types of inputs as follows.

Matrix<float>type The file written is the multichannel audio file that has the number of channels equivalent to the number of rows of the input.

Map<int, ObjectRef>type The written files have the file name with an ID number after BASENAME and monaural audio files are written for each ID.

6.2 Localization category

6.2.1 ConstantLocalization

Outline of the module

The module that continues outputting constant sound source localization results. There are two parameters used for this module, which are ANGLES and ELEVATIONS, and the user sets azimuths (ANGLES) and elevation angles (ELEVATIONS) that the sound source comes at. Since each of these parameters is `Vector`, plural localization results can be output.

Necessary file

No files are required.

Usage

When to use

This module is used in the case that the user wishes to perform evaluations when a source localization result is already known. For example, when wishing to judge if there is a problem in the separation processing or are sound source localization errors, or wishing to evaluate the performance of sound separation under the same sound source localization condition, while evaluating results of sound source separation.

Typical connection

Figure 6.9 shows a connection example. This network displays constant localization results continuously.

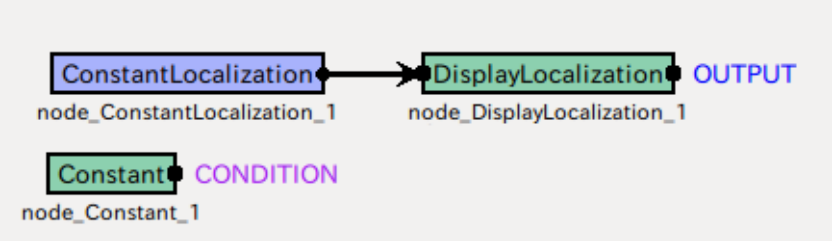


Figure 6.9: Connection example of ConstantLocalization

Input-output and property of the module

Input No inputs.

Output

SOURCES : `Vector< ObjectRef >` type. Fixed sound source localization results are output. The data `ObjectRef`refers are `Source`type data.

Parameter

Table 6.5: Parameter list of ConstantLocalization

Parameter name	Type	Default value	Unit	Description
ANGLES	`Vector< float >`		[deg]	Azimuth (right or left) of the sound source
ELEVATIONS	`Vector< float >`		[deg]	Elevation angle (up or down) of the sound source

ANGLES : `Vector< float >` type. Azimuth (right or left) of the direction that the sound source comes in. The unit of angle is degree.

ELEVATIONS : `Vector< float >` type. Elevation angle (up or down) of the direction that the sound source comes in. The unit of angle is degree.

Details of the module

It is assumed that the number of sound sources is N, the azimuth (ANGLE) of the i th sound source is a_i and the elevation angle (ELEVATION) of that is e_i. Here, parameters are described as follows.

ANGLES: <`Vector< float >` $a_1 \ldots a_N$>

ELEVATIONS: <`Vector< float >` $e_1 \ldots e_N$>

In this way, inputs are performed based on a spherical coordinate system though the data that ConstantLocalizationactually outputs are values of the Cartesian coordinates system (x_i, y_i, z_i), which correspond to points on the unit ball. Conversion from the spherical coordinate system to the Cartesian coordinates system is performed based on the following equations.

$$x_i = \cos(a_i\pi/180)\cos(e_i\pi/180) \tag{6.1}$$
$$y_i = \sin(a_i\pi/180)\cos(e_i\pi/180) \tag{6.2}$$
$$z_i = \sin(e_i\pi/180) \tag{6.3}$$

Other than the coordinate of sound sources, ConstantLocalizationalso outputs power (fixed at 1.0) and ID (indicates as i) of a sound source.

6.2.2 DisplayLocalization

Outline of the module

This module displays sound source localization results by GTK.

Necessary file

No files are required.

Usage

When to use

This module is used when wishing to visually confirm the sound source location results.

Typical connection

This module is connected after localization modules such as ConstantLocalizationor LocalizeMUSIC. The example in Figure 6.10 continuously displays fixed localization results from ConstantLocalization.

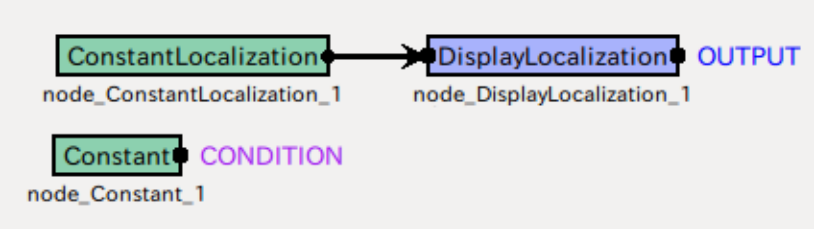


Figure 6.10: Connection example of DisplayLocalization

Input-output and property of the module

Input

SOURCES : `Vector< ObjectRef >` type. Users enter that data (`Source`type) that indicate source positions.

Output

OUTPUT : `Vector< ObjectRef >` type. Input values (`Source`type) are output.

Parameter

Table 6.6: Parameter list of DisplayLocalization

Parameter name	Type	Default value	Unit	Description
LOGISPROVIDED	`bool`	`false`		When users wish to output lngs to the standard output, set `true`.
WINDOWLENGTH	`int`	1000	Frame	Length of the display that displays sound source localization results

WINDOWLENGTH]: `int`type. The default value is 1000. This parameter can be adjusted according to the length of the sound source localization results that the user wishes to see.

Details of the module

Colors are different in each ID though the colors themselves do not have particular meaning.

6.2.3 LocalizeMUSIC

Outline of the module

A sound source direction in a horizontal surface direction is estimated from speech waveform data of multichannels by the Multiple Signal Classification (MUSIC) method in a microphone array coordinate system.

Necessary file

Matrixes (AMATRIX) consisting of steering vectors are required. It is generated based on the positional relationship between a microphone and sound source, or on a measured transfer function.

Usage

This module is used when wishing to estimate a direction of a sound source to, for example, make the robot turn towards the sound source direction, or when wishing to estimate a necessary sound source direction as an input such as sound source separation, like GHDSS.

Typical connection

Figure 6.11 shows a typical connection example.

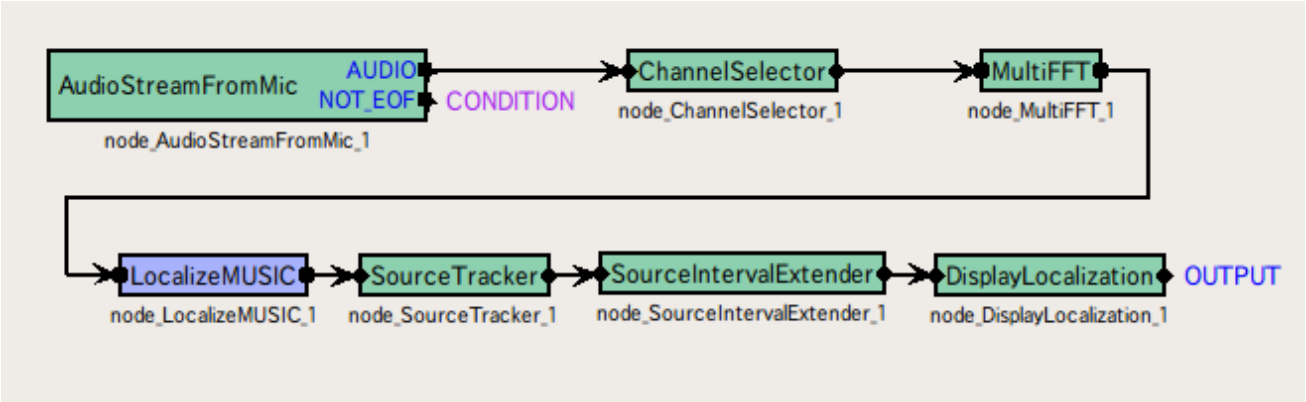


Figure 6.11: Connection example of LocalizeMUSIC

Input-output and property of the module

Input

INPUT : `Matrix<complex<float> >`, Complex frequency representation of input signals $M \times (NFFT/2 + 1)$.

Output

OUTPUT : `Vector<ObjectRef>`, Source position (direction) and `ObjectRef`are structures consisting of sound source directions and power of MUSIC spectra, the number of elements of `Vector`is equivalent to the number of sound sources (N).

Parameter

NBCHANNELS This parameter indicates the number of channels of input signals. Must be equal to the number of input signals.

Table 6.7: Parameter list of LocalizeMUSIC

Parameter name	Type	Default value	Unit	description
NBCHANNELS	int	8	[ch]	Number of channels of input signal
LENGTH	int	512	[pt]	Window length of FFT (=Number of points of FFT)
SAMPLINGRATE	int	16000	[Hz]	Sampling rate
AMATRIX	string			File name of steering vector
ELEVATION	float	16.7	[deg]	Elevation angle of sound source (fixed)
PERIOD	int	50	[frame]	Smoothing frame number for localization
NUMSOURCE	int	2		Number of sound sources assumed in MUSIC
MINDEG	int	-180	[deg]	Minimum value of steering direction (azimuth)
MAXDEG	int	180	[deg]	Maximum value of steering direction (azimuth)
LOWERBOUNDFREQUENCY	int	500	[Hz]	Minimum value of frequency to be used by localization processi
UPPERBOUNDFREQUENCY	int	2800	[Hz]	Maximum value of frequency to be used by localization process
DEBUG	bool	false		ON/OFF of debugging information output

LENGTH Must be equal to those of the modules such as MultiFFT, AudioStreamFromWaveand AudioStreamFromMicin the preceding paragraph.

SAMPLINGRATE Same as LENGTH, it must be equal to those of other modules for this parameter.

AMATRIX See tftool for file creation.

ELEVATION For the elevation angle, set the value that is assumed from a positional relationship between the robot and human. For example, assuming that the height of a microphone array of the robot is 1.2[m], position of the mouth of a human is 1.5[m] and distance between the human and robot is about 1.0[m], arctan $\frac{1.5-1.2}{1.0}$[rad]= 16.7 [deg]

PERIOD This parameter indicates smoothing frame numbers for localization and is the value indicated as *PERIOD* in the equation6.7 in Details of the module. As this value rises, the correlation matrix becomes stable. However, the time interval for obtaining a localization result extends, and moreover, the sound source tracking capacity worsens when the sound source moves.

NUMSOURCE This parameter indicates the number of sound sources assumed in MUSIC and is a value indicated as N_s in Details of the module. Must be determined beforehand. Generally, 1 or 2 is sufficient.

MINDEG This parameter indicates the minimum angle for sound source searching and is a value indicated as θ_{min} in Details of the module. Zero degree indicates the front direction of a robot, negative values indicate its right hand direction and positive values indicate its left hand direction. The designated range is assumed ±180 degrees here for convenience. However, it does not correspond to 360-degree rotations and therefore there are no particular limits.

MAXDEG This parameter indicates the maximum angle for sound source searching and is a value indicated as θ_{max} in Details of the module. Other specifications are same as those of MINDEG.

LOWERBOUNDFREQUENCY This parameter indicates the minimum frequency to be used when integrating in a frequency direction and is indicated as ω_{min} below. Users designate a value in the range from 0 to a half of sampling frequency.

UPPERBOUNDFREQUENCY The maximum frequency to be used when integrating in a frequency direction and is indicated as ω_{max} below. LOWERBOUNDFREQUENCY < UPPERBOUNDFREQUENCY must be satisfied.

DEBUG ON/OFF of debug outputs and the format of debug outputs are as follows. First, sets of ID, direction and power are output in a tab-delimited manner for the number of sound sources. ID is a number given to each frame in the order from zero for convenience and the numbers themselves do not have meaning. Directions are indicated with integers in degree, for which decimal numbers of the azimuth are rounded off. For power, power values of MUSIC spectra are output without modification (the value $P_a(\theta)$ below). Next, "MUSIC spectrum" is output after a line feed. Values of $P_a(\theta)$ are indicated for θ.

Details of the module

The MUSIC method estimates a sound source direction using eigenvalues of correlation of input signals.

Generation of transfer function: First, a transfer function from a sound source to each microphone is generated beforehand. The transfer function from the sound source S_θ in the θ direction to the i th microphone M_i seen from the microphone array is defined with $h_i(\theta, \omega)$ and frequency domain. Collecting up them, the transfer function from the sound source S_θ to the microphone array is defined as follows.

$$\boldsymbol{H}(\theta, \omega) = [h_i(\theta, \omega), \cdots, h_M(\theta, \omega)] \tag{6.4}$$

The user calculates or measures $\boldsymbol{H}(\theta, \omega)$ for each appropriate interval $\Delta\theta$ beforehand. A transfer function for which calculation was performed in a horizontal direction around as $\Delta\theta = 5[deg]$ is available a the website of HARK. Moreover, tftool enables the user to calculate automatically a transfer function by entering microphone layout information. In this way, a transfer function is obtained for each sound source direction and therefore is often called a steering vector since scanning is performed for the direction when localizing with directional vectors or this transfer function.

Estimation of correlation matrix: Next, the input signal of the N channel in time domain is assumed
$\mathrm{x}(t) = [x_1(t), x_2(t), x_3(t), \cdots, x_N(t)]^T$.
In frequency domain, they can be expressed as follows.

$$\boldsymbol{X}(\omega, f) = [X_1(\omega, f), X_2(\omega, f), X_3(\omega, f), \cdots, X_N(\omega, f)]^T \tag{6.5}$$

Here, ω and f indicate frequency and frame, respectively. For relations between the time t and frame f, see MultiFFT. The correlation matrix of the input signal $\boldsymbol{X}(\omega, f)$ is defined for each frame and frequency as follows.

$$\boldsymbol{R}(\omega, f) = \boldsymbol{X}(\omega, f)\boldsymbol{X}^H(\omega, f) \tag{6.6}$$

Theoretically, there would not be a problem if this $\boldsymbol{R}(\omega, f)$ were used for the following processing. However, for practical use, in order to obtain stable correlation matrixes, t is averaged for the time direction for HARK.

$$\boldsymbol{R}'(\omega, f) = \frac{1}{PERIOD} \sum_{i=f}^{f+PERIOD} \boldsymbol{R}(\omega, i) \tag{6.7}$$

For smoothing this, the output of LocalizeMUSICis one per PERIOD frame, not per frame.

Calculation of MUSIC spectrum: In the followings, f is abbreviated for simplification of descriptions. The property of the correlation matrix $\boldsymbol{R}'$ of inputs is positively used for MUSIC method. First, eigenvalue decomposition is performed for $\boldsymbol{R}'$.

$$\boldsymbol{R}'(\omega) = \boldsymbol{P}(\omega)\boldsymbol{D}(\omega)\boldsymbol{P}^{-1}(\omega) \tag{6.8}$$

Here, $\boldsymbol{P}$ represents the matrix $\boldsymbol{P}(\omega) = [e_1(\omega), e_2(\omega), \cdots, e_M(\omega)]$ consisting of eigenvectors perpendicular to each other and $\boldsymbol{D}(\omega)$ represents the diagonal matrix for which an eigenvalue corresponding to each eigenvector is assumed as a diagonal component. Further, $[\lambda_1(\omega), \lambda_2(\omega), \ldots, \lambda_M(\omega)]$, the diagonal component of $\boldsymbol{D}(\omega)$, are sorted in a descending order. It is assumed that among the eigenvalues corresponding to the eigenvectors of $\boldsymbol{P}(\omega)$, those with large and small values are derived from sound sources and from noise, respectively. Assuming the number of sound sources is N_s, $[e_1(\omega), \cdots, e_{N_s}(\omega)]$ is an eigenvector corresponding to sound source and $[e_{N_s+1}(\omega), \cdots, e_M(\omega)]$ is an eigenvector corresponding to noise. In MUSIC method, MUSIC spectra are obtained by the eigenvector that is assumed to be derived from noise and transfer functions obtained beforehand among eigenvectors obtained from input signals.

$$P(\theta, \omega) = \frac{\left|\boldsymbol{H}^H(\theta, \omega)\boldsymbol{H}(\theta, \omega)\right|}{\sum_{i=N_s+1}^{M} \left|\boldsymbol{H}^H(\theta, \omega)e_i(\omega)\right|} \tag{6.9}$$

The numerator on the right-hand side, which is the scalar product of eigenvectors derived from noise source among inputs and the transfer function, is calculated. On the space established with eigenvectors, objective signals corresponding to large eigenvalues are perpendicular to noise corresponding to small eigenvalues and therefore if the transfer function is vector

corresponding to the objective sound source, this scalar product value should be zero theoretically. Therefore, $P(\theta, \omega)$ diverges infinitely. In fact, it does not diverge infinitely for influences of noise and so on though the peak that is sharper than beam forming such as a delayed union is observed, which makes it easier to extract sound sources. The numerator on the right-hand side is a normalizing term for normalization. Since $P(\theta, \omega)$ is an MUSIC spectrum obtained from each frequency, frequency directions are integrated as follows. In HARK, square root of the maximum eigenvalue is used, as a weight for the frequency direction integration for convenience.

$$P_a(\theta) = \sum_{\omega=\omega_{min}}^{\omega_{max}} \sqrt{\lambda_1(\omega)} P(\theta, \omega) \tag{6.10}$$

Sound source serach:s Next, for $P_a(\theta)$, local peaks are detected from the range of θ_{min} - θ_{max}, and direction vectors corresponding to sound source directions and power of MUSIC spectra are output for the top N_s peaks in a descending order. Moreover, when the number of peaks do not reach N_s, outputs are less than N_s in some cases.

References

(1) F. Asano *et. al*, " Real-Time Sound Source Localization and Separation System and Its Application to Automatic Speech Recognition " Proc. of International Conference on Speech Processing (Eurospeech 2001), pp.1013–1016, 2001.

(2) Toshiro Oga, Yutaka Kaneda, Yoshio Yamazaki, "Acoustic system and digital processing" The Institute of Electronics, Information and Communication Engineers.

6.2.4 LoadSourceLocation

Outline of the module

This module reads source localization results stored in the SaveSourceLocationmodule.

Necessary file

Files in the formats saved in SaveSourceLocation.

Usage

When to use

This module is used when wishing to reuse the result of source localization or to evaluate different sound source separation methods based on the completely same source localization result.

Typical connection

Figure 6.12 shows a network that reads and display saved localization results. In this way, this module is used when a localized result is used later.

Figure 6.12: Connection example of LoadSourceLocation

Input-output and property of the module

Input No inputs.

Output

SOURCES : `Vector<ObjectRef>`type. The read localization result is output in the format same as those for source localization module (e.g. LocalizeMUSICor ConstantLocalization). `ObjectRef`refers `Source`type data.

NOTEOF : `bool`type. Since this is an output-terminal of `false`when the file is read till the end, when this terminal is used as a termination condition of a subnetwork of Iteration, files can be read till the end.

Parameter

Table 6.8: Parameter list of LoadSourceLocation

Parameter name	Type	Default value	Unit	Description
FILENAME	`string`	No default		File name of the file to be read

FILENAME : `string`type. Users set a file name of the file to be read.

Details of the module

The source localization result output by this module is `Vector`of the object that has five variables as follows.

1. **Power:** Fixed at 100.0
2. **ID:** ID of a sound source saved in a file
3. *x* **coordinate of sound source position:** Cartesian coordinate corresponding to a sound source direction on a unit ball.
4. *y* **coordinate of sound source position:** Cartesian coordinate corresponding to a sound source direction on a unit ball.
5. *z* **coordinate of sound source position:** Cartesian coordinate corresponding to a sound source direction on a unit ball.

6.2.5 SaveSourceLocation

Outline of the module

This modules saves source localization results.

Necessary file

No files are required

Usage

When to use

This module is used when a localization result is saved in a file analyze the localization result later. For reading the saved file, the LoadSourceLocation module is used.

Typical connection

Figure 6.13 shows a typical connection example. The network shown in the example has a function to save fixed localization results in files. For saving format, see 5.3.4. Moreover, this module can be connected to the modules that output localization results (e.g. LocalizeMUSIC, ConstantLocalizationor LoadSourceLocation).

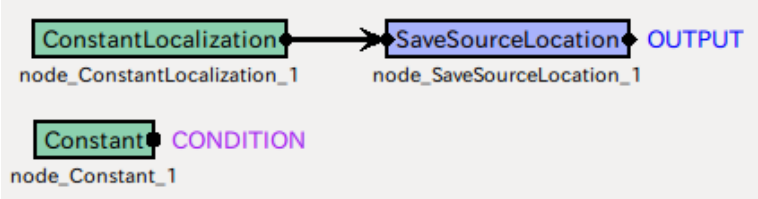


Figure 6.13: Connection example of SaveSourceLocation

Input-output and property of the module

Input

SOURCES : `Vector<ObjectRef>`type. Source localization results are input. `ObjectRef`type refers `Source`type data.

Output

OUTPUT : `Vector<ObjectRef>`type. Inputs (`Source`type) are output without modification.

Parameter

FILENAME : `string`type. No default values.

Table 6.9: Parameter list of SaveSourceLocation

Parameter name	Type	Default value	Unit	Description
FILENAME	`string`			Name of the files that the user wishes to save

6.2.6 SourceIntervalExtender

Outline of the module

This module is used when wishing to output source localization results earlier. Localization results are output sooner than usual by the parameter PREROLLLENGTH given by the user. For example, when PREROLLLENGTH is 5, a localization result is output sooner by five frames than the normal localization result output.

Necessary file

No files are required

Usage

When to use

This module is used as preprocessing when sound source separation is performed after source localization. Since a sound source is localized after sound is input, the sound source localization is slightly delayed from the time when the actual sound occurs. Therefore, the top of the separated sound is cut by this delay time. This module is used to prevent this problem.

Typical connection

Figure 6.14 shows a typical connection example. As shown in the figure, when wishing to separate sound based on a localization result, the delay of sound source separation starting can be avoided by inserting the SourceIntervalExtendermodule in between.

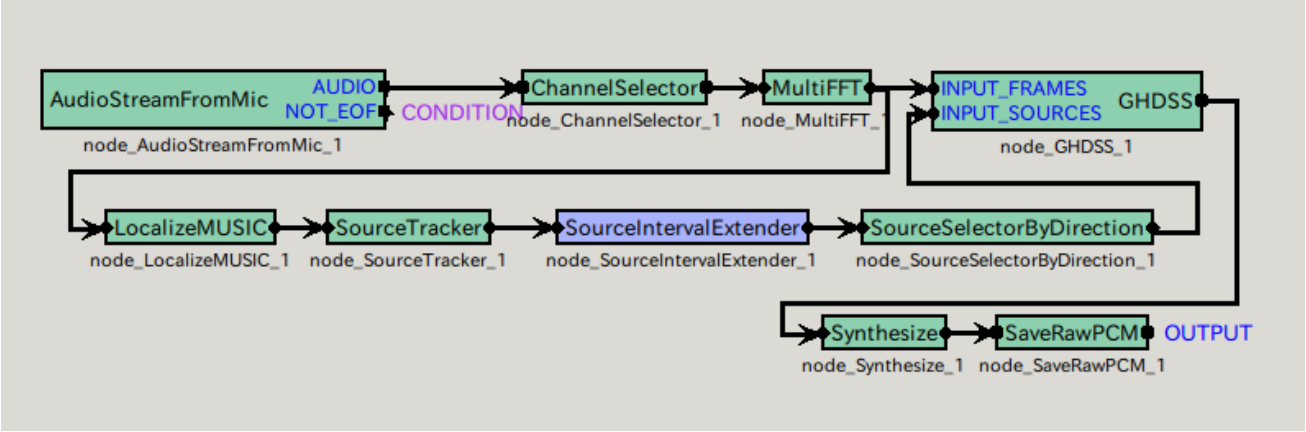


Figure 6.14: Connection example of SourceIntervalExtender

Input-output and property of the module

Input

SOURCES : `Vector<ObjectRef>`type. `Vector`of source localization results expressed in `Source`type are input. `ObjectRef`refers `Source`type data.

Output

OUTPUT : `Vector<ObjectRef>`type. An output source location result is output soon. `ObjectRef`refers of `Source`type data.

Parameter

Table 6.10: Parameter list of SourceIntervalExtender

Parameter name	Type	Default value	Unit	Description
PREROLLLENGTH	int	50	[frame]	The number of frames the localization result is output sooner by.

PREROLLLENGTH : inttype. This value determines how sooner the localization result is output. When this value is too small, start of sound source separation is delayed and therefore users are required to set this value considering the delay of the source location method used in the preceding paragraph.

Details of the module

When sound source separation is performed based on a localization result without SourceIntervalExtender, the beginning part of separated sound is cut by processing time of the source localization as shown in Figure 6.15. In the case of speech recognition in particular, since the cut-off of the beginning part influences on recognition performance, it is necessary to output a localization result beforehand with this module. This module read ahead in each repeat for the length set at PREROLLLENGTH. When a localization result is identified, the output of localization result is started from the time point when it was identified. (See Figure 6.16)

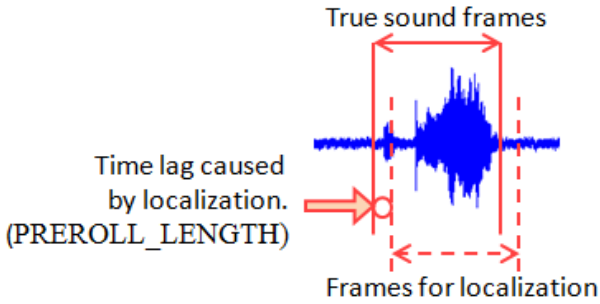


Figure 6.15: Necessity to output a source location result sooner than usual

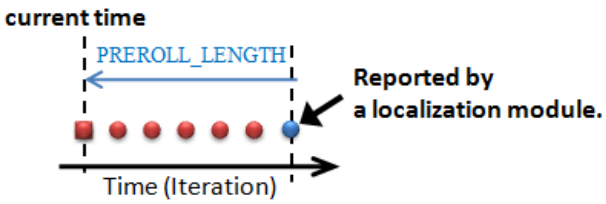


Figure 6.16: Operation of SourceIntervalExtender

6.2.7 SourceTracker

Outline of the module

For source localization results input in time series without IDs, this module gives the same ID to the source localization result obtained from the adjacent direction and a different ID to the source localization result obtained from the different direction. After passing this module, the user can judge if the sound sources are same by IDs.

Necessary file

No files are required.

Usage

When to use

Source localization results vary even though sound sources are fixed (e.g. standing person or fixed speaker). Usually, they are not obtained from the same direction continuously. Therefore, in order to unify source location results so that they can be treated as those from the same sound source, it is necessary to track the source location results. SourceTrackeruses an algorithm that gives IDs of the same sound source to the source localization results whose sound sources are sufficiently close. As a criterion for judging if the sound source is sufficiently close to another, the user may set an angle as a threshold. IDs are given to sound sources by this module, which enables to perform processing for each ID.

Typical connection

Usually, the outputs of source localization modules such as ConstantLocalizationor LocalizeMUSICare connected to the input terminal of this module. Then an appropriate ID is added to a localization result so users connect it to the sound source separation module GHDSSor presentation module for source location results (DisplayLocalization), which are based on source localization. Figure 6.17 shows a connection example. Here, a fixed source location result is displayed through SourceTracker. In this case, if the localization result that ConstantLocalizationoutputs is close to another, they are output together in one sound source. When giving the following property to ConstantLocalizationin the figure, the angle between the two sound sources is less than 20[deg], which is the default value of MINSRCINTERVAL and therefore only one sound source is presented.

ANGLES: `<Vector<float> 10 15>`

ELEVATIONS: `<Vector<float> 0 0>`

See ConstantLocalizationfor the meaning of the set points

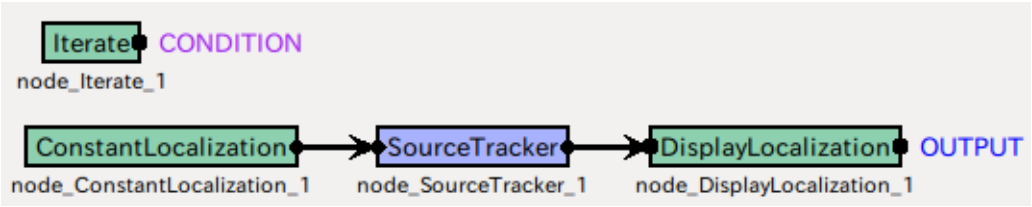


Figure 6.17: Connection example of SourceTracker

Input-output and property of the module

Input

INPUT : `Vector<ObjectRef>`type. Source localization result with no ID given.

Output

OUTPUT : `Vector<ObjectRef>`type. Source localization result for which the same ID is given to the sound source postioned to another

Parameter

Table 6.11: Parameter list of SourceTracker

Parameter name	Type	Default value	Unit	Description
THRESH	`float`			To be ignored if the power of t source is smaller than THRESH.
c.f. LocalizationMUSIC.cpp The 85th line PAUSELENGTH	`float`	800	[frame*10]	Length when assuming that the sound continues.
MINSRCINTERVAL	`float`	20	[deg]	Threshold value of angular diffe judging that the sound source is sa other.

THRESH : `float`type. This parameter judges by the power whether the source localization result is noise to be ignored. The result is judged as noise if the power is smaller than THRESH, and the localization result is not reflected to the output. When it is too small, noise is picked, and when it is too large, it becomes difficult to localize the target sound and therefore it is necessary to find the value that meets this trade-off.

PAUSELENGTH : `float`type. This parameter determines how long the sound source once output as a localization result would last. For a direction that is localized once, even though there are no valid source localization results after the first localization, localization results for such a direction continue being output during the repeat period for PAUSELENGTH / 10 [frame]. （「一度定位した方向は，それ以降に音源定位結果が無くても，PAUSELENGTH / 10 [frame] の繰り返しの間だけ同一方向の定位結果を出力し続ける.」の「それ以降」は「最初の定位後 (first localization)」と意訳しました）

Since the default value is 800, for the direction that is localized once, localization results continue being output for 80 [frame] after the first localization. （上記同様「‥ それ以降」は「最初の定位後 (first localization)」と意訳しましたがよろしいでしょうか）

MINSRCINTERVAL : `float`type. If the source location result is smaller than MINSRCINTERVAL, the two sound sources are judged as an identical sound source, and the influence of fluctuating motion of source localization is reduced by deleting either source localization result.

Details of the module

Definitions of symbols: First, symbols used in this section are defined.

1. ID: ID of sound source
2. Power p: Power of the direction localized.
3. Coordinate x, y, z: Cartesian coordinate on a unit ball corresponding to the source localization direction.
4. Duration r: The index that assumes how long the localized sound source lasts.

The power of the sound source localized is assumed p, and the Cartesian coordinates on a unit ball corresponding to the sound source direction are assumed x, y, z. Assuming N as the number of sound sources that a module presently maintains and M as that of the sound sources newly input, they are distinguished by the subscripts ^{last} and ^{cur}. For example, power of the sound source newly input i th is indicated as p_i^{cur}. The angle between sound sources, which is an index that judges closeness of the sound sources, is assumed θ. **Criterion of closeness of sound source directions:** Assuming two sound

source directions as coordinates on a unit circle of $\boldsymbol{q} = (x, y, z)$ and $\boldsymbol{q}' = (x', y', z')$, the angle θ is expressed as follows.

$$\boldsymbol{q} \cdot \boldsymbol{q}' = |\boldsymbol{q}||\boldsymbol{q}'| \cos\theta \tag{6.11}$$

Here, $\boldsymbol{q}, \boldsymbol{q}'$ are points on the unit circle and therefore expressed as follows. $|\boldsymbol{q}| = |\boldsymbol{q}'| = 1$ Therefore, the term of absolute values is eraced.

$$\boldsymbol{q} \cdot \boldsymbol{q}' = \cos\theta \tag{6.12}$$

Here, θ is obtained by applying an inverse trigonometric function.

$$\theta = \cos^{-1}(\boldsymbol{q} \cdot \boldsymbol{q}') = \cos^{-1}(x \cdot x' + y \cdot y' + z \cdot z') \tag{6.13}$$

In order to simplify the indication, the angle between the ith sound source and the jth sound source is expressed as θ_{ij} below. **Sound source tracking method:** The processing that SourceTrackerperforms in sound source tracking is shown

in Figure 6.18. In the figure, the horizontal axis indicates (=repeat count) and the vertical axis indicates sound source directions. Moreover, the blue circle indicates the source position (^{last}) that the module already has and the green circle indicates the source location (^{cur}) newly input. First, for all the sound sources, if the powers $p_i^{cur} and p_j^{last}$ are smaller than THRESH, they are deleted. Next, comparing the source positions newly input with the localization information that the module already has, if they are sufficiently close (=θ_{ij} is below MINSRCINTERVAL[deg]), they are integrated. The same ID is given to the integrated sound sources and the duration r^{last} is reset at PAUSELENGTH. If source positions newly input are sufficiently close, they are integrated to one sound source. Integration is achieved by leaving one sound source and deleting the other sound positions. The sound sources with θ_{ij} larger than MINSRCINTERVAL [deg] are judged as different sound sources. For the source positions that the module already has but are not newly input, r^{last} is reduced by ten. The sound sources with r^{last} less than zero is judged to be disappeared and deleted. If the source position newly input is different from any of those that the module already has, a new ID is given to the sound source and r^{cur} is initialized at PAUSELENGTH.

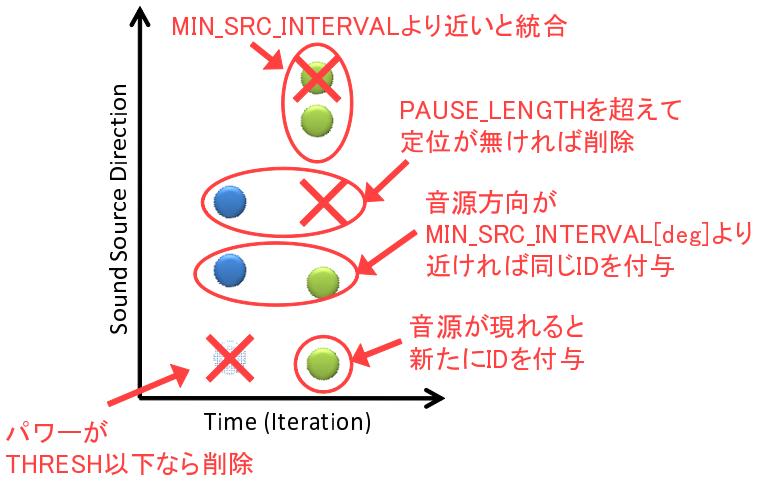


Figure 6.18: Sound source tracking method of SourceTracker

6.3 Separation category

6.3.1 BGNEstimator

Outline of the module

This module estimates stationary noise (or BackGround Noise) such as fan noise contained in signals from power spectra of multichannel signals. The estimated stationary noise is used in the PostFiltermodule.

Necessary file

No files are required,

Usage

When to use

This module is used to estimate stationary noise (or Back Ground Noise) such as fan noise contained in signals from power spectra of multichannel signals. The module that needs this estimated value is PostFilter. The PostFiltermodule suppresses the noise that cannot be subtracted by the separation processing based on this background noise and inter-channel leaks estimated in PostFilter. PostFilterestimates stationary noise though an initial value is needed separately. BGNEstimatoris used to generate such an initial value.

Typical connection example

A connection example of the BGNEstimatormodule is shown in Figure 6.19. In inputs, the user enters the power spectra that are obtained by converting speech waveforms into a frequency domain. The outputs are used in the PostFiltermodule.

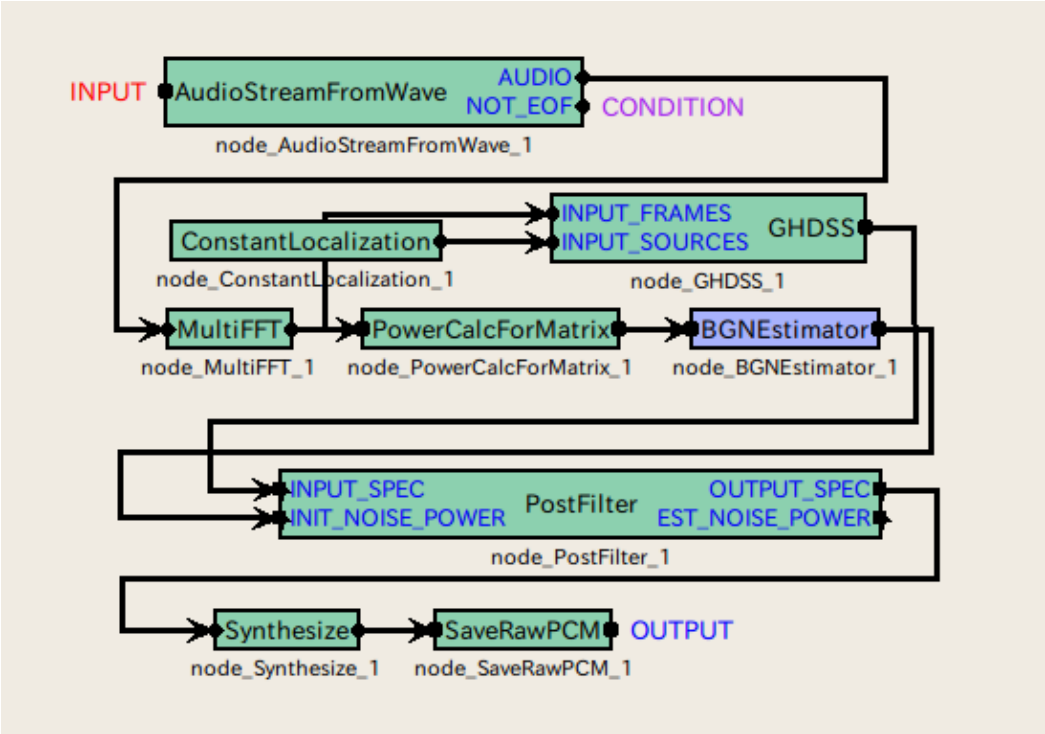


Figure 6.19: Connection example of BGNEstimator

Input-output and property of the module

Table 6.12: Parameter list of BGNEstimator

Parameter name	Type	Default value	Unit	Description
DELTA	float	3.0		Power-ratio threshold value
L	int	150	[frame]	Detection time width
ALPHAS	float	0.7		Smoothing coefficient of input signal
NOISECOMPENS	float	1.0		Mix rate of stationary noise
ALPHADMIN	float	0.05		The minimum value of smoothing coefficient
NUMINITFRAME	int	100	[frame]	Initialization frame number

Input

INPUT : Matrix<float>type. Multichannel power spectrum

Output

OUTPUT : Matrix<float>type. Power spectrum of estimated stationary noise.

Parameter

DELTA : floattype. The fault value is 3.0. This is the threshold value for determining if the target sound such as speech is included in the frequency bin of a power spectrum. Therefore, the greater the value is, the more quantity of power is judged as stationary noise.

L : inttype. The default value is 150. The time of holding the minimum spectrum in the history (stationary noise component), which is the criterion for determining the target sound, is included. This parameter is designated in the AudioStreamFromWavemodule as the number of shifts for the parameter ADVANCE.

ALPHAS : floattype. The default value is 0.7. The coefficient when smoothing input signals in a temporal direction. The greater the value is, the greater weight of the past frame value the smoothing is performed with.

NOISECOMPENS : floattype. The default value is 1.0. This parameter is the weight that weights and adds as stationary noise the frame that is judged not to contain the target sound(smoothing of stationary noise).

ALPHADMIN : floattype. The default value is 0.05. This parameter is the minimum weight when adding the power spectrum of the frame that is judged not to contain the target sound, in smoothing processing of stationary noise.

NUMINITFRAME : inttype. The default value is 100. When starting the processing, all are judged as stationary noise for the number of frames without judging the presence of the target sound.

Details of the module

The process to derive stationary noise is as follows. Time, frequency and channel indices are based on Table 6.1. The derivation flow is as shown in Figure 6.20.

1. Time direction, Frequency direction smoothing: Smoothing of the temporal direction is performed by interior division of the input power spectrum $\boldsymbol{S}(f,k_i)$ and stationary noise power spectrum of the former frame $\boldsymbol{\lambda}(f-1,k_i)$.

$$S_m^{smo,t}(f,k_i) = \alpha_s\lambda_m(f-1,k_i)+(1-\alpha_s)S_m(f,k_i) \tag{6.14}$$

Smoothing of the frequency direction is performed for the temporary smoothed time $S_m^{smo,t}(f,k_i)$.

$$S_m^{smo}(f,k_i) = 0.25S_m^{smo}(f,k_{i-1})+0.5S_m^{smo}(f,k_i)+0.25S_m^{smo}(f,k_{i+1}) \tag{6.15}$$

Table 6.13: Variable

Variable name	Corresponding parameter or description
$\boldsymbol{S}(f,k_i) = [S_1(f,k_i), ..., S_M(f,k_i)]^T$	Time frame f, input power spectrum of the frequency bin k_i
$\boldsymbol{\lambda}(f,k_i) = [\lambda_1(f,k_i), ..., \lambda_M(f,k_i)]^T$	Estimated noise spectrum
δ	DELTA, Default value 0.3
L	L, default value 150
α_s	ALPHAS,Default 0.7
θ	NOISECOMPENS, Default value 1.0
α_d^{min}	ALPHADMIN, Default value 0.05
N	NUMINITFRAME, default value 100

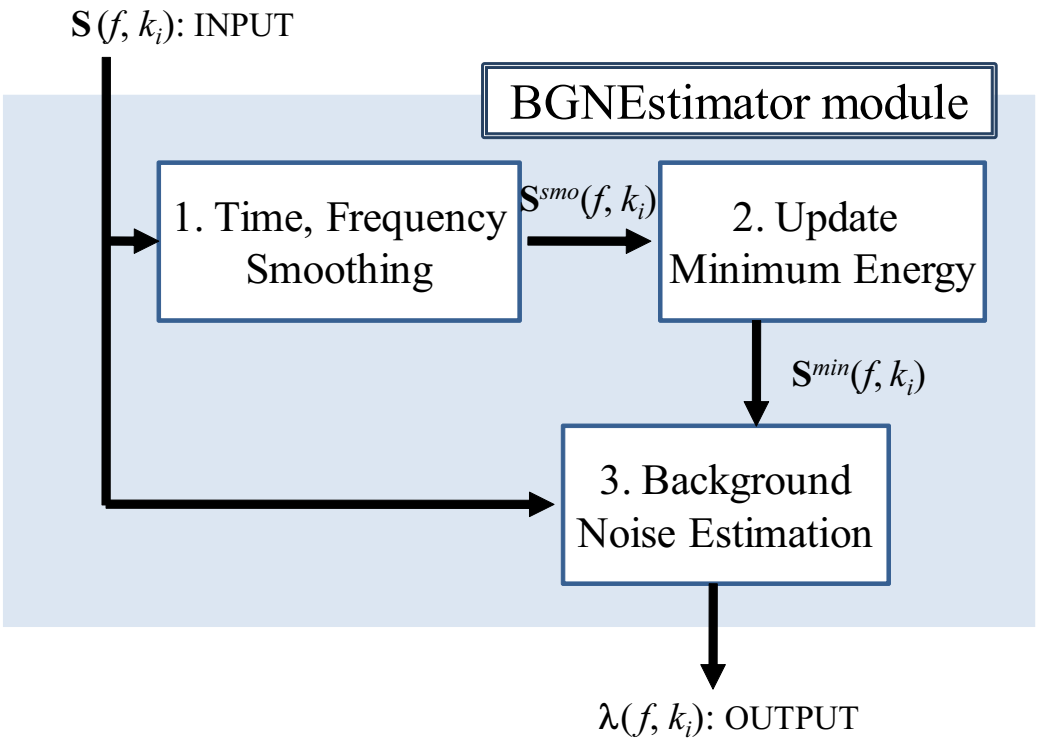


Figure 6.20: Flow of stationary noise estimation

2．Update of the minimum energy: In order to judge presence of the target sound, the minimum energy $\boldsymbol{S}^{min}$ is calculated for each channel and frequency bin after processing is started. $\boldsymbol{S}^{min}$ is the minimum energy for each channel and frequency bin after processing is started $\boldsymbol{S}^{tmp}$ is the provisional minimum energy updated for every L frame.

$$S_m^{tmp}(f,k_i) = \begin{cases} S_m^{smo}(f,k_i), & \text{if } f = nL \\ \min\{S_m^{tmp}(f-1,k_i), S_m^{smo}(f,k_i), & \text{if } f \neq nL \end{cases} \tag{6.16}$$

$$S_m^{min}(f,k_i) = \begin{cases} \min\{S_m^{tmp}(f-1,k_i), S_m^{smo}(f,k_i), & \text{if } f = nL \\ \min\{S_m^{min}(f-1,k_i), S_m^{smo}(f,k_i), & \text{if } f \neq nL \end{cases} \tag{6.17}$$

Here, n is an arbitrary integer.

3．Stationary noise estimation:

1. Judgment of the presence of the target sound
   When either of the equations below is satisfied, it is judged that power of the target sound is not contained in the

concerned time and frequency and only noise exists.

$$
\begin{aligned}
S_m^{smo}(f,k_i) &< \delta S_m^{min}(f,k_i) \text{or} && (6.18)\\
f &< N \text{or} && (6.19)\\
S_m^{smo}(f,k_i) &< \lambda_m(f-1,k_i) && (6.20)
\end{aligned}
$$

2. Calculation of smoothing coefficient
   The smoothing coefficient α_d used when power of stationary noise is calculated as follows.

$$
\alpha_d = \begin{cases} \frac{1}{f+1}, & \text{if } (\frac{1}{f+1} \geq \alpha_d^{min}) \\ \alpha_d^{min}, & \text{if } (\frac{1}{t+1} < \alpha_d^{min}) \\ 0 & \text{(When the target sound is contained)} \end{cases} \quad (6.21)
$$

   Steady noise is obtained by the following equations.

$$
\lambda_m(f,k_i) = (1-\alpha_d)\lambda_m(f-1,k_i) + \alpha_d \theta S_m(f,k_i) \quad (6.22)
$$

6.3.2 CalcSpecSubGain

Outline of the module

This modules determines an optimum gain for how much of the power spectrum of the estimated noise is to be removed when a noise power spectrum is removed from a power spectrum of signals + noise. Further, it outputs probability of speech presence (See Section 6.3.7). However, this module constantly outputs 1 as probability of speech presence. Difference of a power spectrum of the separated sound and that of the estimated noise is output.

Necessary file

No files are required.

Usage

When to use

It is used at the time of the noise estimation with the HRLEmodule.

Typical connection

Figure 6.21 shows a connection example of CalcSpecSubGain. Inputs are power spectra after separation with GHDSSand those of the noise estimated in HRLE. Its outputs connect VOICEPROB and GAIN to SpectralGainFilter.

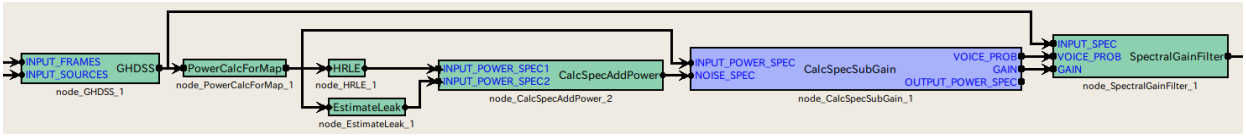


Figure 6.21: Connection example of CalcSpecSubGain

Input-output and property of the module

Input

INPUTPOWERSPEC : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and a power spectrum of the separated sound.

NOISESPEC : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and a power spectrum of the estimated sound.

Output

VOICEPROB : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and a power spectrum of the probability of speech presence.

GAIN : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and a power spectrum of the optimum sound.

OUTPUTPOWERSPEC : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of the sound source ID and the power spectrum for which the estimated noise is deducted from separated sound.

Parameter

No prameters.

Details of the module

This modules determines an optimum gain for how much of the power spectrum of the estimated noise is to be removed when a noise power spectrum is removed from a power spectrum of signals + noise. It also outputs probability of speech presence（See Section 6.3.7. However, this module constantly outputs 1 as probability of speech presence. Difference of a power spectrum of the separated sound and that of the estimated noise is output. Assuming that the power spectrum for which noise is subtracted from the separated sound is $Y_n(k_i)$, a power spectrum of the separated sound is $X_n(k_i)$ and that of the noise estimated is $N_n(k_i)$, the output from OUTPUTPOWERSPEC is expressed as follows.

$$Y_n(k_i) \quad = \quad X_n(k_i) - N_n(k_i) \tag{6.23}$$

Here, n indicates an analysis frame number. k_i indicates a frequency index. The optimum gain $G_n(k_i)$ is expressed as follows.

$$G_n(k_i) \quad = \quad \begin{cases} \frac{Y_n(k_i)}{X_n(k_i)}, & \text{if } Y_n(k_i) > 0, \\ 0, & \text{if otherwise.} \end{cases} \tag{6.24}$$

When processing simply with $Y_n(k_i)$, power can become negative. The purpose of this module is to calculate a gain for removing power spectra of noise beforehand so that power cannot be negative since it might become difficult to treat a power spectrum on the subsequent processing.

6.3.3 CalcSpecAddPower

Outline of the module

This module outputs spectrum for which two input power spectrum are added.

Necessary file

No files are required.

Usage

When to use

This module is used for noise estimation with the HRLEmodule. A power spectrum of the noise estimated with the HRLEmodule is added to that with EstimateLeakso as to obtain a total noise power spectrum.

Typical connection

Figure 6.22 shows a connection example of CalcSpecAddPower. Inputs are the power spectrum of the noise estimated with HRLEand that with EstimateLeak. The output is connected with CalcSpecSubGain.

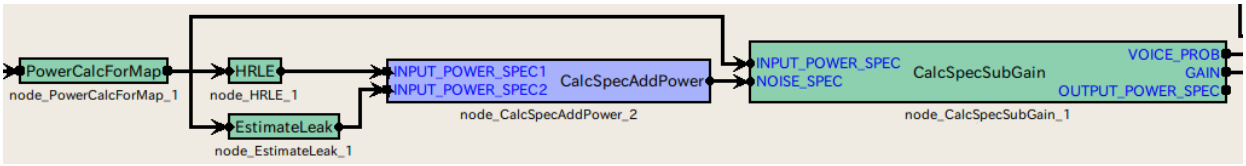


Figure 6.22: Connection example of CalcSpecAddPower

Input-output and property of the module

Input

INPUTPOWERSPEC1 : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and power spectrum.

INPUTPOWERSPEC2 : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and a power spectrum.

Output

OUTPUTPOWERSPEC : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of the sound source ID and the power spectra for which the two inputs are added.

Parameter

No prameters.

Details of the module

This module outputs spectra for which two spectra are added.

6.3.4 EstimateLeak

Outline of the module

This module estimates leakage component from the other channels.

Necessary file

No files are required.

Usage

When to use

This module is used for denoising after source separation with GHDSS.

Typical connection

Figure 6.23 shows a connection example of EstimateLeak. Inputs are power spectrum of speech, which are the outputs of GHDSS. The outputs are connected to CalcSpecAddPower.

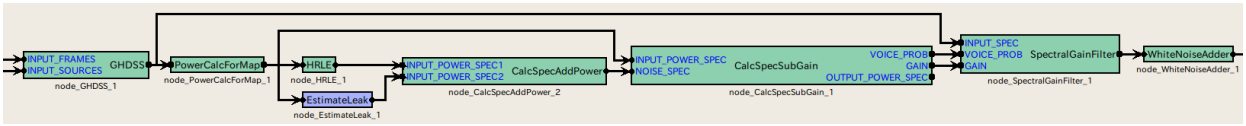


Figure 6.23: Connection example of EstimateLeak

Input-output and property of the module

Input

INPUTPOWERSPEC : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and power spectrum.

Output

LEAKPOWERSPEC : `Map<int, ObjectRef>`type. `Vector<float>`type data pair of a sound source ID and power spectrum of leakage noise.

Parameter

Table 6.14: Parameter list of EstimateLeak

Parameter name	Type	Default value	Unit	Description
LEAKFACTOR	`float`	0.25		Leakage rate.
OVERCANCELFACTOR	`float`	1		Leakage rate weighting factor.

Details of the module

This module estimates leakage component from the other channels. For details, see PostFiltermodule 1-b) Leakage Noise Estimation in Section 6.3.7.

6.3.5 GHDSS

Outline of the module

The GHDSS module performs sound source separation based on the GHDSS (Geometric High-order Dicorrelation-based Source Separation) algorithm. The GHDSS algorithm utilizes microphone arrays and performs the following two processes.

1. Higher-order decorrelation between sound source signals,
2. Direcitivity formation towards the sound source direction.

For directivity formulation, the positional relation of the microphones given beforehand is used as a geometric constraint. The GHDSS algorithm implemented in the current version of HARK utilizes the transfer function of the microphone arrays as the positional relation of the microphones. Node inputs are the multi-channel complex spectrum of the sound mixture and data concerning sound source directions. Note outputs are a set of complex spectrum of each separated sound.

Necessary files

Table 6.15: Necessary files

Corresponding parameter name	Description
TFCONJFILENAME	Transfer function of microphone array
MICFILENAME	Coordinate of microphone position
FIXEDNOISEFILENAME	Coordinate of noise source position
INITWFILENAME	Initial value of separation matrix

Usage

When to use

Given a sound source direction, the module separates a sound source originating from the direction with a microphone array. As a sound source direction, either a value estimated by sound source localization or a constant value may be used.

Typical connection

Figure 6.24 shows a connection example of the GHDSS. The node has two inputs as follows:

1. INPUTFRAMES requires multi-channel complex Spectrum of sound mixture,
2. INPUTSOURCES requires the results of sound source localization.

To recognize the output, that is a separate sound, it may be given to MelFilterBank to convert it to speech features for speech recognition. As a way to improve the performance of automatic speech recognition, it may be given to one of the following modules:

1. The PostFilter module to suppress the inter-channel leak and diffusive noise caused by the source separation processing (shown in the upper right part in Fig.**??**).
2. PowerCalcForMap, HRLE, and SpectralGainFilter cascadedlly to suppress the inter-channel leak and diffusive noise caused by source separation processing (this tuning would be easier than with PostFilter), or
3. PowerCalcForMap, MelFilterBank and MFMGeneration cascadedlly to generate missing feature masks so that the separate speech is recognized by a missing-feature-theory based automatic speech recognition system (shown in the lower right part of Fig.**??**).

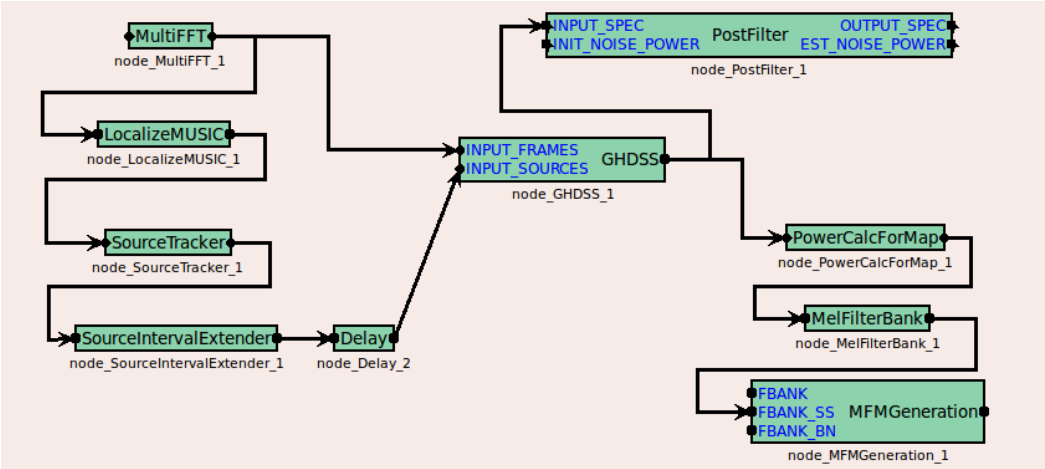


Figure 6.24: Example of Connections of GHDSS

Input-output and property of the module

Input

INPUTFRAMES : `Matrix<complex<float> >`type. Multi-channel complex spectra. Rows correspond to channels, complex spectra of waveforms input from microphones in other words, and columns correspond to frequency bins.

INPUTSOURCES : `Vector<ObjectRef>`type. A `Vector`array of the `Source`type object in which Source localization results are stored. It is typically connected to the SourceTrackermodule and SourceIntervalExtendermodule and its outputs are used.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of a sound source ID of the separated sound and a 1-channel complex spectrum of the separated sound
(`Vector<complex<float> >`type).

Parameter

LENGTH : `int`type. Analysis frame length, which must be equal to the values at a preceding stage value (e.g. AudioStreamFromMicor the MultiFFTnode).

ADVANCE : `int`type. Shift length of a frame, which must be equal to the values at a preceding stage value (e.g. AudioStreamFromMicor the MultiFFTnode).

SAMPLINGRATE : `int`type. Sampling frequency of an input waveform.

LOWERBOUNDFREQUENCY This parameter is the minimum frequency used when GHDSSprocessing is performed. Processing is not performed for frequencies below this value and the value of the output spectrum is zero then. The user designates the parameter in the range from 0 to a half of the sampling frequency.

UPPERBOUNDFREQUENCY This parameter is the maximum frequency used when GHDSSprocessing is performed. Processing is not performed for frequencies above this value and the value of the output spectrum is zero then. LOWERBOUNDFREQUENCY < UPPERBOUNDFREQUENCY must be maintained.

TFCONJ : `string`type. The user selects if a transfer function is to be obtained by a simulation or from measurement values. Select CALC for using simulations and DATABASE for using measurement values. When calculating by a simulation, a conventional transfer function is calculated. However, measurements values need to be complex conjugates of the transfer function. When transfer function data are generated by hark-tools, they are converted into value of complex conjugate values automatically.

1. DATABASE: Set TFCONJFILENAME.
   TFCONJFILENAME: stringtype. This parameter indicates a binary file name in which a transfer function is described. For its file formats, see Section 5.1.2.
2. CALC: Set MICFILENAME, MICPOSSHIFT and SPEEDSOUND.
   MICFILENAME: stringtype. Designate a name of the text file in which a coordinate of a microphone position was described. For its file formats, see Section 5.2.
   MICPOSSHIFT: stringtype. Select FIX or SHIFT. The default value is FIX. When SHIFT is chosen, the origin of the microphone coordinate system is moved to the gravity center of the above-mentioned file. When FIX is selected, nothing is performed. **SPEEDSOUND**: floattype. The default value is 343. Designate the acoustic velocity [m/s]

FIXEDNOISE : booltype. Fixed noise. For example, the user determines if fan noise at the back of the robot is to be included in the GHDSSprocessing. When true, the user designates FIXEDNOISEFILENAME. item [FIXEDNOISEFILENAME]: stringtype. This parameter is valid when FIXEDNOISE is set to true. The user designates a name of the text file in which the position coordinate of the fixed noise is described. For its formats, see 5.2.

INITWFILENAME : stringtype. The file name in which an initial value of a separation matrix is described. Giving a converged separation matrix of an initial-value through preliminary computation enables to separate with a good precision the sound from its beginning part. The file given here must be ready beforehand by setting to truefor EXPORTW. For its formats, see 5.1.3 .

SSMETHOD : stringtype. Select a stepsize calculation method based on the higher-order decorrelation. When wishing to fix at the designated value, select FIX. When wishing to set the value that links with the stepsize based on geometric constraints, select LCMYU. When wishing to perform automatic regulation, select ADAPTIVE.

1. When FIX is chosen: set SSMYU.
   SSMYU: floattype. The default value is 0.01. Designate the stepsize at the time of updating a separation matrix based on higher-order decorrelation. Setting this value and LCMYU to zero and passing the separation matrix of the beamformer of Delay and Sum type as INITWFILENAME enables the processing equivalent to the beamformer of the Delay and Sum type.

SSSCAL : floattype. The default value is 1.0. Designate the scale factor of a hyperbolic tangent function (tanh) in calculation of a higher-order correlation matrix. A positive real number greater than zero must be designated. The smaller the value is, the fewer the non-linearity is, which makes the calculation close to a normal correlation matrix calculation.

NOISEFLOOR : floattype. The default value is 0. The user designates the threshold value (upper limit) of the amplitude for judging the input signal as noise. When the amplitude of the input signal is equal to or less than this value, it is judged as a noise section and the separation matrix is not updated. When noise is large, and a separation matrix becomes stable and does not converge, a positive real number is to be designated.

LCCONST : stringtype. Select a method for geometric constraints. When establishing the geometric constraints for focusing on the target sound source direction only, add DIAG to the focus of the target direction, and when forming a blind spot in a non-target sound source direction, select FULL. Since a blind spot is formed automatically by the higher-order decorrelation, a highly precise separation is achieved in DIAG. The default is DIAG.

LCMETHOD : stringtype. Select a stepsize calculation method based on the geometric constraints. When wishing to fix at the designated value, select FIX. When wishing to perform automatic regulation, select ADAPTIVE.

1. When FIX is chosen: Set LCMYU.
   LCMYU: floattype. The default value is 0.001. Designate the stepsize at the time of updating a separation matrix based on the geometric constraints. Setting this value and LCMYU to zero and passing the separation matrix of the beamformer of Delay and Sum type as INITWFILENAME enables the processing equivalent to the beamformer of the Delay and Sum type.

UPDATEMETHODTFCONJ : stringtype. Select ID or POS. The default value is POS. The user designates if updates of the complex conjugate TFCONJ of a transfer function will be performed based on IDs given to each sound source (in the case of ID) or on a source position (in the case of POS)

UPDATEMETHODW : `string`type. Select ID, POS or IDPOS. The default value is ID. When source position information is changed, recalculation of the separation matrix is required. The user designates a method to judge that the source location information has been changed. A separation matrix is saved with a sound source ID corresponded inside and an angle of a sound source direction for some period of time. Even if the sound stops once, when the sound that is judged to be from the same direction is detected, separation processing is performed with the values of the saved separation matrix again. The user sets criteria to judge if such a separation matrix will be updated in the above case. When ID is selected, it is judged if the sound source is in the same direction by the sound source ID. When POS is selected, it is judged by comparing the sound source directions. When IDPOS is selected, if the sound source is judged not to be the same sound source by sound source ID comparison, further judgment is performed by comparing the angles of the sound source direction.

UPDATEACCEPTANGLE : `float`type. The default value is 5. The unit is [deg]. The user sets an allowable error of angles for judging if the sound is from the same direction when POS or IDPOS are selected for UPDATEMETHODTFCONJ and UPDATEMETHODW.

EXPORTW : `bool`type. The default value is `false`. The user determines if results of the separation matrix updated by GHDSSwill be output. When `true`, select EXPORTWFILENAME.

EXPORTWFILENAME : `string`type. This parameter is valid when EXPORTW is set to `true`. Designate a name of a file in which separation matrix is output. For its format, see Section 5.1.3.

Details of the module

Formulation of sound source separation: Table 6.17 shows symbols used for formulation of a sound source separation problem. Meaning of the indices is based on Table 6.1. Since the calculation is performed in a frequency domain, the symbols generally indicate complex numbers in the frequency domain. Parameters, except transfer functions, generally vary with time but in the case of calculation in the same time frame, they are indicated with the time index f abbreviated. Moreover, the following calculation describes the frequency bin k_i. In a practical sense, the calculation is performed for each frequency bin $k_0, \ldots, k_{K-1}$ of K frequencies.

Mixture model The sound that is emitted from N sound sources is affected by the transfer function $\boldsymbol{H}(k_i)$ in the space and observed through M microphones as expressed by Equation (6.25).

$$\boldsymbol{X}(k_i) = \boldsymbol{H}(k_i)\boldsymbol{S}(k_i) + \boldsymbol{N}(k_i). \tag{6.25}$$

The transfer function $\boldsymbol{H}(k_i)$ generally varies depending on shape of the room and positional relations between microphones and sound sources and therefore it is difficult to estimate it. However, ignoring acoustic reflection and diffraction, in the case that a relative position of microphones and sound source is known, the transfer function limited only to the direct sound $\boldsymbol{H}_D(k_i)$ is calculated as expressed in Equation (6.26).

$$H_{Dm,n}(k_i) = \exp\left(-j2\pi l_i r_{m,n}\right) \tag{6.26}$$

$$l_i = \frac{2\pi\omega_i}{c}, \tag{6.27}$$

However, c indicates the sound speed and l_i is the wave number corresponding to the frequency ω_i in the frequency bin k_i. Moreover, $r_{m,n}$ indicates difference between the distance from the microphone m to the sound source n and that from the reference point of the coordinate system (e.g. origin) to the sound source n. In other words, $\boldsymbol{H}_D(k_i)$ is defined as phase difference occurred by the difference of arrival time from the sound source to each microphone.

Separation model The matrix of a complex spectrum of separated sound $\boldsymbol{Y}(k_i)$ is obtained from the following equation.

$$\boldsymbol{Y}(k_i) = \boldsymbol{W}(k_i)\boldsymbol{X}(k_i) \tag{6.28}$$

The GHDSSalgorithm estimates the separation matrix $\boldsymbol{W}(k_i)$ so that $\boldsymbol{Y}(k_i)$ closes to $\boldsymbol{S}(k_i)$.

Assumption in the model Information assumed to be already-known by this algorithm is as follows.

1. The number of sound sources N
2. Source position (The LocalizeMUSICmodule estimates source location in HARK)
3. microphone position
4. Transfer function of the direct sound component $\boldsymbol{H}_D(k_i)$ (measurement or approximation by Equation (6.26))

As unknown information,

1. Actual transfer function at the time of an observation $\boldsymbol{H}(k_i)$
2. Observation noise $\boldsymbol{N}(k_i)$

Update equation of separation matrix GHDSSestimates $\boldsymbol{W}(k_i)$ so that the followings are satisfied.

1. Higher-order decorrelation of the separated signals
   In other words, the diagonal component of the higher-order matrix $\boldsymbol{R}^{\phi(y)y}(k_i) = E[\phi(\boldsymbol{Y}(k_i))\boldsymbol{Y}^H(k_i)]$ of the separated sound $\boldsymbol{Y}(k_i)$ is made 0. Here, the operators ^{H}, $E[]$ and $\phi()$ indicate a hermite transpose, time average operator and nonlinear function, respectively and a hyperbolic tangent function defined by the followings is used in this module.

$$\phi(\boldsymbol{Y}) = [\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_N)]^T \tag{6.29}$$
$$\phi(Y_k) = \tanh(\sigma|Y_k|)\exp(j\angle(Y_k)) \tag{6.30}$$

   Here, σ indicates a scaling factor (corresponds to SSSCAL).

2. The direct sound component is separated without distortions (geometric constraints)
   The product of the separation matrix $\boldsymbol{W}(k_i)$ and the transfer function of the direct sound $\boldsymbol{H}_D(k_i)$ is made a unit matrix ($\boldsymbol{W}(k_i)\boldsymbol{H}_D(k_i) = \boldsymbol{I}$).

The evaluation function that an upper binary element is matched with is as follows. In order to simplify, the frequency bin k_i is abbreviated.

$$J(\boldsymbol{W}) = \alpha J_1(\boldsymbol{W}) + \beta J_2(\boldsymbol{W}), \tag{6.31}$$
$$J_1(\boldsymbol{W}) = \sum_{i \neq j} |R_{i,j}^{\phi(y)y}|^2, \tag{6.32}$$
$$J_2(\boldsymbol{W}) = \boldsymbol{W}\boldsymbol{H}_D - \boldsymbol{I}^2, \tag{6.33}$$

Here, α and β are weighting factors. Moreover, a norm of a matrix is defined as below. $\boldsymbol{M}^2 = tr(\boldsymbol{M}\boldsymbol{M}^H) = \sum_{i,j}|m_{i,j}|^2$

An update equation of the separation matrix to minimize Equation (6.31) is obtained by the gradient method that uses the complex gradient calculation $\frac{\partial}{\partial \boldsymbol{W}^*}$.

$$\boldsymbol{W}(k_i, f+1) = \boldsymbol{W}(k_i, f) - \mu \frac{\partial J}{\partial \boldsymbol{W}^*}(\boldsymbol{W}(k_i, f)) \tag{6.34}$$

Here, μ indicates a stepsize regulating the quantity of update of a separation matrix. Usually, when obtaining a complex gradient of the right-hand side of Equation (6.34), plural frame values are required for expectation value calculation such as $\boldsymbol{R}^{xx} = E[\boldsymbol{X}\boldsymbol{X}^H]$ and $\boldsymbol{R}^{yy} = E[\boldsymbol{Y}\boldsymbol{Y}^H]$. An autocorrelation matrix is not obtained in calculation in the GHDSSmodule. However, Equation (6.35), which uses only one frame, is used.

$$\boldsymbol{W}(k_i, f+1) = \boldsymbol{W}(k_i, f) - \left[\mu_{SS}\frac{\partial J_1}{\partial \boldsymbol{W}^*}(\boldsymbol{W}(k_i, f)) + \mu_{LC}\frac{\partial J_2}{\partial \boldsymbol{W}^*}(\boldsymbol{W}(k_i, f))\right], \tag{6.35}$$
$$\frac{\partial J_1}{\partial \boldsymbol{W}^*}(\boldsymbol{W}) = \left(\phi(\boldsymbol{Y})\boldsymbol{Y}^H - \mathrm{diag}[\phi(\boldsymbol{Y})\boldsymbol{Y}^H]\right)\tilde{\phi}(\mathrm{boldmath}WX)\boldsymbol{X}^H, \tag{6.36}$$
$$\frac{\partial J_2}{\partial \boldsymbol{W}^*}(\boldsymbol{W}) = 2\left(\boldsymbol{W}\boldsymbol{H}_D - \boldsymbol{I}\right)\boldsymbol{H}_D^H, \tag{6.37}$$

Here, $\tilde{\phi}$ is partial differential of ϕ and is defined as follows.

$$\tilde{\phi}(\boldsymbol{Y}) = [\phi(\tilde{Y}_1), \phi(\tilde{Y}_2), \ldots, \phi(\tilde{Y}_N)]^T \tag{6.38}$$

$$\tilde{\phi}(Y_k) = \phi(Y_k) + Y_k \frac{\partial \phi(Y_k)}{\partial Y_k} \tag{6.39}$$

Moreover, $\mu_{SS} = \mu\alpha$ and $\mu_{LC} = \mu\beta$, which are the stepsizes based on the higher-order decorrelation and geometric constraints. The stepsize for which the regulation is set to automatic calculated by the following equation.

$$\mu_{SS} = \frac{J_1(\boldsymbol{W})}{2\frac{\partial J_1}{\partial \boldsymbol{W}}(\boldsymbol{W})^2} \tag{6.40}$$

$$\mu_{LC} = \frac{J_2(\boldsymbol{W})}{2\frac{\partial J_2}{\partial \boldsymbol{W}}(\boldsymbol{W})^2} \tag{6.41}$$

The indices of each parameter in Equations (6.36, 6.37) are (k_i, f), which are abbreviated above. The initial values of a separation matrix are obtained as follows.

$$\boldsymbol{W}(k_i) = \boldsymbol{H}_D^H(k_i)/M, \tag{6.42}$$

Here, M indicates the number of microphones. **Processing flow** The main processing in the time frame f in the GHDSSmodule

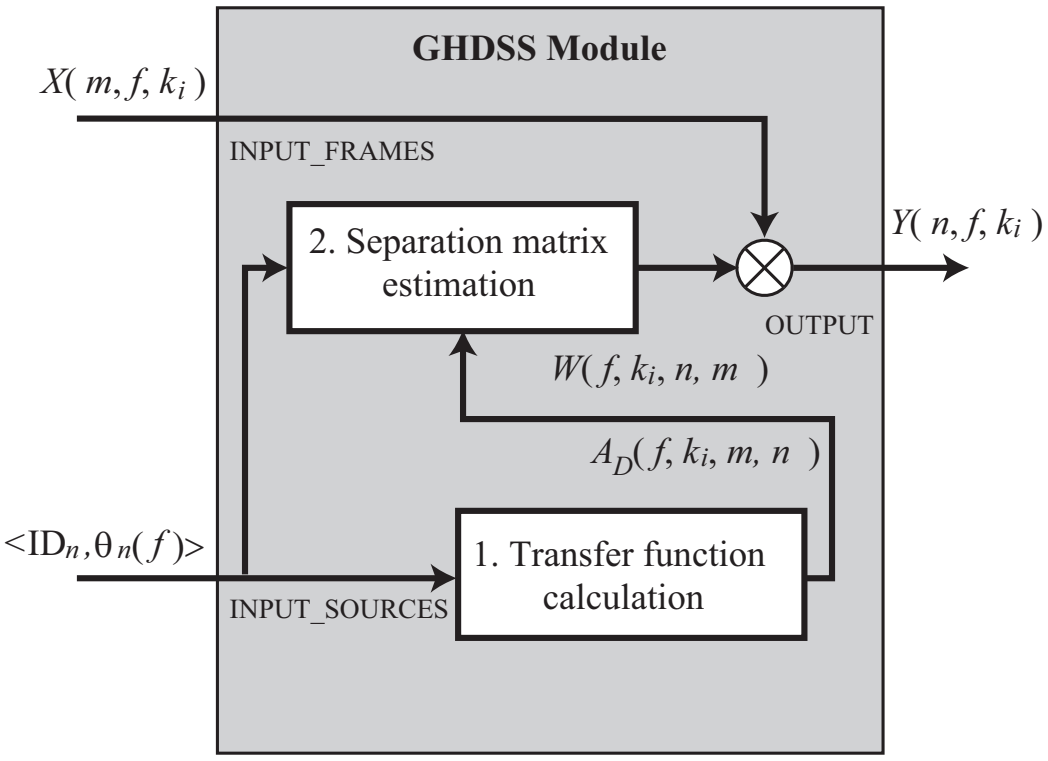


Figure 6.25: Flowchart of GHDSS

is shown in Figure 6.25. In detail, processing related with fixed noise is included as follows.

1. Presence of fixed noise is checked
2. Acquisition of a transfer function (direct sound)
3. Separation matrix $\boldsymbol{W}$ estimation

4. Sound source separation processing is executed in accordance with Equation (6.28)
5. Writing of a separation matrix (When EXPORTW is set to `true`)

Check presence of fixed noise: FIXEDNOISE is set to `true`, If there is sound source from a fixed noise direction in a source localization result, the sound source separation is performed under the condition that -1 is given as an ID. **Acquisition**

of transfer function: The initial value of a transfer function is different for each parameter TFCONJ.
When the parameter TFCONJ is set to CALC, the transfer function $\boldsymbol{H}_D$ is calculated based on Equation (6.26) with the input source location result and the microphone position coordinate designated for the parameter MICFILENAME.
When the parameter TFCONJ is set to DATABASE, the data most close to the direction of the input source localization result are searched for from the transfer function designated in the parameter TFCONJFILENAME.
Processing after the second frame is as follows. The flow till obtaining a transfer function is shown in Figure 6.26.

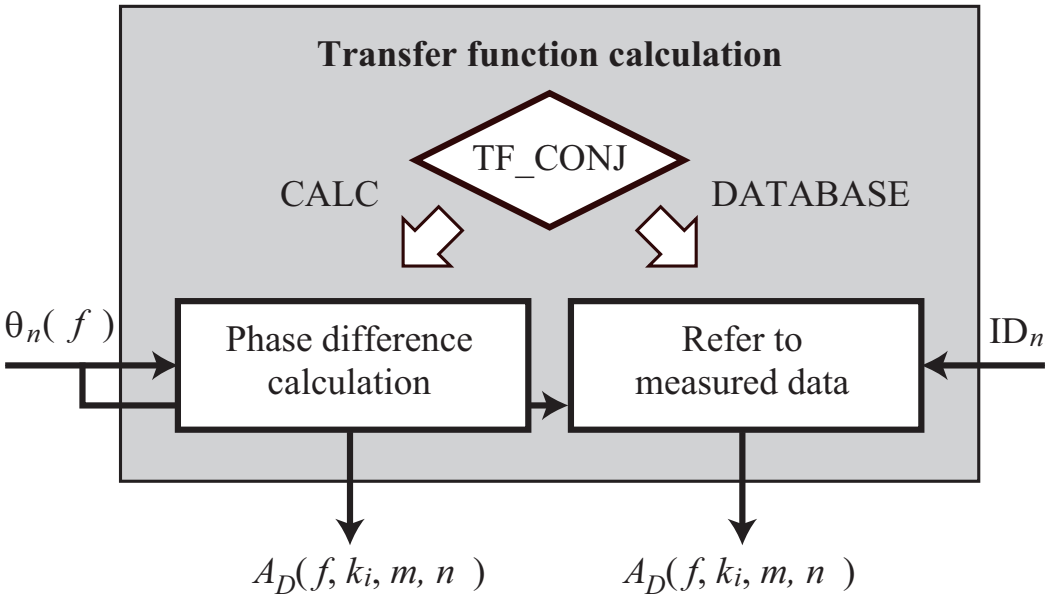


Figure 6.26: Flowchart of transfer function calculation

- When the parameter TFCONJ is set to CALC, the transfer function $\boldsymbol{H}_D$ is calculated based on Equation (6.26) with the input source localization result.
- When the parameter TFCONJ is set to DATABASE, it is determined if the transfer function of the former frame is succeeded or files are read with a value of UPDATEMETHODTFCONJ as follows.

UPDATEMETHODTFCONJ is ID

1. The acquired ID is compared with the ID one frame before.
   - Same: Succeed
   - Different: Read

UPDATEMETHODTFCONJ is POS

1. The acquired direction is compared with the sound source direction one frame before
   - Error is less than UPDATEACCEPTANGLE: Succeed
   - Error is more than UPDATEACCEPTANGLE: Read

Estimation of separation matrix: The initial value of a separation matrix is different depending on if the user designates a value for the parameter INITWFILENAME.
When the parameter INITWFILENAME is not designated, the separation matrix $\boldsymbol{W}$ calculated from the transfer function $\boldsymbol{H}_D$.
When the parameter INITWFILENAME is designated, data most close to the direction of the input source localization result are searched for from the designated separation matrix.
Processing after the second frame is as follows. The flow till estimating a separation matrix is shown in Figure 6.27. Here,

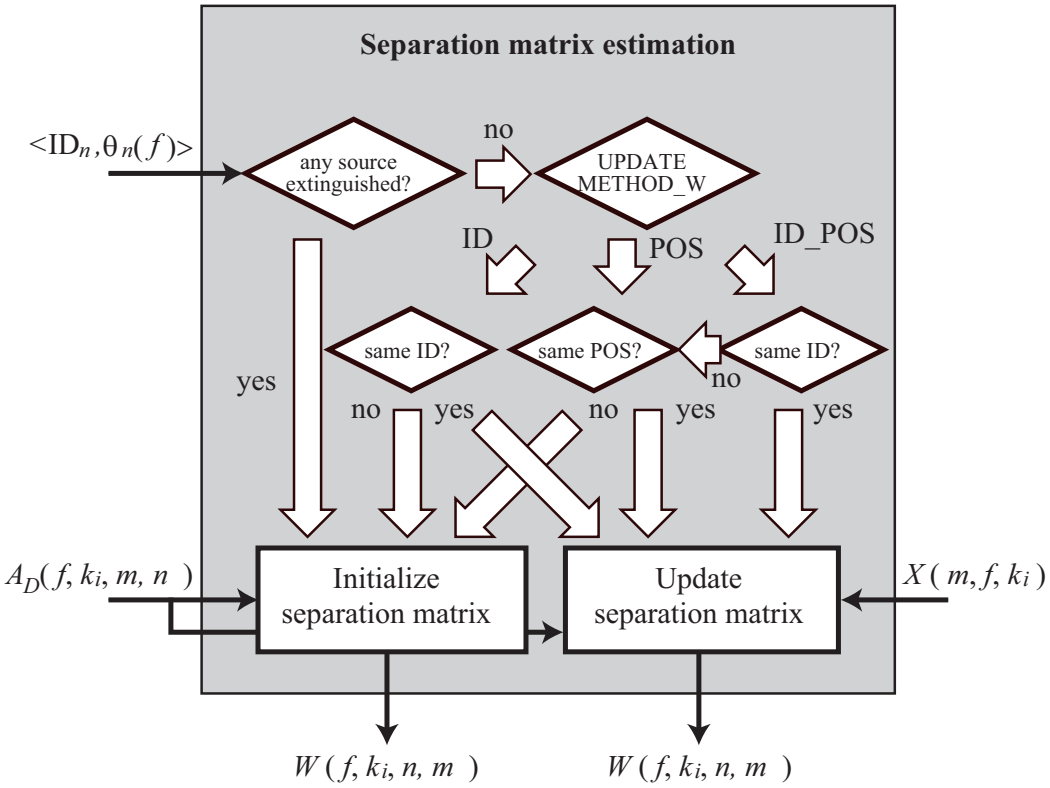


Figure 6.27: Flowchart of separation matrix estimation

the former frame is updated based on Equation (6.35) or an initial value of the separation matrix is derived by the transfer function based on Equation (6.42).

- When it is found that there is a sound source disappeared by referring to source localization information of the former frame, the separation matrix is initialized.
- When the number of sound sources does not change, the separation matrix diverges by the value of UPDATEMETHODW. The sound source ID and localization direction in the former frame are compared with those of the current to determine if the separation matrix will be used continuously or initialized.

UPDATEMETHODW is ID

1. Compare with the former frame ID
   - Same: Update $\boldsymbol{W}$
   - Different: Initialize $\boldsymbol{W}$

UPDATEMETHODW is POS

1. Compare with the former frame localization direction
   - Error is less than UPDATEACCEPTANGLE: Update $\boldsymbol{W}$
   - Error is more than UPDATEACCEPTANGLE: Initialize $\boldsymbol{W}$

UPDATEMETHODW is IDPOS

1. Compare with the former frame ID
   - Same: Update $\boldsymbol{W}$
2. Localization directions are compared when IDs are different
   - Error is less than UPDATEACCEPTANGLE: Update $\boldsymbol{W}$
   - Error more than UPDATEACCEPTANGLE: Initialize $\boldsymbol{W}$

Writing of a separation matrix (When EXPORTW is set to `true`) When EXPORTW is set to `true`, a converged separation matrix is output to a file designated for EXPORTWFILENAME.
When plural sound sources are detected, all those separation matrixes are output to one file. At the time the sound source is disappeared, its separation matrix is written to a file.
When written to a file, it is determined to overwrite the existing sound source or add the sound source as a new sound source by comparing localization directions of the sound sources already saved.

Disappearance of sound source

1. Compare with localization direction of the sound source already saved
   - Error is less than UPDATEACCEPTANGLE: Overwrite and save $\boldsymbol{W}$
   - Error more than UPDATEACCEPTANGLE: Save additionally $\boldsymbol{W}$

Table 6.16: Parameter list of GHDSS

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	Analysis frame length.
ADVANCE	int	160	[pt]	Shift length of frame.
SAMPLINGRATE	int	16000	[Hz]	Sampling frequency.
LOWERBOUNDFREQUENCY	int	0	[Hz]	The minimum value of the frequency used for separation processing
UPPERBOUNDFREQUENCY	int	8000	[Hz]	The maximum value of the frequency used for separation processing
TFCONJ	string	CALC		Select CALC or DATABASE
TFCONJ==DATABASE				The following is valid when is DATABASE is chosen for TFCONJ.
TFCONJFILENAME	string			A file name in which a transfer function of the microphone array is described.
TFCONJ==CALC				The following is valid when is CALC is chosen for TFCONJ.
MICFILENAME	string			A file name in which a coordinate of the microphone position is described.
MICPOSSHIFT	string	FIX		Determine whether or not the origin of the microphone coordinate shifts. Select FIX or SHIFT. When FIX, an origin of microphone does not vary. When SHIFT, a gravity center of the microphone coordinate is assumed to be the origin.
SPEEDSOUND	float	343	[m/s]	Speed of sound.
FIXEDNOISE	bool	false		Designate whether or not the fixed noise sound source is used.
FIXEDNOISE==true				The following is valid when trueis chosen for FIXEDNOISE.
FIXEDNOISEFILENAME	string			A file name in which a coordinate of the fixed noise source position.
INITWFILENAME	string			A file name in which initial value of the separation matrix is described.
SSMETHOD	string	ADAPTIVE		The stepsize calculation method based on the higher-order decorrelation. Select FIX, LCMYU or ADAPTIVE. FIX indicates fixed values, LCMYU indicates the value that links with the stepsize based on geometric constraints and ADAPTIVE indicates automatic regulation.
SSMETHOD==FIX				The following is valid when FIX is chosen for SSMETHOD.
SSMYU	float	0.001		The stepsize based on the higher-order decorrelation at the time of updating a separation matrix
SSSCAL	float	1.0		The scale factor in a higher-order correlation matrix computation
NOISEFLOOR	float	0.0		The threshold value of the amplitude for judging the input signal as noise(upper limit)
LCCONST	string	DIAG		Determine geometric constraints. Select DIAG or FULL. In DIAG, the focus is in the target sound source direction. In FULL, blind spots are formed in non-target sound source directions in addition to the focus in the target direction.
LCMETHOD	string	ADAPTIVE		The stepsize calculation method based on geometric constraints. Select FIX or ADAPTIVE. FIX indicates fixed values and ADAPTIVE indicates automatic regulation.
LCMETHOD==FIX				
LCMYU	float	0.001		The stepsize at the time of updating a separation matrix based on the higher-order decorrelation,
UPDATEMETHODTFCONJ	string	POS		Designate a method to update transfer functions. Select POS or ID.
UPDATEMETHODW	string	ID		Designate a method to update separation matrixes. Select ID, POS or IDPOS.
UPDATEACCEPTANGLE	float	5	[deg]	The threshold value of angular difference for judging a sound source as identical to another in sepa-

Table 6.17: Definitions of the parameters

Parameter	Description
$\boldsymbol{S}(k_i) = [S_1(k_i), \ldots, S_N(k_i)]^T$	The sound source complex spectrum corresponding to the frequency bin k_i.
$\boldsymbol{X}(k_i) = [X_1(k_i), \ldots, X_M(k_i)]^T$	The vector of a microphone observation complex spectrum, which corresponds to INPUTFRAMES.
$\boldsymbol{N}(k_i) = [N_1(k_i), \ldots, N_M(k_i)]^T$	The additive noise that acts on each microphone.
$\boldsymbol{H}(k_i) = [H_{m,n}(k_i)]$	The transfer function matrix including reflection and diffraction ($M \times N$).
$\boldsymbol{H}_D(k_i) = [H_{Dm,n}(k_i)]$	The transfer function matrix of direct sound ($M \times N$).
$\boldsymbol{W}(k_i) = [W_{n,m}(k_i)]$	The separation matrix ($N \times M$).
$\boldsymbol{Y}(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	The separation sound complex spectrum.
μ_{SS}	The stepsize at the time of updating a separation matrix based on the higher-order decorrelation, which co
μ_{LC}	The stepsize at the time of updating a separation matrix based on geometric constraints, which correspon

6.3.6 HRLE

Outline of the module

This module estimates a stationary noise level by the Histogram-based Recursive Level Estimation (HRLE) method. HRLE calculates histograms (frequency distribution) of input spectra and estimates a noise level from the normalization accumulation frequency designated by the cumulative distribution and parameter *Lx*. A histogram is calculated by a past input spectrum weighted with an exponent window and a position of the exponent window is updated every one frame.

Necessary file

No files are required

Usage

When to use

This module is used when noise is suppressed by spectrum subtraction

Typical connection

As shown in Figure 6.28, the input is connected after separation modules such as GHDSSand the output is connected to the modules that obtain an optimal gain such as CalcSpecSubGain. Figure 6.29 shows a connection example when EstimateLeakis used together.

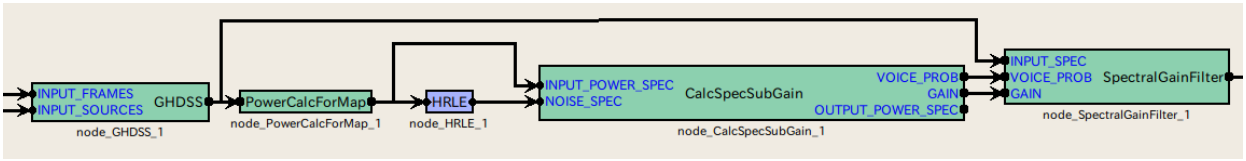


Figure 6.28: Connection example of HRLE1

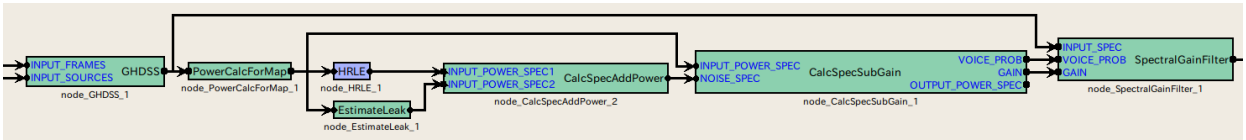


Figure 6.29: Connection example of HRLE2

Input-output and property of the module

Table 6.18: Parameter list of HRLE

Parameter name	Type	Default value	Unit	Description
LX	float	0.85		Normalization accumulation frequency (*Lx* value).
TIMECONSTANT	float	16000	[pt]	Time constant.
NUMBIN	float	1000		Number of bins of a histogram.
MINLEVEL	float	-100	[dB]	The minimum level of a histogram.
STEPLEVEL	float	0.2	[dB]	Width of a bottle of a histogram.
DEBUG	bool	false		Debugging mode.

Input

INPUTSPEC : Map<int, float>type. Power spectrum of input signal

Output

NOISESPEC : Map<int, float>type. Power spectrum of estimated noise

Parameters

LX : floattype. The default value is 0.85. Normalization accumulation frequency on an accumulation frequency distribution is designated in the range from 0 to 1. When designating 0, the minimum level is estimated. When designating 1, the maximum level is estimated. Median is estimated when 0.5 are designated.

TIMECONSTANT : floattype. The default value is 16000. A time constant (more than zero) is designated in time sample unit.

NUMBIN : floattype. The default value is 1000. Designate the number of bins of a histogram.

MINLEVEL : floattype. The default value is -100. Designate the minimum level of a histogram in dB.

STEPLEVEL : floattype. The default value is 0.2. Designate a width of a bin of a histogram in dB.

DEBUG : boolThe default value is `false`. Designate the debugging mode. In the case of the debugging mode (`true`), values of the cumulative histogram are output once every 100 frames as a standard output in the comma-separated text file format. Output values are in the complex matrix value format with plural rows and columns The rows indicate positions of frequency bins and columns indicate positions of histograms. Each element indicates the complex values separated with parenthesis (right side is for real numbers and left side is for imaginaries). (since the cumulative histogram is expressed with real numbers and imaginary parts are usually 0. However, it does not necessarily mean that it will be 0 in the future version.) The additional value of a cumulative histogram for one sample is not 1 and they increase exponentially (for speedup). Therefore, note that cumulative histogram values do not indicate accumulation frequency itself. Most of the cumulative histogram values in each row are 0. When values are contained only in the positions that are close to the last column, the input values are great, exceeding the level range of the set histogram (overflow status). Therefore, part or all of NUMBIN, MINLEVEL and STEPLEVEL must be set to high values. On the other hand, when most of the cumulative histogram values of each row are constant values and different low values are contained only in the positions that are close to the first column, the input values are small below the level range of the set histogram (underflow status). Therefore, MINLEVEL must be set to low values. Example of the output:

```
---------- Compmat.disp()
----------
[(1.00005e-18,0), (1.00005e-18,0), (1.00005e-18,0), ...
, (1.00005e-18,0);
(0,0), (0,0), (0,0), ...
, (4.00084e-18,0);
...
(4.00084e-18,0), (4.00084e-18,0), (4.00084e-18,0), .., , (4.00084e-18,0)]
^T
Matrix size = 1000 x 257
```

Details of the module

Figure 6.30 shows a processing flow of HRLE. HRLEobtained a histogram of a level is from the input power and estimates the Lx level from the cumulative distribution. The Lx level, as shown in Figure 6.31, is the level that normalization accumulation frequency in an accumulation frequency distribution becomes x. x is a parameter. For example, when $x = 0$, the minimum value is estimated, when $x = 1$, maximum value is estimated and when $x = 0.5$, a median is estimated in its processing. The details of the processing by HRLEare expressed by the following seven equations (corresponding to Figure 6.30). In the equations, t indicates time (frame), y_p indicates input power (INPUTSPEC) and n_p indicates estimated

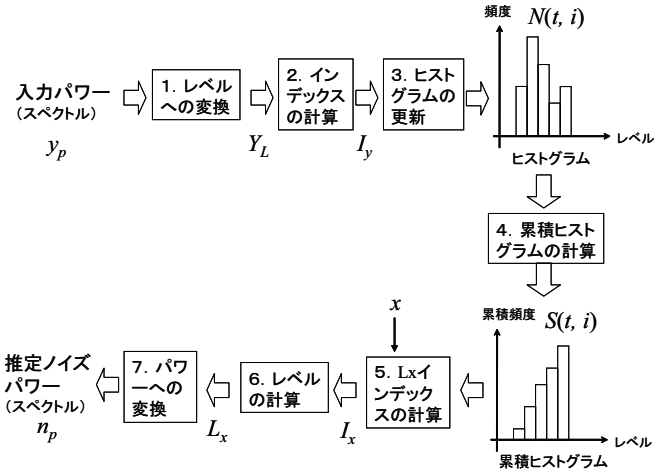


Figure 6.30: Processing flow of HRLE

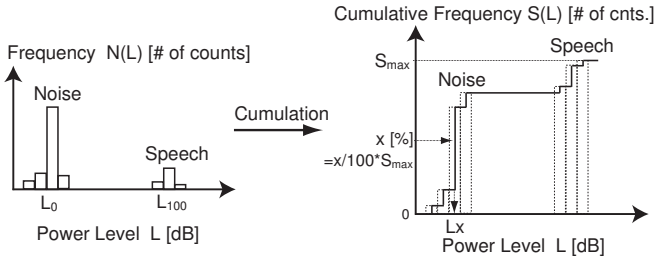


Figure 6.31: Estimation of Lx value

noise power (NOISESPEC]). x, α, L_{min} and L_{step} are the parameters related to histograms and indicate normalization accumulation frequency (LX), time constant (TIMECONSTANT), the minimum level (MINLEVEL) of a bin, and a level width (STEPLEVEL) of a bin, respectively. $\lfloor a \rfloor$ indicates an integer most close to a below a. Moreover, all variables except the parameters are functions of frequency and the same processing is performed independently for every frequency. In the equations, frequency is abbreviated for simplification.

$$Y_L(t) = 10 \log_{10} y_p(t), \tag{6.43}$$

$$I_y(t) = \lfloor (Y_L(t) - L_{min})/L_{step} \rfloor, \tag{6.44}$$

$$N(t, l) = \alpha N(t-1, l) + (1-\alpha)\delta(l - I_y(t)), \tag{6.45}$$

$$S(t, l) = \sum_{k=0}^{l} N(t, k), \tag{6.46}$$

$$I_x(t) = \underset{I}{\operatorname{argmin}} \left[S(t, I_{max}) \frac{x}{100} - S(t, I) \right], \tag{6.47}$$

$$L_x(t) = L_{min} + L_{step} \cdot I_x(t), \tag{6.48}$$

$$n_p(t) = 10^{L_x(t)/10} \tag{6.49}$$

References

(1) H.Nakajima, G. Ince, K. Nakadai and Y. Hasegawa: “ An Easily-configurable Robot Audition System using Histogram-based Recursive Level Estimation ”, Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2010 (to be appeared).

6.3.7 PostFilter

Outline of the module

This module performs postprocessing to improve the accuracy of speech recognition with the sound source separation module GHDSSfor a separated complex spectrum. At the same time, it generates noise power spectra to generate Missing Feature Masks.

Necessary file

No files are required.

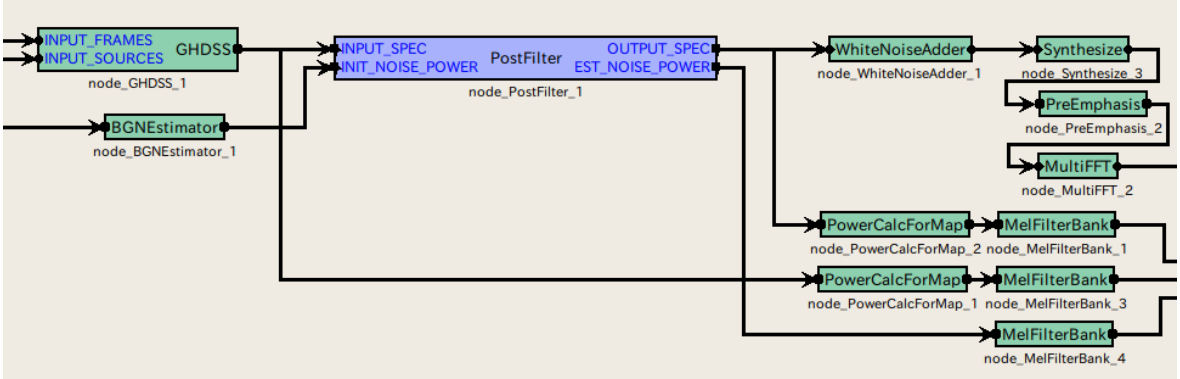
When to use

This module is used to form the spectrum that are separated by the GHDSSmodule and generate the noise spectra required to generate Missing Feature Masks.

Typical connection

Figure **??** shows a connection example of the PostFiltermodule. For connections at the input, INPUTSPEC is connected to the output of the GHDSSmodule and INITNOISEPOWER is connected to the output of the BGNEstimatormodule. For the output, Figure **??** shows connection examples of the followings

1. Speech feature extraction from separated sound (OUTPUTSPEC) (MSLSExtractionmodule)
2. Generation of Missing Feature Masks from separated sound and power (ESTNOISEPOWER) of noise contained in it at the time of speech recognition (MFMGenerationmodule)



Input-output and property of the module

Input

INPUTSPEC : `Map<int, ObjectRef>`type. The type same as the output from the GHDSSmodule. A pair of a sound source ID and a complex spectrum of the separated sound as `Vector<complex<float> >`type data.

INPUTNOISEPOWER : `Matrix<float>`type. A power spectrum of the stationary noise estimated by the BGNEstimatormodule.

Output

OUTPUTSPEC : `Map<int, ObjectRef>`type. A complex spectrum for which noise is removed from the input INPUTSPEC.

ESTNOISEPOWER : `Map<int, ObjectRef>`type. Power of the noise estimated to be contained is paired with IDs as `Vector<float>`type data for each separation sound of OUTPUTSPEC.

Parameter

Details of the module

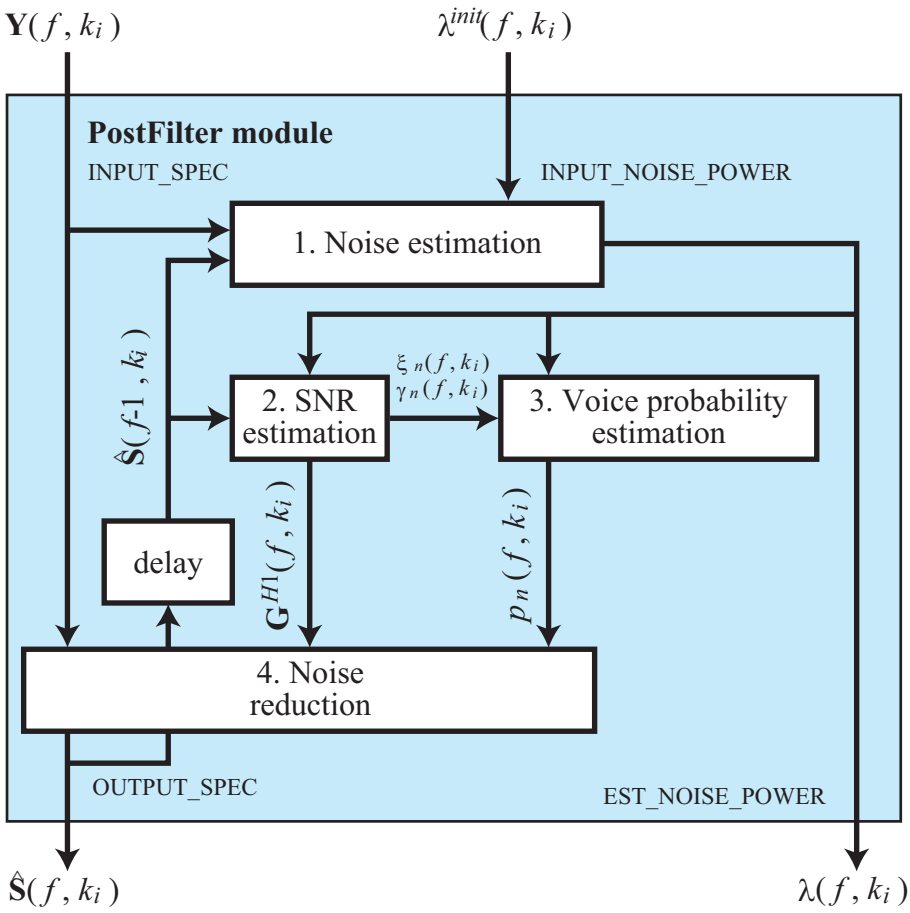


Figure 6.32: Flowchart of PostFilter

The subscripts used in the equations are based on the definitions in Table 6.1. Moreover, the time frame index f is abbreviated in the following equations unless especially needed. Figure 6.32 shows a flowchart of the PostFiltermodule. A separated sound spectrum from the GHDSSmodule and a stationary noise power spectrum of the BGNEstimatormodule are obtained as inputs. Outputs are the separated sound spectrum for which the speech is emphasized and a power spectrum of noise mixed with separated sound. The processing flow is as follows.

1. Noise estimation
2. SNR estimation
3. Speech presence probability estimation
4. Noise removal

Table 6.19: Parameter list of PostFilter(first half)

Parameter name	Type	Default value	Unit
MCRASETTING	bool	false	
MCRASETTING			The followings are valid when MCRASETTING is set to tr
STATIONARYNOISEFACTOR	float	1.2	
SPECSMOOTHFACTOR	float	0.5	
AMPLEAKFACTOR	float	1.5	
STATIONARYNOISEMIXTUREFACTOR	float	0.98	
LEAKFLOOR	float	0.1	
BLOCKLENGTH	int	80	
VOICEPTHRESHOLD	float	3	
ESTLEAKSETTING	bool	false	
ESTLEAKSETTING			
LEAKFACTOR	float	0.25	
OVERCANCELFACTOR	float	1	
ESTREVSETTING	bool	false	
ESTREVSETTING			
REVERBDECAYFACTOR	float	0.5	
DIRECTDECAYFACTOR	float	0.2	
ESTSNSETTING	bool	false	
ESTSNSETTING			
PRIORSNRFACTOR	float	0.8	
VOICEPPROBFACTOR	float	0.9	
MINVOICEPPROB	float	0.05	
MAXPRIORSNR	float	100	
MAXOPTGAIN	float	20	
MINOPTGAIN	float	6	

Table 6.20: Parameter list of PostFilter(latter half)

Parameter name	Type	Default value	Unit
ESTVOICEPSETTING	bool	false	
ESTVOICEPSETTING			The followings are valid when ESTVOICEPSETTINGis set to true
PRIORSNRSMOOTHFACTOR	float	0.7	
MINFRAMESMOOTHSNR	float	0.1	
MAXFRAMESMOOTHSNR	float	0.316	
MINGLOBALSMOOTHSNR	float	0.1	
MAXGLOBALSMOOTHSNR	float	0.316	
MINLOCALSMOOTHSNR	float	0.1	
MAXLOCALSMOOTHSNR	float	0.316	Maximum value of the frequency smoothing SNR (local).
UPPERSMOOTHFREQINDEX	int	99	
LOWERSMOOTHFREQINDEX	int	8	
GLOBALSMOOTHBANDWIDTH	int	29	
LOCALSMOOTHBANDWIDTH	int	5	
FRAMESMOOTHSNRTHRESH	float	1.5	
MINSMOOTHPEAKSNR	float	1.0	
MAXSMOOTHPEAKSNR	float	10.0	
FRAMEVOICEPPROBFACTOR	float	0.7	Speech probability smoothing coefficient (frame).
GLOBALVOICEPPROBFACTOR	float	0.9	Speech probability smoothing coefficient (global).
LOCALVOICEPPROBFACTOR	float	0.9	
MINVOICEPAUSEPROB	float	0.02	
MAXVOICEPAUSEPROB	float	0.98	Maximum value of speech quiescent probability.

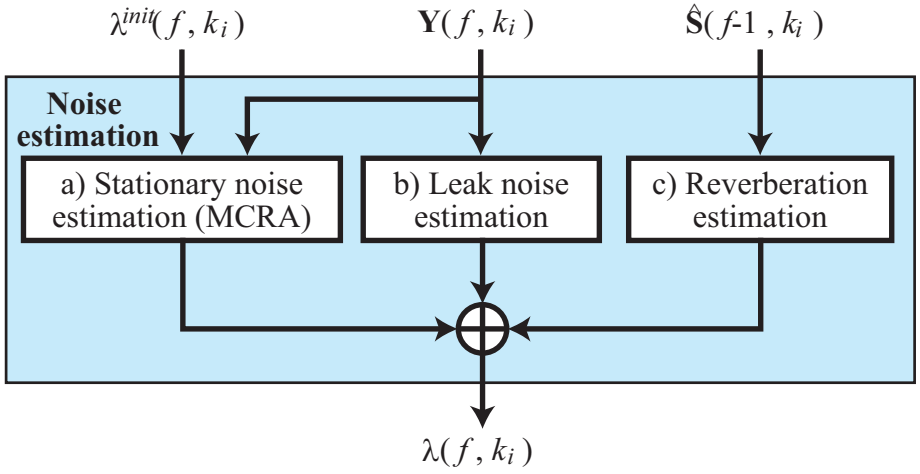


Figure 6.33: Procedure of noise estimation

1) Noise estimation: Figure 6.33 shows the processing flow of noise estimation . The following three are the noise that the PostFiltermodule processes.
a) The stationary noise for which contact points of microphones are a factor,
b) The sound of other sound sources that cannot be completely removed (leakage noise),
c) Reverberant of the former frame,

The noise contained in the final separated sound $\boldsymbol{\lambda}(f, k_i)$ is obtained by the following equation.

$$\boldsymbol{\lambda}(f, k_i) = \boldsymbol{\lambda}^{sta}(f, k_i) + \boldsymbol{\lambda}^{leak}(f, k_i) + \boldsymbol{\lambda}^{rev}(f-1, k_i) \tag{6.50}$$

However, $\boldsymbol{\lambda}^{sta}(f, k_i), \boldsymbol{\lambda}^{leak}(f, k_i) and \boldsymbol{\lambda}^{rev}(f-1, k_i)$ indicate reverberant of stationary noise, leakage noise and the former frame, respectively.

1-a) Stationary noise estimation by MCRA method The parameters used in 1-a) are based on Table 6.21. First, calculate the power spectrum for which the input spectrum is smoothed with the power one frame before. $\boldsymbol{S}(f, k_i) = [S_1(f, k_i), \ldots, S_N(f, k_i)]$.

$$S_n(f, k_i) = \alpha_s S_n(f-1, k_i) + (1-\alpha_s)|Y_n(k_i)|^2 \tag{6.51}$$

Next, update $\boldsymbol{S}^{tmp}$, $\boldsymbol{S}^{min}$.

$$S_n^{min}(f, k_i) = \begin{cases} \min\{S_n^{min}(f-1, k_i), S_n(f, k_i) & \text{if } f \neq nL \\ \min\{S_n^{tmp}(f-1, k_i), S_n(f, k_i) & \text{if } f = nL \end{cases}, \tag{6.52}$$

$$S_n^{min}(f, k_i) = \begin{cases} \min\{S_n^{tmp}(f-1, k_i), S_n(f, k_i) & \text{if } f \neq nL \\ S_n(f, k_i) & \text{if } f = nL \end{cases}, \tag{6.53}$$

Here, n indicates an arbitrary integer. $\boldsymbol{S}^{min}$ maintains the minimum power after the noise estimation begins $\boldsymbol{S}^{tmp}$ maintains an extremely small power of a recent frame. $\boldsymbol{S}^{tmp}$ is updated for every L frame. Next, judge if speech is contained based on

Table 6.21: Definition of variable

Parameter	Description, Corresponding parameter
$\boldsymbol{Y}(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	Complex spectrum of separated sound corresponding to the frequency bin k_i
$\boldsymbol{\lambda}^{init}(k_i) = \left[\lambda_1^{init}(k_i), \ldots, \lambda_N^{init}(k_i)\right]^T$	Initial value power spectrum used for the stationary noise estimation
$\boldsymbol{\lambda}^{sta}(k_i) = \left[\lambda_1^{sta}(k_i), \ldots, \lambda_N^{sta}(k_i)\right]^T$	Estimated stationary noise power spectrum.
α_s	Smoothing coefficient of the input power spectrum. Parameter SPECSMOOTHFACTOR. The default value is 0.5
$\boldsymbol{S}^{tmp}(k_i) = \left[S_1^{tmp}(k_i), \ldots, S_N^{tmp}(k_i)\right]$	Temporary parameter for minimum power calculation.
$\boldsymbol{S}^{min}(k_i) = \left[S_1^{min}(k_i), \ldots, S_N^{min}(k_i)\right]$	The parameter that maintains the minimum power.
L	Maintained frame numbers of $\boldsymbol{S}_{tmp}$. Parameter BLOCKLENGTH. The default value is 80
δ	Threshold value of speech presence judgment. Parameter VOICEPTHRESHOLD. The default value is 3.0
α_d	Mixing ratio of estimated stationary noise. Parameter STATIONARYNOISEMIXTUREFACTOR. The default value is 0.98
$\boldsymbol{Y}^{leak}(k_i)$	Power spectrum of leakage noise estimated, to be contained in separated sound
q	Coefficient for when leakage noise is removed from the input separated sound power. Parameter AMPLEAKFACTOR. The default value is 1.5.
S_{floor}	Minimum value of leakage noise. Parameter LEAKFLOOR. The default value is 0.1.
r	Coefficient at the time of stationary noise estimation. Parameter STATIONARYNOISEFACTOR. The default value is 1.2

the power ratio of the minimum power and the input separated sound.

$$S_n^r(k_i) = \frac{S_n(k_i)}{S^{min}(k_i)}, \tag{6.54}$$

$$I_n(k_i) = \begin{cases} 1 & \text{if } S_n^r(k_i) > \delta \\ 0 & \text{if } S_n^r(k_i) \leq \delta \end{cases} \tag{6.55}$$

When $I_n(k_i)$ included in speech, this value is 1 and when it is not included, it is 0. $I_n(k_i)$, $I_n(k_i)$ Based on this judgment result, determine the mixing ratio $\alpha_{d,n}^C(k_i)$ of the frame stationary noise of the former frame and power of the current frame.

$$\alpha_{d,n}^C(k_i) = (\alpha_d - 1)I_n(k_i) + 1. \tag{6.56}$$

Next, subtract leakage noise contained in the power spectrum of the separated sound.

$$S_n^{leak}(k_i) = \sum_{p=1}^{N} |Y_p(k_i)|^2 - |Y_n(k_i)|^2, \tag{6.57}$$

$$S_n^0(k_i) = |Y_n(k_i)|^2 - qS_n^{leak}(k_i), \tag{6.58}$$

Here, when $S_n^0(k_i) < S_{floor}$, the valued is changed to below.

$$S_n^0(k_i) = S_{floor} \tag{6.59}$$

Obtain stationary noise of the current frame by mixing the power spectrum with leakage noise removed $S_n^0(f, k_i)$ and the estimated stationary noise of the former frame $\boldsymbol{\lambda}^{sta}(f-1, k_i)$ or $bf\lambda^{init}(f, k_i)$, which is the output from BGNEstimator.

$$\lambda_n^{sta}(f, k_i) = \begin{cases} \alpha_{d,n}^C(k_i)\lambda_n^{sta}(f-1, k_i) + (1 - \alpha_{d,n}^C(k_i)rS_n^0(f, k_i) & nochangeinsourceposition \\ \alpha_{d,n}^C(k_i)\lambda_n^{init}(f, k_i) + (1 - \alpha_{d,n}^C(k_i)rS_n^0(f, k_i) & \text{ifChange in source position} \end{cases} \tag{6.60}$$

Table 6.22: Definition of variable

Variable	Description, Corresponding parameter
$\boldsymbol{\lambda}^{leak}(k_i)$	Power spectrum of leakage noise. Vector comprising elements of each separated sound.
α^{leak}	Leakage rate for the total of separated sound power. LEAKFACTOR × OVERCANCELFACTOR
$S_n(f, k_i)$	Smoothing power spectrum obtained by Equation (6.51)

Table 6.23: Definition of variable

Variable	Description, Corresponding parameter
$\boldsymbol{\lambda}^{rev}(f, k_i)$	Power spectrum of reverberant in the time frame f
$\hat{\boldsymbol{S}}(f-1, k_i)$	

1-b)Leakage noise estimation The variables used in 1-b) are based on Table 6.22.
Some parameters are calculated as follows.

$$\beta = -\frac{\alpha^{leak}}{1-(\alpha^{leak})^2+\alpha^{leak}(1-\alpha^{leak})(N-2)} \tag{6.61}$$

$$\alpha = 1-(N-1)\alpha^{leak}\beta \tag{6.62}$$

With this parameter, mix the smoothed spectrum $\boldsymbol{S}(k_i)$, the power spectrum for which the power of the own separated sound is removed from the power of other separated sound $S_n^{leak}(k_i)$ obtained by Equation (6.57).

$$Z_n(k_i) = \alpha S_n(k_i) + \beta S_n^{leak}(k_i), \tag{6.63}$$

Here, when $Z_n(k_i) < 1$, assume $Z_n(k_i) = 1$. The power spectrum of final leakage noise $\boldsymbol{\lambda}^{leak}(k_i)$ is obtained as follows.

$$\lambda_n^{leak} = \alpha^{leak}\left(\sum_{n'\neq n} Z_{n'}(k_i)\right) \tag{6.64}$$

1-c) Reverberant estimation The variables used in 1-c) are based on Table 6.23.

$$\lambda_n^{leak} = \alpha^{leak}\left(\sum_{n'\neq n} Z_{n'}(k_i)\right) \tag{6.65}$$

$$\lambda_n^{rev}(f, k_i) = \gamma\left(\lambda_n^{rev}(f-1, k_i) + \Delta|\hat{S}_n(f-1, k_i)|^2\right) \tag{6.66}$$

2)SNR estimation: Figure 6.34 shows the flow of the SNR estimation. The SNR estimation consists of the followings
a) Calculation of SNR
b) Preliminary SNR estimation before noise mixture
c) Estimation of a speech content rate
d) Estimation of an optimal gain

The vector elements in Table 6.24 indicate value of each separated sound.

2-a) Calculation of SNR The variables used in 2-a) are based on Table 6.24. Here, SNR $\gamma_n(k_i)$ is calculated based on the complex spectra $\boldsymbol{Y}(k_i)$ of an input and the power spectrum of the noise estimated in above.

$$\gamma_n(k_i) = \frac{|Y_n(k_i)|^2}{\lambda_n(k_i)} \tag{6.67}$$

$$\gamma_n^C(k_i) = \begin{cases} \gamma_n(k_i) & \text{if } \gamma_n(k_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.68}$$

Here, when $\gamma_n(k_i) < 0$ is satisfied, $\gamma_n(k_i) = 0$.

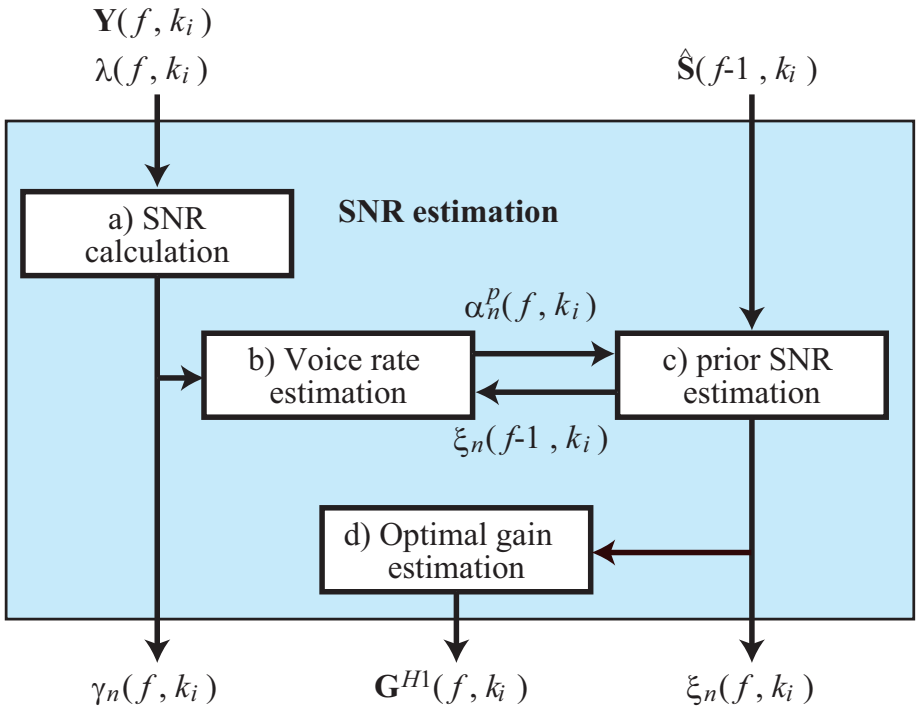


Figure 6.34: Procedure of SNR estimation

Table 6.24: Definition of major variable

Variable	Description, corresponding parameter
$\boldsymbol{Y}(k_i)$	Complex spectra of the separated sound, which is an input of the PostFiltermodule
$\hat{\boldsymbol{S}}(k_i)$	Complex spectra of the formed separated sound, which is an output of the PostFiltermodule
$\boldsymbol{\lambda}(k_i)$	Power spectrum of noise estimated above
$\gamma_n(k_i)$	SNR of the separated sound n
$\alpha_n^p(k_i)$	Speech content rate
$\xi_n(k_i)$	Preliminary SNR
$\boldsymbol{G}^{H1}(k_i)$	Optimal gain to improve SNR of the separated sound

2-b)Estimation of speech content rate The variables used in 2-b) are based on Table 6.25. The speech content rate

Table 6.25: Definition of variable

Variable	Description, corresponding parameter
α_{mag}^p	Preliminary SNR coefficient. Parameter VOICEPPROBFACTOR. The default value is 0.9.
α_{min}^p	Minimum speech content rate. Parameter MINVOICEPPROB. The default value is 0.05.

$\alpha_n^p(f, k_i)$ is calculated as follows, with the preliminary SNR $\xi_n(f-1, k_i)$ of the former frame.

$$\alpha_n^p(f, k_i) = \alpha_{mag}^p \left(\frac{\xi_n(f-1, k_i)}{\xi_n(f-1, k_i)+1} \right)^2 + \alpha_{min}^p \tag{6.69}$$

2-c) Preliminary SNR estimation before noise mixture The variables used in 2-c) are based on Table 6.26. The preliminary SNR $\xi_n(k_i)$ is calculated as follows.

$$\xi_n(k_i) = \left(1 - \alpha_n^p(k_i)\right) \xi_{tmp} + \alpha_n^p(k_i) \gamma_n^C(k_i) \tag{6.70}$$

$$\xi_{tmp} = a \frac{|\hat{S}_n(f-1, k_i)|^2}{\lambda_n(f-1, k_i)} + (1-a) \xi_n(f-1, k_i) \tag{6.71}$$

Table 6.26: Definition of variable

Variable	Description, corresponding parameter
a	Internal ratio of the former frame SNR. Parameter PRIORSNRFACTOR. The default value is 0.8.
ξ^{max}	Upper limit of the preliminary SNR. Parameter MAXPRIORSNR. The default value is 100.

Here, ξ_{tmp} is a temporary variable in the calculation, which is an interior division value of the estimated SNR$\gamma_n(k_i)$ and preliminary SNR $\xi_n(k_i)$ of the former frame. Moreover, when $\xi_n(k_i) > \xi^{max}$ is satisfied, shange the value as $\xi_n(k_i) = \xi^{max}$.

2-d)Estimation of optimal gain The variables used in 2-d) are based on Table 6.27. Prior to calculating an optimal gain,

Table 6.27: Definition of variable

Variable	Description, corresponding parameter
θ^{max}	Intermediate variable $v_n(k_i)$ maximum value. Parameter MAXOPTGAIN. The default value is 20.
θ^{min}	The intermediate variable $v_n(k_i)$ minimum value. Parameter MINOPTGAIN. The default value is 6

the following intermediate variable $v_n(k_i)$ is calculated with the preliminary SNR$\xi_n(k_i)$ obtained above and the estimated SNR$\gamma_n(k_i)$.

$$v_n(k_i) = \frac{\xi_n(k_i)}{1+\xi_n(k_i)}\gamma_n(k_i) \tag{6.72}$$

When $v_n(k_i) > \theta^{max}$ is satisfied, $v_n(k_i) = \theta^{max}$. The optimal gain $\boldsymbol{G}^{H1}(k_i) = [G_1^{H1}(k_i), \ldots, G_N^{H1}(k_i)]$ when speech exists is obtained as sollows.

$$G_n^{H1}(k_i) = \frac{\xi_n(k_i)}{1+\xi_n(k_i)} \exp\left\{\frac{1}{2}\mathtt{int}_{v_n(k_i)}^{\inf}\frac{e^{-t}}{t}\mathrm{d}t\right\} \tag{6.73}$$

Here,

$$\begin{aligned} G_n^{H1}(k_i) &= 1 \quad \text{if} v_n(k_i) < \theta^{min} \\ G_n^{H1}(k_i) &= 1 \quad \text{if} G_n^{H1}(k_i) > 1. \end{aligned} \tag{6.74}$$

3) Eestimation of probability of speech presence Figure 6.35 shows the flow of estimation of probability of speech presence. Estimation of the probability of speech presence consists of the followings.
a) Smoothing of the preliminary SNR for every 3 types of bands
b) Estimation with the temporal probability of speech presence based on the smoothed SNR in each band
c) Speech quiescent probability is estimated based on three provisional probability.
d) Estimation of the final probability of speech presence.

3-a) Smoothing of preliminary SNR The variables used in 3-a) are summarized in Table 6.28. First, temporally-smoothing is performed with the preliminary SNR $\xi_n(f, k_i)$ calculated by Equation (6.70) and the temporally-smoothed preliminary SNR $\zeta_n(f-1, k_i)$ of the former frame.

$$\zeta_n(f, k_i) = b\zeta_n(f-1, k_i) + (1-b)\xi_n(f, k_i) \tag{6.75}$$

Smoothing of the frequency direction is reduced in the order of frame, global, local depending on the size of the frame.

- Frequency smoothing in frame
  Smoothing by averaging is performed in the frequency bin range $F_{st} \sim F_{en}$.

$$\zeta_n^f(k_i) = \frac{1}{F_{en}-F_{st}+1}\sum_{k_j=F_{st}}^{F_{en}}\zeta_n(k_j) \tag{6.76}$$

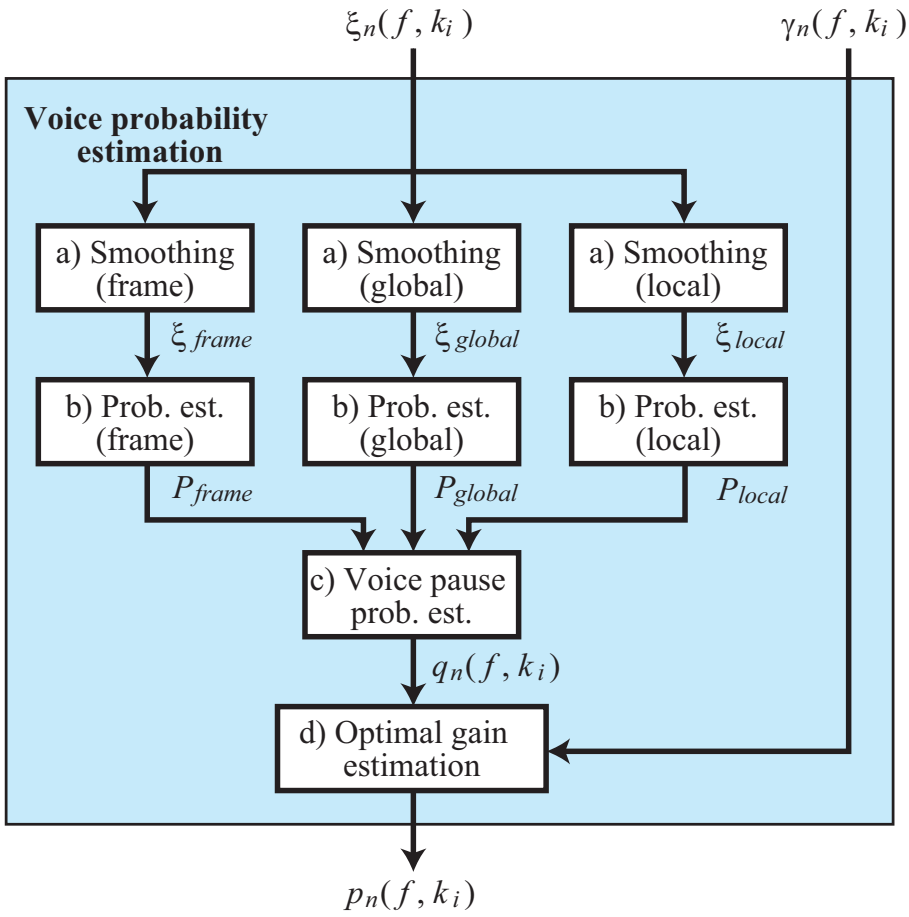


Figure 6.35: Procedure for estimation of probability of speech presence:

Table 6.28: Definition of variable

Variable	Description, corresponding parameter
$\zeta_n(k_i)$	Time preliminary SNR temporally-smoothed
$\xi_n(k_i)$	Preliminary SNR
$\zeta_n^f(k_i)$	Frequency-smoothed SNR (frame)
$\zeta_n^g(k_i)$	Frequency-smoothed SNR (global)
$\zeta_n^l(k_i)$	Frequency smoothing SNR (local)
b	Parameter PRIORSNRSMOOTHFACTOR. The default value is 0.7
F_{st}	Parameter LOWERSMOOTHFREQINDEX. The default value is 8
F_{en}	Parameter UPPERSMOOTHFREQINDEX. The default value is 99
G	Parameter GLOBALSMOOTHBANDWIDTH. The default value is 29
L	Parameter LOCALSMOOTHBANDWIDTH. The default value is 5

- Frequency smoothing in global
  In global, the frequency smoothing with a hamming window in width G is performed.

$$\zeta_n^g(k_i) = \sum_{j=-(G-1)/2}^{(G-1)/2} w_{han}(j + (G-1)/2)\zeta_n(k_{i+j}), \tag{6.77}$$

$$w_{han}(j) = \frac{1}{C}\left(0.5 - 0.5\cos\left(\frac{2\pi j}{G}\right)\right), \tag{6.78}$$

Here, C is a normalization coefficient so that $\sum_{j=0}^{G-1} w_{han}(j) = 1$ can be satisfied.

- Frequency smoothing in local
  In local, the frequency smoothing with a triangle window in width F is performed.

$$\zeta_n^l(k_i) = 0.25\zeta_n(k_i - 1) + 0.5\zeta_n(k_i) + 0.25(k_i + 1) \tag{6.79}$$

3-b Estimation of the probability of provisional speech The variables used in 3-b) are shown in Table 6.29.

Table 6.29: Definition of variable

Variable	Description, corresponding parameter
$\zeta_n^{f,g,l}(k_i)$	SNR smoothed in each band
$P_n^{f,g,l}(k_i)$	Probability of provisional speech in each band
$\zeta_n^{peak}(k_i)$	Peak of smoothed SNR
Z_{min}^{peak}	Parameter MINSMOOTHPEAKSNR. The default value is 1.
Z_{max}^{peak}	Parameter MAXSMOOTHPEAKSNR. The default value is 10.
Z_{thres}	FRAMESMOOTHSNRTHRESH. The default value is 1.5.
$Z_{min}^{f,g,l}$	Parameter MINFRAMESMOOTHSNR, MINGLOBALSMOOTHSNR, MINLOCALSMOOTHSNR. The default value is 0.1.
$Z_{max}^{f,g,l}$	Parameter MAXFRAMESMOOTHSNRF, MAXGLOBALSMOOTHSNR, MAXLOCALSMOOTHSNR. The default value is 0.316.

- Calculation of $P_n^f(k_i)$ and $\zeta_n^{peak}(k_i)$
  First, calculate $\zeta_n^{peak}(f, k_i)$ as follows.

$$\zeta_n^{peak}(f, k_i) = \begin{cases} \zeta_n^f(f, k_i), & \text{if } \zeta_n^f(f, k_i) > Z_{thres}\zeta_n^f(f-1, k_i) \\ \zeta_n^{peak}(f-1, k_i), & \text{if otherwise.} \end{cases} \tag{6.80}$$

Here, the value of $\zeta_n^{peak}(k_i)$ must be within the range of the parameter $Z_{min}^{peak}, Z_{max}^{peak}$. That is,

$$\zeta_n^{peak}(k_i) = \begin{cases} Z_{min}^{peak}, & \text{if } \zeta_n^{peak}(k_i) < Z_{min}^{peak} \\ Z_{max}^{peak}, & \text{if } \zeta_n^{peak}(k_i) > Z_{max}^{peak} \end{cases} \tag{6.81}$$

Next, $P_n^f(k_i)$ is obtained as follows.

$$P_n^f(k_i) = \begin{cases} 0, & \text{if } \zeta_n^f(k_i) < \zeta_n^{peak}(k_i) Z_{min}^f \\ 1, & \text{if } \zeta_n^f(k_i) > \zeta_n^{peak}(k_i) Z_{max}^f \\ \frac{\log\left(\zeta_n^f(k_i)/\zeta_n^{peak}(k_i)Z_{min}^f\right)}{\log\left(Z_{max}^f/Z_{min}^f\right)}, & \text{otherwise} \end{cases} \tag{6.82}$$

- Calculation of $P_n^g(k_i)$
  Calculate as follows.

$$P_n^g(k_i) = \begin{cases} 0, & \text{if } \zeta_n^g(k_i) < Z_{min}^g \\ 1, & \text{if } \zeta_n^g(k_i) > Z_{max}^g \\ \frac{\log\left(\zeta_n^g(k_i)/Z_{min}^g\right)}{\log\left(Z_{max}^g/Z_{min}^g\right)}, & \text{otherwise} \end{cases} \tag{6.83}$$

Calculation of $P_n^l(k_i)$
Calculate as follows.

$$P_n^l(k_i) = \begin{cases} 0, & \text{if } \zeta_n^l(k_i) < Z_{min}^l \\ 1, & \text{if } \zeta_n^l(k_i) > Z_{max}^l \\ \frac{\log\left(\zeta_n^l(k_i)/Z_{min}^l\right)}{\log\left(Z_{max}^l/Z_{min}^l\right)}, & \text{otherwise} \end{cases} \tag{6.84}$$

Table 6.30: Definition of variable

Variable	description, a corresponding parameter
$q_n(k_i)$	Probability of speech pause.
a^f	FRAMEVOICEPPROBFACTOR. The default value is 0.7.
a^g	GLOBALVOICEPPROBFACTOR. The default value is 0.9.
a^l	LOCALVOICEPPROBFACTOR. The default value is 0.9.
q_{min}	MINVOICEPAUSEPROB. The default value is 0.02.
q_{max}	MAXVOICEPAUSEPROB. The default value is 0.98.

3-c) Estimation of the probability of speech pause The variables used in 3-c) are shown in Table 6.30. As shown below, the probability of speech pause $q_n(k_i)$ is obtained by integrating the provisional probability of speech calculated from a smoothing result of the three frequency bands $P_n^{f,g,l}(k_i)$.

$$q_n(k_i) = 1 - \left(1 - a^l + a^l P_n^l(k_i)\right)\left(1 - a^g + a^g P_n^g(k_i)\right)\left(1 - a^f + a^f P_n^f(k_i)\right), \tag{6.85}$$

Here, when $q_n(k_i) < q_{min}$, $q_n(k_i) = q_{min}$, and when $q_n(k_i) > q_{max}$, $q_n(k_i) = q_{max}$.

3-d) Estimation of the probability of speech presence The probability of speech presence $p_n(k_i)$ is obtained by the probability of speech suspension pause $q_n(k_i)$, the preliminary SNR $\zeta_n(k_i)$ and the intermediate variable $v_n(k_i)$ derived by Equation (6.72).

$$p_n(k_i) = \left\{1 + \frac{q_n(k_i)}{1 - q_n(k_i)}\left(1 + \zeta_n(k_i)\right)\exp\left(-v_n(k_i)\right)\right\}^{-1} \tag{6.86}$$

4 Noise removal:

The enhanced separated sound as an output $\hat{S}_n(k_i)$ is derived by activating the optimal gain $G_n^{H1}(k_i)$ and the probability of speech presence $p_n(k_i)$ for the separated sound spectrum as the input $Y_n(k_i)$.

$$\hat{S}_n(k_i) = Y_n(k_i) G_n^{H1}(k_i) p_n(k_i) \tag{6.87}$$

6.3.8 SpectralGainFilter

Outline of the module

This module multiplies the separated sound spectrum input by the optimal gain and the probability of speech presence (see PostFilter) and output.

Necessary file

No files are required.

Usage

When to use

This module is used when obtaining a speech spectrum with suppressed noise from a separated sound spectrum.

Typical connection

Figure 6.36 shows a connection example of SpectralGainFilter. Inputs are separation spectra output from GHDSSand the optimal gain, and the probability of speech presence output from CalcSpecSubGainand so on. In the figure, an audio file is created with this module being connected to Synthesizeand SaveRawPCMas an example of output.

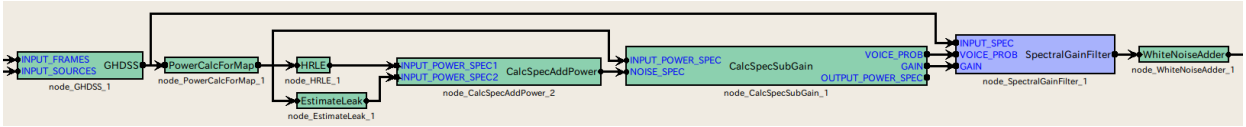


Figure 6.36: Connection example of SpectralGainFilter

Input-output and property of the module

Input

INPUTSPEC : `Map<int, ObjectRef>`type. The same type same as that of the output of GHDSS. A pair of the sound source ID and the complex spectrum of separated sound as `Vector<complex<float> >`type data.

VOICEPROB : `Map<int, ObjectRef>`type. A pair of the sound source ID and the probability of speech presence as `Vector<float>`type data.

GAIN : `Map<int, ObjectRef>`type. A pair of the sound source ID and the optimal gain as `Vector<float>`type data.

Output

OUTPUTSPEC : `Map<int, ObjectRef>`type. The same type as that of the output of GHDSS. A pair of the sound source ID and the complex spectrum of separated sound as `Vector<complex<float> >`type data.

Parameter

No parameters

Details of the module

This module multiplies the input speech spectrum by the optimal gain and the probability of speech presence and outputs a separated sound spectrum with emphasized speech. Assuming that the separated sound spectrum, which is the input $X_n(k_i)$, the optimal gain is $G_n(k_i)$ and the probability of speech presence is $p_n(k_i)$, the separated sound for which the input speech is enhanced $Y_n(k_i)$ is expressed as follows.

$$Y_n(k_i) = X_n(k_i)G_n(k_i)p_n(k_i) \tag{6.88}$$

6.4 FeatureExtraction category

6.4.1 Delta

Outline of the module

This module acquires dynamic feature vectors from static feature vectors. It is usually connected to the posterior half of MSLSExtractionand MFCCExtraction, which are feature extraction modules. These feature-extracting modules acquire static feature vectors while they secure the regions where dynamic features are saved. The dynamical feature of this time is set to 0. The Deltamodule calculates dynamic feature vector values with static feature vector values and set the values. Therefore, dimension numbers are same at the input and output.

Necessary file

No files are required.

Usage

When to use

This module is used for obtaining dynamic features from static features. It is usually used after MFCCExtractionand MSLSExtraction.

Typical connection

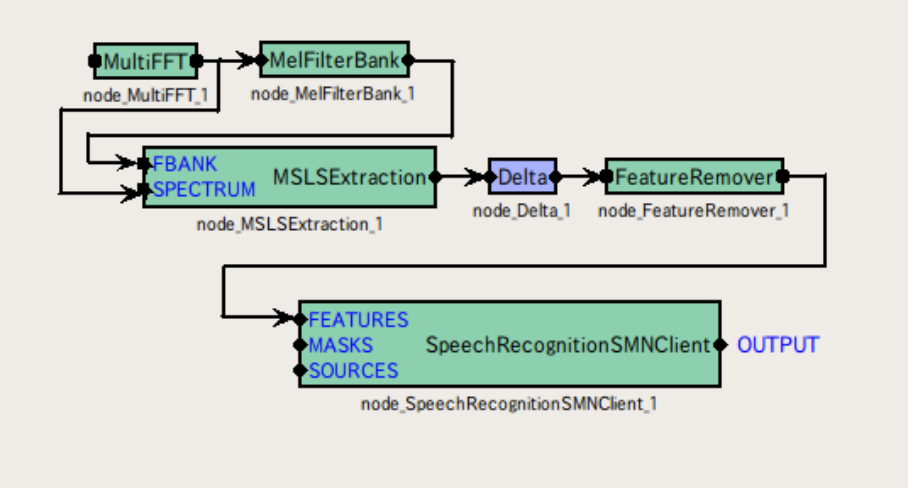


Figure 6.37: Typical connection example of Delta

Input-output and property of the module

Table 6.31: Parameter list of Delta

Parameter name	Type	Default value	Unit	Description
FBANKCOUNT	int	13		

INPUT : Map<int, ObjectRef>type. A pair of the sound source ID and feature vector as Vector<float>type data.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and feature vector as `Vector<float>`type data.

Parameter

FBANKCOUNT : `int`type. Dimension numbers of features to be processed. Its range is positive integer. When connecting it just after the feature extract module, designate FBANKCOUNT designated in feature extraction. However, in the case that `true`is selected for the option used for the power term in feature extraction, designate FBANKCOUNT +1.

Details of the module

This module obtains dynamic feature vectors from static feature vectors. The dimension number of inputs is the total dimension number of dynamic and static features. Dynamic features are calculated with an assumption that the dimension elements less than FBANKCOUNT are static features. Dynamic features are added to the dimension elements higher than FBANKCOUNT. The input feature vector at the frame time f is expressed as follows.

$$\boldsymbol{x}(f) = [x(f,0), x(f,1), \ldots, x(f,P-1)]^T \tag{6.89}$$

Here, P is FBANKCOUNT.

$$\boldsymbol{y}(f) = [x(f,0), x(f,1), \ldots, x(f,2P-1)]^T \tag{6.90}$$

Each output vector element is expressed as,

$$y(f,p) = \begin{cases} x(f,p), & \text{if } p = 0,\ldots,P-1, \\ w\sum_{\tau=-2}^{2} \tau \cdot x(f+\tau,p), & \text{if } p = P,\ldots,2P-1, \end{cases} \tag{6.91}$$

Here, $w = \frac{1}{\sum_{\tau=-2}^{\tau=2} \tau^2}$. Figure 6.38 shows the input-output flow of Delta.

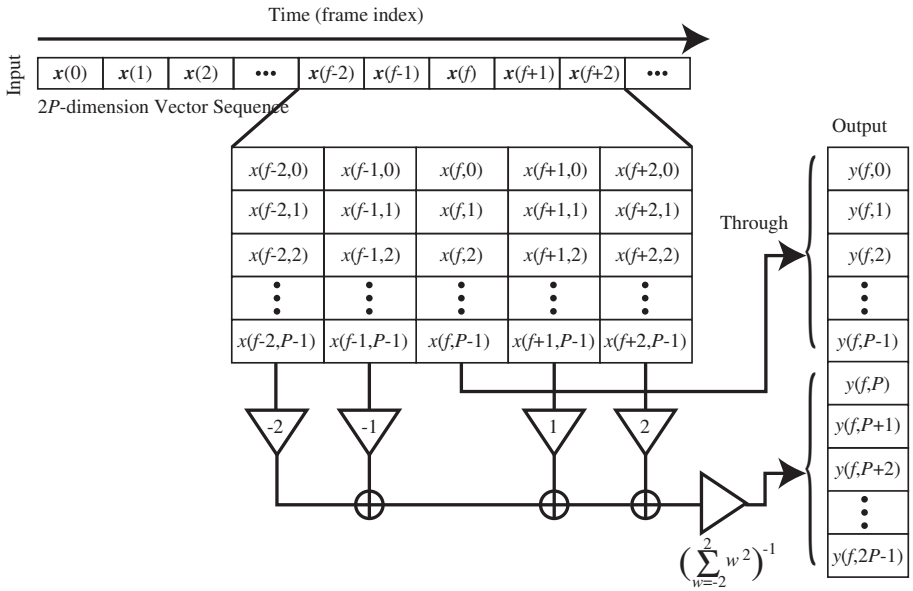


Figure 6.38: Input-output flow of Delta.

6.4.2 FeatureRemover

Outline of the module

This module deletes the dimension element specified from the input vector and outputs the vector with reduced vector dimension.

Necessary file

No files are required.

Usage

When to use

This module is used to delete unnecessary elements from the acoustic features and elements of vector types such as Missing Feature Mask so as to reduce the dimension number. The feature extraction processing usually extracts static features followed by dynamic features. In this processing, static features are not needed in some cases. This module is used to delete unnecessary features. In particular, logarithmic power terms are often deleted.

Typical connection

The delta logarithmic power term can be calculated by calculating the logarithmic power term with MSLSExtractionand MFCCExtractionand then using Delta. Since the delta logarithmic power term cannot be calculated unless the logarithmic power is calculated, the logarithmic power term is removed after calculating once the acoustic features amount including the logarithmic power. This module is usually connected with the posterior half of Deltaand used to remove logarithmic power terms.

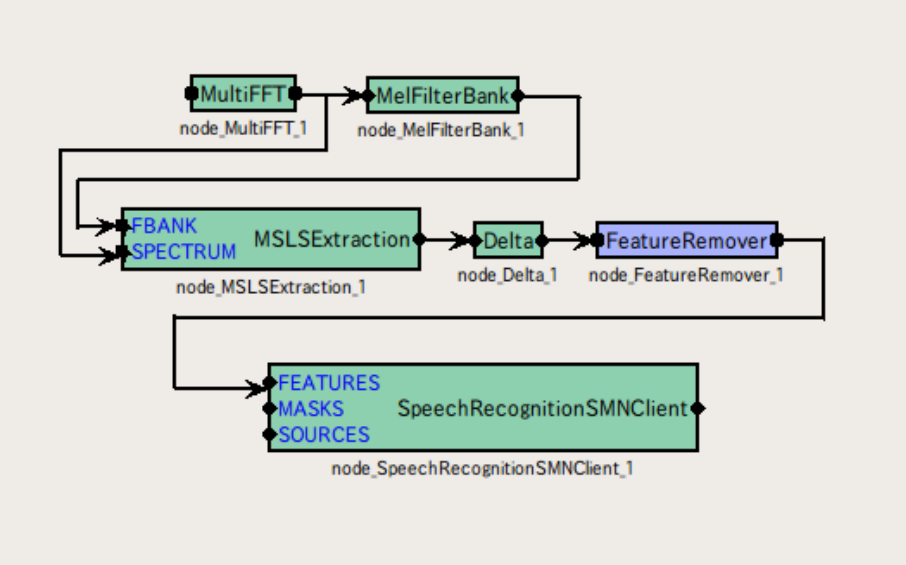


Figure 6.39: Typical connection example of FeatureRemover

Input-output and property of the module

Input

INPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and the feature vector as `Vector<float>`type data.

Table 6.32: Parameter list of FeatureRemover

Parameter name	Type	Default value	Unit	Description
SELECTOR	Vector<int>		Vector consisting of dimension index	(plural parameters can be designated)

OUTPUT : Map<int, ObjectRef>type. A pair of the sound source ID and the feature vector as Vector<float>type data.

Parameter

SELECTOR : Vector<int>type. The range is from 0 to the dimension number of the input feature. The user may specify this as many as wished. When the elements of the first and third dimensions are deleted and the dimension of input vector is reduced by two dimensions, <Vector<int> 0 2>. Note that the index of dimension designation begins with 0.

Details of the module

This module deletes unnecessary dimension elements from the input vectors to reduce the dimension number of the vectors. It is shown by analyzing audio signals that the logarithmic power of an analysis frame tends to be large in speech sections, vocal sound parts in particular. Therefore, improvement of recognition accuracy can be expected by adopting the logarithmic power term to acoustic features in speech recognition. However, when the logarithmic power term is used directly as features, difference in the sound pickup level is reflected directly to the acoustic features. When difference occurs in the logarithmic power level used for creation of an acoustic model and sound pickup level, the speech recognition accuracy falls. Even when fixing the general instrument setting, the utterers do not necessarily speak with the same level. Therefore, the delta logarithmic power term, which is the dynamic feature of the logarithmic power term, is used. This enables to capture the features that are strong for the difference of sound pickup levels and indicate utterance sections and vocal sound parts.

6.4.3 MelFilterBank

Outline of the module

This module performs the mel-scale filter bank processing for input spectra and outputs energy of each filter channel. Note that there are two types of input spectra and output results differ depending on inputs.

Necessary file

No files are required.

Usage

When to use

This module is used as preprocessing for acquiring acoustic features. It is used just after MultiFFT, PowerCalcForMapor PreEmphasis. It is used before MFCCExtractionor MSLSExtraction.

Typical connection

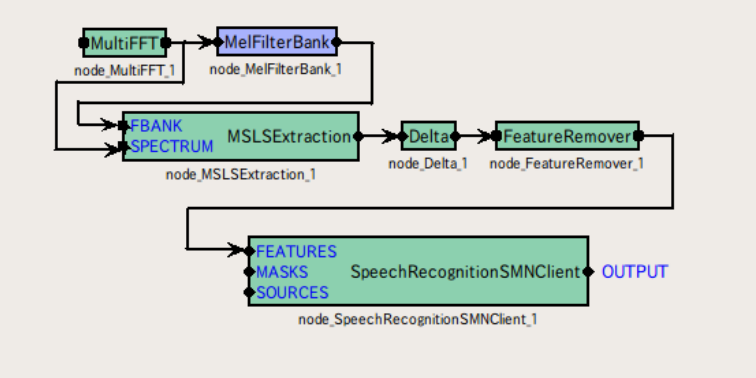


Figure 6.40: Connection example of MelFilterBank

Input-output and property of the module

Table 6.33: Parameter list of MelFilterBank

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	Analysis frame length
SAMPLINGRATE	int	16000	[Hz] Sampling frequency	
CUTOFF	int	8000	[Hz]	Cut-off frequency of lowpass filter
MINFREQUENCY	int	63	[Hz]	Lower cut-off frequency of filter bank
MAXFREQUENCY	int	8000	[Hz]	Upper limited frequency of filter bank
FBANKCOUNT	int	13		Filter bank numbers

Input

INPUT : Map<int, ObjectRef>type. A pair of the sound source ID and power spectrum as Vector<float>type or complex spectrum Vector<complex<float> >type data. Note that when the power spectrum is selected, output energy doubles, different from the case that the complex spectrum is selected.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and the vector consisting of output energy of the filter bank as `Vector<float>`type data. The dimension number of output vectors is twice as large as FBANKCOUNT. Output energy of the filter bank is in the range from 0 to FBANKCOUNT-1 and 0 is in the range from FBANKCOUNT to 2 *FBANKCOUNT-1. The part that 0 is in is a placeholder for dynamic features. When dynamic features are not needed, it is necessary to delete with FeatureRemover.

Parameter

LENGTH : `int`type. Analysis frame length. It is equal to the number of frequency bins of the input spectrum. Its range is positive integer.

SAMPLINGRATE : `int`type. Sampling frequency. Its range is positive integer.

CUTOFF : Cut-off frequency of the anti-aliasing filter in a discrete Fourier transform. It is below 1/2 of SAMPLINGRATE.

MINFREQUENCY : `int`type. Lower cut-off frequency of the filter bank. Its range is positive integer and less than CUTOFF.

MAXFREQUENCY : `int`type. Upper limited frequency of the filter bank. Its range is less than in positive integer and CUTOFF.

FBANKCOUNT : `int`type. The number of filter banks. Its range is positive integer.

Details of the module

This module performs the mel-scale filter bank processing and outputs energy of each channel. Center frequency of each bank is positioned on mel-scale [(1)] at regular intervals. Center frequency for each channel is determined by performing FBANKCOUNT division from the minimum frequency bin SAMPLINGRATE/LENGTH to SAMPLINGRATECUTOFF/LENGTH. Transformation of the linear scale and mel scale is expressed as follows.

$$m = 1127.01048 \log(1.0 + \frac{\lambda}{700.0}) \tag{6.92}$$

However, expression on the linear scale is λ (Hz) and that on the mel scale is m. Figure 6.41 shows an example of the transformation by 8000 Hz. The red points indicate the center frequency of each bank when SAMPLINGRATE is 16000Hz, CUTOFF is 8000Hz and FBANKCOUNT is 13. The figure shows that the center frequency of each bank is at regular intervals on the mel scale. Figure 6.42 shows the window functions of the filter banks on the mel scale. It is a triangle window that becomes 1.0 on the center frequency parts and 0.0 on the center frequency parts of adjacent channels. Center frequency for each channel is at regular intervals on the mel scale and in symmetric shape. These window functions are represented as shown in Figure 6.43 on the linear scale. A wide band is covered in high frequency channels. The input power spectrum expressed on the linear scale is weighted with the window functions shown in Figure 6.43 and energy is obtained for each channel and output.

References:

(1) Stanley Smith Stevens, John Volkman, Edwin Newman: " A Scale for the Measurement of the Psychological Magnitude Pitch ", Journal of the Acoustical Society of America 8(3), pp.185–190, 1937.

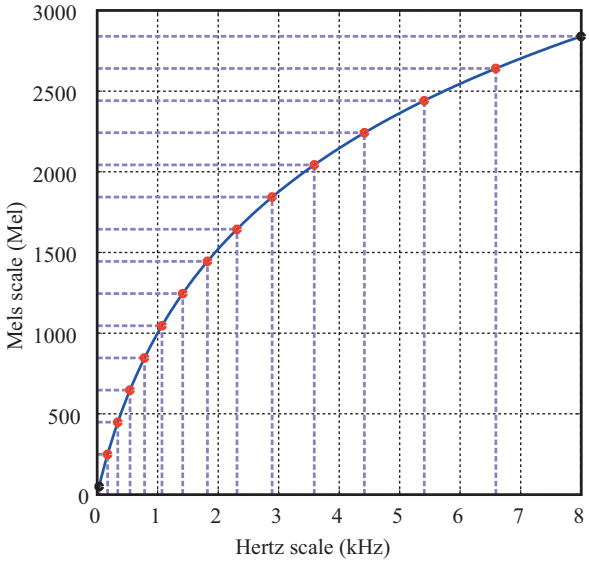


Figure 6.41: Correspondence between linear scale and a mel scale

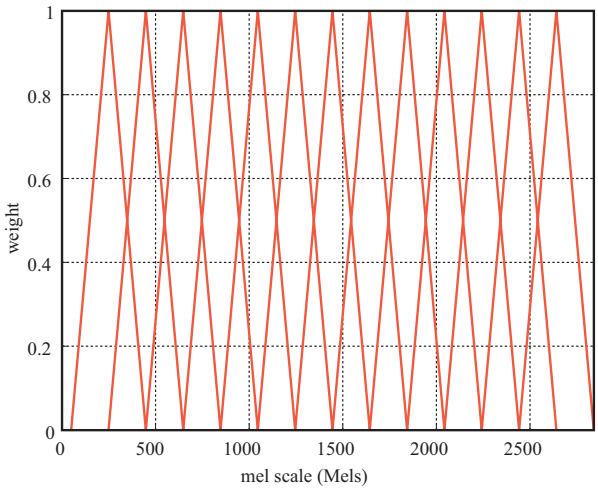


Figure 6.42: Window function on mel scale

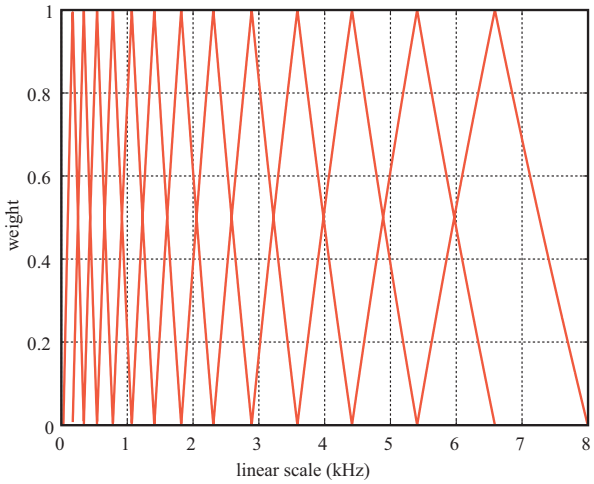


Figure 6.43: Window function on linear scale

6.4.4 MFCCExtraction

Outline of the module

This module acquires mel-cepstrum coefficients (MFCC), which are one of the acoustic features. It generates acoustic feature vectors consisting of the mel-cepstrum coefficients and logarithmic spectrum power as elements.

Necessary file

No files are required.

Usage

When to use

This module is used to generate an acoustic feature with mel-cepstrum coefficients as elements and acoustic feature vectors. For example, acoustic feature vectors are input to the speech recognition module to identify phoneme and speaker.

Typical connection

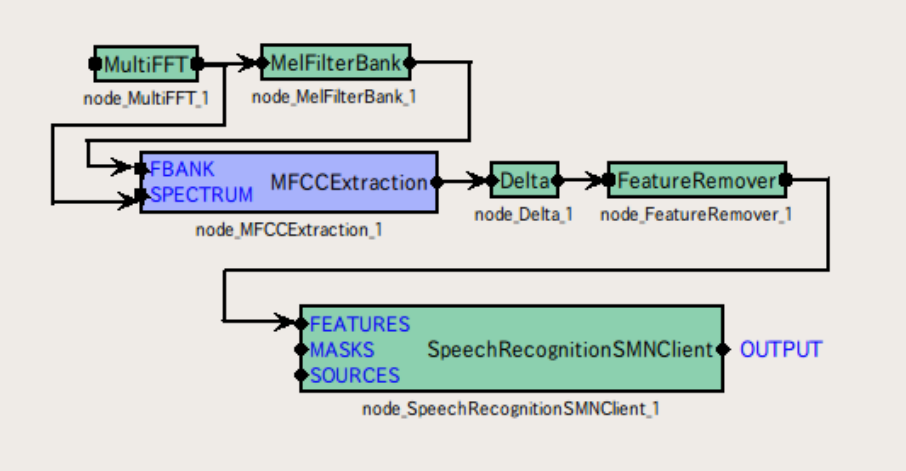


Figure 6.44: Typical connection example of MFCCExtraction

Input-output and property of the module

Table 6.34: Parameter list of MFCCExtraction

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	int	24		The number of filter banks for input spectrum
NUM_CEPS	int	12		The number of cepstral coefficients for liftering
USE_POWER	bool	false		Select whether or not to include logarithmic power in features

Input

FBANK : Map<int, ObjectRef>type. A pair of the sound source ID and the vector consisting of sound source output energy of the filter bank as Vector<float>type data.

SPECTRUM : `Map<int, ObjectRef>`type. A pair of the sound source ID and the vector consisting of complex spectra as `Vector<float>`type data.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and the vector consisting of MFCC and a logarithmic power term as `Vector<float>`type data.

Parameter

FBANKCOUNT : `int`type. The number of filter banks for the input spectrum. The default value is 24. Its range is positive integer. The frequency band for 1 bank narrows as the value is raised and acoustic features of high frequency resolution are obtained. Acoustic features are expressed more finely by setting greater FBANKCOUNT. Precise expression is not necessarily optimal for speech recognition and it depends on acoustic environment for uttering.

NUMCEP : `int`type. The number of cepstrum coefficients left to liftering. The default value is 12. The range is positive integer. When raising the value, dimension of acoustic features increases. Acoustic features that express a finer spectral change are obtained.

USEPOWER : When selecting `true`, the logarithmic power term is added to acoustic features. When selecting `false`, it is omitted. It is rare to use the power term for acoustic features though it is assumed that delta logarithmic power is effective for speech recognition. When `true`is selected, delta logarithmic power is calculated in the posterior half and its result is used as acoustic features.

Details of the module

This module acquires mel-cepstrum coefficients (MFCC), which are one of the acoustic features, and logarithmic power. It generates acoustic features consisting of the mel-cepstrum coefficients and log spectrum power as elements. Filter bank of a triangle window is performed for log spectra. Center frequencies of the triangle window are positioned at regular intervals on the mel-scale. Output logarithmic energy of each filter bank extracted and Discrate Cosine Transform is performed for it. The coefficient for which liftering is performed of the obtained coefficient is MFCC. It is premised that output logarithmic energy of each filter bank is input to FBANK of the input unit of this module. The vector input to FBANK at frame time f is expressed as follows.

$$\boldsymbol{x}(f) = [x(f,0), x(f,1), \cdots, x(f,P-1)]^T \tag{6.93}$$

Here, P indicates FBANKCOUNT in the dimension number of the input feature vector. The vector output is a $P+1$ dimensional vector and consists of the mel-cepstrum coefficient and power term. The dimensions from the first to the P are for mel-cepstrum coefficients and the dimension $P+1$ is for the power term. The output vector of this module is expressed as;

$$\boldsymbol{y}(f) = [y(f,0), y(f,1), \ldots, y(f,P-1), E]^T \tag{6.94}$$

$$y(f,p) = L(p) \cdot \sqrt{\frac{2}{P}} \cdot \sum_{q=0}^{P-1} \left\{ \log(x(q)) \cos\left(\frac{\pi(p+1)(q+0.5)}{P} \right) \right\} \tag{6.95}$$

Here, E indicates the power term (see later description). The liftering coefficient is expressed as;

$$L(p) = 1.0 + \frac{Q}{2} \sin\left(\frac{\pi(p+1)}{Q} \right), \tag{6.96}$$

Here, $Q = 22$. The power term is obtained from the input vector of SPECTRUM part. The input vector is expressed as;

$$\boldsymbol{s} = [s(0), \ldots, s(K-1)]^T, \tag{6.97}$$

Here, K indicates FFT length. K is determined by the dimension number of `Map`connected to SPECTRUM. The logarithmic power term is expressed as;

$$E = \log\left(\frac{1}{K} \sum_{k=0}^{K-1} s(k) \right) \tag{6.98}$$

6.4.5 MSLSExtraction

Outline of the module

This module acquires mel-scale logarithmic spectra (MSLS), which are one of the acoustic features, and logarithmic power. It generates acoustic feature vectors consisting of the mel-scale logarithmic spectrum coefficients and logarithmic spectrum power as elements.

Necessary file

No files are required.

Usage

When to use

The mel-scale logarithmic spectrum coefficients and logarithmic spectrum power are used as the elements of this module. This module is used to generate acoustic feature vectors. For example, acoustic feature vectors are input to the speech recognition module to identify phoneme and speaker.

Typical connection

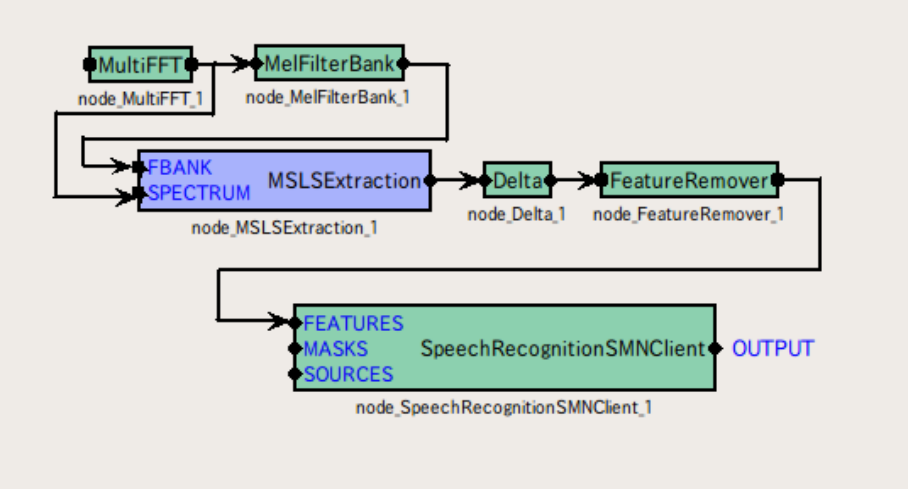


Figure 6.45: Typical connection example of MSLSExtraction

Input-output and property of the module

Table 6.35: Parameter list of MSLSExtraction

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	int	13		The number of filter banks for input spectrum
NORMALIZATIONMODE	string	CEPSTRAL		Feature normalization method
USEPOWER	bool	false		Select whether or not to include logarithmic power in features

Input

FBANK : `Map<int, ObjectRef>`type. A pair of the sound source ID and the vector consisting of sound source output energy of the filter bank as `Vector<float>`type data.

SPECTRUM : `Map<int, ObjectRef>`type. A pair of the sound source ID and the vector consisting of complex spectra as `Vector<float>`type data.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and the vector consisting of MSLS and a logarithmic power term as `Vector<float>`type data. This module obtains static features of MSLS though vectors containing a dynamic feature part are output. Zero is set to the dynamic feature part. Figure 6.46 shows the operation.

Parameter

FBANKCOUNT : `int`type. The number of filter banks for the input spectrum. Its range is positive integer. The frequency band 1 bank narrows as the value is raised and acoustic features of high frequency resolution are obtained. Typical set values are from 13 to 24. Acoustic features are expressed more finely by setting greater FBANKCOUNT. Precise expression is not necessarily optimal for speech recognition and it depends on acoustic environment for uttering.

NORMALIZATIONMODE : `string`type. The user can designate CEPSTRAL or SPECTRAL. The user selects whether or not to perform normalization in the cepstrum domain / spectrum domain.

USEPOWER : When selecting `true`, the logarithmic power term is added to acoustic features. When selecting `false`, it is omitted. It is rare to use the power term for acoustic features though it is assumed that delta logarithmic power is effective for speech recognition. When `true`is selected, delta logarithmic power is calculated in the posterior half and its result is used as acoustic features.

Details of the module

This module acquires mel-scale garithmic spectra (MSLS), which are one of the acoustic features, and logarithmic power. It generates acoustic features consisting of the mel-scale garithmic spectrum coefficients and log spectrum power as elements. Output logarithmic energy of each filter bank is input to FBANK input terminal of this module. Calculation methods of MSLS to be output differ depending on normalization method the user designates. The followings are calculation methods of output vectors for this module for each normalization method.

CEPSTRAL : The input to the FBANK terminal is expressed as;

$$\boldsymbol{x} = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.99}$$

Here, P indicates FBANKCOUNT in the dimension number of the input feature vector. The vector output is a $P+1$ dimensional vector and consists of the MSLS coefficient and power term. The dimensions from the first to the P are for MSLS and the dimension $P+1$ is for the power term. The output vector of this module is expressed as;

$$\boldsymbol{y} = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.100}$$

$$y(p) = \frac{1}{P}\sum_{q=0}^{P-1}\left\{L(q)\cdot\sum_{r=0}^{P-1}\left\{\log(x(r))\cos\left(\frac{\pi q(r+0.5)}{P}\right)\right\}\cos\left(\frac{\pi q(p+0.5)}{P}\right)\right\} \tag{6.101}$$

Here, the liftering coefficient is expressed as;

$$L(p) = \begin{cases} 1.0, & (p = 0, \ldots, P-1), \\ 0.0, & (p = P, \ldots, 2P-1), \end{cases} \tag{6.102}$$

Here, $Q = 22$.

SPECTRAL : The input to the FBANK part is expressed as;

$$\boldsymbol{x} = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.103}$$

Here, P indicates FBANKCOUNT in the dimension number of the input feature vector. The vector output is a $P+1$ dimensional vector and consists of the MSLS coefficient and power term. The dimensions from the first to the P are for mel-cepstrum coefficients and the dimension $P+1$ is for the power term. The output vector of this module is expressed as;

$$\boldsymbol{y} = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.104}$$

$$y(p) = \begin{cases} (\log(x(p)) - \mu) - 0.9(\log(x(p-1)) - \mu), & if \;\; p = 1, \ldots, P-1 \\ \log(x(p)), & if \;\; p = 0, \end{cases} \tag{6.105}$$

$$\mu = \frac{1}{P}\sum_{q=0}^{P-1} \log(x(q)), \tag{6.106}$$

Mean subtraction of the frequency direction and peak enhancement processing are applied.

For the logarithmic power term, the input to the SPECTRUM terminal is expressed as;

$$\boldsymbol{s} = [s(0), s(1), \ldots, s(N-1)]^T \tag{6.107}$$

Here, N is determined by the size of Mapconnected to the SPECTRUM terminal. For Map, assuming that spectral representation from 0 to π is stored in B bins, $N = 2(B-1)$. Now, the power term is expressed as;

$$p = \log\left(\frac{1}{N}\sum_{n=0}^{N-1} s(n)\right) \tag{6.108}$$

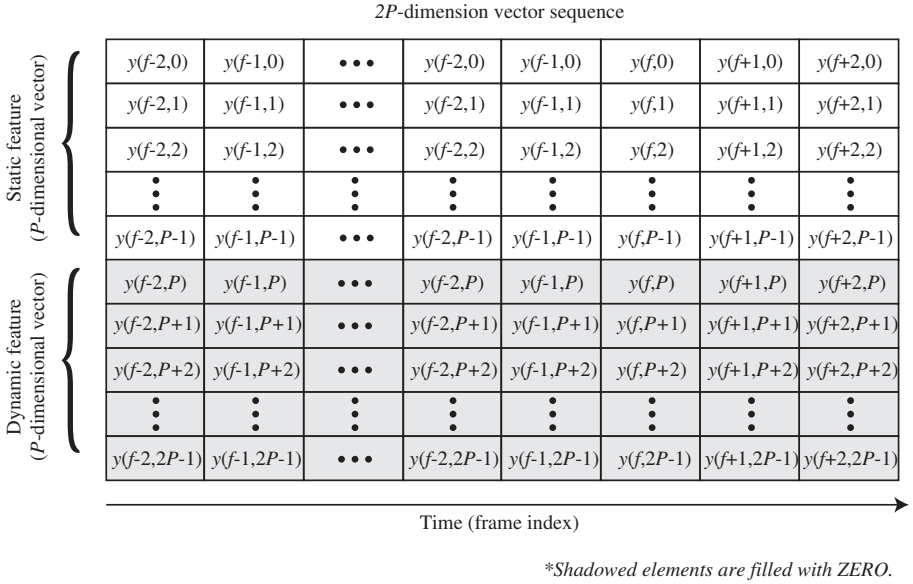


Figure 6.46: Output parameter of MSLSExtraction

6.4.6 PreEmphasis

Outline of the module

This module performs processing to emphasize upper frequency (pre-emphasis) when extracting acoustic features for speech recognition, so as to raise robustness to noise.

Necessary file

No files are required.

Usage

This module is generally used before extracting MFCC features. Moreover, it can be used as preprocessing when extracting MSLS features generally used for HARK.

Typical connection

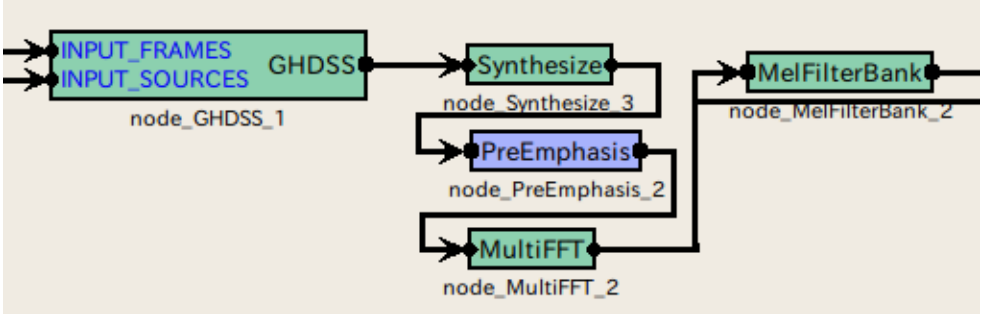


Figure 6.47: Connection example of PreEmphasis

Input-output and property of the module

Table 6.36: Parameter list of PreEmphasis

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	Signal length or window length of FFT
SAMPLINGRATE	int	16000	[Hz]	Sampling rate
PREEMCOEF	float	0.97		Preemphasis coefficient
INPUTTYPE	string	WAV		Input signal type

Input

INPUT : Map<int, ObjectRef>, When input signals are time domain waveforms, ObjectRefis used as Vector<float>. Moreover, in the case of the signals of frequency domain, it is used as Vector<complex<float> >.

Output

OUTPUT : Map<int, ObjectRef>, Signals for which upper frequency is emphasized are output. Corresponding with type of inputs,ObjectRefis changed to Vector<float>for time domain waveforms and to Vector<complex<float> >for frequency domain signals.

Parameter

LENGTH When INPUTTYPE is SPECTRUM, it indicates FFT length and must be equal to the value set for the former module. When INPUTTYPE is WAV, it indicates signal length of the signal contained in one frame and must be equal to the value set for the former module. Signal length is same as FFT length in a normal configuration.

SAMPLINGRATE Same as LENGTH, it is necessary to make equal the value to those of other modules.

PREEMCOEF A pre-emphasis coefficient expressed as c_p below. 0.97 are generally used for speech recognition.

INPUTTYPE Two input types of WAV and SPECTRUM are available. WAV is used for time domain waveform inputs. Moreover, SPECTRUM is used for frequency domain signal inputs.

Details of the module

Necessity and effects on common speech recognition of the pre-emphasis are described in various books and theses. Please read them. Although it is commonly said that this processing makes the system strong against noise, not much performance difference is obtained by this processing for HARK. This is probably because the microphone array processing is performed for HARK. However, it is necessary to make equal the audio data used when learning acoustic model used for speech recognition to parameters. In other words, when preemphasis is performed for the data used for learning acoustic model, the performance is improved by performing pre-emphasis also for input data. Concretely, PreEmphasisconsists of two types of processing in accordance with input signal types.

Upper frequency emphasis in time domain:

In the case of time domain, assuming t is the index indicating a sample in a frame, input signals are $s[t]$, the signal for which upper frequency is emphasized is $p[t]$ and the pre-emphasis coefficient is c_p, the upper frequency emphasis in time domain is expressed as follows.

$$p[t] = \begin{cases} s[t] - c_p \cdot s[t-1] & t > 0 \\ (1 - c_p) \cdot s[0] & t = 0 \end{cases} \tag{6.109}$$

Upper frequency emphasis in frequency domain:

In order to realize a filter of time domain equivalent to the filter of a frequency domain, the spectral filter for which a frequency analysis is performed for impulse responses of the aforementioned filter. Moreover, 0 is set to the low domain (for four bands from the bottom) and high domain (more than $fs/2$ -100Hz) considering errors. Here, fs indicates sampling frequency.

6.4.7 SaveFeatures

Outline of the module

This module saves feature vectors in files.

Necessary file

No files are required.

Usage

When to use

This module is used to save acoustic features such as MFCC and MSLS.

Typical connection

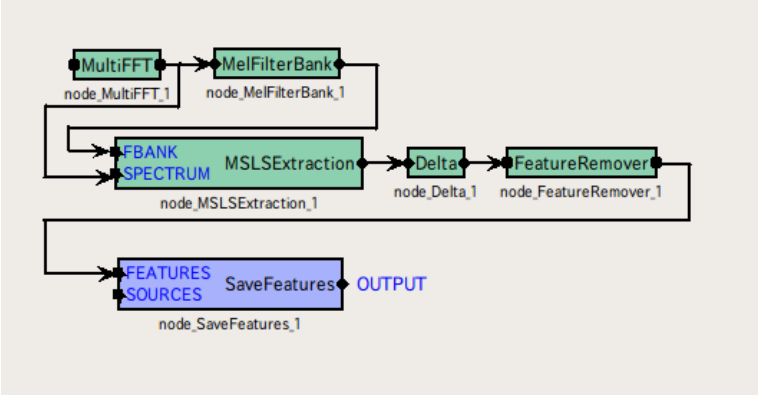


Figure 6.48: Connection example of SaveFeatures

Input-output and property of the module

Table 6.37: Parameter list of SaveFeatures

Parameter name	Type	Default value	Unit	Description
BASENAME	`string`			Prefix of file name when saving

Input

FEATURES : `Map<int, ObjectRef>`type or `Matrix<float>`type.

SOURCES : `Vector<ObjectRef>`type. This input is optional.

Output

OUTPUT : `Map<int, ObjectRef>`type.

Parameter

BASENAME : `string`type. Prefix of a file name when saving. An ID of SOURCES is given after Prefix and features are stored when saving.

Details of the module

Feature vectors are saved. For the file format for saving, vector elements are saved in 32bit floating-point number format of IEEE 758 little endian. For the naming rule, an ID number is given after Prefix given in the BASENAME property.

6.4.8 SaveHTKFeatures

Outline of the module

This module saves of feature vectors in the file format that can be treated by HTK (The Hidden Markov Model Toolkit).

Necessary file

No files are required.

Usage

When to use

This module is used when saving acoustic features such as MFCC, MSLS. Different from SaveFeatures, an exclusive header is added, for saving so that they can be treated by HTK.

Typical connection

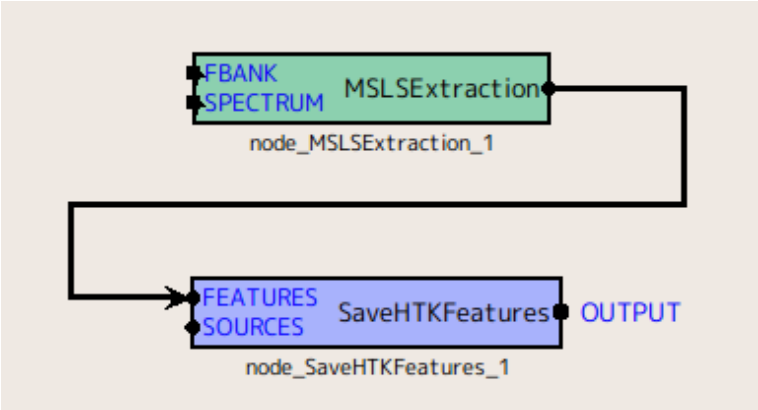


Figure 6.49: Connection example of SaveHTKFeatures

Input-output and property of the module

Table 6.38: Parameter list of SaveHTKFeatures

Parameter name	Type	Default value	Unit	Description
BASENAME	`string`			Prefix of file name when saving
HTKPERIOD	`int`	100000	[100 nsec]	Frame period
FEATURETYPE	`string`	USER		Type of feature

Input

FEATURES : `Map<int, ObjectRef>`type or `Matrix<float>`type.

SOURCES : `Vector<ObjectRef>`type. This input is optional.

Output

OUTPUT : `Map<int, ObjectRef>`type.

Parameter

BASENAME : `string`type. Prefix of a file name when saving. An ID of SOURCES is given after Prefix and features are stored when saving.

HTKPERIOD : Setting of frame period. Its unit is [100 nsec]. When shift length is 160 at the sampling frequency of 16000[Hz], the frame period is 10[msec]. Therefore, 10[ms] = 100000 * 100[nsec], which indicates that 100000 is obtained as an appropriate set point.

FEATURETYPE : Designation of type of features to be treated in HTK. Follow the HTK's original type. For example, when `MFCC_E_D`, "(MFCC+ power) + delta (MFCC+ power)", set this parameter so that it can match with the contents of features calculated in HARK. See HTKbook for the details.

Details of the module

This module saves feature vectors in the format that can be treated by HTK. For the file format for saving, vector elements are saved in 32bit floating-point number format of IEEE 758 little endian. For the naming rule, an ID number is given after Prefix given in the BASENAME property. Setting of the header of HTK can be changed in Property.

6.4.9 SpectralMeanNormalization

Outline of the module

This module subtracts the mean of features from the input acoustic features. However, as a problem, to realize real-time processing, the mean of the utterance concerned cannot be subtracted. It is necessary to estimate or approximate mean values of the utterance concerned using some values.

Necessary file

No files are required.

Usage

When to use

This module is used to subtract the mean of acoustic features. This module can subtract mismatches between the mean values of recording environments of audio data for acoustic model training and audio data for recognition. Properties of a microphone cannot be standardized often for some speech recording environments. In particular, speech-recording environments of acoustic model training and recognition are not necessarily the same. Since different persons are usually in charge of speech corpus creating for training and recording of audio data for recognition, it is difficult to arrange the same environment. Therefore, it is necessary to use features that do not depend on speech recording environments. For example, microphones used for acquiring training data and those used for recognition are usually different. Difference of the properties of microphones appears as a mismatch of the acoustic features of the recording sound, which causes recognition performance degradation. The difference of properties of microphones does not change with time and appears as difference of mean spectra. Therefore, the components that simply depend on recording environments can be subtracted from features by subtracting the mean spectra.

Typical connection

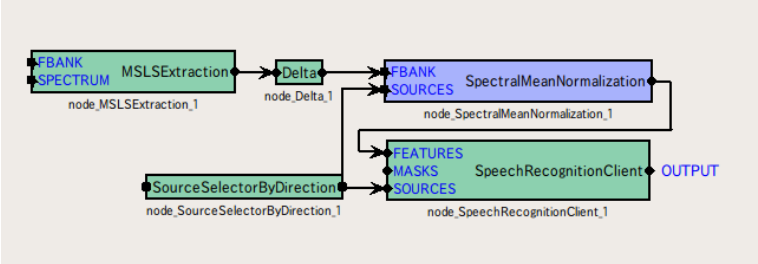


Figure 6.50: Connection example of SpectralMeanNormalization

Input-output and property of the module

Table 6.39: Parameter list of SpectralMeanNormalization

Parameter name	Type	Default value	Unit	Description
FBANKCOUNT	`int`	13		Dimension number of input feature parameter

Input

FBANK : `Map<int, ObjectRef>`type. A pair of the sound source ID and feature vector as `Vector<float>`type data.

SOURCES : It is `Vector<ObjectRef>`type. Sound source position.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and feature vector as `Vector<float>`type data.

Parameter

FBANKCOUNT : `int`type. Its range is 0 or a positive integer.

Details of the module

This module subtracts the mean of features from the input acoustic features. However, as a problem, to realize real-time processing, the mean of the utterance concerned cannot be subtracted. It is necessary to estimate or approximate mean values of the utterance concerned using some values. Real time mean subtraction is realized by assuming the mean of the former utterance as an approximate value and subtracting it instead of subtracting the mean of the utterance concerned. In this method, a sound source direction must be considered additionally. Since transfer functions differ depending on sound source directions, when the utterance concerned and the former utterance are received from different directions, the mean of the former utterance is inappropriate compared with the mean approximation of the utterance concerned. In such a case, the mean of the utterance that is uttered before the utterance concerned from the same direction is used as a mean approximation of the utterance concerned. Finally, the mean of the utterance concerned is calculated and it is maintained in a memory as a mean of the direction of the utterance concerned for mean subtraction subsequently performed. When the sound source moves by more than 10[deg] during utterance, a mean is calculated as another sound source.

6.5 MFM category

6.5.1 DeltaMask

Outline of the module

This module obtains missing feature mask vectors of dynamic features from those of static features and generates mas vectors consisting of the missing feature mask vectors of static and dynamic features.

Necessary file

No files are required.

Usage

When to use

This module is used to perform speech recognition by masking features depending on reliability based on the missing feature theory. It is usually used for the latter half of MFMGeneration.

Typical connection

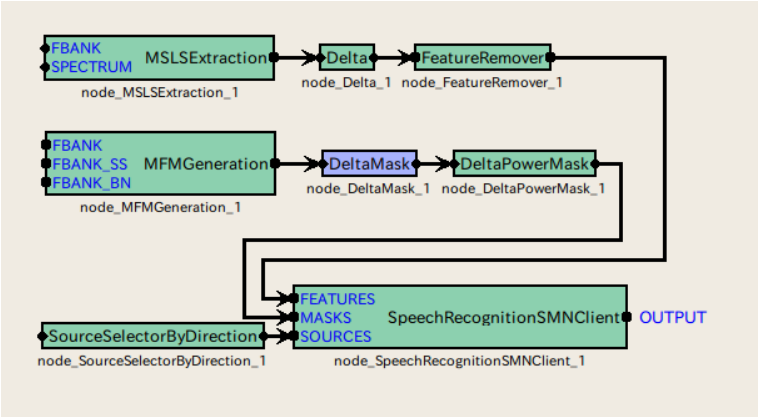


Figure 6.51: Typical connection example of DeltaMask

Input-output and property of the module

Table 6.40: Parameter list of DeltaMask

Parameter name	Type	Default value	Unit	Description
FBANKCOUNT	`int`			Dimension number of static feature

Input

INPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and mask vectorof the feature as `Vector<float>`type data. The mask value is a real number from 0.0 to 1.0. 0.0 indicates the feature is not reliable and 1.0 indicates it is reliable.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and mask vectorof the feature as `Vector<float>`type data. The mask value is a real number from 0.0 to 1.0. 0.0 indicates the feature is not reliable and 1.0 indicates it is reliable.

Parameter

FBANKCOUNT : `int`type. A dimension number of the feature to be processed. Its range is positive integer.

Details of the module

This module obtains missing feature mask vectors of dynamic features from those of static features and generates mas vectors consisting of the missing feature mask vectors of static and dynamic features. The input mask vector at the frame time f is expressed as;

$$\boldsymbol{m}(f) = [m(f,0), m(f,1), \ldots, m(f,2P-1)]^T \tag{6.110}$$

Here, P indicates the dimension number that indicates the static feature among the input mask vectors and is given in FBANKCOUNT. The mask value of the dynamic features is obtained with that of the static features, the output vector is generated by substituting it to the dimensional element from P to $2P-1$. The output vector $\boldsymbol{m}'(f)$ is expressed as follows.

$$\boldsymbol{y}'(f) = [m'(f,0), m'(f,1), \ldots, m'(f,2P-1)]^T \tag{6.111}$$

$$m'(f,p) = \begin{cases} m(f,p), & \text{if } p = 0,\ldots,P-1, \\ \prod_{\tau=-2}^{2} m(f+\tau,p), & \text{if } p = P,\ldots,2P-1, \end{cases} \tag{6.112}$$

Figure 6.52 shows an input-output flow of DeltaMask.

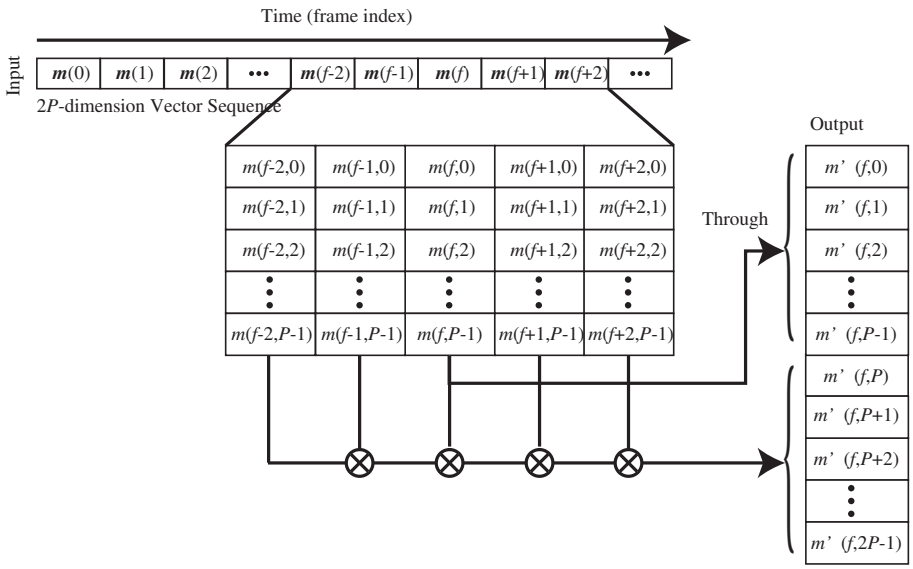


Figure 6.52: Input-output flow of DeltaMask.

6.5.2 DeltaPowerMask

Outline of the module

This module generates a mask value of the dynamic logarithmic power, which is one of the acoustic features. The generated mask value is added to the mask vector element of an input.

Necessary file

No files are required.

Usage

When to use

This module is used to perform speech recognition by masking features depending on reliability based on the missing feature theory. It is usually used for the posterior half of DeltaMask.

Typical connection

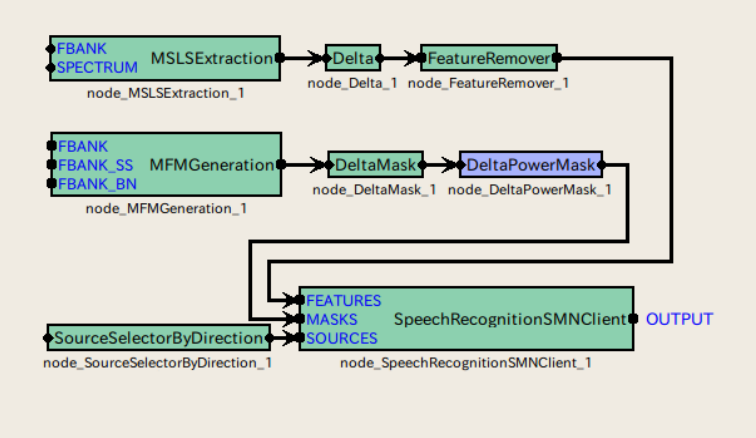


Figure 6.53: Typical connection example of DeltaPowerMask

Input-output and property of the module

INPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and mask vector of the feature as `Vector<float>`type data. The mask value is a real numbers from 0.0 to 1.0. 0.0 indicates the feature is not reliable and 1.0 indicates it is reliable.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and mask vector of the feature as `Vector<float>`type data. The mask value is a real numbers from 0.0 to 1.0. 0.0 indicates the feature is not reliable and 1.0 indicates it is reliable.

Parameter

Details of the module

This module generates a mask value of the dynamic logarithmic power, which is one of the acoustic features. The mask value generated is 1.0 consistently. The dimension number of the output mask is the mask's dimension number +1.

6.5.3 MFMGeneration

Details of the module

This module generates Missing Feature Masks (MFM) for speech recognition based on the missing feature theory.

Necessary file

No files are required.

When to use

This module is used to perform speech recognition based on the missing feature theory. MFMGenerationgenerates Missing Feature Masks from the outputs of PostFilterand GHDSS. Therefore, PostFilterand GHDSSare used as a premise.

Typical connection

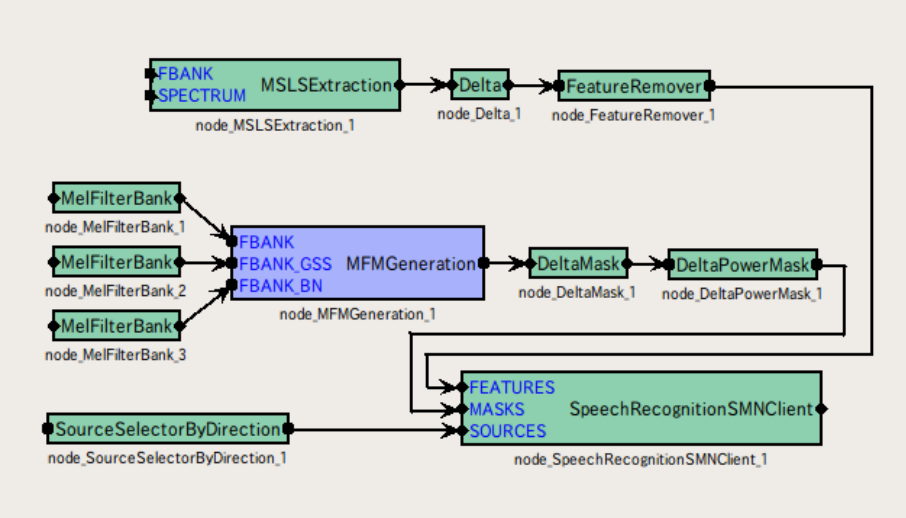


Figure 6.54: Connection example of MFMGeneration

Input-output and property of the module

Table 6.41: Parameter list of MFMGeneration

Parameter name	Type	Default value	Unit	Description
FBANKCOUNT	`int`	13		Dimension number of acoustic feature
THRESHOLD	`float`	0.2		Threshold value to quantize continuous values between 0.0 and 1.0 to 0.0 (not reliable) or 1.0 (reliable)

Input

FBANK : `Map<int, ObjectRef>`type. A pair of the sound source ID and vector that consists of mel filter bank output energy obtained from the output of PostFilteras `Vector<float>`type data.

FBANKSS : `Map<int, ObjectRef>`type. A pair of the sound source ID and vector that consists of mel filter bank output energy obtained from the output of GHDSSas `Vector<float>`type data.

FBANKBN : `Map<int, ObjectRef>`type. A pair of the sound source ID and vector that consists of mel filter bank output energy obtained from the output of BGNEstimatoras `Vector<float>`type data.

Output

OUTPUT : `Map<int, ObjectRef>`type. A pair of the sound source ID and vector that consists of missing feature vectors as `Vector<float>`type data. Vector elements are 0.0 (not reliable) or 1.0 (reliable). Output vectors are 2*FBANKCOUNT dimension and dimension elements more than FBANKCOUNT are all 0. It is a place electrode holder of Missing Feature Masks for quantity of dynamical characteristics.

Parameter

FBANKCOUNT : `int`type. A dimension of acoustic features.

THRESHOLD : `float`type. The threshold value to quantize continuous values between 0.0 (not reliable) and 1.0 (reliable). When setting to 1.0, all features are trusted continuously and it becomes equivalent to the normal speech recognition processing.

Details of the module

This module generates missing feature masks (MFM) for speech recognition based on the missing feature theory. Threshold processing is performed for the reliability $r(p)$ with the threshold value THRESHOLD and the mask value is quantized to 0.0 (not reliable) or 1.0 (reliable). The reliability is obtained from the output energy $f(p)$, $b(p)$, $g(p)$, of the mel filter bank obtained from the output of PostFilter, GHDSSand BGNEstimator. Here, the mask vector of the frame number f is expressed as;

$$\boldsymbol{m}(f) = [m(f,0), m(f,1), \ldots, m(f,P-1)]^T \tag{6.113}$$

$$m(f,p) = \begin{cases} 1.0, & r(p) > \text{THRESHOLD} \\ 0.0, & r(p) \leq \text{THRESHOLD} \end{cases}, \tag{6.114}$$

$$r(p) = \min(1.0, (f(p) + 1.4 * b(p))/(fg(p) + 1.0)), \tag{6.115}$$

$$\tag{6.116}$$

Here, P is the dimension number of the input feature vector and is positive integer designated in FBANKCOUNT. The dimension number of the vector actually output is 2*FBANKCOUNT. Dimension elements more than FBANKCOUNT are filled up with 0. This is a placeholder to be put in dynamic feature values. Figure 6.55 shows a schematic view of an output vector sequence.

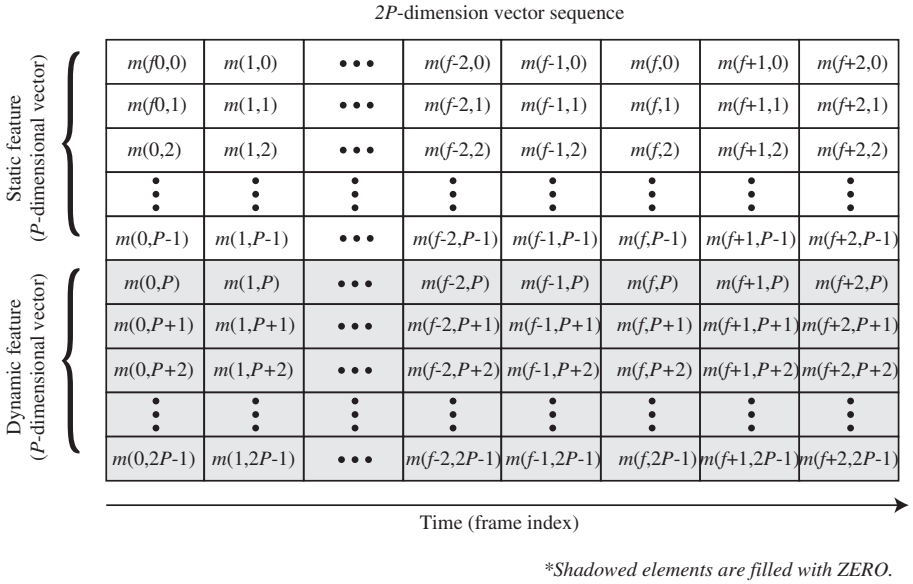


Figure 6.55: Output vector sequence of MFMGeneration

6.6 ASRIF category

6.6.1 SpeechRecognitionClient

Outline of the module

This module sends acoustic features to the speech recognition module via a network.

Necessary file

No files are required.

When to use This module is used to send acoustic features to software out of HARK. For example, it sends them to the large vocabulary continuous speech recognition software Julius [(1)] and performs speech recognition.

Typical connection

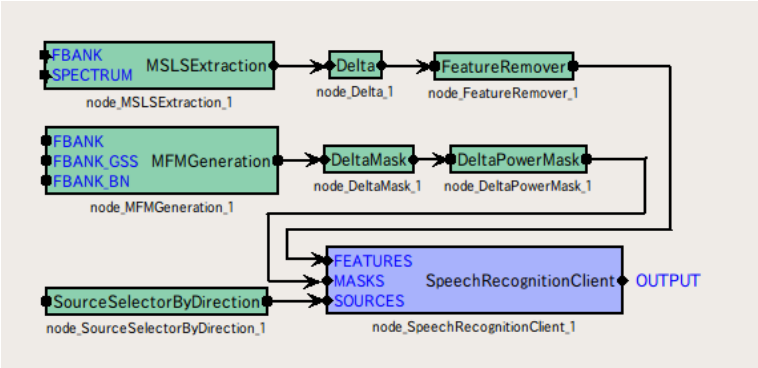


Figure 6.56: Connection example of SpeechRecognitionClient

Input-output and property of the module

Table 6.42: Parameter list of SpeechRecognitionClient

Parameter name	Type	Default value	Unit	Description
MFTENABLED	bool	true		Select whether or not to send out missing feature masks
HOST	string	127.0.0.1		Host name /IP address of the server in which Julius/Julian works
PORT	int	5530		Port number for sending out to network
SOCKETENABLED	bool	true		The flag that determines whether or not to output to the socket

Input

FEATURES : Map<int, ObjectRef>type. A pair of the sound source ID and feature vector as Vector<float>type data.

MASKS : Map<int, ObjectRef>type. A pair of the sound source ID and mask vector as Vector<float>type data.

SOURCES : Sourcetype.

Output

OUTPUT : `Vector<ObjectRef>`type.

Parameter

MFTENABLED : `bool`type. When `true`is selected, MASKS is transmitted. When `false`is selected, MASKS of an input is ignored, mask of 1 is transmitted for all.

HOST : `string`type. IP address of a host that transmits acoustic parameters. When SOCKETENABLED is set to `false`, it is invalid.

PORT : `int`type. A socket number to transfer acoustic parameters. When SOCKETENABLED is set to `false`, it is invalid.

SOCKETENABLED : `bool`type. When `true`, acoustic parameters are transmitted to the socket and when `false`, they are not transmitted.

Details of the module

When MFTENABLED is set to `true`and SOCKETENABLED, this module sends acoustic features and mask vectors to the speech recognition module via the network port. When `false`is selected for MFTENABLED, the speech recognition that is not based on the missing feature theory is performed. In actual operations, mask vectors are sent out with all mask vectors as 1, all acoustic features as reliable in other words. When `false`is selected for SOCKETENABLED, the features are not sent to the speech recognition module. This module is used to perform network operation checks of HARK without running the external program since the speech recognition engine depends on an external program. For HOST, designate an IP address of HOST for which the external program that sends vectors runs. For PORT, designate a network port number to send vector.

References:

(1)http://julius.sourceforge.jp/enindex.php

6.6.2 SpeechRecognitionSMNClient

Outline of the module

This module sends acoustic features to the speech recognition module via a network. The different point from SpeechRecognitionClientis that this module performs mean subtraction (Spectral Mean Normalization: SMN) of input feature vectors. However, as a problem, in order to realize real-time processing, the mean of the utterance concerned cannot be subtracted. It is necessary to estimate or approximate mean values of the utterance concerned using some values. For the details of the approximation processing, see Details of the module.

Necessary file

No files are required.

Usage

When to use

This module is used to send acoustic features to software out of HARK. For example, it sends them to the large vocabulary continuous speech recognition software Julius [(1)] and performs speech recognition.

Typical connection

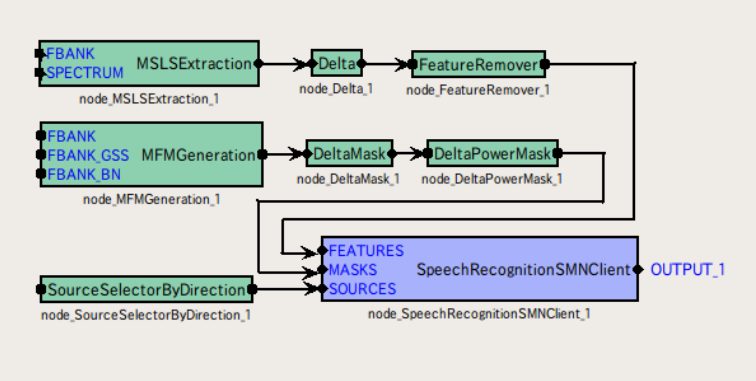


Figure 6.57: Connection example of SpeechRecognitionSMNClient

Input-output and property of the module

Table 6.43: Parameter list of SpeechRecognitionSMNClient

Parameter name	Type	Default value	Unit	Description
MFTENABLED	bool	true		Select whether or not to send out missing feature masks
HOST	string	127.0.0.1		Host name /IP address of the server in which Julius/Julian works
PORT	int	5530		Port number for sending out to network
SOCKETENABLED	bool	true		The flag that determines whether or not to output to the socket

Input

FEATURES : Map<int, ObjectRef>type. A pair of the sound source ID and feature vector as Vector<float>type data.

MASKS : `Map<int, ObjectRef>`type. A pair of the sound source ID and mask vector as `Vector<float>`type data.

SOURCES : `Source`type.

Output

OUTPUT : `Vector<ObjectRef>`type.

Parameter

MFTENABLED : `bool`type. When `true`is selected, MASKS is transmitted. When `false`is selected, MASKS of an input is ignored, mask of 1 is transmitted for all.

HOST : `string`type. The IP address of a host that transmits acoustic parameters. When SOCKETENABLED is set to `false`, it is invalid.

PORT : `int`type. The socket number to transfer acoustic parameters.When SOCKETENABLED is set to `false`, it is invalid.

SOCKETENABLED : `bool`type. When `true`, acoustic parameters are transmitted to the socket and when `false`, they are not transmitted.

When MFTENABLED is set to `true`and SOCKETENABLED, this module sends acoustic features and mask vectors to the speech recognition module via the network port. When `false`is selected for MFTENABLED, the speech recognition that is not based on the missing feature theory is performed. In actual operations, mask vectors are sent out with all mask vectors as 1, all acoustic features as reliable in other words. When `false`is selected for SOCKETENABLED, the features are not sent to the speech recognition module. This module is used to perform network operation checks of HARK without running the external program since the speech recognition engine depends on an external program. For HOST, designate an IP address of HOST for which the external program that sends vectors runs. For PORT, designate a network port number to send vector.

References:

(1)http://julius.sourceforge.jp/enindex.php

6.7 MISC category

6.7.1 ChannelSelector

Outline of the module

This module extracts only the data of specified channels from speech waveform data of the multichannels in a specified order.

Necessary file

No files are required.

Usage

When to use

This module is used when deleting unnecessary channels from speech waveform data of input multichannels, changing the alignment of channels or copying channels.

Typical connection

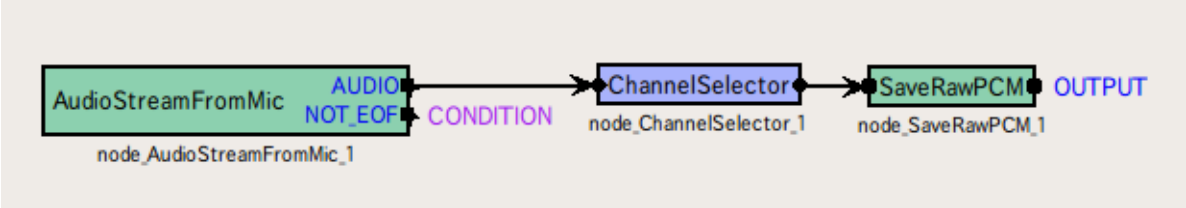


Figure 6.58: Typical connection example of ChannelSelector

Figure 6.58 shows a typical connection example. Only several channels of the audio files of the multichannels can be extracted with this network file. The main input sources are AudioStreamFromMicand AudioStreamFromWaveand the main output sources are SaveRawPCM, MultiFFT.

Input-output and property of the module

Input

INPUT : `Matrix<float>`type. Speech waveform data of multichannels.

Output

OUTPUT : `Matrix<float>`type. Speech waveform data of multichannels.

Parameter

Parameter list of ChannelSelector

Parameter name	Type	Default value	Unit	Description
SELECTOR	`Vector< int >`	`Vector< int >`		Designate the number of channels to output

SELECTOR : Vector<int>type. No default value. Designate the numbers of the channels to be used. Channel numbers begin with 0. Example: When using only Channels 2, 3 and 4 among the five channels (0-4), designate as (1) and when furthermore replacing Channels 3 and 4 furthermore designate as (2).

(1) <Vector<int> 2 3 4>

(2) <Vector<int> 2 4 3>

Details of the module

This module extracts only the speech waveform data of the channels designated from the $N \times M$ type matrix (Matrix) of an input and outputs data of a new $N' \times M$ type matrix. Here, N indicates the input channel number and N' indicates the number of output channels.

6.7.2 DataLogger

Outline of the module

This module gives labels specified for parameters to input data and outputs data as standard outputs or to files.

Necessary file

No files are required.

Usage

When to use

This module is used for debugging for modules or when saving outputs of a module and use them for experiments and analyses.

Typical connection

For example, when outputting features of each sound source in text and analyzing them, connect the module as follows.

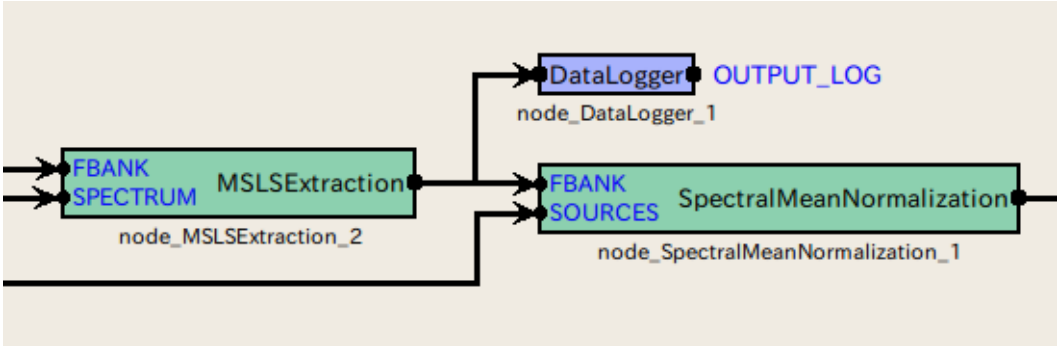


Figure 6.59: Connection example of DataLogger

Input-output and property of the module

Table 6.44: Parameter list of ModuleName

Parameter name	Type	Default value	Unit	Description
LABEL	`string`			Label be given to output data

Input

INPUT : `any`. Note that the supported types are `Map<int, float>`, `Map<int, int>`or `Map<int, ObjectRef>`. `ObjectRef`of `Map<int, ObjectRef>`supports only `Vector<float>`or `Vector<complex<float> >`.

Output

OUTPUT : Same as inputs.

Parameter

LABEL : Specify character strings to be given to output data so that the user can figure out which DataLoggeroutputs the result when plural DataLoggerare used.

Details of the module

This module gives labels specified for parameters to input data and outputs data as standard outputs or to files. The type supported by this module is only the Maptype with the sound source IDs that are often used for HARK as a key. The output format is as follows. Label frame count key 1 value-1 key two values-two ... The output of 1 frame count of this module is in one row. As above, the label specified for parameters is output first and the frame count is output next followed by all Maptype keys and values in a space-delimited format. When values are Vectortype, all elements are output in a space-delimited format.

6.7.3 MatrixToMap

Outline of the module

This module converts `Matrix<float>`type and `Matrix<complex<float> >`type data into `Map<int, ObjectRef>`type data.

Necessary file

No files are required.

Usage

When to use

This module is used when the input is connected to the nodes that accept only `Map<int, ObjectRef>`type, such as PreEmphasis, MelFilterBankor SaveRawPCM.

Typical connection

Figures 6.60 and 6.61 show connection examples of the MatrixToMapmodule. In Figure 6.60, speech waveform data are taken from microphones in the AudioStreamFromMicmodule, necessary channels are sorted in the ChannelSelectormodule and `Matrix<float>`type data are converted into `Map<int, ObjectRef>`type with the MatrixToMapmodule. The output is connected to the SaveRawPCMmodule and waveforms are saved in files. Figure 6.61 shows the usage of `Map<int, ObjectRef>`type to obtain wave form spectra as `Map<int, ObjectRef>`type. As shown in the figure, the MultiFFTmodule can be connected to either side ofMatrixToMap.

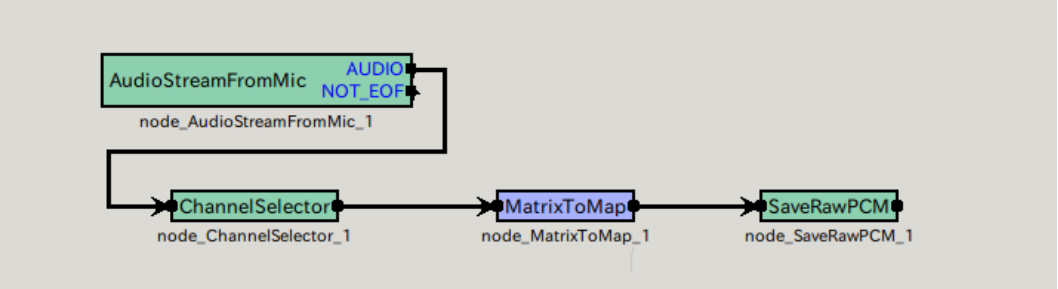


Figure 6.60: Connection example – Connection to MatrixToMap

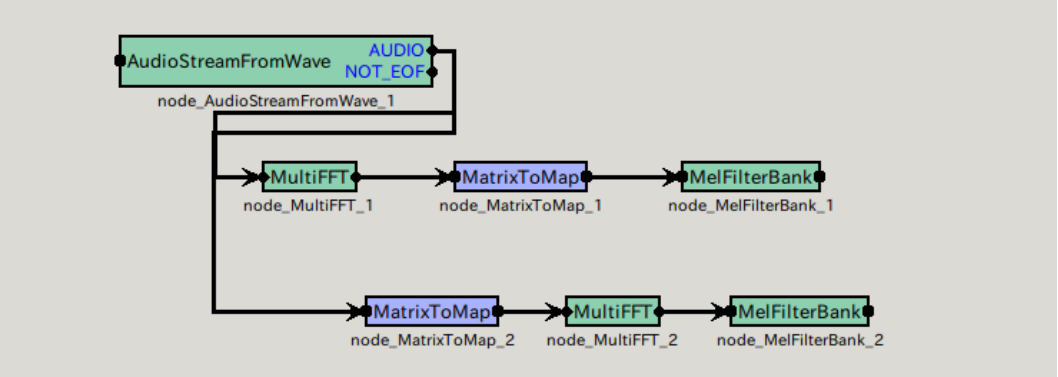


Figure 6.61: Connection example of MatrixToMap– Connection to MultiFFT

Input-output and property of the module

Input

INPUT : `Matrix<float>`or `Matrix<complex<float> >`types.

Output

OUTPUT : `Map<int, ObjectRef>`type. The structure which ID was given to in input data.

Parameter No parameters.

Details of the module

ID: Values of ID are 0 continuously.

6.7.4 MultiDownSampler

Outline of the module

This module performs downsampling of input signals and outputs its results. The window method is used for low-pass filters and its window function is Kaiser window.

Necessary file

No files are required.

Usage

When to use

This module is used when sampling frequency of input signals is not 16kHz. For the HARK modules, sampling frequency is supposed to be 16kHz basically. Therefore, in the case that input signals are of 48kHz for example, it is necessary to perform downsampling to lower the sampling frequency to 16kHz. **Note 1 (Range of ADVANCE):** In the convenience of processing, it is necessary to give limitations to the parameter setting of input modules that are connected before the modules such as AudioStreamFromMicand AudioStreamFromWave. The difference of those parameters LENGTH and ADVANCE: OVERLAP = LENGTH - ADVANCE must be sufficiently large. More concretely, it must be greater than the low-pass filter length N of this module. Values over 120 are sufficient for the setting of the default of this module and therefore no problems occur if ADVANCE is more than a quarter of LENGTH. Moreover, it is necessary to satisfy the requirements below **Note 2**. **Note 2 (Setting of ADVANCE):** The ADVANCE value of this module must be SAMPLINGRATEIN / SAMPLINGRATEOUT times as great as the ADVANCE value of the module connected after (e.g. GHDSS). Since this is a specification, its operation is not guaranteed with the values other than the above. For example, in the case of ADVANCE = 160 and of SAMPLINGRATEIN / SAMPLINGRATEOUT = 3 for the module that is connected after, it is necessary to set to 480 for ADVANCE of this module and that connected before. **Note 3 (Requirements for the LENGTH value of the module connected before this module):** The LENGTH value of the module connected before this module (e.g. AudioStreamFromMic) must be SAMPLINGRATEIN / SAMPLINGRATEOUT times as great as the ADVANCE value of the module connected after (e.g. GHDSS). For example, in the case of SAMPLINGRATEIN / SAMPLINGRATEOUT = 3, if LENGTH = 512 and ADVANCE = 160 for GHDSS, it is desired to have LENGTH = 1536 and ADVANCE = 480 for AudioStreamFromMic.

Typical connection

Typical connection examples are shown below. This network file reads Wave file inputs, performs downsampling and saves files as Raw files. The Wave file input is achieved by connecting Constant, InputStreamand AudioStreamFromMic. After that, it performs downsampling with MultiDownSamplerand saves output waveforms in SaveRawPCM.

Input-output and property of the module

Input

INPUT : `Matrix<float>`type. Multichannel speech waveform data (time domain waveform).

Output

OUTPUT : `Matrix<float>`type. The multichannel speech waveform data for which downsampling is performed (time domain waveform).

Parameter

Low-pass filters, frequency characteristics of Kaiser window here, are mostly set for the parameters. Figure 6.64 shows relations between symbols and filter properties. See carefully the correspondence when reading them.

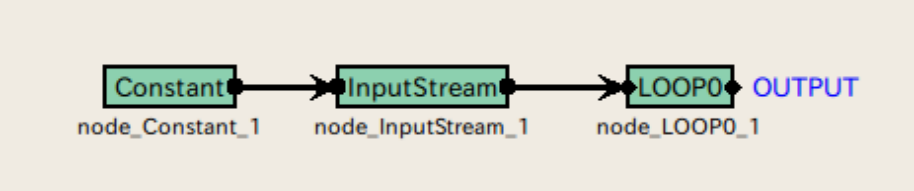


Figure 6.62: Connection example of MultiDownSampler: Main network

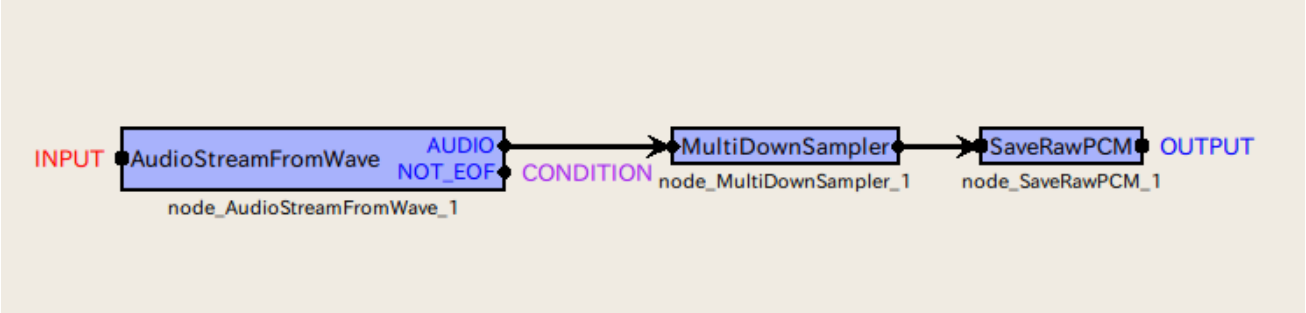


Figure 6.63: Connection example of MultiDownSampler: Iteration(LOOP0) Network

Table 6.45: Parameter list of MultiDownSampler

Parameter name	Type	Default Value	Unit	Description
ADVANCE	int	480	[pt]	Frame shift length for every iteration in INPUT signals. Since special setting is required, see the parameter description.
SAMPLINGRATEIN	int	48000	[Hz]	Sampling frequency of INPUT signals.
SAMPLINGRATEOUT	int	16000	[Hz]	Sampling frequency of OUTPUT signals.
Wp	float	0.28	$[\frac{\omega}{2\pi}]$	Low-pass filter pass band end. Designate normalized frequency [0.0 - 1.0] with INPUT as reference.
Ws	float	0.34	$[\frac{\omega}{2\pi}]$	Low-pass filter stopband end. Designate normalized frequency [0.0 - 1.0] with INPUT as reference.
As	float	50	[dB]	Minimum attenuation in stopband.

ADVANCE : inttype. The default value is 480. For processing frames for speech waveform, designate shift width on waveforms in sampling numbers. Here, use values that are used for the modules connected before INPUT. **Note:** This value must be SAMPLINGRATEIN / SAMPLINGRATEOUT times as great as the ADVANCE value set for after OUTPUT.

SAMPLINGRATEIN : inttype. The default value is 48000. Designate sampling frequency for input waveforms.

SAMPLINGRATEOUT : inttype. The default value is 16000. Designate sampling frequency for output waveforms. Note that values that are 1 / integer of SAMPLINGRATEIN can be used for this value.

Wp : floattype. The default value is 0.28. Designate the low-pass filter pass band end frequency by values of normalized frequency [0.0 - 1.0] with INPUT as reference. When the sampling frequency of inputs is 48000[Hz] and this value is set to 0.48, gains of low-pass filter begin to decrease from around $48000 * 0.28 = 13440$ [Hz].

Ws : floattype. The default value is 0.34. Designate the low-pass filter stopband end frequency by values of normalized frequency [0.0 - 1.0] with INPUT as reference. When the sampling frequency of inputs is 48000[Hz] and this value is set to 0.38, gains of low-pass filter begin to be stable from around $48000 * 0.34 = 16320$ [Hz].

As : floattype. The default value is 50. The value that indicates the minimum attenuation in stopband in [dB]. When using the default value, the gain of the stopband is around -50 [dB] with the passing band as 0.

Here, when Wp, Ws and As are set to severe values, when Wp and Ws are set to the values around the cutoff frequency Ws for example, the frequency response characteristic accuracy of Kaiser window improves. However, dimensions of the low-pass filter increase and processing time is increased. This relation is trade off.

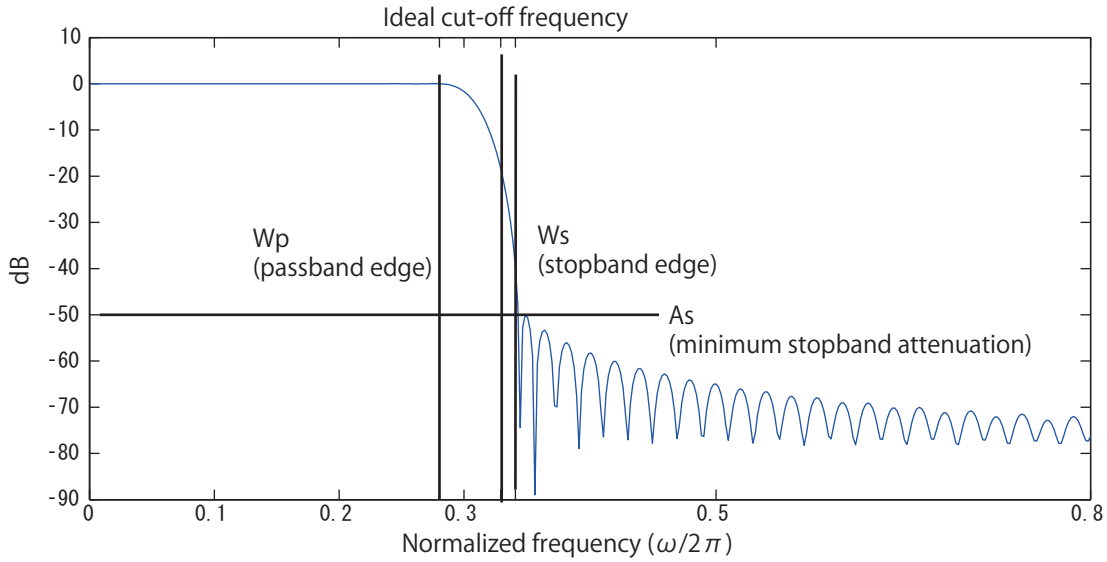


Figure 6.64: Filter property of default setting: Horizontal axis normalized frequency [$\omega/2\pi$], vertical axis gain [dB]

Details of the module

MultiDownSampleris the module that performs band limiting by the low-pass filter using the Kaiser window for multichannel signals and downsampling. Concretely, this module performs downsampling of SAMPLINGRATEOUT/SAMPLINGRATEIN after creating / executing a FIR low-pass filter by synthesizing 1) Kaiser window and 2) ideal low-pass responses. **FIR filter:**

The filtering with the finite impulse response $h(n)$ is performed based on the following equation.

$$s_{\text{out}}(t) = \sum_{i=0}^{N} h(n) s_{\text{in}}(t-n) \tag{6.117}$$

Here, $s_{\text{out}}(t)$ indicates output signals and $s_{\text{in}}(t)$ indicates input signals. In the case of the multichannel signals, the filtering is performed for signals of each channel independently. The same finite impulse response $h(n)$ is used here. **Ideal low-pass**

response: The ideal low-pass response whose cutoff frequency is ω_c is obtained by the following equation.

$$H_i(e^{j\omega}) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \text{otherwise} \end{cases} \tag{6.118}$$

This impulse response is expressed as follows.

$$h_i(n) = \frac{\omega_c}{\pi}\left(\frac{sin(\omega n)}{\omega n}\right), \quad -\infty \leq n \leq \infty \tag{6.119}$$

This impulse response does not satisfy the acausal and bounded input-bounded output (BIBO) stability conditions. It is necessary to cut off the impulse response in the middle to obtain the FIR filter from this ideal filter.

$$h(n) = \begin{cases} h_i(n), & |n| \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.120}$$

Here, N indicates a dimension of the filter. In this filter, a ripple occurs in the pass band and stopband by cutoff of the impulse response. Moreover, the minimum attenuation in stopband As also stays around 21dB and sufficient attenuation is not obtained. **Low-pass filter by the window method with Kaiser window:** In order to improve the properties by

the above cutoff method, the following impulse response, for which the ideal impulse response $h_i(n)$ is multiplied by the window function $v(n)$, is used instead.

$$h(n) = h_i(n)v(n) \tag{6.121}$$

Here, the low-pass filter is designed with the Kaiser window. The Kaiser window is defined by the following equation.

$$v(n) = \begin{cases} \frac{I_0\left(\beta\sqrt{1-(nN/2)^2}\right)}{I_0(\beta)}, & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.122}$$

Here, β indicates the parameter to determine shape of the window and $I_0(x)$ indicates the modified Bessel function of 0th order. The Kaiser window is obtained by the following equation.

$$I_0(x) = 1 + \sum_{k=1}^{\infty}\left(\frac{(0.5x)^k}{k!}\right) \tag{6.123}$$

The parameter β is determined by the attenuation obtained by the low-pass filter. Here, it is determined by the following index.

$$\beta = \begin{cases} 0.1102(As - 8.7) & As > 50, \\ 0,5842(As - 21)^{0.4} + 0.07886(As - 21) & 21 < As < 50, \\ 0 & As < 21 \end{cases} \tag{6.124}$$

For the rest, if the cutoff frequency ω_c and the filter order are determined, the low-pass filter is realized by the window method. The filter order N is estimated with the minimum attenuation in stopband As and the transition region $\Delta f = (Ws - Wp)/(2\pi)$ as follows.

$$N \approx \frac{As - 7.95}{14.36\Delta f} \tag{6.125}$$

Moreover, the cutoff frequency ω_c is set to $0.5(Wp + Ws)$. **Downsampling:** Downsampling is realized by thinning the

sample points of SAMPLINGRATEIN / SAMPLINGRATEOUT from the signals that pass the low-pass filter. For example, 48000/16000 = 3 in the default setting and therefore input samples taken once in three times will be output samples.

References:

(1) Author: Translated by P. Vaidyanathan: Akinori Nishihara, Eiji Watanabe, Toshiyuki Yoshida, Nobuhiko Sugino: "Multirate signal processing and filter bank", Science and technology publication, 2001.

6.7.5 MultiFFT

Outline of the module

This module performs Fast Fourier Transform (FFT) for multichannel speech waveform data.

Necessary file

No files are required.

Usage

When to use

This module is used to convert multichannel speech waveform data into a spectrum and analyze with time frequency domain. It is often used as preprocessing of feature extraction for speech recognition.

Typical connection

Figure 6.65 shows an example of giving inputs of `Matrix<float>`and `Map<int, ObjectRef>`types to the MultiFFTmodule. The path in Figure 6.65 receives multichannel acoustic signals of the `Matrix<float>`type from the AudioStreamFromWavemodule. The signals are then converted into `Matrix<complex<float> >`type complex spectrum in the MultiFFTmodule and input to the LocalizeMUSICmodule.

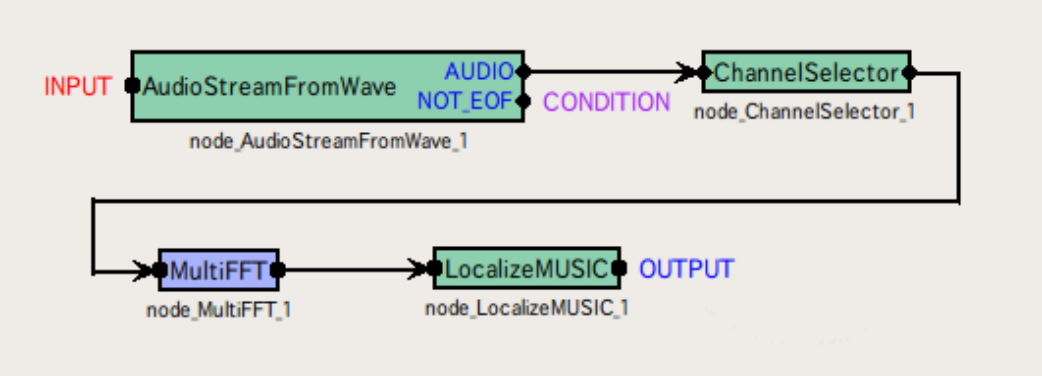


Figure 6.65: Connection example of MultiFFT

Input-output and property of the module

Table 6.46: Parameter list of MultiFFT

Parameter name	Type	Default value	Unit	Description
LENGTH	`int`	512	[pt]	Length of signals to be Fourier transformed
WINDOW	`string`	CONJ		Type of window function when performing Fourier transform. Select from CONJ, HAMMING and RECTANGLE, which indicate complex window, hamming window and rectangle window, respectively.
WINDOWLENGTH	`int`	512	[pt]	Length of window function when performing Fourier transform.

Input

INPUT : `Matrix<float>`or `Map<int, ObjectRef>`types. Multichannel speech waveform data. Assuming the matrix size is $M \times L$, M indicates the number of channels and L indicates the sample numbers of waveforms. L must be equal to the parameter LENGTH.

Output

OUTPUT : `Matrix<complex<float> >`or `Map<int, ObjectRef>`types. Multichannel complex spectrum corresponding to inputs. When the inputs are `Matrix<float>`type, the outputs are `Matrix<complex<float> >`type and when the inputs are `Map<int, ObjectRef>`type, the outputs are `Map<int, ObjectRef>`type. When the input matrix size is $M \times L$, the output matrix size is $M \times L/2 + 1$.

Parameter

LENGTH : `int`type. The default value is 512. Designate length of signals to be Fourier transformed. Its value is powers of 2 for the property of the algorithm. Moreover, it must be greater than WINDOWLENGTH.

WINDOW : `string`type. The default value is CONJ. Select from CONJ, HAMMING and RECTANGLE, which indicate complex window, hamming window and rectangle window, respectively. The HAMMING window is often used for audio signal analyses.

WINDOWLENGTH : `int`type. The default value is 512. Designate length of a window function. When increasing this value, the frequency resolution increases while the temporal resolution decreases. Intuitively, when increasing this value, it becomes sensitive to difference of sound pitches while it becomes insensitive to change of sound pitches.

Details of the module

Rough indication of LENGTH and WINDOWLENGTH: It is appropriate to analyze audio signals with frame length of 20 ~ 40 [ms]. Assuming the sampling frequency is f_s [Hz] and the temporal length of a window is x [ms], the frame length L [pt] is expressed as;

$$L = \frac{f_s x}{1000}$$

For example, in the case that the sampling frequency is 16 [kHz], the default value 512 [pt] corresponds to 32 [ms]. Powers of 2 are suited for the parameter LENGTH for the property of Fast Fourier Transform. Select 512. WINDOWLENGTH, the window function, is set to 400 [pt] (corresponds to 25 [ms] when the sampling frequency is 16 [kHz]) in some cases so as to designate frame length more suited for acoustic analyses. **Shape of each window function:** The shape of each window function $w(k)$ is as follows. k indicates the index of a sample, L indicates the length of window function and $NFFT$ indicates the FFT length. k moves within the range of $0 \leq k < L$. When the FFT length is greater than the window length, the window function value for function in $NFFT \leq k < L$ is 0. **CONJ, Complex window:**

$$w(k) = \begin{cases} 0.5 - 0.5\cos\frac{4kC}{L}, & \text{if } 0 \leq k < L/4 \\ \sqrt{\left(1.5 - 0.5\cos 2C - \frac{4kC}{L}\right)\left(0.5 + 0.5\cos 2C - \frac{4kC}{L}\right)}, & \text{if } L/4 \leq k < 2L/4 \\ \sqrt{\left(1.5 - 0.5\cos\frac{4kC}{L} - 2C\right)\left(0.5 + 0.5\cos\frac{4kC}{L} - 2C\right)}, & \text{if } 2L/4 \leq k < 3L/4 \\ 0.5 - 0.5\cos\left(4C - \frac{4kC}{L}\right), & \text{if } 3L/4 \leq k < L \\ 0, & \text{if } NFFT \leq k < L \end{cases},$$

$$w(k) = \begin{cases} 0.5 - 0.5\cos\left(\frac{4k}{L}C\right), & \text{if} 0 \leq k < L/4 \\ \sqrt{1 - \left\{0.5 - 0.5\cos\left(\frac{2L-4k}{L}C\right)\right\}^2}, & \text{if} L/4 \leq k < 2L/4 \\ \sqrt{1 - \left\{0.5 - 0.5\cos\left(\frac{4k-2L}{L}C\right)\right\}^2}, & \text{if} 2L/4 \leq k < 3L/4 \\ 0.5 - 0.5\cos\left(\frac{4L-4k}{L}C\right), & \text{if} 3L/4 \leq k < L \\ 0, & \text{if} NFFT \leq k < L \end{cases},$$

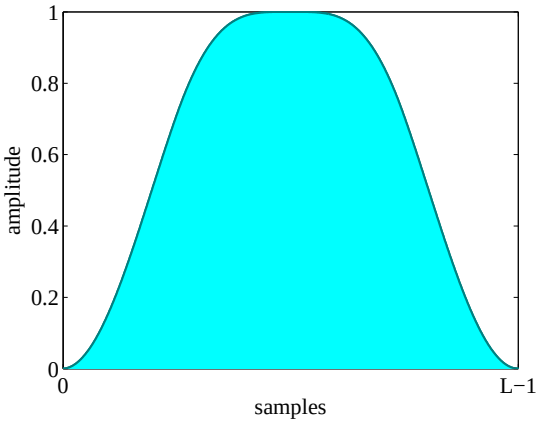


Figure 6.66: Shape of complex window function

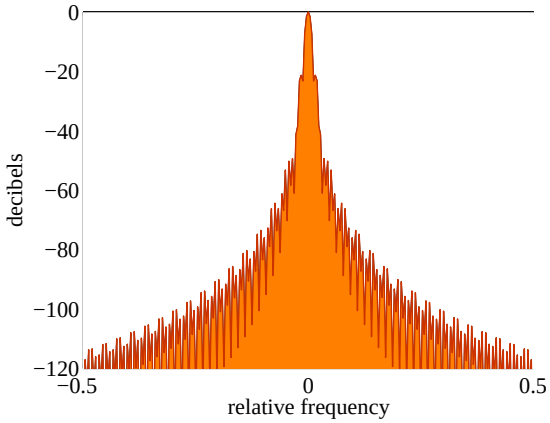


Figure 6.67: Frequency response of complex window function

Here, $C = 1.9979$. Figures 6.66 and 6.67 show shape and frequency responses of the complex window function, respectively. The horizontal axis in Figure 6.67 indicates mean values of relative frequency for sampling frequency. Generally, performance of frequency responses of a window function is better with a sharper peak at 0 in the horizontal axis. Components existing outside the center of the frequency response indicate the amount that the power of other frequency components that leak to a certain frequency bin when performing Fourier transformation. Values in the vertical axis indicate the amount that the power of other frequency components that leak to a certain frequency bin when performing Fourier transformation and so on.

HAMMING, Hamming window:

$$w(k) = \begin{cases} 0.54 - 0.46\cos\frac{2\pi k}{L-1}, & \text{if } 0 \le k < L, \\ 0, & \text{if } L \le k < NFFT \end{cases}$$

Here, π indicates the circular constant. Figures 6.68 and 6.68 show shape and frequency responses of the hamming window

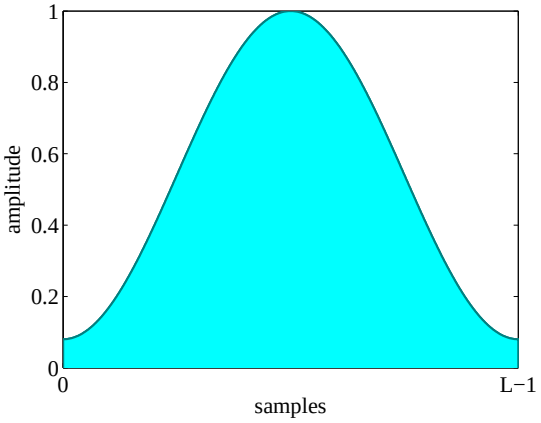


Figure 6.68: Shape of hamming window function

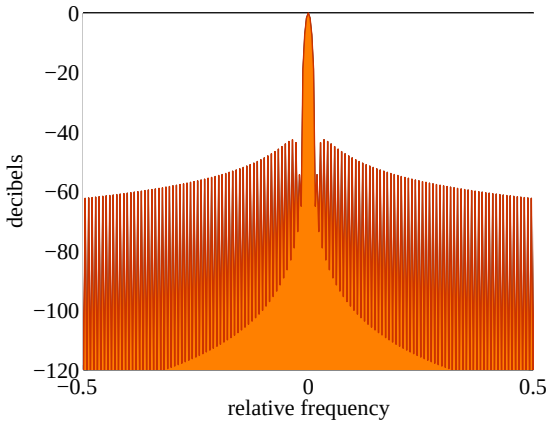


Figure 6.69: Frequency response of hamming window function

function. **RECTANGLE, Rectangle window:**

$$w(k) = \begin{cases} 1, & \text{if } 0 \le k < L \\ 0, & \text{if } L \le k < NFFT \end{cases}$$

Figures 6.70 and 6.70 show shape and frequency responses of the rectangle window function.

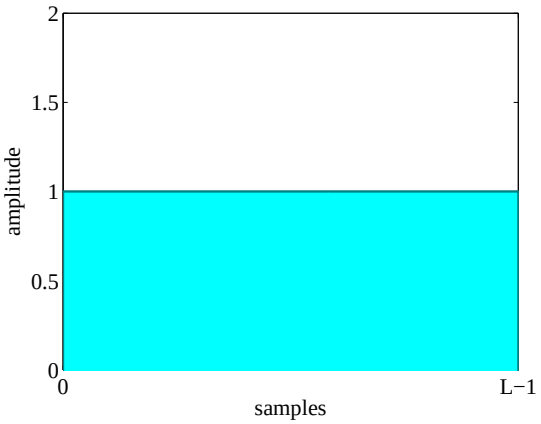


Figure 6.70: Shape of rectangle window function

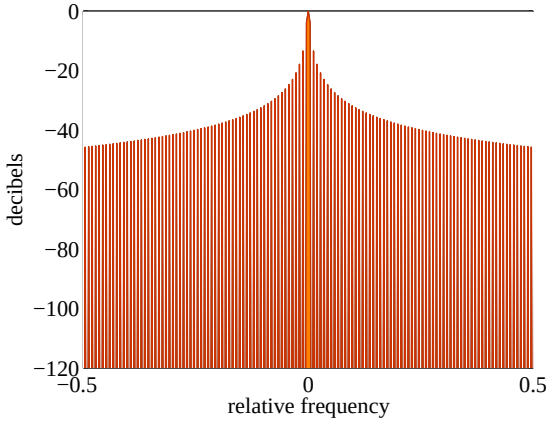


Figure 6.71: Frequency response of rectangle window function

6.7.6 MultiGain

Outline of the module

This module regulates gains of input signals.

Necessary file

No files are required.

Usage

When to use

This module is used to amplify input signals or so that they cannot be clipped. For example, when speech waveform data quantized with 24 [bit] are used for inputs and when using a system built for 16 [bit], this module is used to decrease the gains for 8 [bit].

Typical connection

This module is usually positioned just after AudioStreamFromMicor AudioStreamFromWavedirectly, or positioned with ChannelSelectorin between.

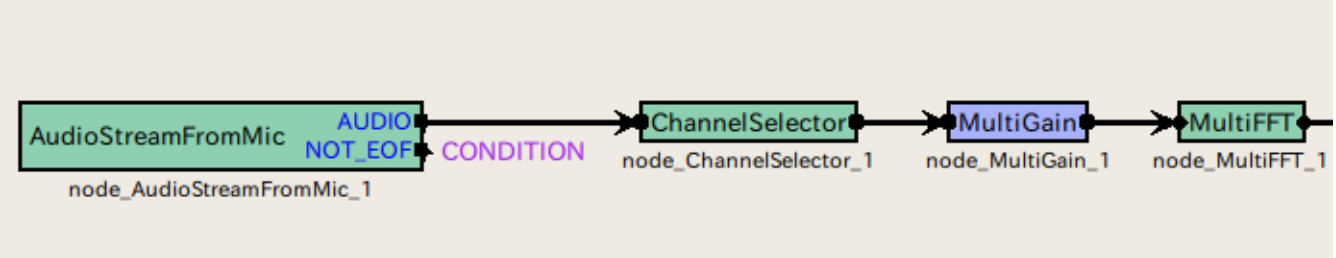


Figure 6.72: Connection example of MultiGain

Input-output and property of the module

Table 6.47: Parameter list of MultiGain

Parameter name	Type	Default value	Unit	Description
GAIN	`float`	1.0		Gain value

Input

INPUT : `Matrix<float>`type. Multichannel speech waveform data (time domain waveform).

Output

OUTPUT : `Matrix<float>`type. Multichannel speech waveform data (time domain waveform) for which gains are regulated.

Parameter

GAIN : `float`type. A gain parameter. Inputting 1.0 corresponds to outputting the inputs without change.

Details of the module

Each channel of inputs is output as the values multiplied by the value designated for the GAIN parameter. Note that the inputs are time domain waveforms. For example, when decreasing gains by 40 dB, calculate as follows and designate 0.01.

$$20 \log x = -40 \tag{6.126}$$

$$x = 0.01 \tag{6.127}$$

6.7.7 PowerCalcForMap

Outline of the module

This module converts multichannel complex spectrum with `Map<int, ObjectRef>`type IDs into the real power spectra.

Necessary file

No files are required.

Usage

When to use

This module is used to convert complex spectrum into the real power spectra and is used when inputs are `Map<int, ObjectRef>`type. When inputs are `Matrix<complex<float> >`type, use the PowerCalcForMatrixmodule.

Typical connection

Figure 6.73 shows a usage example of the PowerCalcForMapmodule. The `Map<int, ObjectRef>`type complex spectrum obtained from the MultiFFTmodule are converted into `Map<int, ObjectRef>`type power spectrum and input to the MelFilterBankmodule.

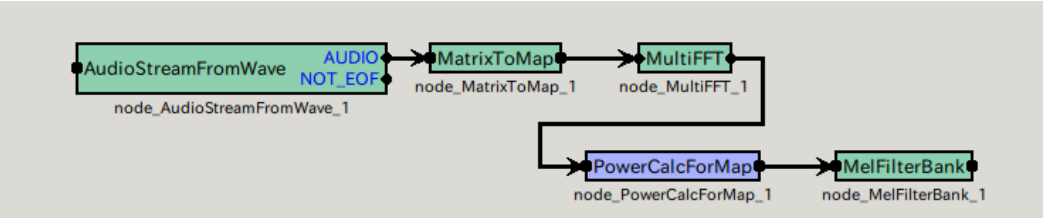


Figure 6.73: Connection example of PowerCalcForMap

Input-output and property of the module

Input

INPUT : `Map<int, ObjectRef>`type. Complex matrixes of `Matrix<complex<float> >`type is stored in the `ObjectRef`part.

Output

OUTPUT : `Map<int, ObjectRef>`type. Real matrixes of absolute values are stored in the `ObjectRef`part for each element of the complex matrixes of the inputs.

Parameter

No parameters.

Details of the module

The real matrix $N_{i,j}$ of the output for the complex matrix $M_{i,j}$ of an input (i, j indicate rows and columns of the index, respectively) is obtained as follows.

$$N_{i,j} = M_{i,j}M^*_{i,j},$$

Here, $M^*_{i,j}$ indicates the complex conjugate of $M_{i,j}$.

6.7.8 PowerCalcForMatrix

Outline of the module

This module converts `Matrix<complex<float> >`type multichannel complex spectrum into the real power spectra.

Necessary file

No files are required.

Usage

When to use

This module is used to convert complex spectrum into the real power spectra and is used when inputs are `Matrix<complex<float> >`type. When inputs are `Map<int, ObjectRef>`type, use the PowerCalcForMapmodule.

Typical connection

Figure 6.74 shows a usage example of the PowerCalcForMatrixmodule. The `Matrix<complex<float> >`type complex spectrum obtained from the MultiFFTmodule are converted into `Matrix<float>`type power spectrum and input to the BGNEstimator module.

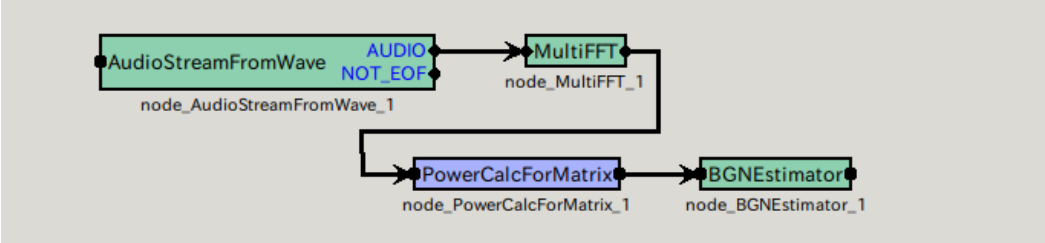


Figure 6.74: Connection example of PowerCalcForMatrix

Input-output and property of the module

Input

INPUT : `Matrix<complex<float> >`type. A matrix with each element as a complex number.

Output

OUTPUT : `Matrix<float>`type. A real matrix consisting of absolute values of each element of the input.

Parameter No parameters.

Details of the module

The real matrix $N_{i,j}$ of the output for the complex matrix $M_{i,j}$ of an input (i, j indicate rows and columns of the index, respectively) is obtained as follows.

$$N_{i,j} = M_{i,j}M^*_{i,j},$$

Here, $M^*_{i,j}$ indicates the complex conjugate of $M_{i,j}$.

6.7.9 SegmentAudioStreamByID

Outline of the module

This module extracts the acoustic streams that use ID information and output them adding ID information to them.

Necessary file

No files are required.

Usage

When to use

This module is useful to extract and process specific parts such as audio parts, not to process the entire acoustic signals as one stream. Since the extraction is performed with IDs as keys, ID information is essential for its inputs. This module extracts the section where the same ID is continuously seen as a signal section and outputs it as Mapdata of one channel adding its ID information to it.

Usage

When to use

It is presumed that the input stream is sound mixture of two audio signals. When comparing the signals separated with GHDSSwith the original sound mixture from the section that is temporally same as that of them, input the acoustic stream of 1 ch and the sound source detected by sound source localization to this module. In this case, its outputs are in the formats (Map) that are completely same as those of GHDSSand PostFilterseparated sounds.

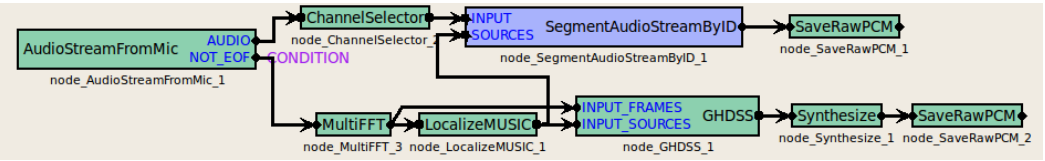


Figure 6.75: Connection example of SegmentAudioStreamByID

Input-output and property of the module

Input

INPUT : any, Matrix<complex<float> >or Matrix<float>types.

SOURCES : Vector<ObjectRef>type. Sound source directions with IDs. The contents of each Vector are Source type indicating the sound source information with IDs. Feature vectors are stored for each sound source. This parameter must be designated for any cases.

Output

OUTPUT : Map<int, ObjectRef>and ObjectRefare slender pointers to Vector<float>and Vector<complex<float> >.

Details of the module

This module extracts the acoustic streams that use ID information and outputs them adding ID information to them. Note that this module converts `Matrix<complex<float> >`and `Matrix<float>`into `Map<int, ObjectRef>`with ID given in the input though it presently supports only 1 ch data as inputs.

6.7.10 SourceSelectorByDirection

Outline of the module

This module is a filtering module that filters only the results in the specified angle range for horizontal direction among the input source localization results.

Necessary file

No files are required.

Usage

When to use

This module is used obtain localization results only for the direction whose prior information is available (e.g. in the case that it is known the sound sources exist only in the front). Or, when a direction of noise source direction is already known, this module can subtract noise localization results by designating the directions other than that.

Usage

When to use

Source location results such as ConstantLocalization, LoadSourceLocationand LocalizeMUSICare mainly connected. Connection example in Figure 6.76 is a network to extract only the directions in the specified range from the log files of source localization results.

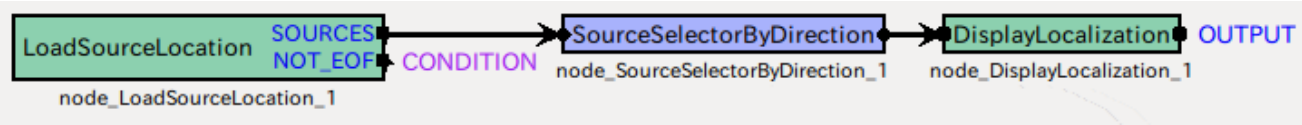


Figure 6.76: Connection example of SourceSelectorByDirection

Input-output and property of the module

Input

SOURCES : `Vector<ObjectRef>`type. Source location results as inputs are connected. `ObjectRef`refers to the `Source`type data.

Output

OUTPUT : `Vector<ObjectRef>`type. Source location results after filtered. `ObjectRef`refers to the `Source`type data.

Parameter

Table 6.48: Parameter list of SourceSelectorByDirection

Parameter name	Type	Default value	Unit	Description
MINAZIMUTH	`float`	-20.0	[deg]	Minimum value of a sound source direction to be passed
MAXAZIMUTH	`float`	20.0	[deg]	[deg] Maximum value of a sound source direction to be passed

MINAZIMUTH，MAXAZIMUTH : `float`type. The angles indicate right and left directions (azimuth angle) of the sound source to be passed.

Details of the module

This module does not perform spatial filtering by microphone array signal processing such as the beamformer though it only performs filtering based on the information on sound source directions of localization results.

6.7.11 SourceSelectorByID

Outline of the module

This module is used to output only the sound source separation results with the IDs over the designated value among plural sound source separation results. In particular, when setting the FIXEDNOISE property of the GHDSSmodule to `true`, negative IDs are given to stationary noise separation results. This module is used as a filter to process the sounds other than those.

Necessary file

No files are required.

Usage

When to use

The sound source separation module GHDSSswitches on the robot though it does not move it. Concretely, when performing the sound source separation under the condition that the noise (e.g. sound of fan) is stationary and known, an ID of the separation result of the noise is output as -1. In such a case, when connecting SourceSelectorByIDafter the GHDSSmodule and setting the threshold value to 0, the separated sound of the stationary noise can be ignored in subsequent processing.

When to use

Figure 6.77 shows connection example. As shown in the figure, This module is connected to the posterior half of GHDSStypically.

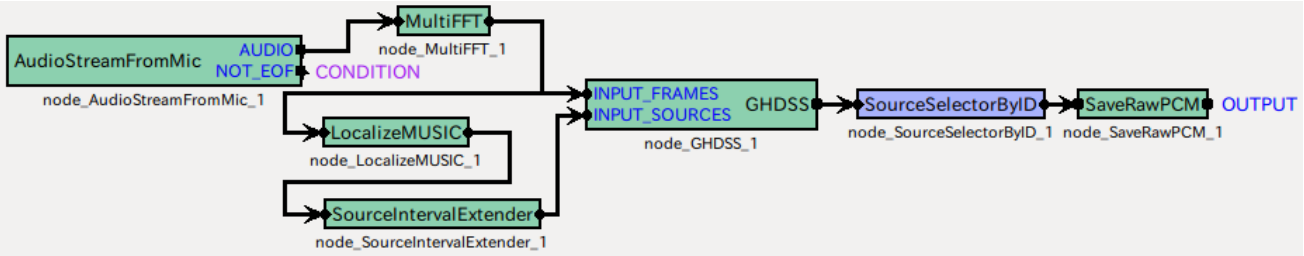


Figure 6.77: Connection example of SourceSelectorByID

Input-output and property of the module

Input

INPUT : `Map<int, ObjectRef>`type. Since this module is usually connected after the sound source separation module, sound source IDs correspond to `int`, which is the key of `Map`. `ObjectRef`is `Vector<float>`type(power spectrum) or `Vector<complex<float> >`type (complex spectra), which indicate separation .

Output

OUTPUT : `Map<int, ObjectRef>`type. The data for which only the data whose sound source IDs are greater than MINID are extracted are output. The content of `Map`is same as that of INPUT.

Parameter

MINID : `int`type. Separated sounds with sound source IDs greater than this parameter value are passed. The default value is 0. When this module is connected after GHDSS, no changes are required to the default value.

Table 6.49: Parameter list of SourceSelectorByID

Parameter name	Type	Default value	Unit	Description
MINID	int	0		Sound sources with IDs greater than this value are passed.

6.7.12 Synthesize

Outline of the module

This module converts signals of the frequency domain into waveforms of the time domain.

Necessary file

No files are required.

Usage

This module is used to convert signals of the frequency domain into waveforms of the time domain.

When to use

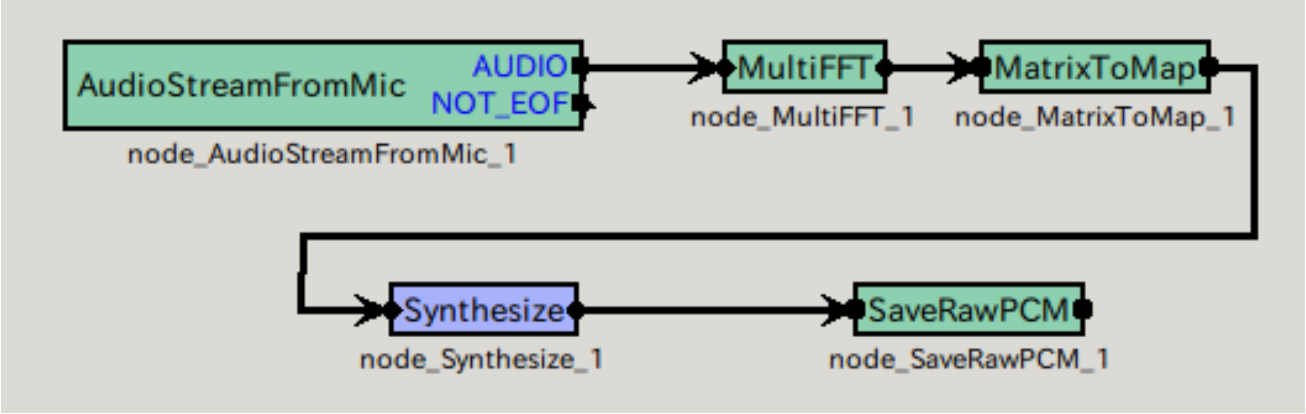


Figure 6.78: Connection example of Synthesize

Input-output and property of the module

Table 6.50: Parameter list of Synthesize

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	FFT length
ADVANCE	int	160	[pt]	Shift length
SAMPLINGRATE	int	16000	[Hz]	Sampling rate
MINFREQUENCY	int	125	[Hz]	Minimum frequency
MAXFREQUENCY	int	7900	[Hz]	Maximum frequency
WINDOW	string	HAMMING		Window function
OUTPUTGAIN	float	1.0		Output gain

Input

INPUT : `Map<int, ObjectRef>`type. `ObjectRef`is `Vector<complex<float> >`.

Output

OUTPUT : `Map<int, ObjectRef>`type. `ObjectRef`is `Vector<float>`.

Parameter

LENGTH FFT length. Its value must be equal to the other module (MultiFFT).

ADVANCE Shift length. Its value must be equal to the other module (MultiFFT).

SAMPLINGRATE Sampling rate. Its value must be equal to the other modules.

MINFREQUENCY The minimum frequency used for waveform generation

MAXFREQUENCY The maximum frequency used for waveform generation

WINDOW Window function. Select HAMMING, RECTANGLE or CONJ

OUTPUTGAIN Output gain

Details of the module

The reverse FFT is performed for the input signals of the frequency domain for four low bands and is performed by substituting 0 for frequency bins over $\omega_s/2 - 100$ [Hz]. Next, a designated window is adopted to perform overlap-add processing. The overlap-add processing is a method to reduce the influence of a window by performing the inverse transformation for every frame to add the signals for which the time domains are shifted back while shifting them. For the details, see the web pages in References. Finally, the temporal waveforms are multiplied by the output gain and output. Further, it is necessary to read ahead the frames to perform the overlap-add processing and therefore this module brings delay to the entire processor as a result. The length of delay is calculated as follows.

$$delay = \begin{cases} |\text{LENGTH}/\text{ADVANCE}| - 1, & \text{if LENGTH } mod \text{ ADVANCE} \neq 0, \\ \text{LENGTH}/\text{ADVANCE}, & \text{otherwise.} \end{cases} \tag{6.128}$$

Since the default values are LENGTH = 512 and ADVANCE = 160 for HARK, the delay is 3 [frame], which means that the delay for the entire system is 30 [ms].

References

(1) http://en.wikipedia.org/wiki/Overlap-add

6.7.13 WhiteNoiseAdder

Outline of the module

This module adds white noise to input signals.

Necessary file

No files are required.

Usage

This module is used to add noise daringly so as to reduce the influence of non-linear distortions after separation. For example, since PostFilterperforms nonlinear processing, it is difficult to avoid the occurrence of musical noise. Such noise greatly influences the speech recognition performance in some cases. It is known that the influence of such noise can be reduced by adding appropriate amount of the known noise. When to use An example is shown in the figure with

concrete module names.

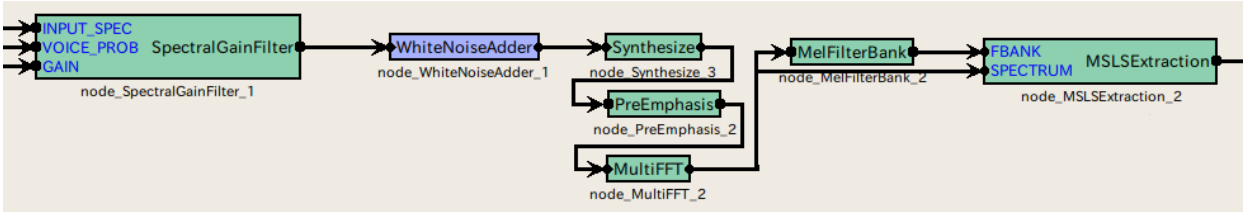


Figure 6.79: Connection example of WhiteNoiseAdder

Input-output and property of the module

Table 6.51: Parameter list of WhiteNoiseAdder

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	FFT length
WNLEVEL	float	0		Additional noise level

Input

INPUT : Map<int, ObjectRef>type. Since ObjectRefis Vector<complex<float> >, it is a premise to input signals of the frequency domain.

Output

OUTPUT : Map<int, ObjectRef>type. ObjectRefis Vector<complex<float> >. The signals to which white noise is added are output.

Parameter

LENGTH : FFT length. Its value must be equal to the other modules.

WNLEVEL : Additional noise level. Designate the maximum amplitude value in the time domain.

Details of the module

The following is added to each frequency bin of input signals.

$$\frac{\sqrt{\text{LENGTH}}}{2} \cdot \text{WNLEVEL} \cdot e^{2\pi jR} \tag{6.129}$$

R indicates a random number of $0 \leq R \leq 1$ (different value for each frequency bin). $\sqrt{\text{LENGTH}}/2$ is the term to revise distortions of scaling between the time domain and frequency domain that occur a frequency analysis by FFT.

6.8 Modules independent of FlowDesigner

6.8.1 JuliusMFT

Outline

JuliusMFTis the speech recognition module obtained by remodeling the large glossary speech recognition system Juliusfor HARK. It had been provided for HARK 0.1.x systems as a patch for the multiband edition Julius[1] improved based on the large glossary speech recognition system Julius 3.5. However, for HARK 1.0, we reviewed its implementation and functions with the 4.1 Julius system as a base. Compared with the original Julius, modifications have been made to JuliusMFTof HARK 1.0 for accepting the following four points.

- Introduction of the Missing Feature theory
- Acceptance of network inputs (mfcnet) of MSLS features
- Acceptance of addition of sound source information (SrcInfo)
- Acceptance of simultaneous utterance (exclusion)

Implementation was achieved with plug-in feature introduced from Julius4.0 with the minimum modification to the main body of Julius. This section describes difference from Julius and connection with the HARK modules in FlowDesigner as well as the installation method and usage of it.

Start up and setting

Execution of JuliusMFTis performed as follows when assuming the setting file name as julius.conf for example.

```
> julius -C julius.jconf
```

In HARK, after starting JuliusMFT, the socket connection with JuliusMFTis performed by starting a network that contains SpeechRecognitionClient(or SpeechRecognitionSMNClient) for which an IP address and a port number are correctly set to enable the speech recognition. The abovementioned julius.jconf is a text file that describes setting of JuliusMFT. The content of the setting file consists of from an argument options that begin with "-" basically and therefore the user can designate arguments directly as an option of Julius when starting. Moreover, descriptions that come after # are treated as comments. The options used for Julius are summarized in http://julius.sourceforge.jp/juliusbook/ja/descoption.html and the users are recommended to refer to the website. The minimum required setting is the following seven items.

- -notypecheck
- -plugindir /usr/lib/juliusplugin
- -input mfcnet
- -gprune addmasktosafe
- -gram grammar
- -h hmmdefs
- -hlist allTriphones

- **-notypecheck** Setting to skip type checks for feature parameters. It is an option that can be designated arbitrarily in the original Juliusthough it is an option that must be designated in JuliusMFT. Type check is performed unless this option is designated. However, mask data, as well as features, are calculated in plug-ins of JuliusMFT(1.0 is output even in the case of without masks). Therefore, it is judged that the sizes do not match in the type check and recognition is not performed.
- **-plugindir Plug in directory name** Designate a directory where plug ins (*.jpi) exist. Designate an absolute path from a current directory or a complete path of the plug in as an argument. The default value of this path is /usr/lib/juliusplugin when apt-get is installed and /usr/local/lib/juliusplugin when the source code is compiled and installed without designating the path. Further, it is necessary to designate this path before designating the functions that are realized in plug ins such as -input mfcnet or -gprune addmasktosafe. Note that all extensive plug in files in this path are read entirely in execution.

[1] http://www.furui.cs.titech.ac.jp/mbandjulius/

- **-input mfcnet** -input itself is an option implemented in the original Juliusand microphones, files and inputs through a network are supported. In JuliusMFT, this option is extended so that acoustic features transmitted by SpeechRecognitionClient(or SpeechRecognitionSMNClient) and masks can be received through a network and the user can designate mfcnet as an audio input source. This function is validated by designating -input mfcnet. Moreover, the port numbers when designating mfcnet are used to designate port numbers for the acoustic input source adinnet in the original Julius. The user can designate like "-adport port number" with adport.
- **-gprune** Designate a pruning algorithm used when masks are used for existing output probability calculation. They basically are the algorithms that are transplanted from the functions equipped with juliusmft(ver3.5) that was provided in HARK 0.1.x. The user selects an algorithm from *add_mask_to_safe*, *add_mask_to_heu*, *add_mask_to_beam*, *add_mask_to_none* (when it is not designated, the default calculation method is adopted). They correspond to *safeheuristicbeamnone* in the original Julius, respectively. Further, since the calculation method with eachgconst of juliusmft(ver3.5) is not precise strictly, errors occurred in calculation results (score) different from the original. This time, the calculation method same as that of the original is adopted to solve this error problem.
- **-gram grammar** Designate a language model. Same as the original Julius.
- **-h hmmdefs** Designate acoustic models (HMM). Same as the original Julius.
- **-hlist allTriphones** Designate a HMMList file. Same as the original Julius.

Further, when using the above in the module modes described later, it is necessary to designate the -module option same as the original Julius.

Detail description

mfcnet communication specification

To use mfcnet as an acoustic input source, designate "-input mfcnet" as an argument when starting up JuliusMFTas mentioned above. In such a case, JuliusMFTacts as a TCP/IP communications server and receives features. Moreover, SpeechRecognitionClientand SpeechRecognitionSMNClient, which are modules of HARK, work as a client to transmit acoustic features and Missing Feature Mask to JuliusMFT. The client connects to JuliusMFTfor every utterance and cuts off the connection after transmitting completion promptly. The data to be transmitted must be little endian (Note that it is not a network byte order). Concretely, communication is performed as follows for one utterance.

1. Socket connection The client opens the socket and connects to JuliusMFT.
2. **Communication initialization (data transmitted once at the beginning)** The client transmits information on the sound source that is going to be transmitted shown in Table 6.52 only once just after the socket connection. The sound source information is expressed in a SourceInfo structure (Table 6.53) and has a sound source ID, sound source direction and time of starting transmitting. The time is indicated in a timeval structure defined in ¡sys/time.h¿ and is elapsed time from the starting time point (January 1, 1970 00:00:00) in the time zone of the system. The time indicates the elapsed time from the time starting point thereafter.

Table 6.52: Data to be transmitted only once at the beginning

Size [byte]	type	Data to be transmitted
4	`int`	28 (= sizeof(SourceInfo)))
28	SourceInfo	Sound source information on features that is going to be transmitted

Table 6.53: SourceInfo structure

Member variable name	Type	Description
sourceid	`int`	Sound source ID
azimuth	`float`	Horizontal direction [deg]
elevation	`float`	Vertical direction [deg]
time	timeval	Time (standardized to 64 bit processor and the size is 16 bytes)

3. **Data transmission (every frame)** Acoustic features and Missing Feature Mask are transmitted. Features of one utterance are transmitted repeatedly till the end of the speech section with the data shown in Table 6.54 as one frame. It is assumed inside the JuliusMFTthat the dimension number of feature vectors and mask vectors are same.

Table 6.54: Data to be transmitted for every frame

Size [byte]	Type	Data to be transmitted
4	`int`	N1=(dimension number of feature vector) × sizeof(`float`)
N1	`float`[N1]	Array of feature vector
4	`int`	N2= (dimension number of mask vector) × sizeof(`float`)
N2	`float`[N2]	Array of mask vector

4. **Completing process** Finishing transmitting features for one sound source, data (table 6.55) that indicate completion are transmitted and the socket is closed.

Table 6.55: Data to indicate completion

Size [byte]	Type	Data to be transmitted
4	`int`	0

Module mode communication specification When designating -module, JuliusMFT operates in the module mode same as the original Julius. In the module mode, JuliusMFTfunctions as a server of TCP/IP communication and provides clients such as jcontrol with statuses and recognition results of JuliusMFT. Moreover, its operation can be changed by transmitting a command. EUC-JP is usually used as a character code of a Japanese character string and can be changed with the argument. An XML-like format is used for data representation and as a mark to indicate completion of data. "." (period) is transmitted for every one message. Table **??** shows an example of the outputs of the module mode. Meaning of representative tags transmitted in JuliusMFTis as follows.

- INPUT tag
  This tag indicates information related to inputs and there are STATUS and TIME as attributes. The values of STATUS are LISTEN, STARTREC or ENDREC. LISTEN indicates that Juliusis ready to receive speech. STARTREC indicates that reception of features is started. ENDREC indicates that the last feature of the sound source being received is received. TIME indicates the time at that time.

- SOURCEINFO tag
  This tag indicates information related to tag sound sources and is an original tag of JuliusMFT. There are ID, AZIMUTH, ELEVATION, SEC and USEC as attributes. The SOURCEINFO tag is transmitted when starting the second path. Its ID indicates a sound source ID (not speaker IDs but numbers uniformly given to each sound source) given in HARK. AZIMUTH and ELEVATION indicate horizontal and vertical direction (degrees) seen from a microphone array coordinate system for the first frame of the sound source, respectively. SEC and USEC indicate time of the first frame of the sound source. SEC indicates seconds and USEC indicates microsec digit.

- RECOGOUT tag
  This tag indicates recognition results, and subelement is a gradual output, the first path output or the second path output. In the case of the gradual output, this tag has the PHYPO tag as a subelement. In the case of the first path output and the second path output, this tag has the SHYPO tag as a subelement. In the case of the first path, the result that becomes the maximum score is the output, and in the case of the second path, candidates for the number specified for the parameter are the output and therefore SHYPO tags for the number of candidates are the output.

- PHYPO tag
  This tag indicates gradual candidates and columns of the candidate word WHYPO tag are included as a subelement. There are PASS, SCORE, FRAME and TIME as attributes. PASS indicates the order of the path and always is 1. SCORE indicates conventionally accumulated scores of this candidate. FRAME indicates the number of frames that have been processed to output this candidate. TIME indicates time (sec) at that time.

- SHYPO tag
  This tag indicates sentence assumptions and columns of the candidate word WHYPO tag are included as a subelement. There are PASS, RANK, SCORE, AMSCORE and LMSCORE as attributes. PASS indicates the order of the path

and always is 1 when an attribute PASS exists. RANK indicates a rank order of an assumption and exists only in the case of the second path. SCORE indicates a logarithmic likelihood of this assumption, AMSCORE indicates a logarithmic acoustic likelihood and LMSCORE indicates a logarithmic language probability.

- WHYPO tag
  This tag indicates word assumptions and WORD, CLASSID and PHONE are included as attributes. WORD indicates notations, CLASSID indicates word names that become a key to a statistics language model, PHONE indicates phoneme lines and CM indicates word reliability. Word reliability is included only in the results of the second path.

- SYSINFO tag
  This tag indicates statuses of the system and there is PROCESS as an attribute. When PROCESS is EXIT, it indicates normal termination. When PROCESS is ERREXIT, it indicates abnormal termination. When PROCESS is ACTIVE, it indicates the status that speech recognition can be performed. When PROCESS is SLEEP, it indicates the status that speech recognition is halted. It is determined whether or not to output these tags and attributes by the argument designated when starting Julius MFT. The SOURCEINFO tag is always output and the others are same as those of the original Julius and therefore users are recommended to refer to Argument Help of the original Julius.

When comparing with the original Julius, the changes made to JuliusMFTare two points as follows.

- Addition of items related to the SOURCEINFO tag, which is a tag for the information on source localization above and embedding of sound source ID(SOURCEID) to the following tags related. STARTRECOG, ENDRECOG, INPUTPARAM, GMM, RECOGOUT, REJECTED, RECOGFAIL, GRAPHOUT, SOURCEINFO

- To improve the processing delay caused by the exclusion control at the time of simultaneous utterance, changes were made to the format of the module mode. Concretely, exclusion control has been performed for each utterance conventionally. An output is divided into multiple times so that the exclusion control can be performed for each unit and therefore modifications were made to the output of the following tags, which need to be output at a time. ¡¡Tags separated to start-tag / end-tag ¿¿

  – <RECOGOUT>... </RECOGOUT>
  – <GRAPHOUT>... </GRAPHOUT>
  – <GRAMINFO>... </GRAMINFO>
  – <RECOGPROCESS>... </RECOGPROCESS>

  ¡¡One-line tags that are output devided into multiple times inside¿¿

  – <RECOGFAIL .../>
  – <REJECTED .../>
  – <SR .../>

Example output of JuliusMFT

1. Example output of standard output mode

```
Stat:
server-client:
connect from 127.0.0.1
forked process [6212] handles this request
waiting connection...
source_id = 0, azimuth = 5.000000, elevation = 16.700001, sec = 1268718777, usec = 474575
###
Recognition:
1st pass (LR beam)
.....................................................................................................
read_count < 0, read_count=-1, veclen=54
```

```
pass1_best:
<s> Please order </s> pass1_best_wordseq:
0 2 1
pass1_best_phonemeseq:
silB |
ch u:
m o N o n e g a i sh i m a s u |
silE
pass1_best_score:
403.611420
###
Recognition:
2nd pass (RL heuristic best-first)
STAT:
00 _default:
19 generated, 19 pushed, 4 nodes popped in 202
transmittedence1:
<s> Please order </s> wseq1:
0 2 1
phseq1:
silB |
ch u:
m o N o n e g a i sh i m a s u |
silE
cmscore1:
1.000 1.000 1.000
score1:
403.611786
connection end
ERROR:
an error occurred while recognition, terminate stream
<- This error log is part of the specification
```

2. Output sample of module mode Output to clients (e.g. jcontrol)in following XML-like format. " > " on the line head is output by jcontrol when using jcontrol (not included in output information).

```
> <STARTPROC/>
> <STARTRECOG SOURCEID="0"/>
> <ENDRECOG SOURCEID="0"/>
> <INPUTPARAM SOURCEID="0" FRAMES="202" MSEC="2020"/>
> <SOURCEINFO SOURCEID="0" AZIMUTH="5.000000" ELEVATION="16.700001" SEC="1268718638" USEC="10929"/>
> <RECOGOUT SOURCEID="0">
> <SHYPO RANK="1" SCORE="403.611786" GRAM="0">
>
<WHYPO WORD="<s>" CLASSID="0" PHONE="silB" CM="1.000"/>
>
<WHYPO WORD="Please order "CLASSID="2" PHONE="ch u:
m o N o n e g a i sh i m a s u" CM="1.000"/>
>
<WHYPO WORD="</s>" CLASSID="1" PHONE="silE" CM="1.000"/>
> </SHYPO>
> </RECOGOUT>
```

Notice

- Restraint of the -outcode option
  Since the tag outputs are implemented with plug-in functions, modifications were made so that the -outcode option for which the user can designate an output information type can be realized with plug-in functions. An error occurs when designating the -outcode option with the status that the plug ins are not read.

- Error message in utterance completion in the standard output mode The error output in the standard output mode "ERROR: an error occurred while recognition, terminate stream" (see the example output) is output because the error code is returned to the main body of julius forcibly when finishing the child process generated in the feature input plug in created (mfcnet). As a specification, measures are not taken for this error so as to avoid modifications to the main body of Julius as much as possible. Further, in the module mode, this error is not output.

Installation method

- Method using apt-get
  If setting of apt-get is ready, installation is completed as follows. Furthermore, since the original Juliusis made packaged in Ubuntu, in the case that the original Juliusis installed, execute the following after deleting this.

  ```
  > apt-get install julius-4.1.4-hark julius-4.1.3-hark-plugin
  ```

- Method to install from source

  1. Download julius-4.1.4-hark and julius4.1.3plugin and expand them in an appropriate directory.
  2. Move to the julius-4.1.4-hark directory and execute the following command. Since it is installed in /usr/local/bin with the default, designate - prefix as follows to install in /usr/bin same as package.

     ```
     ./configure --prefix=/usr --enable-mfcnet;
     make;
     sudo make install
     ```

  3. If the following indication is output after the execution, installation of Juliusis completed normally.

     ```
     > /usr/bin/julius
     Julius rev.4.1.4 - based on JuliusLib? rev.4.1.4 (fast) built for
     i686-pc-linux
     Copyright (c)
     1991-2009 Kawahara Lab., Kyoto University Copyright
     (c)
     1997-2000 Information-technology Promotion Agency, Japan Copyright
     (c)
     2000-2005 Shikano Lab., Nara Institute of Science and Technology
     Copyright (c)
     2005-2009 Julius project team, Nagoya Institute of
     Technology
     Try '-setting' for built-in engine configuration.
     Try '-help' for run time options.
     >
     ```

  4. Install plug ins next. Move to the julius4.1.3plugin directory and execute the following command.

     ```
     > export JULIUSSOURCEDIR=../julius4.1.4-hark;
     make;
     sudo make install
     ```

     Designate the path of the source of julius4.1.4-hark in JULIUS_SOURCE_DIR. Here, the example shows the case of developing the sources of Juliusand plug-ins to the same directory. Now the installation is completed.

  5. Confirm if there are plug in files under /usr/lib/juliusplugin properly.

```
> ls /usr/lib/juliusplugin
calcmixbeam.jpi calcmixnone.jpi mfcnet.jpi calcmixheu.jpi calcmixsafe.jpi
>
```

If five plug in files are indicated as above, installation is completed normally.

Chapter 7

Support Tools

7.1 harktool

7.1.1 Outline

harktool is a tool to generate transfer functions of GHDSSand LocalizeMUSICand to visualize them. The following two files are required for generation of transfer functions.

1. Microphone layout (see 5.2)
2. Inpulse response file (5.3.1)

Transfer functions are generated by designating these two types of files from the window. LocalizeMUSICcan be generated in the same way in simulations with the tool mkmusictf provided. In this case, transfer functions are generated only with the file described in 1, in which microphone layouts are described. However, since transfer functions are generated on the premise that the microphones are arranged in free space, echo (e.g. reflection of the robot's the head) of the object on which microphones are actually set is ignored. Shape can be confirmed with gnuplot after generating transfer functions.

7.1.2 Installation method

Same as HARK, it is shown in the web page http://winnie.kuis.kyoto-u.ac.jp/HARK. For compilation, perform "make" in the directory where the archive is decompressed.

7.1.3 Usage

Presently, this tool performs generation and visualization of transfer functions for GHDSSand LocalizeMUSIC. Select a name of the node with the tab on the top in the window. See the attached manual for details.

Chapter 8

HARK-compatible Multi-Channel AD Devices and Their Installation

The user becomes able to perform the microphone array processing by connecting multichannel A/D devices (hereinafter called A/D device) to HARK. The users who need to perform the microphone array processing set required A/D devices referring to this chapter. An A/D device to support in HARK is three following model.

1. System In Fronteir, Inc. the RASP series.
2. ALSA-based devices (e.g. the RME Hammerfall DSP Multiface series),
3. Tokyo Electron device TD-BD-16ADUSB.

The installation and setting methods for each device are described here. However, only the 3rd party version HARK corresponds to TD-BD-16ADUSB.

8.1 System In Fronteir, Inc. the RASP series, Radio RASP

Among the multichannel A/D devices that the HARK development team checks the operation of, this chapter describes the setting procedure to use the radio RASP, which is one of the RASP series of System In Fronteir, Inc. (hereinafter called radio RASP) in HARK. Figure 8.1 shows overview of the radio RASP in. The radio RASP is the A/D, D/A conversion device that solely performs 16ch AD conversion and 2ch D/A conversion. It transmits and receives commands via TCP/IP and transmits and receives setting and data of A/D, D/A conversion processing from a host PC. Its usability is good as the filter of sampling frequency and analog input parts can be changed easily by software. The installation method for the radio RASP and the method to record the speech from the microphones connected to the radio RASP on HARK are introduced here. Description and operation check are performed in Ubuntu 9.10.

- Connection to a PC of the radio RASP (connection of hardware)
- Installation and setting of software for the radio RASP
- Recording test on HARK with the radio RASP (use from HARK)

8.1.1 Connection to a PC of the radio RASP

The radio RASP consists of a sole box. Since a host PC is connected with a wireless LAN, a router corresponding to a wireless LAN is required. For the radio RASP, it is necessary to set an IP address beforehand and register an SSID of a radio router. IP address set at the time of shipping can be used though it is essential to set an SSID. Complete the setting referring to the attached manual.

Figure 8.1: System In Fronteir, Inc. Wireless RASP

8.1.2 Installation and setting of software for the radio RASP

It is necessary to install a library and a sample program in a host PC to be operated so as to operate the radio RASP. Copy files from the bundled CD to the radio RASP and install them. Refer to the attached materials for details. It is necessary at least to prepare the environment where the setting of the analog filter of the radio RASP can be initialized for use of HARK. Installation of the command wsfpaaconfig is required concretely. wsfpaaconfig is generated by installing a library and compiling a sample program. The set value of wsfpaaconfig is deleted when turning off the radio RASP and therefore make sure to execute it right after turning it off.

8.1.3 Recording test in HARK with the radio RASP

Confirm the equipment prior to recording. Show use equipment in table 8.1. For a speech recording test with HARK,

Table 8.1: List of device to be used

Device name	description
Wireless RASP	A/D, D/A conversion device
Router for wireless LAN	For connection between RASP and host PC
PC	PC to operate the radio

use the network file test. n shown in Figure 8.2. The connection route is from MatrixToMapto SaveRawPCM, from AudioStreamFromMicvia ChannelSelector. Table 8.2 shows setting of each property.

```
> ./test.n
```

By setting the above, the speech of the analog input 1 of the radio RASP is recorded for three seconds. Speech of PCM 16-bit quantization and 16kHz sampling is saved in the little endian RAW format.

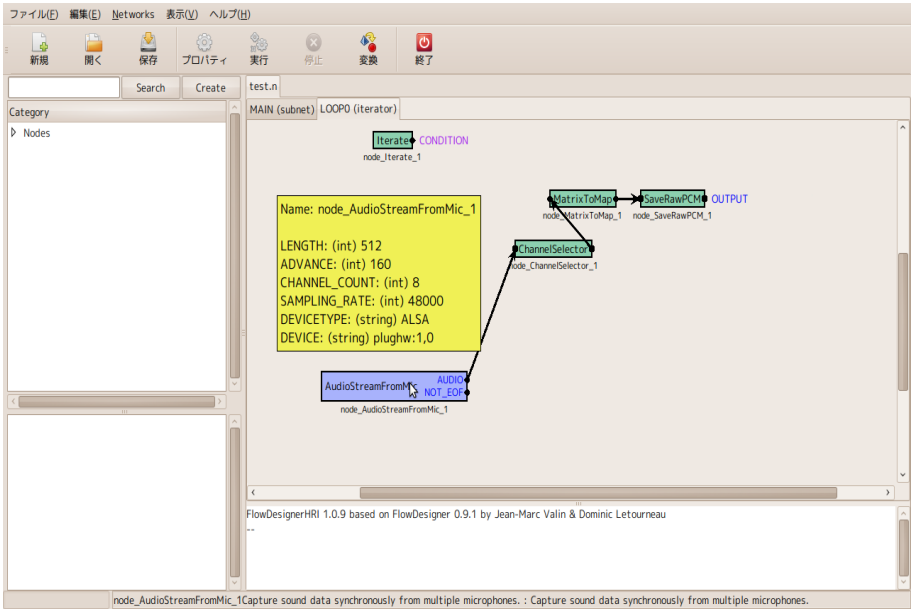


Figure 8.2: Network (test.n)

Table 8.2: Property of speech recording network

Module name	Property name	Type of value	Set value
AudioStreamFromMic	LENGTH	int	512
	ADVANCE	int	160
	CHANNELCOUNT	int	16
	SAMPLINGRATE	int	16000
	DEVICETYPE	string	WS
	DEVICE	string	192.168.0.1
ChannelSelector	SELECTOR	Object	<Vector<int> 0 >
SaveRawPCM	BASENAME	string	sep
	ADVANCE	int	160
	BITS	int	16
Iterate	MAXITER	int	300

8.2 RME Hammerfall DSP series Multiface AE

Among the multichannel A/D devices that the HARK development team checks the operation of, this chapter describes the setting procedure to use the Hammerfall DSP Multiface series of RME (the Hammerfall DSP Multiface series is hereinafter called Multiface) in HARK. Multiface is the A/D, D/A conversion device that solely performs 10ch AD conversion and 10ch D/A conversion. Among the 10ch, analog parts are 8ch and SPDIF parts are 2ch. Multiface is the ALSA compliant device, which is a standard interface in Linux. This chapter introduces the setting method as a representative of ALSA compliant devices. If OS is enabled to recognize other ALSA compliant devices through an ALSA driver, multichannel recording can be performed in HARK same as Multiface. Although ALSA compliant devices except Multiface are out of our support scope, this chapter will be helpful when using A/D, D/A devices that accept ALSA out of out support. The installation method for Multiface and the method to record the speech from the microphones connected to Multiface on HARK are introduced here. Description and operation check are performed in Ubuntu 9.10.

8.2.1 Connection of Multiface to PC

The 24bit/96kHz multiple-channel A/D device made by RME consists of the following three devices.

- **RME HDSP I/O box**: RME HDSP Multiface Avena unit,
- **RME HDSP interface**: Either of the RME HDSP PCI card or RME HDSP CardBus card
  Operation of the RME HDSP PCI-Express card and RME HDSP Express card has not been confirmed and therefore they are out of our support scope.
- **Preamplifier for microphone**: RME OctaMic-II RME OctaMic and Yamaha HA -8 have ever been used though these are discontinued models.

However, a preamplifier for microphones is not necessarily needed. If a good recording level is secured, it is not necessary to connect the device to a preamplifier. Connect these hardware to a PC referring to the attached manual.

Installation and setting of software for Multiface

To enable Linux to recognize Multiface as a multichannel A/D device, it is necessary to install a general-purpose driver for ALSA devices and firmware and software for Multiface and set parameters. For the following installation, users are strongly recommended to install them from the package. Although compiling and installing from the source can be one of the choices, such a method may cause a conflict with other software, which may lead to the failure of recording/replaying in actual operations even if compiling and installing seem to be completed successfully. Note that a conflict with pulseAudio often occurs in particular. The detail description strongly depends on individual system environments and they are omitted here. From the following package, this chapter describes the installation method of a general-purpose driver for ALSA devices first and then the installation work to enable the hdsp command, which is required for initialization of Multiface followed by setting by the hdsp command. Finally, an example of the recording network with HARK is described.

Installation of general-purpose driver for ALSA device

General-purpose drivers (alsa-lib, alsa-driver) for ALSA devices are often already installed. Input the following bold-faced parts to confirm they are installed. > indicates the command prompt. If the italic part message is displayed, alsa-lib and alsa-drive are already installed and therefore it is not necessary to install them. Since version numbers differ depending on usage environments, different version numbers may be displayed in the actual operation. In this test environment, the version 1.0.23 is used.

> **cat /proc/asound/version**
> *Advanced Linux Sound Architecture Drive Version 1.0.23.*

If they are not installed, they will be installed automatically by the following operation and therefore proceed to the next operation in the next chapter.

Installation to enable command required for initialization of Multiface

The following two packages are required.

- alsa-firmware-loaders
- alsa-tools-gui

For installation of these packages, use either of the items below.

- Synaptic package manager
- apt-get

Installing the above-mentioned package makes operational the following three commands required for setting Multiface.

- hdsploader (Package : alsa-firmware-loaders)
- hdspconf (Package : alsa-tools-gui)

- hdspmixer (Package : alsa-tools-gui)

An installation example with apt-get is shown below.

```
> sudo apt-get upadte
> sudo apt-get install alsa-firmware-loaders
> sudo apt-get install alsa-tools-gui
```

Finally, download the source of alsa-firmware from the website of alsa to acquire multifacefirmwarerev11.bin. (http://www.alsa-project.org/main/index.php/MainPage). The available latest version in August 9, 2010 is 1.0.23. Download alsa-firmware-1.0.23.tar.bz2. After the download, the file is saved in "location" "download" of the system menu. It will be easier to work if copying it to "location" "home". Copy the downloaded file alsa-firmware-1.0.23.tar.bz2 to an appropriate directory. Develop a file in bunzip2 and tar firstly. Select "application" "accessories" "terminal". If the downloaded file is copied to "location" "home", execute the following commands on the window opened. When the file is copied to another directory, move to the directory by the cd command and then start compilation. Concrete procedures are as follows.

```
> bunzip2 -d alsa-firmware-1.0.23.tar.bz2
> tar vfx alsa-firmware-1.0.23.tar
alsa-firmware-1.0.23/
alsa-firmware-1.0.23/multisound/
   Omitted
alsa-firmware-1.0.23/hdsploader/
alsa-firmware-1.0.23/hdsploader/digifacefirmwarerev11.dat
alsa-firmware-1.0.23/hdsploader/Makefile.in
alsa-firmware-1.0.23/hdsploader/tobin.c
alsa-firmware-1.0.23/hdsploader/digifacefirmware.dat
alsa-firmware-1.0.23/hdsploader/README
alsa-firmware-1.0.23/hdsploader/Makefile.am
alsa-firmware-1.0.23/hdsploader/multifacefirmwarerev11.dat
alsa-firmware-1.0.23/hdsploader/multifacefirmware.dat
   omitted
alsa-firmware-1.0.23/echoaudio/Makefile.am
alsa-firmware-1.0.23/echoaudio/fwwriter.c
> cd alsa-firmware-1.0.23
> sh configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
   omitted
config. status: creating aica/Makefile
config. status: executing depfiles commands
> make
> sudo mkdir -p /lib/firmware/hdsploader
> sudo cp hdsploader/multifacefirmwarerev11.dat /lib/firmware/hdsploader/
```

The hdsp command is installed by the above operation and setting and operation of Multiface are enabled.

Setting by hdsp command

The necessary three hdsp commands are hdsploader, hdspconf and hdspmixer. These commands explained sequentially.

- **hdsploader**
  hdsploader is command to upload the firmware program (multifacefirmwarerev11.dat) to initialize FPGA of multiface. An execution example is shown below. Although the error message (Hwdep ioctl error on card hw: 0 : Device or resource busy.) is displayed, it is to be ignored since it is not a problem here. This error message is indicated when executing more than twice on the same system. This error message is always indicated in this operation in the case of a

system that is executed during the start-up of OS.

```
# hdsploader
hdsploader - firmware loader for RME Hammerfall DSP cards
Looking for HDSP + Multiface or Digiface cards :
Card 0: RME Hammerfall DSP + Digiface at 0xf9df0000, irq 16
Upload firmware for card hw:0
Hwdep ioctl error on card hw:0:Device or resource busy.
Card 1: HDA Intel at 0xfdffc000 irq 30
```

- **hdspconf**
  A setting window of Multiface opens when executing hdspconf. Set sampling frequency in this window. For other items, refer to the documents of Multiface. Figure 8.3 shows a window example for setting. The settable sampling rates are 32kHz, 44.1kHz, 48kH, 64kHz, 88.2KkHz and 96kHz. In this example, 32kHz is selected.

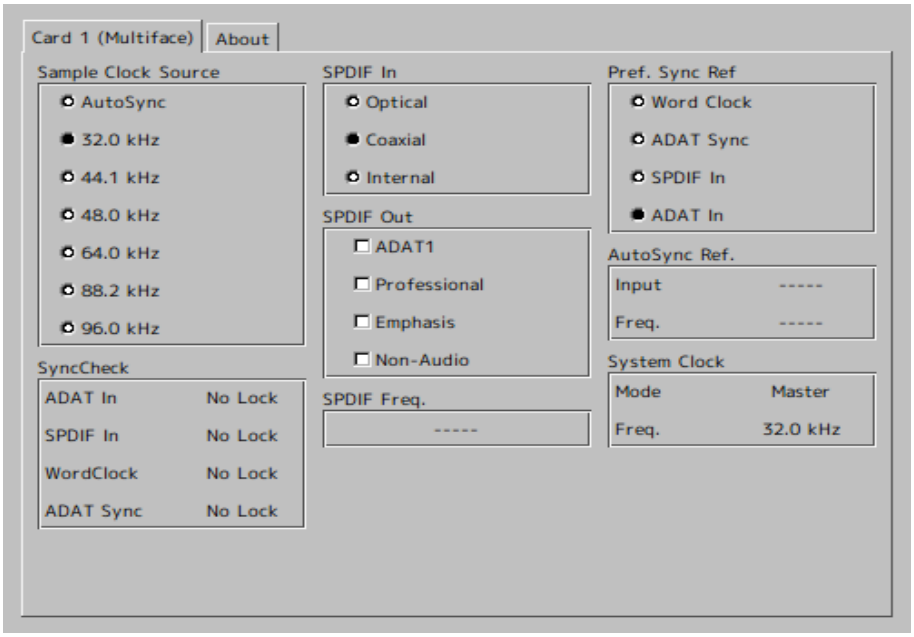


Figure 8.3: Parameter setting window

- **hdspmixer**
  GUI of the mixer is displayed when executing hdspmixer. Control and confirmation of input and output levels can be performed. Figure 8.4 shows an example of the mixer.

8.2.2 Sound recording test with Multiface in HARK

Prior to sound recording, confirm devices. Table 8.3 shows devices to be used. For the sound recording test for HARK,

Table 8.3: List of device to be used

Device name	Description
Hammerfall DSP Multiface	A/D, D/A conversion device
Digital Audio CardBus Interface	Card to add Multiface connection ports to PCMCIA slot
Octa Mic	Microphone preamplifier

change only the property of each module with the network file test.n same as that shown in Figure 8.2. Table 8.4 shows setting of each property.

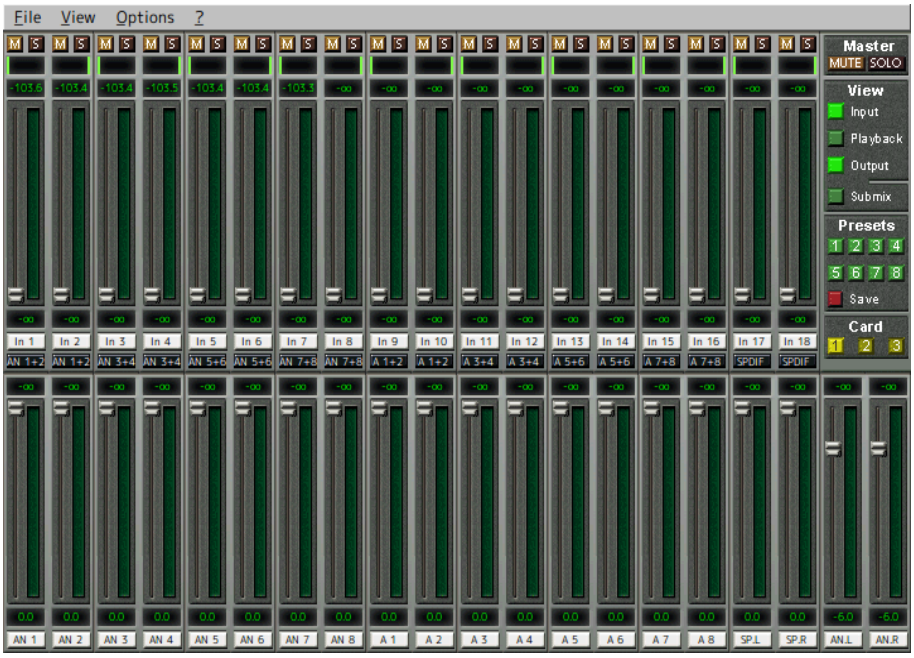


Figure 8.4: Parameter setting window

Figure 8.5: RME Hammerfall DSP Multiface (front)

Figure 8.6: RME Hammerfall DSP Multiface (Rear)

> **./test.n**

Entering the above, speech of the analog input 1 of Multiface is recorded for three seconds. Speech of PCM 16-bit quantization and 32kHz

Figure 8.7: PCMCIA CardBus Interface for Hammerfall DSP System

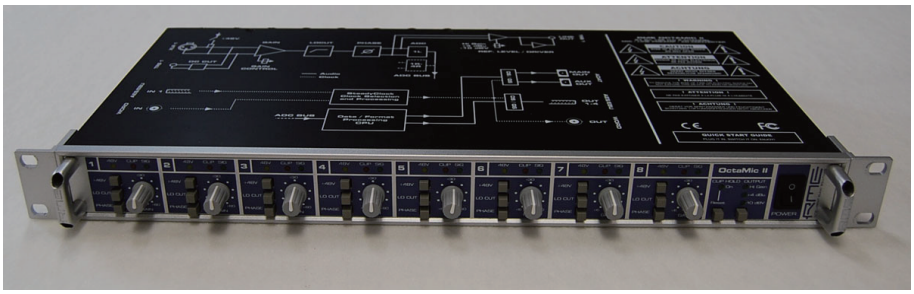


Figure 8.8: RME OctaMic (Front)

Figure 8.9: RME OctaMic (Rear)

sampling are saved in the little endian RAW format.

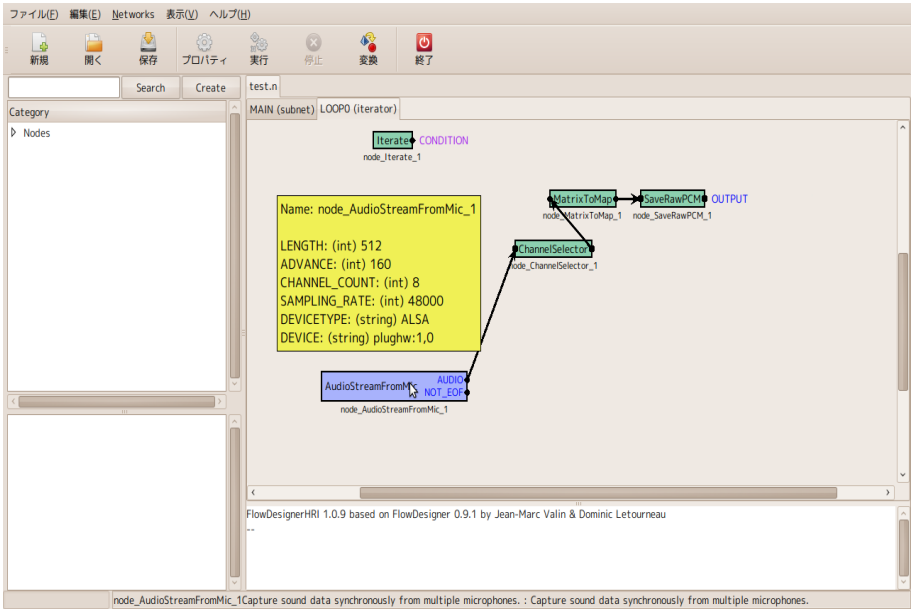


Figure 8.10: Network (test.n)

Table 8.4: Property of sound recording network

Module name	Property name	Type of value	Set value
AudioStreamFromMic	LENGTH	int	1024
	ADVANCE	int	320
	CHANNELCOUNT	int	8
	SAMPLINGRATE	int	32000
	DEVICETYPE	string	ALSA
	DEVICE	string	plughw:1,0
ChannelSelector	SELECTOR	Object	<Vector<int> 0 >
SaveRawPCM	BASENAME	string	sep
	ADVANCE	int	320
	BITS	int	16
Iterate	MAXITER	int	300

8.3 Tokyo Electron Device TD-BD-16ADUSB

Among the multichannel A/D devices that the HARK development team checks the operation of, this chapter describes the setting procedure to use TD-BD-16ADUSB, which is one of the Inrevium series of Tokyo Electron Device (hereinafter Tokyo Electron Device TD-BD-16ADUSB is called 16ADUSB) in HARK. Further, to use TD-BD-16ADUSB in HARK, it is necessary to use AudioStreamFromMic2, which is included only in the 3rd party version. Figure **??** shows an overview of 16ADUSB. 16ADUSB is the A/D, D/A conversion device that solely performs 16ch AD conversion and 4ch D/A conversion. The driver software of 16ADUSB is attached to the product. The kernel mode driver for Linux is attached and is convenient. For connection with microphones, it corresponds to plug in power supply and therefore condenser microphones for plug in power supply can be connected without modification.

The installation method of 16ADUSB and the sound recording method from the microphones connected to 16ADUSB on HARK are described. Description and operation check are performed in Ubuntu 9.10.

8.3.1 Connection to PC of 16ADUSB

16ADUSB is implemented on a business card size base. Since it is connected to a host PC through USB, it is easy to connect. Connect hardware to a PC referring to the attached manuals of them.

8.3.2 Installation and setting of software for 16ADUSB

It is necessary to install a kernel mode driver in a host PC to operate 16ADUSB.

8.3.3 Sound recording test in HARK with TD-BD-16ADUSB

Prior to sound recording, confirm devices to be used. Table 8.5 shows devices to be used. For the sound recording test for

Table 8.5: List of device to be used

Device name	Description
TD-BD-16ADUSB	A/D, D/A conversion device
PC	PC to operate TD-BD-16ADUSB chastity

HARK, change only the property of each module with the network file test.n same as that shown in Figure 8.2. Table 8.4 shows setting of each property. Here, note that it is necessary to use AudioStreamFromMic2, which is included only in the 3rd party version, not AudioStreamFromMic. Table 8.6 shows setting of each property.

```
> ./test.n
```

By entering the above, speech of the analog input 1 of 16ADUSB is recorded for three seconds. Speech of PCM 16-bit quantization and 32kHz sampling are saved in the little endian RAW format.

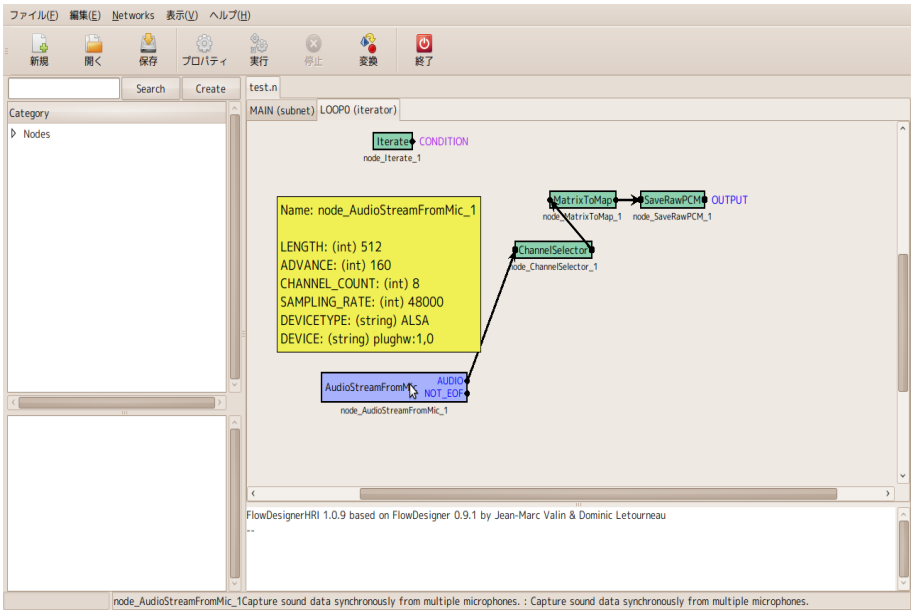


Figure 8.11: Network (test.n)

Table 8.6: Property of sound recording network

Module name	Property name	Type of value	Set value
AudioStreamFromMic2	LENGTH	int	512
	ADVANCE	int	160
	CHANNELCOUNT	int	16
	SAMPLINGRATE	int	16000
	DEVICETYPE	string	SINICH
	DEVICE	string	SINICH
ChannelSelector	SELECTOR	Object	<Vector<int> 0 >
SaveRawPCM	BASENAME	string	sep
	ADVANCE	int	160
	BITS	int	16
Iterate	MAXITER	int	300