HARK version 1.2.0 Document

Hiroshi G. Okuno
Kazuhiro Nakadai
Toru Takahashi
Ryu Takeda
Keisuke Nakamura
Takeshi Mizumoto
Takami Yoshida
Angelica Lim
Takuma Otsuka
Kohei Nagira
Tatsuhiko Itohara

[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

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 a multitude of sounds are heard, processes them to communicate with people and to enjoy TV, music or movies. A robot audition system that provides such recognition functionality needs to process sounds heard in various environments at various levels, and therefore it cannot be defined easily, similar to in robot vision.

Indeed, the OpenCV open source image processing software is an aggregate of a huge number of processing modules and therefore it is essential also for robot audition software to consist of aggregates that include the minimum required functions. The robot audition software HARK is the system that aims at being an "auditory OpenCV". HARK, like OpenCV, contains signal processing algorithms, measurement tools and GUI from a device level for the module necessary to "recognize," and is also is published as open source. The three major tasks in Computational Auditory Scene Analysis – the understanding of the environment from sound information – are 1) sound source localization, 2) sound source separation and 3) automatic speech recognition. HARK has been developed as an achievement obtained from research in these fields. HARK is currently available for free [1] as an open source project for research use.

1.2 Design philosophy of HARK

The 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,
4. **Provision of optimal sound processing module and advice** For the signal processing algorithms, each algorithm is based on effective premises, multiple 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.

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

We have continuously released and updated HARK under the above design concepts as follows:

Time	Version	Note
Apr., 2008	HARK 0.1.7	Released as open source software
Nov., 2009	HARK 1.0.0 pre-release	many improvements, bug-fix, documentation
Oct., 2010	HARK 1.0.0	bug-fix, released support tools
Feb., 2012	HARK 1.1	functionality improvements, 64-bit support, ROS support, bug-fix
Mar., 2013	HARK 1.2	3D localization, Windows support, English speech recognition, bug-fix

Basically, users' feedback which was sent to hark-support ML plays a great role for improvements, and bug-fix.

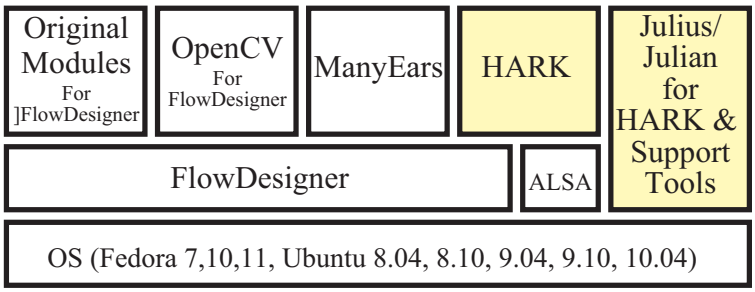


Figure 1.1: Relation between the robot audition software HARK and the middleware FlowDesigner and OS

As shown in Figure 1.1, HARK uses FlowDesigner [2] 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 the API called ALSA (Advanced Linux Sound Architecture) is used so as to support multiple multichannel A/D systems. HARK for PortAudio is also being developed since PortAudio has recently become available for Windows systems.

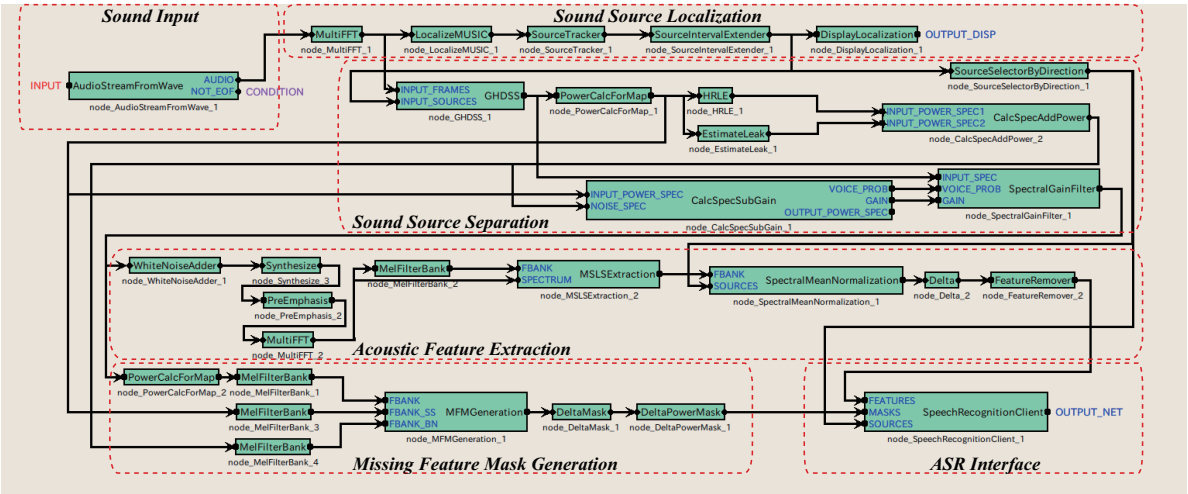


Figure 1.2: Module construction on FlowDesigner for typical robot audition with HARK

Middleware FlowDesigner

In robot audition, sound sources are typically separated based on sound source localization data, and speech recognition is performed for the separated speech. Each processing would become flexible by comprising it of multiple modules so that an algorithm can be partially substituted. Therefore, it is essential to introduce a middleware that allows efficient integration 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 [2] 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 (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 settings) has been improved based on the past use experience.
[2]

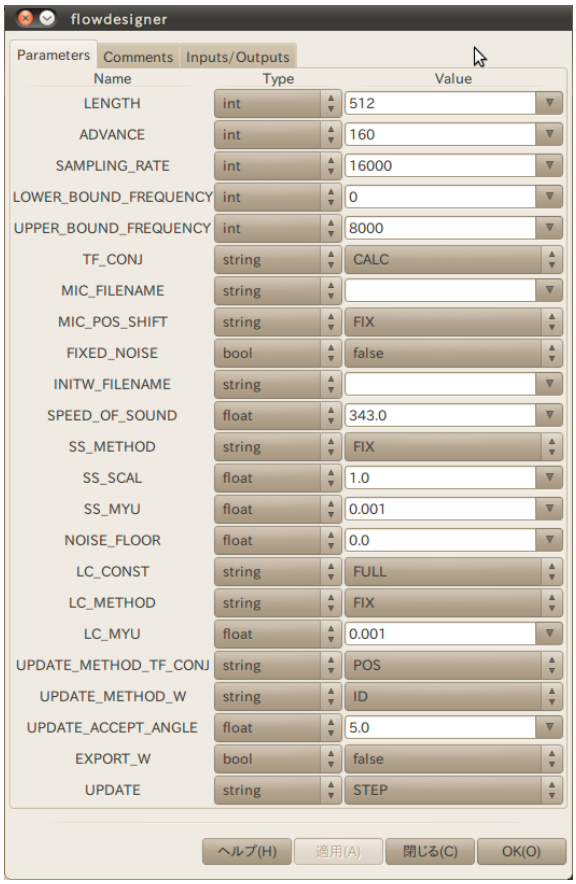


Figure 1.3: Attribute setting screen of GHDSS


SIG2 (whole body)


Robovie-R2


HRP-2 (8ch)

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

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.

[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.

Input device

Multiple 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 are the multichannel A/D conversion devices supported by HARK.

- System in Frontier, Inc., The RASP series,
- A/D conversion device of ALSA base, (e.g. RME, Hammerfall DSP series, Multiface AE)
- Microsoft Kinect
- Sony PS-EYE
- Dev-Audio Microcone

These A/D systems have different number of input channels. Any number of channels can be used in HARK by changing internal parameters in HARK. However, the processing speed might fall under such a condition with a large number of channels. Both 16 bits and 24 bits are supported. Further, the sampling rate assumed for HARK is 16kHz and therefore a downsampling module can be used for 48KHz sampling data. Note that Tokyo Electron Device Ltd., TD-BD-16ADUSB (USB interface) is now not supported, since Linux kernel supported by them is too old.

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.

Sound source localization

MUltiple SIgnal Classification (MUSIC) method, which has shown the best performance in 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 [3] 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. In HARK 1.1, we supported GEVD-MUSIC and GSVD-MUSIC which are extended version of MUSIC. By the extension, we can suppress or *whiten* a known high power noise such as robot ego-noise and localize desired sounds under this noise. In HARK 1.2, we extended the algorithm to localize sound sources in a 3-dimensional way.

Sound source separation

For sound source separation, Geometric-Constrained High-order Source Separation (GHDSS) [8], which is known to have the highest total performance in various acoustic environments from the past usage experience, PostFilter and the noise estimation method Histogram-based Recursive Level Estimation HRLE are employed for HARK 1.0.0. Presently, the best performance and stability in various acoustic environments are obtained by the combination of GHDSS and 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 [4], which was supported as an external module with HARK 0.1.7,

Table 1.1: Nodes and Tools provided by HARK 1.1.0

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
		SaveWavePCM	Save sound in wav-formatted file
		HarkDataStreamSender	Socket-based data communication
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
		CMLoad	Load a Correlation Matrix (CM) file
		CMSave	Save a CM file
		CMChannelSelector	Channel selection for CM
		CMMakerFromFFT	Create a CM
		CMMakerFromFFTwithFlag	Create a CM
		CMDivideEachElement	Division of each element of CM
		CMMultiplyEachElement	Multiplication of each element of CM
		CMInverseMatrix	Inverse of CM
		CMMultiplyMatrix	Multiplication of CM
		CMIdentityMatrix	Output identity CM
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 → 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	harktool4	Visualize data / Generate setting file
		wios	Sound recording tool for transfre function measurements

3. Combination of GSS and PostFilter as an original design [5] employed for in 1.0.0 HARK prerelease,

4. Combination of GHDSS and HRLE employed for HARK 1.0.0 [6, 8].

GSS of 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. GSS was 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 removed. 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 GHDSS of 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 GSS and GHDSS and 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 PostFilter and 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 GHDSS separation algorithm and estimates inter-channel leak energy, in combination with HRLE .

MFT-ASR: Speech recognition based on MFT

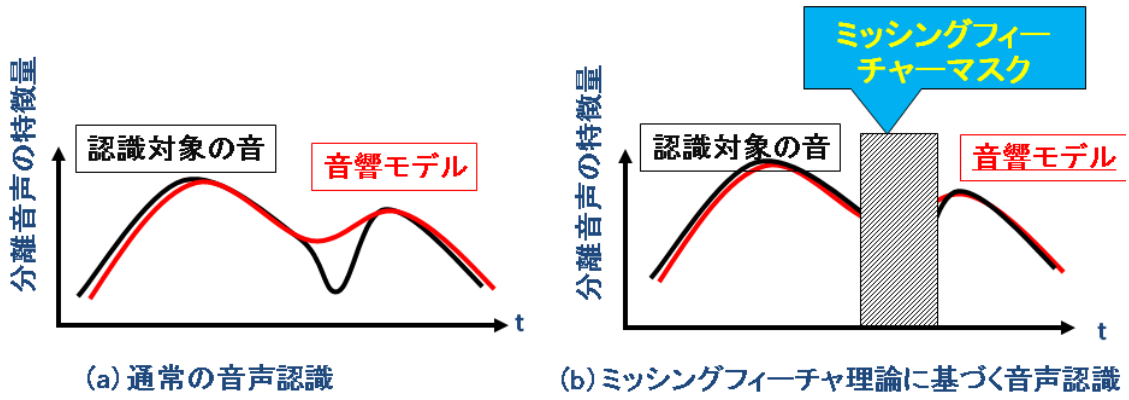


Figure 1.5: Schematic diagram of speech recognition using Missing Feature Theory

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) [4]. The concept of MFT-ASR is shown in Figure 1.5. The black and red lines in the figure indicate the time variation of acoustic features in a separated sound and that of an acoustic model for corresponding speech, used 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 a Hard Mask) is usually used. Masks with continuous values from 0 to 1 are called Soft Masks. 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 a 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 software Julius developed by Tokyo Institute of Technology Furui Laboratory is used, reinterpreted as MFT-ASR [13]. 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 to be updated without having to modify MFT-ASR. 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 and trap the spectral distortion only for specific acoustic features, Mel Scale Log Spectrum (MSLS) [4] 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 infrequent, better performance is achieved by speech recognition with MFCC in some cases. HARK 1.0.0 provides a new module to use the power term Δ with MSLS features [6]. The effectiveness of the 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 with white noise added. Then 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 [6] for one speaker's speech.

1.3 Application of HARK

We have developed an auditory function robot with binaural hearing using two microphones and used three-speaker simultaneous speech recognition as a kind of benchmark. For auditory functions on SIG and SIG2 (upper body humanoid robots), simultaneous speech recognition at 1m away from three speakers who stand at an interval of 30 degrees is possible at a certain level of accuracy [16]. 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 an 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 a variation of three-speaker simultaneous speech recognition, a referee robot that judges the winner of the rock-paper-scissors game played orally by three persons has been developed on Robovie-R2 [17]. Moreover, as a non-robot application, we have developed a system that visualizes the sounds that HARK localizes and separates 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 [18, 19].

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

1.3.1 Three-speaker simultaneous speech recognition

The three-speaker simultaneous speech recognition system returns speech recognition results for 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 FlowDesigner is shown in Figure 1.2. The dialogue management module performs the following:

1. Listen to speech of a user. When judging if 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 any following 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 multiple 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 to face the speaker 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 performed through voice. The fun of the “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 [17]. 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 [17]. 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 [20], In the case that multiple 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 improve **auditory awareness** (sound recognition),

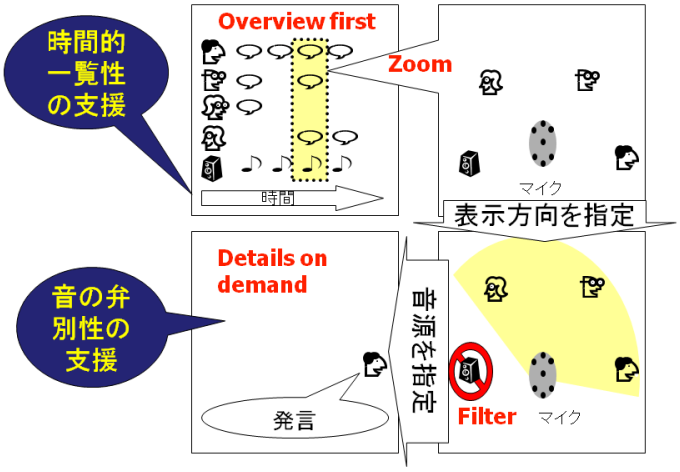


Figure 1.7: CASA 3D Visualizer: CASA 3D Visualizer: Visual Information-Seeking Matra Visualization of outputs from HARK in accordance with “Overview first, zoom and filter, then details on demand”

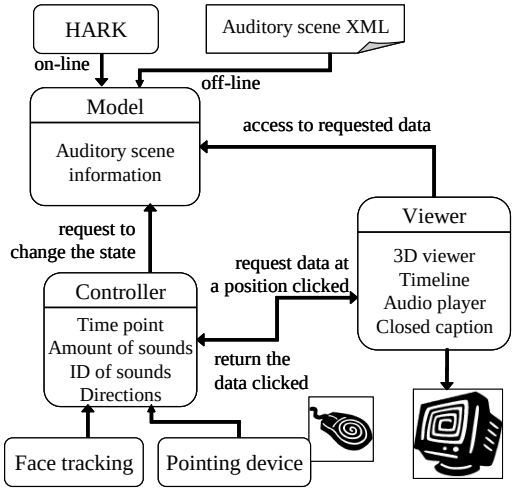


Figure 1.8: Implementation with the MVC (Model-View-Control) model of CASA 3D Visualizer

we modified HARK, and designed and implemented a three-dimensional sound environment visualization system that supports sound environment comprehension [18, 19]. 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

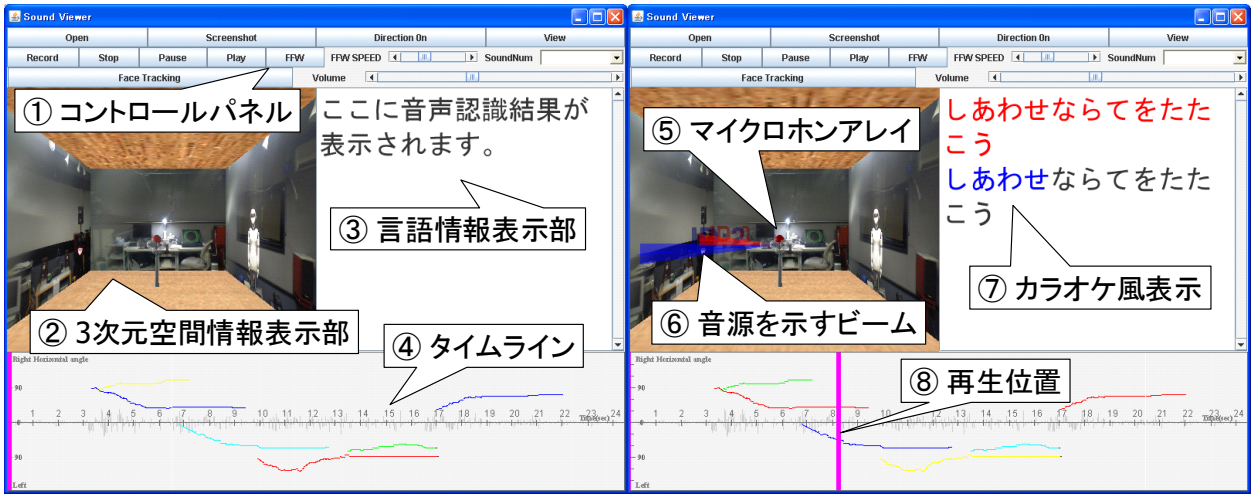


Figure 1.9: GUI of CASA 3D Visualizer

and rotation can be 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 [18],

2. Display results of Visualizer on Head Mount Display (HMD) [21].

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 the analogy of Google Maps, 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

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.

are separated to the following two processes.

1. Installation of microphones in Texai, measurement of impulse response and installation 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 is 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.

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

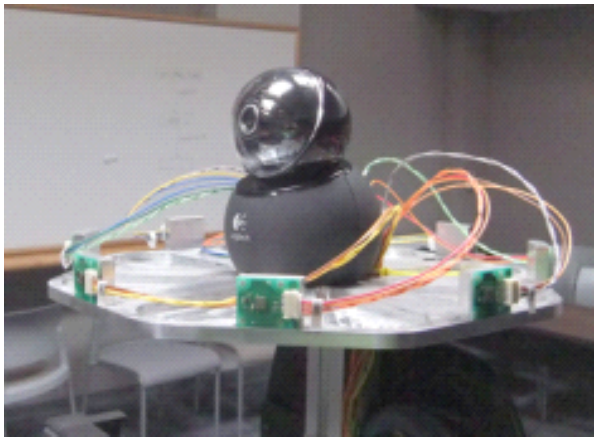


Figure 1.11: Closeup of the first head of Texai: Eight MEMS microphones are embedded on a disk

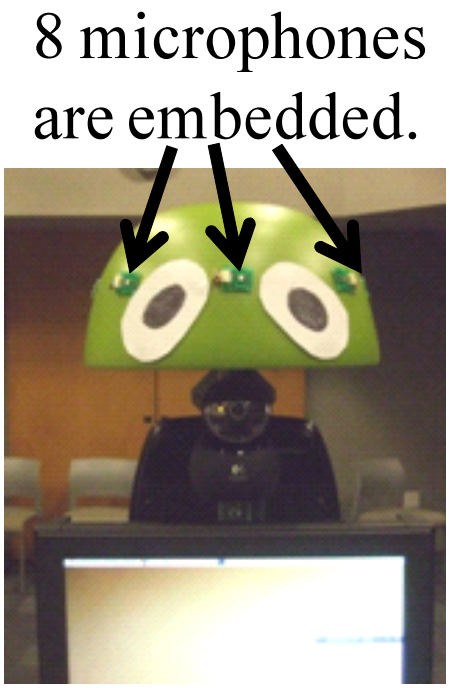


Figure 1.12: Closeup of the head of Texai: Eight MEMS microphones are embedded in the shape of circle

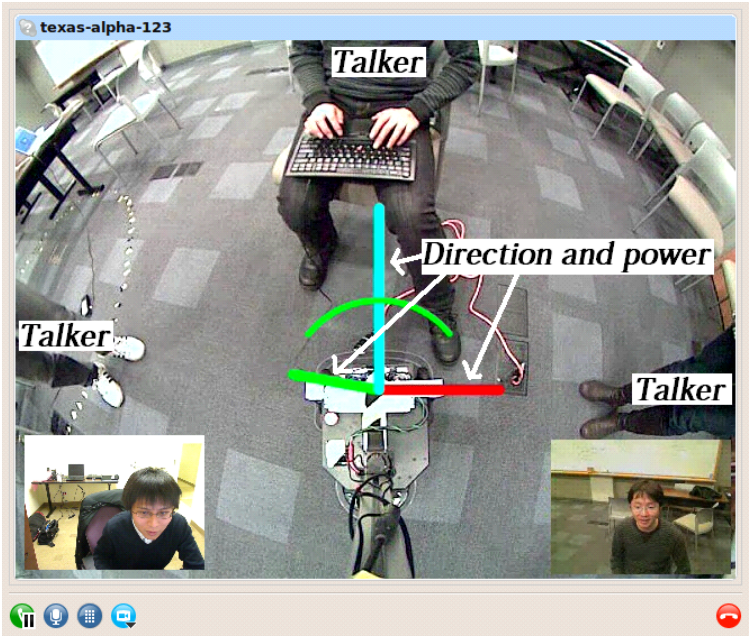


Figure 1.13: Image seen by the remote operator through Texai

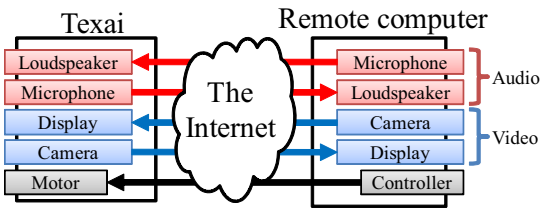


Figure 1.14: Teleoperation of Texai

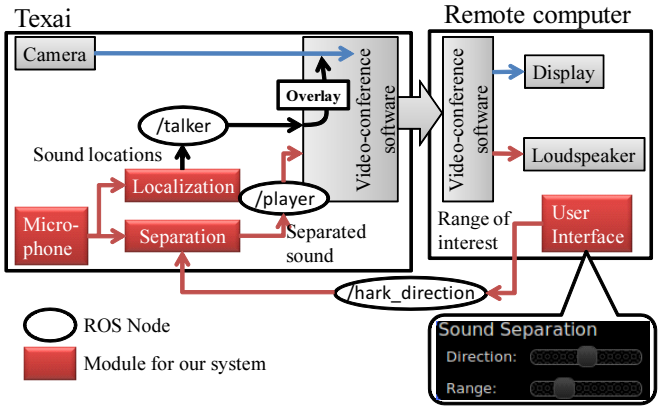


Figure 1.15: Built-in of HARK in Texai

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 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 document 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 HARK and problems on it.

2.1 Technology to distinguish sounds is the base for robot audition

According to the Astro Boy Encyclopedia (written by Mitsumasa Oki, Shobunsha), 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 the "Cocktail party effect" discovered by Cherry in 1953, which distinguishes speeches selectively. Hearing-impaired persons or elderly adults with weak hearing may ask, "Isn't a function that distinguishes simultaneous utterances enough, though it's not a super device?". The chronicle of Japan, Suiko-ki, introduces an anecdote of "Prince Shotoku" who understood utterances from ten speakers at the same time and judged. The old tale "Ear Hood," in which people can hear and understand animals, trees and plants, inspires the imagination of kids. If we could give such a separation function to a robot, it would be able to interact with humans much more easily.

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 word acquisition and back channels by non-voice and has many various functions. Indeed, the importance of research on Automatic Speech Recognition (ASR) has been highly recognized and huge amounts of funding and effort 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 stance of the authors has been to develop a processing method for sounds using 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 a 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 sound environment understanding (Computational Auditory Scene Analysis) [9], focusing on the importance of dealing with general sounds including music, environmental sounds and their mixture with speech. An important task in a study on sound environment understanding is sound mixture processing. 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 a sound mixture with mixed sound 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. In order to achieve this goal, the middleware that can effectively provide a combination with a good balance is also important. The robot audition software **HARK** is constructed on the

[1] http://www31.ocn.ne.jp/ goodold60net/atm_gum3.htm

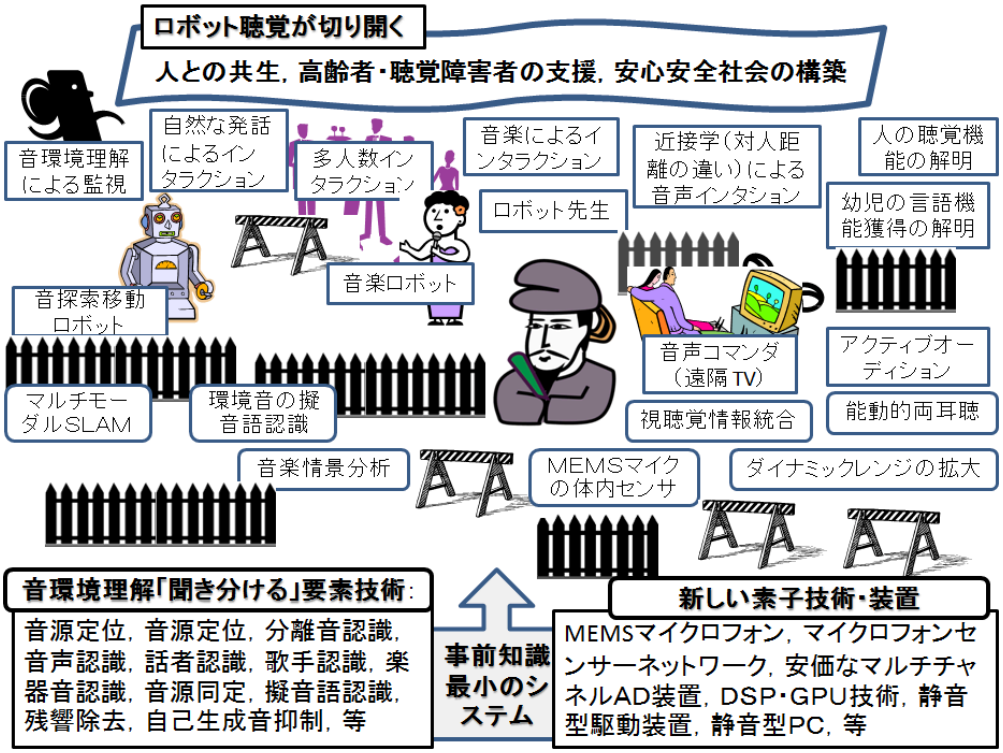


Figure 2.1: Development of robot audition based on sound environment understanding

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 removing the need for prior knowledge as much as possible, and is a system that aims at being the "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 useful. 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 a 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, taking a cue 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[11, 12]. 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 [14, 15, 27]. "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 the sound source is located at 70 - 80 degrees. Indeed,

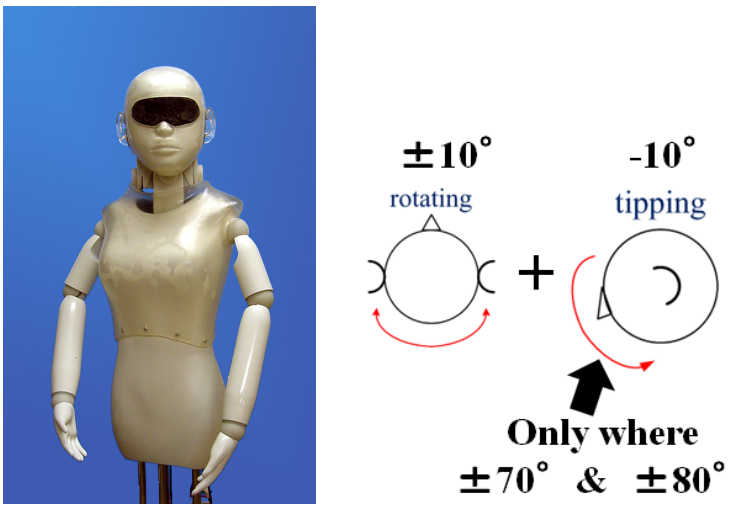


Figure 2.2: Active audition of SIG2: For the surrounding sound, ambiguity of front and back is removed by moving the neck right, left and down.

the 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 [12]. 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 [12]. Nakadai et al. incorporated a function to reject motor noise from specific directions into HARK [12]. 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 speaker's direction do not corresponds to the radiated

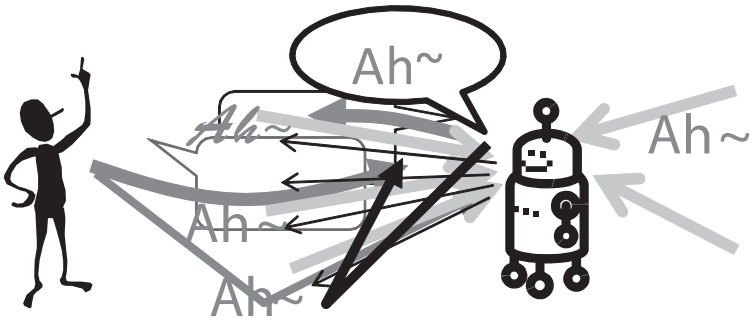


Figure 2.3: The robot's own voice enters its own ears accompanied with reverberations and another person's speech (called barge-in)

sound's direction, so as to suppress self-generated sound [12]. 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 [12].

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 speak 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 adults, their own voice enters the primary auditory cortex of the temporal lobe, though is not transmitted to the associative auditory cortex of the cerebral cortex, and thus their 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 a single technology but is a process consisting of multiple systems**. There are a number of elemental technologies for component parts and in addition, component parts vary considerably in their performance. Therefore they need to interact well with each other during processing. Moreover, better interaction enables better functioning of the processes as a whole. 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 audio-visual information integration. At the lowermost level, a speaker is detected from audio signals and lip 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 Applications enabled by robot audition

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 multiple 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 [12]. 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 [1]. 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.

Learning to listen to what others say at sixty years old, from "The Analects of Confucius / Government"
It is said that a person learns to listen to what others say at age sixty. However, the sensitivity of auditory organs for high frequencies drops through overwork or age, and the person loses the ability to hear conversations so he/she cannot depend on their ears even if they wish to.

Bibliography

[1] Nakadai, Mitsunaga, Okuno (Eds): Special Issue on Robot Audition (Japanese), Journal of Robotics Society of Japan, Vol.28, No.1 (Jan. 2010).

[2] C. Côté, et al.: Code Reusability Tools for Programming Mobile Robots, *IEEE/RSJ IROS 2004*, pp.1820–1825.

[3] J.-M. Valin, F. Michaud, B. Hadjou, J. Rouat: Localization of simultaneous moving sound sources for mobile robot using a frequency-domain steered beamformer approach. *IEEE ICRA 2004*, pp.1033–1038.

[4] S. Yamamoto, J.-M. Valin, K. Nakadai, T. Ogata, and H. G. Okuno. Enhanced robot speech recognition based on microphone array source separation and missing feature theory. *IEEE ICRA 2005*, pp.1427–1482.

[5] H. G. Okuno and K. Nakadai: Robot Audition Open-Sourced Software HARK (Japanese) Journal of Robotics Society of Japan, Vol.28, No.1 (Jan. 2010) pp. 6–9, Robotics Society of Japan.

[6] K. Nakadai, T. Takahasi, H.G. Okuno, H. Nakajima, Y. Hasegawa, H. Tsujino: Design and Implementation of Robot Audition System "HARK", *Advanced Robotics*, Vol.24 (2010) 739-761, VSP and RSJ.

[7] K. Nakamura, K. Nakadai, F. Asano, Y. Hasegawa, and H. Tsujino, "Intelligent Sound Source Localization for Dynamic Environments", in *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS 2009)*, pp. 664–669, 2009.

[8] H. Nakajima, K. Nakadai, Y. Hasegawa, H. Tsujino: Blind Source Spearation With Parameter-Free Adaptive Step-Size Method for Robot Audition, *IEEE Transactions on Audio, Speech, and Language Processing*, Vol.18, No.6 (Aug. 2010) 1467–1485, IEEE.

[9] D. Rosenthal, and H.G. Okuno (Eds.): *Computational Auditory Scene Analysis*, Lawrence Erlbaum Associates, 1998.

[10] Bregman, A.S.: *Auditory Scene Analysis – the Perceptual Organization of Sound*, MIT Press (1990).

[11] H.G. Okuno, T. Nakatani, T. Kawabata: Interfacing Sound Stream Segregation to Automatic Speech Recognition – Preliminary Results on Listening to Several Sounds Simultaneously, *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-1996)*, 1082–1089, AAAI, Portland, Aug. 1996.

[12] Special interest group of AI Challenge, The Japanese Society of Artificial Intelligence. Papers are available on the web page: http://winnie.kuis.kyoto-u.ac.jp/AI-Challenge/

[13] Y. Nishimura, T. Shinozaki, K. Iwano, S. Furui: Speech recognition using band-dependent weight likelihood (Japanese), Annual Meeting of the Acoustical Society of Japan, Vol.1, pp.117–118, 2004.

[14] Nakadai, K., Lourens, T., Okuno, H.G., and Kitano, H.: Active Audition for Humanoid. In *Proc. of AAAI-2000*, pp.832–839, AAAI, Jul. 2000.

[15] Nakadai, K., Hidai, T., Mizoguchi, H., Okuno, H.G., and Kitano, H.: Real-Time Auditory and Visual Multiple-Object Tracking for Robots, In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pp.1425–1432, IJCAI, 2001.

[16] Nakadai, K., Matasuura, D., Okuno, H.G., and Tsujino, H.: Improvement of recognition of simultaneous speech signals using AV integration and scattering theory for humanoid robots, *Speech Communication*, Vol.44, No.1–4 (2004) pp.97–112, Elsevier.

[17] Nakadai, K., Yamamoto, S., Okuno, H.G., Nakajima, H., Hasegawa, Y., Tsujino H.: A Robot Referee for Rock-Paper-Scissors Sound Games, *Proceedings of IEEE-RAS International Conference on Robotics and Automation (ICRA-2008)*, pp.3469–3474, IEEE, May 20, 2008. doi:10.1109/ROBOT.2008.4543741

[18] Kubota, Y., Yoshida, M., Komatani, K., Ogata, T., Okuno, H.G.: Design and Implementation of 3D Auditory Scene Visualizer towards Auditory Awareness with Face Tracking, *Proceedings of IEEE International Symposium on Multimedia (ISM2008)*, pp.468–476, Berkeley, Dec. 16. 2008. doi:10.1109/ISM.2008.107

[19] Kubota, Y., Shiramatsu, S., Yoshida, M., Komatani, K., Ogata, T., Okuno, H.G.: 3D Auditory Scene Visualizer With Face Tracking: Design and Implementation For Auditory Awareness Compensation, *Proceedings of 2nd International Symposium on Universal Communication (ISUC2008)*, pp.42–49, IEEE, Osaka, Dec. 15. 2008. doi:10.1109/ISUC.2008.59

[20] Kashino, M., and Hirahara, T.: One, two, many – Judging the number of concurrent talkers, *Journal of Acoustic Society of America*, Vol.99, No.4 (1996), Pt.2, 2596.

[21] K. Tokuda, K. Komatani, T. Ogata, H. G. Okuno: Hearing-Impared Supporting System in Understanding Auditory Scenes by Presenting Sound Source Localization and Speech Recognition Results in Integrated manner on HMD (Japanese), The 70th Annual Conference of Information Processing Society ofJapan, 5ZD-7, Mar. 2008.

[22] H. G. Okuno, K. Nakadai: Research Issues and Current Status of Robto Audition (Japanese), IPSJ Magazine "Joho Shori" Vol.44, No.11 (2003) pp.1138–1144, Information Processing Society of Japan.

[23] H. Okuno, H. Mizoguchi: Information Integration for Robot Audition: State-of-the-art and Issues (Japanese) Journal of the Sciety of Instrument and Control Engineers, Vol.46, No.6 (2007) pp.415–419.

[24] H. G. Okuno, S. Yamamoto: Computing for Computational Auditory Scene Analysis (Japanese) Journal of the Japanese Society of Artificial Intelligence, Vol.22, No.6 (2007) pp.846–854.

[25] Takeda, R., Nakadai, K., Komatani, K., Ogata, T., and Okuno, H.G.: Exploiting Known Sound Sources to Improve ICA-based Robot Audition in Speech Separation and Recognition, In *Proc. of IEEE/RSJ IROS-2007*, pp.1757–1762, 2007.

[26] Tasaki, T., Matsumoto, S., Ohba, H., Yamamoto, S., Toda, M., Komatani, K. and Ogata, T. and Okuno, H.G.: Dynamic Communication of Humanoid Robot with Multiple People Based on Interaction Distance, Journal of The Japanese Society of Artificial Intelligence, Vol.20, No.3 (Mar. 2005) pp.209–219.

[27] H-D. Kim, K. Komatani, T. Ogata, H.G. Okuno: Binaural Active Audition for Humanoid Robots to Localize Speech over Entire Azimuth Range, *Applied Bionics and Biomechanics*, Special Issue on "Humanoid Robots", Vol.6, Issue 3 & 4(Sep. 2009) pp.355-368, Taylor & Francis 2009.

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

You can find the HARK software and its installation instructions in HARK web site (http://winnie.kuis.kyoto-u.ac.jp/HARK/) You can download the installers for Windows from the site and the repositories for Ubuntu where the package exists. The source code is available in the repository.

If you are the first time to use HARK, we strongly recommend to use the windows packages. This document do not cover the installation from the source code.

3.2 Installation of the Software

3.2.1 For Linux Users

Instructions in this section show operation examples for all the actions required to complete installation. > 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.

For Ubuntu

Ubuntu 12.04 users can install from the package. First, add the HARK repository to your system. The following web page describes how to do this: HARK installation instructions
Then, install the package using the following commands:

```
> sudo apt-get update
> sudo apt-get install install harkfd harktool4 julius-4.2.2-hark-plugin
```

If you are using other environments, you need compile from the source code. See the HARK installation instructions to get the source code.

3.2.2 For Windows Users

For Windows, you can install using the installer. This program installs the following software.

1. FlowDesigner for HARK
2. hark-fd
3. HARK-Julius
4. HARK-WIOS

First, run `HARK_1.2.0_setup.exe`. If your OS is Windows 8, right click the installer, and select *Run as administrator*. Then, the installation of the FlowDesigner for HARK begins. If you agree the license, click the *Next* button. The installation will proceed. Similarly, you can install hark-fd, HARK Julius, and HARK-WIOS.

3.3 FlowDesigner

FlowDesigner is an open source middleware. Presently, the only middleware that HARK is based on is FlowDesigner. Future implementations on other middleware are also planned.

The main feature of FlowDesigner is that you can construct a system using a graphical user interface. 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 nodes, and setting property values of the nodes.

The rest of this section includes the description of FlowDesigner for Linux in Section 3.3.1 and the description of FlowDesigner for Windows in Section 3.3.2. Since the FlowDesigner for Windows mainly explains the difference from the linux version, refer to the Section 3.3.1.

3.3.1 FlowDesigner for Linux

An overview of FlowDesigner is shown in Figure3.4 FlowDesigner is started using the following command.

```
> flowdesigner
```

Basic operation of FlowDesigner

A node is a processing unit that processes data. First, an example of a node is shown in Figure 3.5. The green part enclosed with a rectangle is the node, and the node's name is displayed in the center. In this example, PostFilter is displayed. The name node_Postfilter_1 displayed below the node is an instance name of the node. The same node, when applied multiple times, is named after the order in which it is created. For example, node_Postfilter_2,

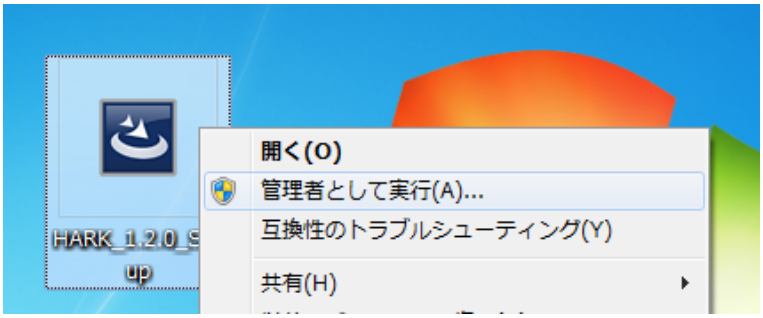


Figure 3.1: Run as administrator

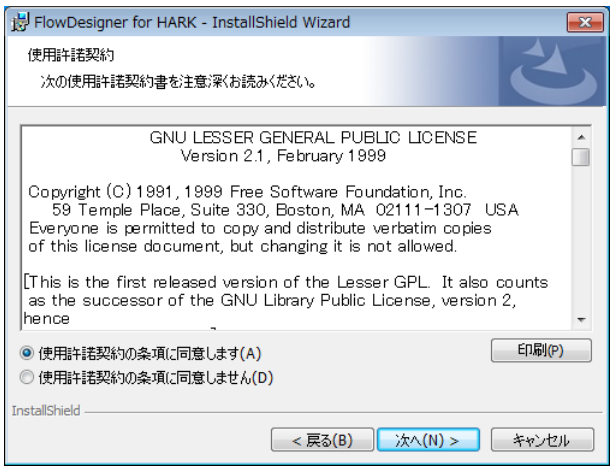


Figure 3.2: End-User License Agreement

Figure 3.3: Installation

node_Postfilter_3. It is useful to distinguish each node. Each node may have multiple terminals for inputs and outputs of data processed. The black points on the right and left of the node 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 node has at least one output terminal though there are no input terminals in some cases. For example, the **constant** node included in FlowDesigner by default does not have inputs but one output. Data processing flows are defined by positioning multiple nodes and connecting the terminals of the nodes. A series of connected nodes is called a network, and creating a network itself is programming. Networks can be saved using the file save menu. Once a network is saved, it can be loaded by FlowDesigner anytime. When opening a saved file with a text editor, an XML-like description is seen. The user may edit such a file directly by a text editor once they get used to HARK.

Since some nodes have properties, the user can modify the processing of the nodes in detail by setting their property values. To set property values, move the mouse cursor over a node and left-double click on it. The setting DIALOG window opens and the user may input property values using the 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 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.

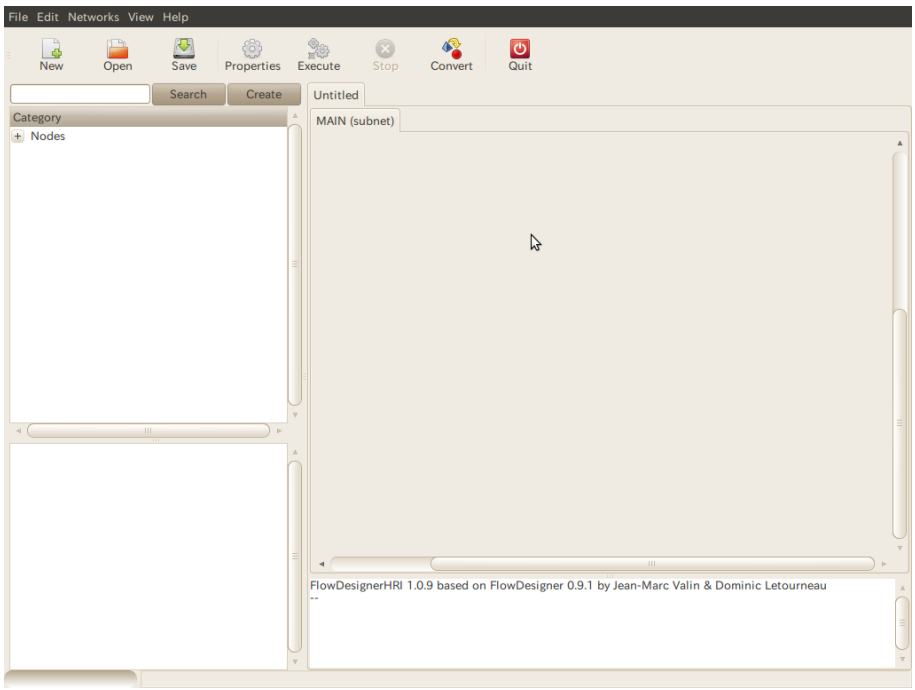


Figure 3.4: Overview of FlowDesigner

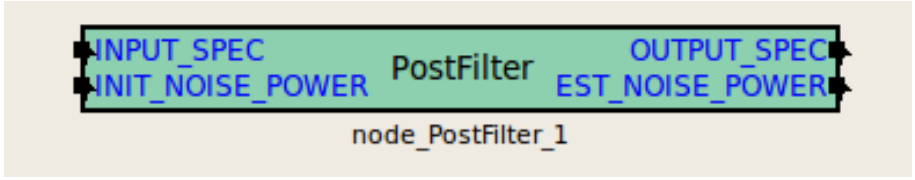


Figure 3.5: Overview of node

Module operation / terminal connection / property setting

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

- **Creating a node**
  Right-click on the network construction tab to display a list of node categories. Choose a category and left click on your desired node. To cancel the addition of the node, click the background part of the network construction tab (either of right or left).
- **Moving a node**
  To relocate a node to an easy-to-see position, click and drag the node with the left mouse button.
- **Deleting a node**
  Right-click on the node and select ”delete”.
- **Copying a node**
  Hold down the Shift key and left-click the node to copy it. Since attribute values are copied along with the node itself, this function is useful in constructing a network that executes the same processes in parallel.
- **Connecting a node**
  Modules are connected by connecting the terminals with arrows. Left-click on the source terminal, drag and release it on the connection point. The user can connect input to output terminals but cannot connect between input terminals and between output terminals. Multiple connections can be drawn from one output terminal to

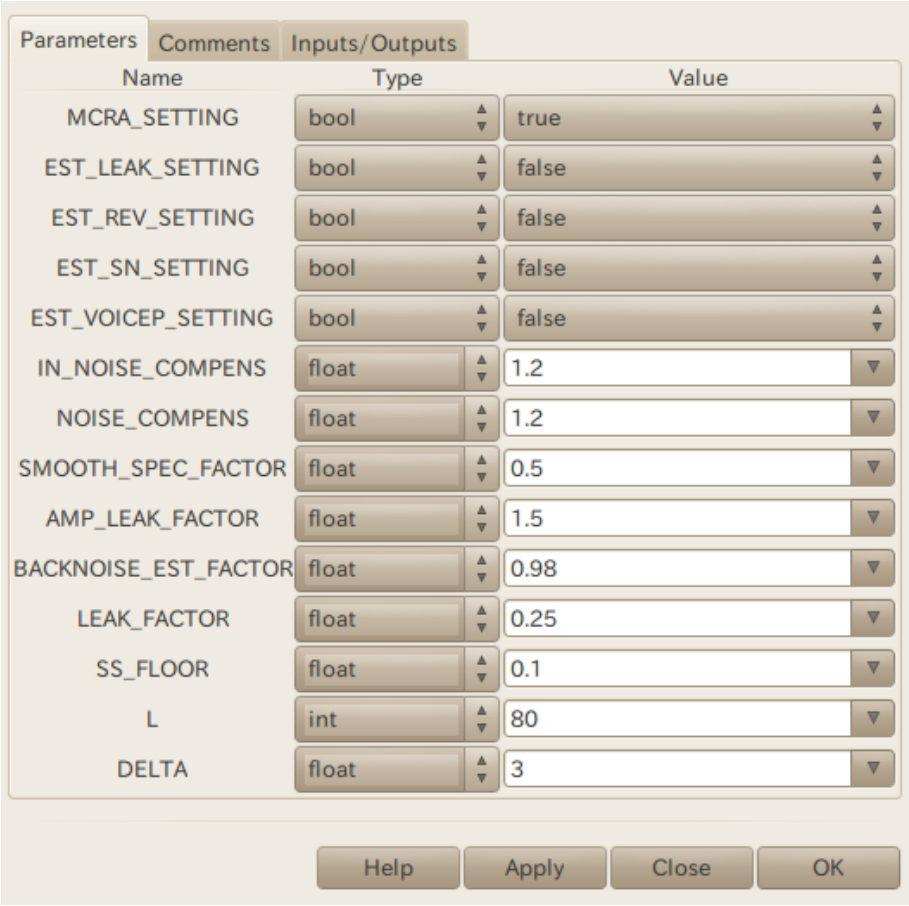


Figure 3.6: Overview of property

send the result of a node to multiple nodes. On the other hand, multiple connections cannot be established onto one input terminal.

In some cases, input and output terminals cannot be established, for instance when the terminal cannot process a particular data type. Such a connection is indicated with a red arrow. In some cases, even though the arrow is black, an error may occur in a program execution when failing to judge the data type 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 node reference for accurate connections.

- **Disconnecting a node**
  Hold down Shift and right-click on the starting point or end point of the arrow of the connection that the user wish to disconnect.

- **Bending a node connection line**
  When moving a node, the node moves with the connection maintained. When positioning a large number of nodes, nodes and their connection lines overlap, which makes them difficult to see. Therefore, it is recommended to bend the lines so that network can be easily seen with fewer overlaps. Left-click the connection line and drag it to bend the line at that point.

- **Setting a property value**
  Right-click on a node and select “Properties” to display its settings dialog. Enter your desired values and data types, referring to the node reference, and click on “Apply”. The settings are saved and the dialog is closed by clicking on “ OK”.

Next, we describe how to create a network, using the example of a sound source localizing network.

Creating a network for beginners

This section shows an example of building a sound source localizing network for users who are creating such a network for the first time. As well as the main network tab, the subnetwork tab for Iteration is needed. First, we will build a sound source waveform reader in the main tab, then create a subnetwork and build localization and result display parts in the subnetwork.

We first describe the modular arrangement of the sound source waveform reading part, as well as its connections and property settings. Start FlowDesigner, and the MAIN (subnet) tab will appear on the right side of the window. Proceed to work in this tab first. Add the Constant node and the InputStream node as shown in Figure 3.7. Right-click on the grey background in the tab, and a pull-down menu will appear. Move the mouse cursor onto New Node, and the list of nodes registered in FlowDesigner will be displayed. Here, select the General category, and the list of nodes registered in General will be displayed. The Constant node is included in this list. From there, move the mouse cursor onto Constant to add a Constant node in the MAIN (subnet) tab. In the following descriptions, an operation to select this node out of a pull-down menu is indicated as “New Node → General → Constant”. In the same way, add the InputStream node as New Node → IO → InputStream. Next, connect the Constant node and an InputStream node as

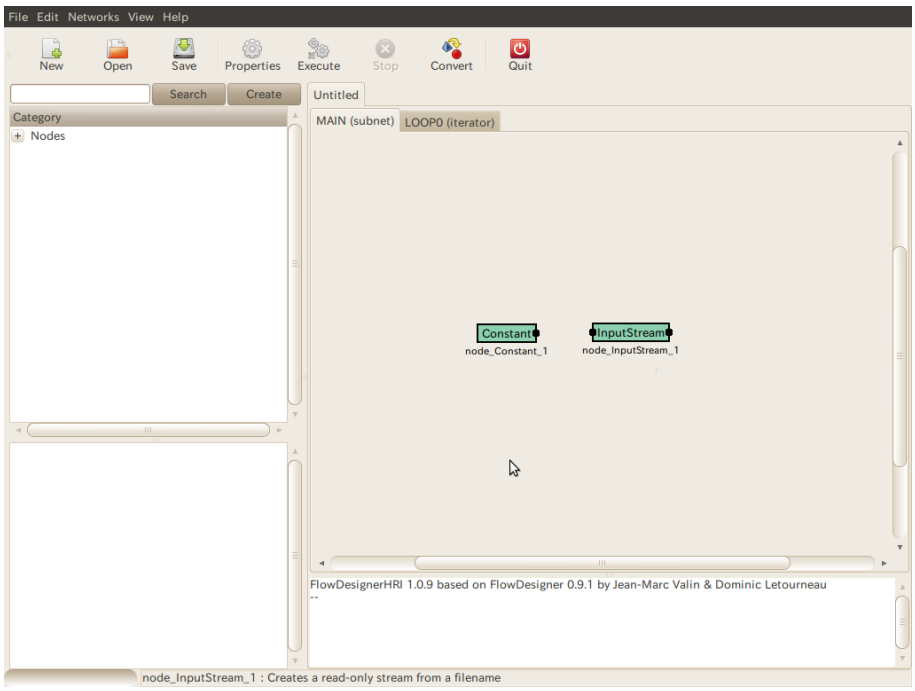


Figure 3.7: Addition of two nodes

shown in Figure3.8.

Next, set properties of the node. Here, set only the Constant node. The InputStream node does not have settable property values. Right-click on the Constant node and select Properties to reveal the property setting dialog. The dialog contains Parameters, Comments and Inputs/Outputs tabs, with the Parameters tab open by default. The Comments tab and Inputs/Outputs tab are not used. The Parameters tab contains three items: 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 node has only one settable property, which has the attribute name VALUE. Since a file name is to be indicated in this node, give tutorial1.wav for the file name in the property value. The data type is `string` . Finally, save the settings by pressing “Apply” in the ”Properties” window, followed by ”Close”. Alternatively, pressing “OK” to perform these two operations simultaneously. Since the user cannot operate other windows of FlowDesigner while the ”Properties” window is displayed, it is necessary to finish the property setting before returning to such operations. Here, the property setting is complete, and a network to read from a sound source waveform has been created.

Next, we describe subnetwork creation. A subnetwork is created by selecting ”Add Iterator” from the ”Networks” menu in FlowDesigner. Select “Add Iterator” to enter a name for the Iterator tab. If using the default name ”LOOP0”, press ”OK”. Subnetwork creation is cancelled by pressing the cancel button. Clicking on ”OK” for LOOP0, a tab

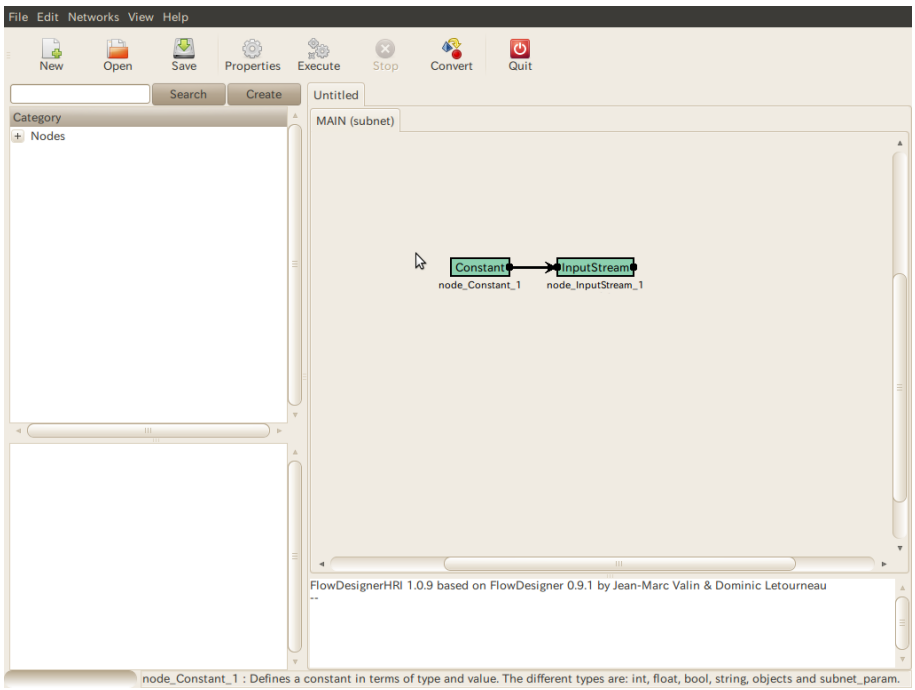


Figure 3.8: Connections of two nodes

named ”LOOP0” appears next to the MAIN (subnet) tab and the LOOP0 tab becomes active. Build a localization part and result display part in this tab. This sound source localization subnetwork is built with nodes for file reading, FFT, sound source localization using the MUSIC method, sound source tracking and localization result display. Figure 3.9 shows nodes added using the following procedures.

```
New Node -> HARK -> AudioIO       -> AudioStreamFromWave
New Node -> HARK -> MISC          -> MultiFFT
New Node -> HARK -> Localization -> LocalizeMUSIC
New Node -> HARK -> Localization -> SourceTracker
New Node -> HARK -> Localization -> DisplayLocalization
```

Connect these nodes as shown in Figure 3.10. After completing the connection, set node properties, assuming the audio file to be processed is 16,000 [Hz]. In the AudioStreamFromWave node, there are three property values: LENGTH, ADVANCE and USE_WAIT. LENGTH and ADVANCE indicate the analysis frame length and frame shift length of speech by unit samples, respectively. The 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 USE_WAIT of the data type `bool` . The user does not need to change these values.

In the MultiFFT node, there are three properties to set: LENGTH, WINDOW and WINDOW_LENGTH. The 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 WINDOW_LENGTH 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. LocalizeMUSIC nodes have nineteen properties to set: MUSIC_ALGORITHM, TF_CHANNEL_SELECTION, LENGTH, SAMPLING_RATE, A_MATRIX, ELEVATION, WINDOW, PERIOD, NUM_SOURCE, MIN_DEG, MAX_DEG, LOWER_BOUND_FREQUENCY, UPPER_BOUND_FREQUENCY, SPECTRUM_WEIGHT_TYPE, A_CHAR_SCALING, MANUAL_WEIGHT_SPLINE, MANUAL_WEIGHT_SQUARE, ENABLE_EIGENVALUE_WEIGHT, and DEBUG. The details of those parameters are given in the Parameters section in LocalizeMUSIC . The example of the parameter setting is shown below.

- **MUSIC_ALGORITHM** : SEVD
- **TF_CHANNEL_SELECTION** : `<Vector<int> 0 1 2 3 4 5 6 7>`
- **LENGTH** : 512[samples]

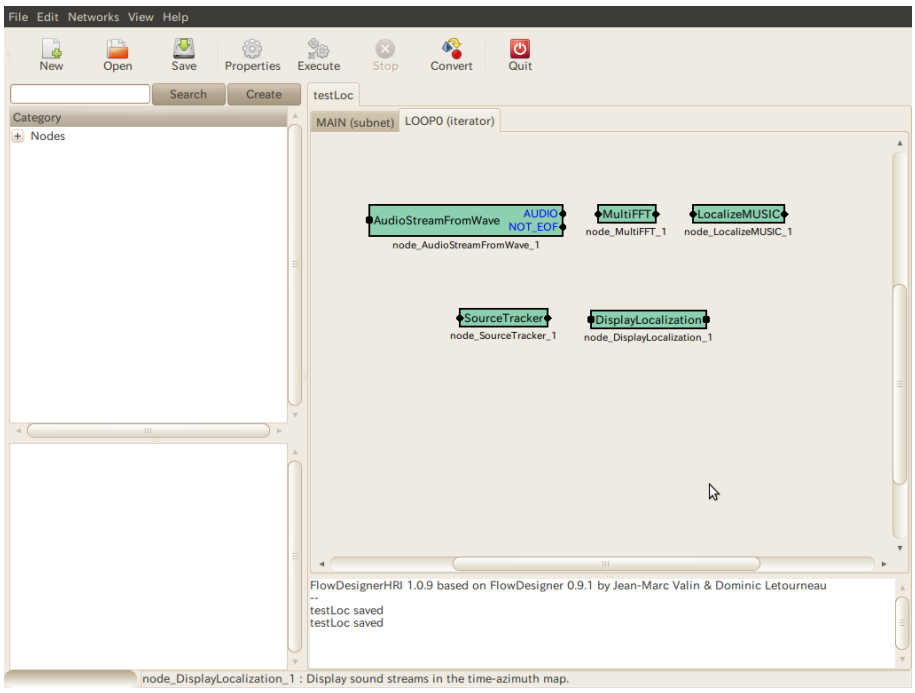


Figure 3.9: Nodes for sound source localizing part

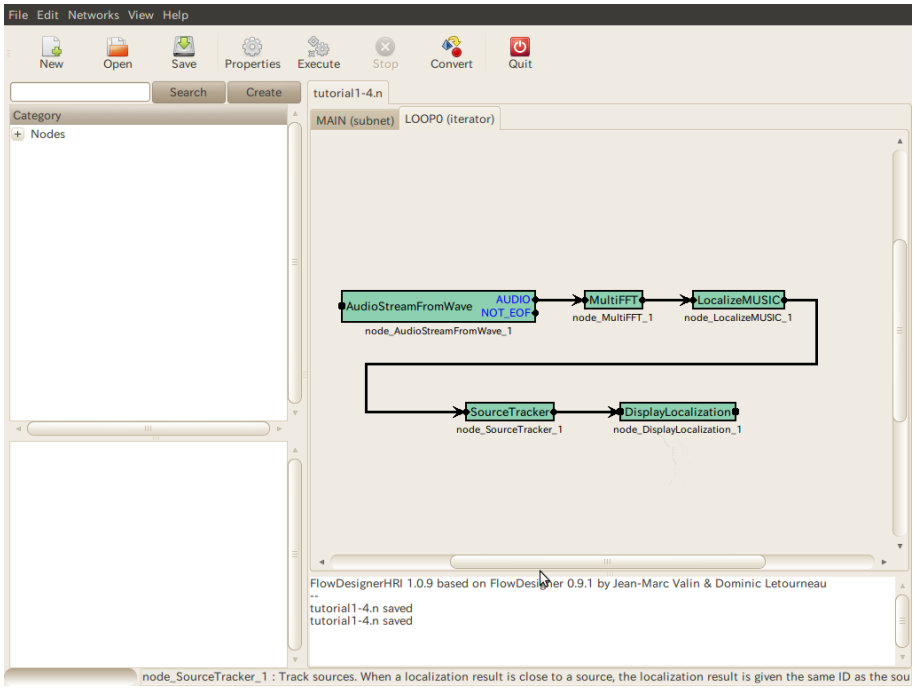


Figure 3.10: Connection of node for sound source localizing part

- **SAMPLING_RATE** : 16000[Hz]
- **A_MATRIX** : Filename of your transfer function database
- **ELEVATION** : 16.7[deg]

- **WINDOW** : 50[frames]
- **PERIOD** : 50[frames]
- **NUM_SOURCE** : 2[sources]
- **MIN_DEG** : -180[deg]
- **MAX_DEG** : 180[deg]
- **LOWER_BOUND_FREQUENCY** : 500[Hz]
- **UPPER_BOUND_FREQUENCY** : 2800[Hz]
- **SPECTRUM_WEIGHT_TYPE** : Uniform
- **A_CHAR_SCALING** : Not displayed
- **MANUAL_WEIGHT_SPLINE** : Not displayed
- **MANUAL_WEIGHT_SQUARE** : Not displayed
- **ENABLE_EIGENVALUE_WEIGHT** : true
- **DEBUG** : false

For the DisplayLocalization node, there is one property for the value of LOG_IS_PROVIDED. To perform logging of the source locations, set `true`. Its data type is `bool` , and the default value is `false`. The user does not need to change these values.

Next, set Iterator. Iterator can be considered as a subroutine of MAIN (subnet) and requires INPUT, OUTPUT and CONDITION. INPUT and OUTPUT can be likened to the input and output of the subroutine. Iterator repeatedly performs processing while it is the subroutine; therefore, the user needs to describe CONDITION, which is the loop's halt condition. Set it as shown in Figure 3.11. To set INPUT, left-click on the input terminal while holding down Shift. Press "OK" and "INPUT" will be displayed on the input terminal in red letters. To set OUTPUT, left-click on the input terminal while holding down Shift. Press "OK," and "OUTPUT" will be displayed on the terminal in blue letters. CONDITION is set by left-clicking on the output terminal while holding down 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 while pressing Shift.

Finally, integrate Iterator and Subnet. Return to the network display of MAIN (subnet) by pressing the MAIN (subnet) tab. Left-click on the grey background and move the mouse cursor onto New Node; a new item named `subnet` should have appeared on the node category name, with an item named LOOP0. Left-click on LOOP0, similar to when creating a node, and a node named LOOP0 with one input and one output will be added. In other words, the subnetwork described in LOOP0 (Iterator) is made with this virtual node. To complete the source localization network, connect InputStream to LOOP0 and set OUTPUT on LOOP0 as shown in Figure 3.12.

3.3.2 FlowDesigner for Windows

This section describes the FlowDesigner for Windows including how to use it and the difference from the Linux version.

Launch the FlowDesigner

You can launch the FlowDesigner in two ways:

```
    double click the  FlowDesigner icon on Desktop
```

or

```
Click Start menu -> Programs -> HARK -> FlowDesigner for HARK
```

You can also run from the command prompt.

```
> flowdesigner
```

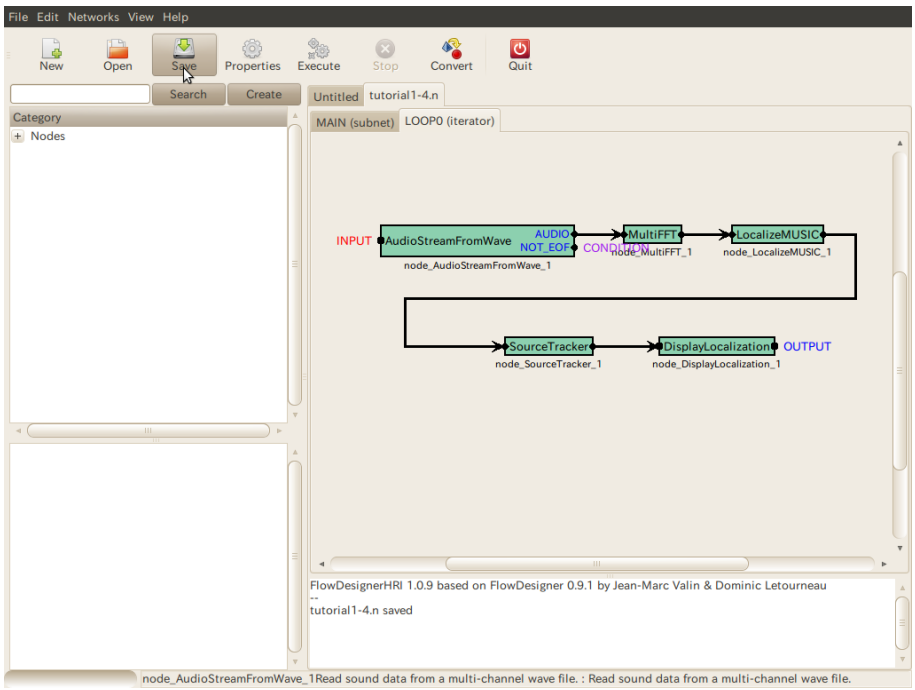


Figure 3.11: Setting of INPUT, OUTPUT and CONDITION on sound source localization node

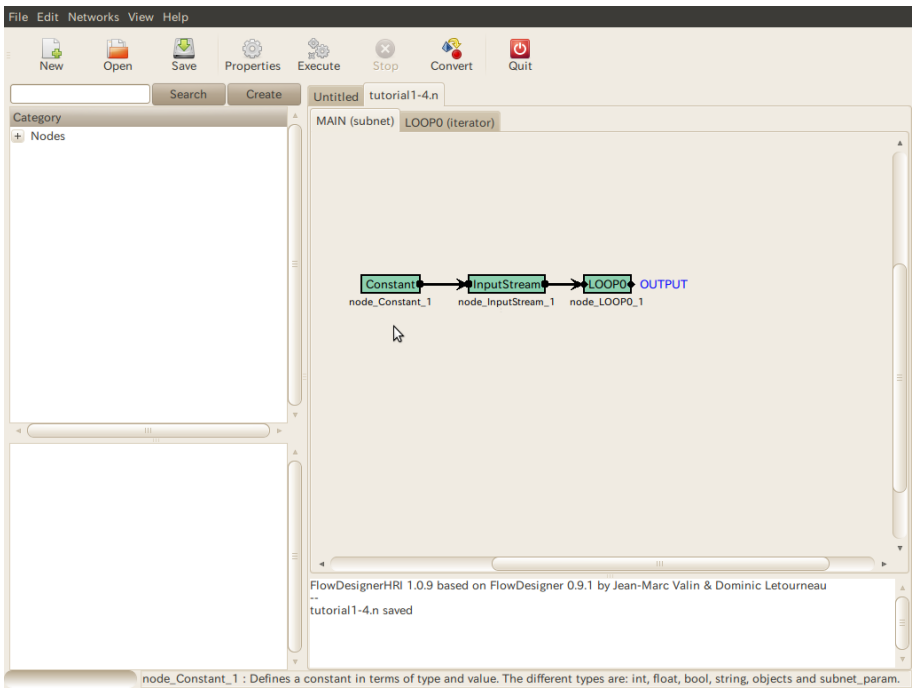


Figure 3.12: Source localization network completed

Manipulating Nodes

- **Add node**
  First, open the tree named *Node* in the *Toolbox* sub-window on the left. Then, drag-and-drop a node.
  For example, if you want to add SpectralGainFilter , Select Nodes → HARK → Separation, and drag-and-drop

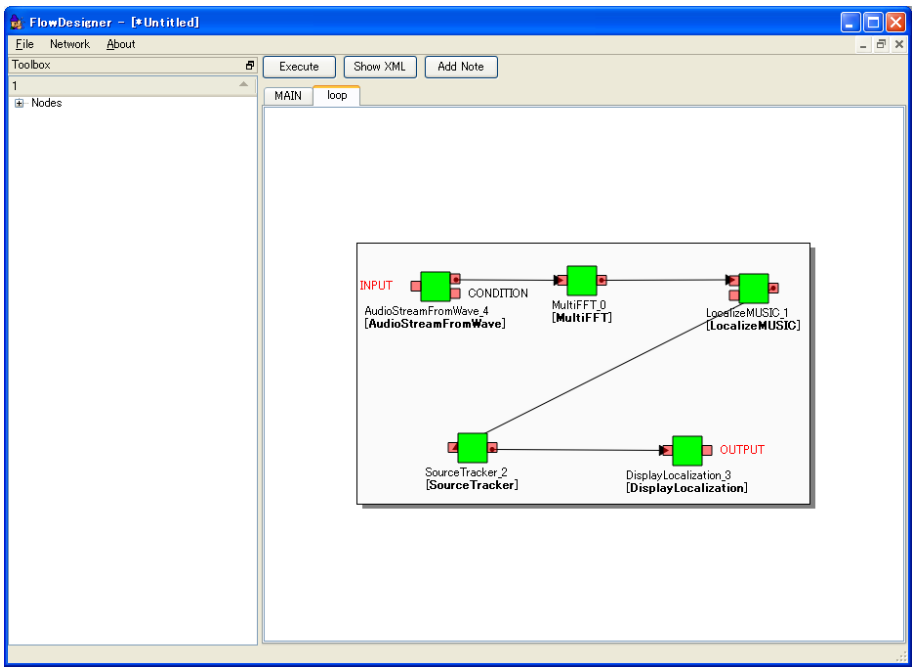


Figure 3.13: Overview of FlowDesigner for Windows

Figure 3.14: The icon of FlowDesigner for HARK

the SpectralGainFilter to the Network.

- **Delete node**
  Click the node, and press the Delete button.
- **Copy node**
  not implemented.

Connecting Terminals

- **Connect terminals**
  The operation is similar to the FlowDesigner for Linux. Drag from the source terminal and drop at the destination terminal, then, these terminals will be connected by an arrow.
- **Disconnect terminal connections**
  Left click the connection, then, press the Delete button.
- **Show terminal names**
  FlowDesigner for Windows does not always show the terminal names. You can see them by putting the mouse cursor on the terminal. During the connection, you can see the terminal name and its type when you put the mouse on the terminal.

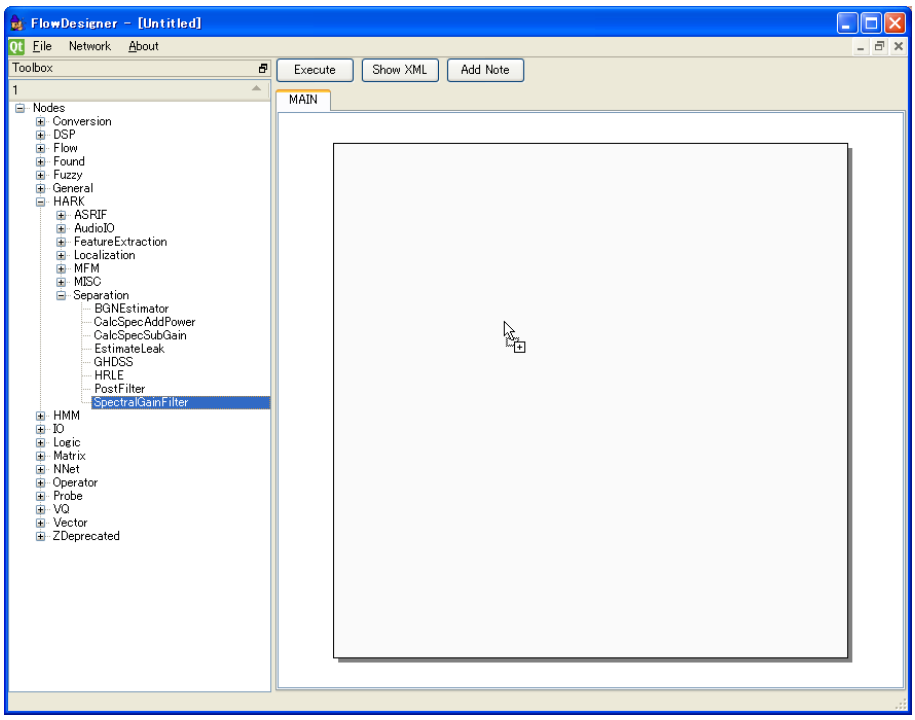


Figure 3.15: Adding the node

- **Not implemented functions**
  FlowDesginer for Windows does not support the function to turn the arrow color to red if the source and destination terminal types are different.

  FlowDesginer for Windows also does not support the function to reorganize the connections.

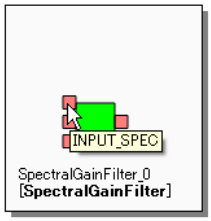


Figure 3.16: Showing the terminal name

Chapter 4

Data Types

This chapter describes the data types used FlowDesigner and HARK . Users need to be aware of HARK data types for the following two cases.

- Setting the properties of a node
- Connecting between nodes (inter-node communication)

Data types used for setting node properties
The data types that can be currently set as a property of a node are the following five types.

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 HARK uses the basic C++ data types for int , float , string and bool , their specifications conform to C++. Data types particular to FlowDesigner are Object and subnet_param . Object is the generic name for data types in FlowDesigner . In HARK , Vector or Matrix are types of Object s that can be set as properties. As described later, since all types except for primitive types are Object s, any class whose values are able to be specified using text-type input can be used as a property. Even a primitive type can be designated as an Object by wrapping it in an Object (e.g. <Int >, for example). subnet_param is a special data type that is used for sharing one parameter with a label between multiple nodes.

Data types used for connections between nodes

Connection of nodes (inter-node communication) is achieved by connecting lines between the terminals of two nodes (indicated as a black point on the left and right of a node) in the GUI of FlowDesigner . Data types used in such cases 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	FlowDesigner object type
Matrix	Matrix type	FlowDesigner object type
Int	Integral type	FlowDesigner object type
Float	Single-precision floating point real type	FlowDesigner object type
String	String type	FlowDesigner object type
Complex	Complex type	FlowDesigner object type
TrueObject	Logical type (true)	FlowDesigner object type
FalseObject	Logical type (false)	FlowDesigner object type
nilObject	object type (nil)	FlowDesigner object type
ObjectRef	Object reference type	FlowDesigner -specific type
Map	Map type	HARK -specific type
Source	Sound source information type	HARK -specific type

The any data type is a discrete data type containing various data types and is defined specifically in FlowDesigner . Since the C++ primitive data types are used for int , float , double , complex<float> , complex<double> , char , string , bool , their specifications conform to C++. When a user attempts to use a primitive type as an Object , it is converted into GenericType<T> automatically and can be treated as a class with the first letter in upper case, as a subclass of Object , such as Int and Float . String and Complex , however, are not GenericType and they are defined as std:string and std:complex, respectively. Likewise, they are used when string and complex are used as an Object type. In this way, a data type defined as a subclass of Object of FlowDesigner are called "FlowDesigner object type". TrueObject , FalseObject and nilObject are also defined as Object s that correspond to true, false and nil, respectively.

The types used most often as FlowDesigner objects are Vector and Matrix . These are FlowDesigner object types that basically conform to the same data types in the C++ STL. ObjectRef is a FlowDesigner -specific data type that is realized as a smart pointer for object types and is often used as elements of Vector , Matrix and Map . Map also conforms to the C++ STL and is a HARK -specific data type, not FlowDesigner . Source is a HARK -specific data type, defined as a sound source information type.

The node's terminal type is displayed at the bottom of the screen in FlowDesigner by focusing the cursor on a terminal of the node for which the user wishes to know the type. Figure 4.1 shows an example of focusing on an AUDIO terminal of a AudioStreamFromMic node. The message "AUDIO (Matrix<float> Windowed wave form). The row indices specify the channel, and the column indices are time" is displayed at the bottom of FlowDesigner , describing that the AUDIO port supports Matrix<float> , Windowed wave forms are output, and that the matrix rows indicate channel and the columns indicates time. Terminals can be successfully connected when the data types of the terminals are the same. This is shown by black arrows when attempting to connect them. When a user connects terminals without meeting this condition, the connection is displayed with red arrows as a warning. The following section describes details of the above for each of the primitive type, FlowDesigner object type, FlowDesigner -specific type and HARK -specific type.

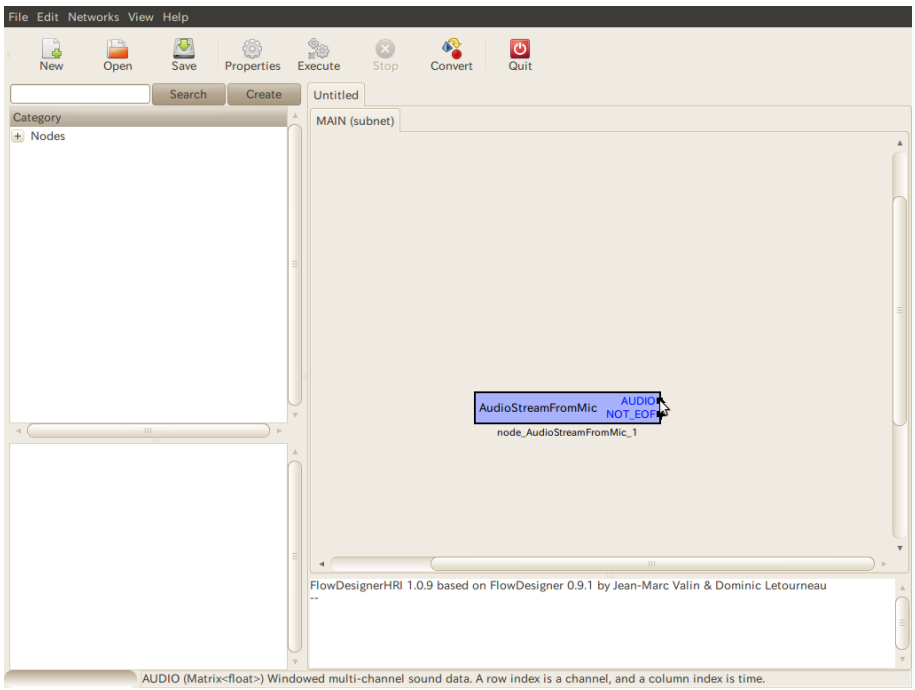


Figure 4.1: Display example of message bar

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. HARK often 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 node 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 FlowDesigner though is used only for `Source` in HARK .

Module converted into this type: The To* node 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 , String and Complex are Object types of int , float , string and complex , respectively. TrueObject and FalseObject is the Object that corresponds to true and false of bool type, nilObject is the Object that corresponds to nil. Descriptions for these are abbreviated. The types that are redefined as the Object type in FlowDesigner in forms conforming to the standard template library (STL) of C++ include Vector and 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 ObjectRef as an element, and Vector< int > and Vector< float > , which have values (int , float) as elements are typically used. It is often used for value pairs, for example to designate a group of angles in ConstantLocalization and to present a group of localization results, which are the output of LocalizeMUSIC . The following are definitions for the Vector type. Base Vector is the type for which methods in FlowDesigner are implemented. As shown below, the Vector type conforms to the vector type of the STL.

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

The ToVect node in the Conversion category accepts inputs such as int and float and outputs a Vector with only one value as an element. Moreover, when the user wishes to use Vector as a parameter of a node, Object should be used as the parameter type, which can be entered in text as shown below. For example, when the user wishes to enter a Vector of int s that has two elements, 3 and 4, as the parameter, character strings are to be entered as shown in Figure 4.2. However, for character strings, note that Vector must come next to the first character < without any spaces.

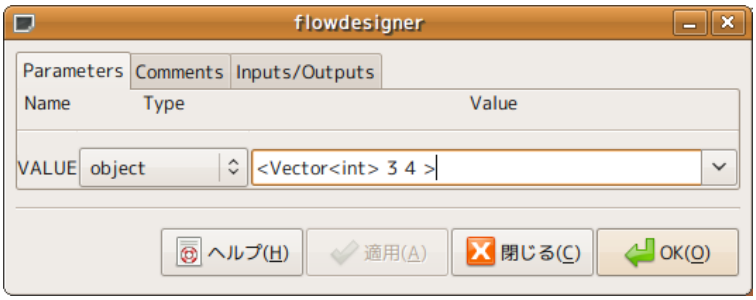


Figure 4.2: Input example of Vector

4.2.2 Matrix

Matrix indicates a matrix. The representative types are Matrix<complex<float> > type and Matrix<float> type, a matrix of complex numbers and a matrix with real numbers as elements, respectively. Matrix type is defined as follows. However, Base Matrix is the class in which FlowDesigner methods are implemented.

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

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

The nodes that use `Matrix` for inter-node communication include MultiFFT (frequency analysis) and LocalizeMUSIC (sound source localization). Further, in a robot audition system with typical functions like sound source location / source tracking / separation and speech recognition for which HARK is used, an ID is given to each sound source in sound source tracking (SourceTracker). `Matrix` is used for inter-node communication in the processing before an ID is given, and `Map` is often used in processing after that.

4.3 Types particular to FlowDesigner

Types particular to FlowDesigner include any , ObjectRef , Object and subnet_param .

4.3.1 any

any is the generic name of various data types. When a terminal of a node is of the any type, the terminal can be connected to any type of terminal without warning (with black line). However, since availability of actual communication depends on implementations in the node, it is recommended to use it as little as possible when attempting to implement nodes yourself. In HARK , its use is limited to the nodes such as MultiFFT , DataLogger , SaveRawPCM and MatrixToMap , which are universally used.

4.3.2 ObjectRef

This data type presents reference of the Object type defined in FlowDesigner to a data type to be succeeded to. Concretely, it is defined as a smart pointer for the Object type. Since all of the FlowDesigner object type, FlowDesigner -specific type and HARK -specific type are data types that have Object as the parent class, these data types can be referred-to. Primitive data types can also be used since it is, as described above, also eventually converted into NetCType<T> when substituting it for ObjectRef and becomes a subclass of Object .

4.3.3 Object

This is a data type mainly used within node properties. In HARK , it is used in ChannelSelector , for example. Basically, this is a data type used when setting data types other than int , float , string , bool and subnet_param as properties. As described in the Section 4.3.2, it can be used as Object type including the primitive data types and therefore all data types can be designated as Object s, theoretically. However, classes whose values cannot be set using text are not used for properties. This function is implemented for Vector and Matrix , although it is not for Map , and therefore Map cannot be specified as a property at present. Examples of available input types are shown below for reference.

	< Vector< float > 0.0 1.0>	Common input method
For Vector	Complex <float (0,0)>	complex can be entered as Complex
	Vector<complex<float> > (0.0, 1.0)>	Not possible since input of complex is not supporte
	Vector< ObjectRef > Complex <float > (0.0, 1.0)>>	No problem if entered as Complex
	<Int 1>	int can be entered as Int

4.3.4 subnet_param

This is the data type used in node properties. When setting the same parameter as a property in multiple subnet nodes, all values can be updated simultaneously by editing the value under this label in MAIN, when designating subnet_param and setting a common label. For example, consider the case of creating an Iterator network (with name LOOP0, for example) and using nodes which need sampling frequency to be set, such as LocalizeMUSIC and GHDSS . The sampling rate can be set in properties of LocalizeMUSIC and GHDSS within the virtual network LOOP0 by setting the sampling rate value to subnet_param type, and setting the value to the label "SAMPLING_RATE". The SAMPLING_RATE of LocalizeMUSIC and GHDSS are guaranteed to always be same by setting the SAMPLINGRATE property in the higher level MAIN loop to the int type and entering 16000. Moreover, as another usage, setting the property of a node in 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 parameter in question can be set as an argument by clicking the property in FlowDesigner . The value shown in the dialog of this property is used as a default value in batch execution.

4.4 HARK-specific type

The data types that HARK defines originally are Map type and Source type.

4.4.1 Map

The Map type is the data type consisting of a key and an ObjectRef as a set. ObjectRef designates a pointer to any data types that is an Object , such as Matrix , Vector or Source . Since HARK provides a speech recognition function, processing is performed for each utterance. In such cases with processing for each utterance, 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 a Map<int, ObjectRef> with an utterance ID as a key, and a pointer to a Vector< complex > which contains the spectrum of the separated utterance is stored in ObjectRef . MatrixToMap is for connecting such nodes with nodes that perform communication based on Matrix . Matrix is used before sound source tracking processing, and Map is used afterwards.

4.4.2 Source

This is a type that indicates source localization information. In HARK , it is used as the information that ObjectRef of Map<int, ObjectRef> points to during processing of sound source localization to the sound source separation, namely, LocalizeMUSIC (output), SourceTracker (input-output), GHDSS (input). Source type contains the following information.

1. **ID:** int type. ID of sound source
2. **Power:** float type. Power of the direction localized.
3. **Coordinate:** An array that is 3 in length of float type. The Cartesian coordinate on a unit ball corresponding to a source localizing direction.
4. **Duration:** double type. The number of frames before a localized sound source is terminated. When a corresponding sound is not detected, this value decreases. When it becomes 0, the sound source is terminated. This is an internal variable which is only used in SourceTracker .

Problem

Read this when wishing to know about the data type (Map < ·· >) used for inputs and outputs of nodes such as MFCCExtraction or SpeechRecognitionClient .

Solution

The Map type is a type consisting of groups of keys and data corresponding to the keys. For example, when performing three-speaker simultaneous recognition, it is necessary to distinguish the features used for the speech recognition for each speaker. Therefore, the key is the ID that indicates which of the three speakers the feature corresponds to, and what number utterance the utterance is. Speakers / utterances are distinguished by treating the key and data as a set.

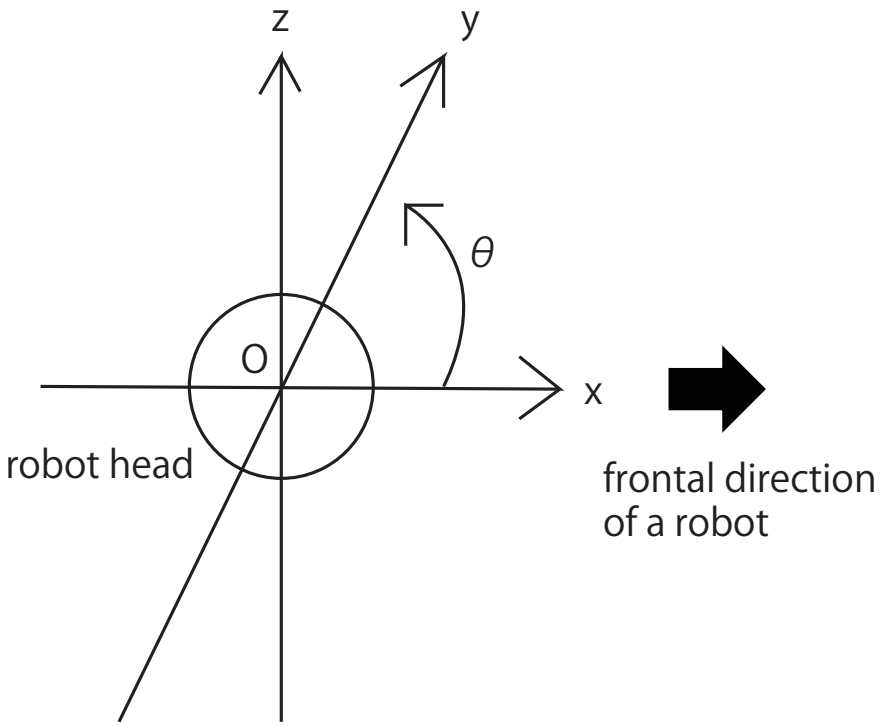


Figure 4.3: HARK standard coordinate system

4.5 HARK standard coordinate system

The center of the coordinate system used in HARK is the origin specified by a user (usually the center of a microphone array). The positive direction of the x axis indicates the front, the positive y axis indicates the left side and the positive z axis indicates the upwards direction. The unit for the coordinate system is meters. Moreover, for angles, 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 the file types used with HARK and their formats. In HARK , a file can be used for module input, module output, or for storing property settings. Table 5.1 gives a list of the file types used in HARK .

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
SaveHTKFeatures	Output	Feature file	HTK format
DataLogger	Output	`Map` data file	`Map` text
CMSave	Property setting	Correlation matrix file	correlation matrix text
CMLoad	Output	Correlation matrix file	correlation matrix 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

Here, Julius formats basically conform to the file formats of Julius . For differences from the original Julius , see the description of JuliusMFT . File formats other than Julius are described below. The input-output of HGTF (HARK General Transfer Function), binary format and HARK text format are collectively supported by the input-output library harkio of HARK and their generic name is HARK file format. Other formats, including srcinf text, PCM binary, `float` binary, localization result text, `Map` text, and correlation matrix text formats will be modified in future versions of HARK so that they can be supported by 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 for LocalizeMUSIC source localization transfer function files, GHDSS sound source separation transfer function files and separation matrix files. Its data structure is shown in Table 5.2. Versions of the form type

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)	

”HGTF” are 3 types as follows.

- M2PG 1.0
- GTF 1.0
- SM 1.0

M2PG (MUSIC 2D Polar Grid) is used for source localization transfer function files for the LocalizeMUSIC module. GTF (GHDSS Transfer Function) and SM (Separation `Matrix`) are used for the sound source separation transfer function files for GHDSS and separation matrix files for GHDSS , respectively. fmt is the basic sub chunk used to enter fundamental data. Its details are shown in Table 5.3. Further, the data chunk size is the total size minus the 4 bytes of chunk name and the 4 bytes of chunk size. The M2PG/GTF/SM sub chunk is a chunk that stores the data of

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]

individual files (source localization transfer function 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 the formats of SPEC. The null terminator is added after each item, the character code is ASCII (in general, only single byte characters such as alphanumerics are used) and the linefeed code is LF.

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 the A_MATRIX property of LocalizeMUSIC . For details of usage, see LocalizeMUSIC . Table 5.5 shows definitions of M2PG. Further, the rows in memory of a multi-dimensional array are row major (same as in the C language). This file is created from a file in the impulse response format using harktool , which is a support tool. Details of the creation method are described in Section 7.1.

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)

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

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 in HGTF format with GTF sub chunks. The sound source separation transfer function file is designated in TF_CONJ_FILENAME when the property TF_CONJ of GHDSS is `true`. For details of usage, see GHDSS . Table 5.1.2 shows the definition of the GTF sub chunk. The rows in memory of a multi-dimensional array are row major (same as in the C language). This file is

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

created from a file in the impulse response format or from a microphone position and source location list file using harktool, which is a support tool. Details of the creation method are described in Section 7.1.

5.1.3 Separation matrix for GHDSS

The separation matrix is expressed as a file in HGTF format with SM sub chunks. The sound source separation transfer function file is designated in the property INITW_FILENAME of GHDSS . Moreover, a separation matrix file is output by designating EXPORT_W_FILENAME, which appears when EXPORT_W is set to `true`. For details of usage, see GHDSS . Table 5.6 shows the definition of SM sub chunks. The rows in memory of a multi-dimensional

array are 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 HARK text format is a text file that supports microphone position information and noise position information. Headers with file identifiers and version numbers are to be included in each of them.

Description format
until hark1.1 is TEXT format
since hark1.2 is XML format.

5.2.1 Microphone position text format

The microphone position text format is a file that describes each microphone position of the microphone array used in GHDSS and harktool. When selecting CALC in the TF_CONJ property of GHDSS node, the MIC_FILENAME property appears. Give the coordinates of each microphone in the microphone array (saved in the microphone position format) into this column to generate an impulse response and perform sound source separation in simulation. The format is as follows.

The description of the tag

Table 5.7: microphone position Tag

tag name	Description
hark_config	file type id : ”MicArrayLocation” fix major: Major number (2) minor: Minor number (0)
Comment	Comment
Mic size	number of Microphone
MicIndex	Mic number
	[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 $\sqrt{x^2y^2z^2}$ becomes 1.0. unit: meter type: cartesian
X	sound source position: X axis (Refer to Position tag)
Y	sound source position: Y axis (Refer to Position tag)
Z	sound source position: Z axis (Refer to Position tag)

Example: microphone position file

example xml file: microphone array of 4 channels

```
<hark_xml>
  <hark_config id="MicArrayLocation" major="2" minor="0">
    <Comment>created by harktool4</Comment>
    <Mics size="4">
      <Mic>
        <MicIndex>0</MicIndex>
        <Position unit="meter" type="cartesian">
          <X>-0.957823</X>
          <Y>8.37355e-08</Y>
          <Z>0.287361</Z>
        </Position>
      </Mic>
      <Mic>
        <MicIndex>1</MicIndex>
        <Position unit="meter" type="cartesian">
          <X>1.14219e-08</X>
          <Y>-0.957823</Y>
          <Z>0.287361</Z>
        </Position>
      </Mic>
      <Mic>
        <MicIndex>2</MicIndex>
        <Position unit="meter" type="cartesian">
          <X>0.957823</X>
          <Y>0</Y>
          <Z>0.287361</Z>
        </Position>
      </Mic>
      <Mic>
        <MicIndex>3</MicIndex>
        <Position unit="meter" type="cartesian">
          <X>-4.18678e-08</X>
          <Y>0.957823</Y>
          <Z>0.287361</Z>
        </Position>
      </Mic>
    </Mics>
  </hark_config>
</hark_xml>
```

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 . By selecting `true`in the FIXED_NOISE property of GHDSS node, the FIXED_NOISE_FILENAME property appears. Giving this column a sound source coordinate in the noise source format, process a noise of the direction as known noise source.

The description of the tag

Table 5.8: Noise position tag

tag name	Description
hark_config	File Type id : ”NoiseLocation” fix major: Major number (2) minor: Minor number (0)
Comment	Comment
NoiseSources size	”1” fix
NoiseIndex	”0” fix
Position	Coordinate position of sound source position [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 $\sqrt{x^2y^2z^2}$ becomes 1.0. unit: meter type: cartesian
X	sound source position: X axis (Refer to Position tag)
Y	sound source position: Y axis (Refer to Position tag)
Z	sound source position: Z axis (Refer to Position tag)

Example: Noise position

```
<hark_xml>
<hark_config id="NoiseLocation" major="2" minor="0">
  <Comment>created by harktool4</Comment>
  <NoiseSources size="1">
    <NoiseSource>
      <NoiseIndex>0</NoiseIndex>
      <Position unit="meter" type="cartesian">
        <X>-4.37114e-08</X>
        <Y>-3.82137e-15</Y>
        <Z>1</Z>
      </Position>
      </NoiseSource>
    </NoiseSources>
  </hark_config>
</hark_xml>
```

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 described 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 A_MATRIX of LocalizeMUSIC) and a sound source separation transfer function file (set in TF_CONJ_FILE of GHDSS).

Description format
until hark1.1 is TEXT format
since hark1.2 is XML format.

The description of the tag

Table 5.9: sound source position Tag1

tag name	Description
hark_config	File Type id : "ImpulseResponseList" or "TSPList" fix major: Major number (2) minor: Minor number (0)
NumMic	number of microphones
SourceFileType	"float" or "wav"
Sources size	number of elements
Comment	Comment
Position	Coordinate position of sound source position [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 $\sqrt{x^2y^2z^2}$ becomes 1.0. unit: meter type: cartesian
X	sound source position: X axis (Refer to Position tag)
Y	sound source position: Y axis (Refer to Position tag)
Z	sound source position: Z axis (Refer to Position tag)
Files size	number of Files size

Table 5.10: sound source position Tag2

tag name	Description
Name	file name impulse response file name or TSP response file name **impulse response**: example_D**ddd**_E**eee**_R**rrr**_ch**cc**.flt (“example” is Any **ddd** is Azimuth **eee** is Elevation,**rrr** is Radius,**cc** is Channel number) In addition, flt is little-endian 32bit real number `float` Format file Example: Azimuth:180degree Elevation:16.7degree Radius:0.1m /path/example_D180_E16.7_R100_ch00.flt **TSP response:** example_D**ddd**_E**eee**_R**rrr**_ch**cc**.wav (“example” is Any **ddd** is Azimuth **eee** is Elevation,**rrr** is Radius,**cc** is Channel number) Each file can use only PCM wave format of RIFF form. Example: Azimuth:180degree Elevation:16.7degree Radius:0.1m /path/example_D180_E16.7_R100_ch00.wav
MicIndices size	number of mic
MicIndex	Mic number
CutStart	cut-start index in float vector 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.
CutEnd	cut-end index in float vector Data is read until the sample number specified by **End Offset**. For example, when **Start offset** and **End offset** are set to 100 and 200, respectivly, 100th to 200th samples are read in the file named **File name**. The value should be a positive integer.
SynchronousAverge	number of Synchronous Averge
OriginalImpulseFile	Original Impulse File
TSPOffset	offset of tsp signal in samples
TSPLength	length of one tsp in sample
SignalMax	maximum Amplitude of tsp signal
NoiseSources size	number of elements
NoiseIndex	number of Microphone

Example:ImpulseResponseList file

```
<hark_xml>
  <hark_config id="ImpulseResponseList" major="2" minor="0">
    <NumMic>8</NumMic>
    <SourceFileType>float</SourceFileType>
    <Comment>created by harktool4</Comment>
    <Sources size="72">
      <Source>
        <Position unit="meter" type="cartesian">
          <X>-0.957823</X>
          <Y>8.37355e-08</Y>
          <Z>0.287361</Z>
        </Position>
        <Files size="8">
          <File>
            <Name>/YOUR_PATH/D-180_E16.7_R100_ch00.flt</Name>
            <MicIndices size="1">
              <MicIndex>0</MicIndex>
            </MicIndices>
          </File>
          <File>
            <Name>/YOUR_PATH/D-180_E16.7_R100_ch01.flt</Name>
            <MicIndices size="1">
              <MicIndex>1</MicIndex>
            </MicIndices>
          </File>
            .
            .
        </Files>
        <CutStart>1</CutStart>
        <CutEnd>0</CutEnd>
      </Source>
            .
            .
    </Sources>
  </hark_config>
</hark_xml>
```

Example: TSP List file

```
  <hark_config id="TSPList" major="2" minor="0">
      <SynchronousAverage>16</SynchronousAverage>
      <OriginalImpulseFile type="float">
        /usr/bin/harktool4_utils/16384.little_endian.tsp
      </OriginalImpulseFile>
      <TSPOffset>16384</TSPOffset>
      <TSPLength>16384</TSPLength>
      <SignalMax>32768</SignalMax>
      <NumMic>8</NumMic>
      <SourceFileType>float</SourceFileType>
      <Comment>created by harktool4</Comment>
      <Sources size="72">
          <Source>
              <Position unit="meter" type="cartesian">
                <X>-0.957823</X>
                <Y>8.37355e-08</Y>
                <Z>0.287361</Z>
              </Position>
              <Files size="1">
                <File>
                  <Name>/YOUR_PATH/D180_E16.7_R100.wav</Name>
                    <MicIndices size="8">
                      <MicIndex>0</MicIndex>
                      <MicIndex>1</MicIndex>
                      <MicIndex>2</MicIndex>
                      <MicIndex>3</MicIndex>
                      <MicIndex>4</MicIndex>
                      <MicIndex>5</MicIndex>
                      <MicIndex>6</MicIndex>
                      <MicIndex>7</MicIndex>
                    </MicIndices>
                </File>
              </Files>
              <CutStart>1</CutStart>
            <CutEnd>0</CutEnd>
          </Source>
            .
            .
      </Sources>
    </hark_config>
</hark_xml>
```

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 in little endian and additional information such as headers is not used. In HARK , SaveRawPCM outputs in this format and the standard extension is .sw. Acoustic waveforms with multiple 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, track numbers and quantization bit rate must be designated.

5.3.3 `float` binary

This is a file that stores real 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 real-valued. They are output in SaveFeatures and 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 HARK are power spectra, MFCC, MSLS and Missing Feature Mask. Note that, the header information for the features is not saved in SaveFeatures , so the number of dimensions and vectors are not recorded. HARK supports SaveHTKFeatures which records HTK(The Hidden Markov Model Toolkit)-formatted header information to the features. Therefore, you can use either SaveFeatures or SaveHTKFeatures depending on your purpose.

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] = 32 [bit] */
if( ret != 4){break;}
printf("\%f\n", data);}
close(fd);return (0);}
```

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 LoadSourceLocation and SaveSource-Location nodes. Designating a file saved in this format in FILENAME of the property enables reading and writing. 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 sound source direction.

5.3.5 Map text

This is the 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 >`). The 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.

5.3.6 Correlation matrix text for sound source localization

This is a file format for saving/loading the correlation matrix of a multi-channel signal in the frequency domain. The correlation matrix is utilized in LocalizeMUSIC to whiten (suppress) pre-measured noise from sound source localization. Currently, CMSave and CMLoad use this file format for input/output correlation matrices.

Let M and N denote the number of microphones and frequency bins, respectively. A correlation matrix is an M-th order square complex matrix, and the matrix is calculated for each frequency bin. Let $\boldsymbol{R}(n)$ represent the correlation matrix of n-th frequency bin ($1 \leq n \leq N$), described as follows:

$$\boldsymbol{R}(n) = \begin{bmatrix} r_{11}(n) & \cdots & r_{1M}(n) \\ \vdots & \ddots & \vdots \\ r_{M1}(n) & \cdots & r_{MM}(n) \end{bmatrix}, \tag{5.1}$$

where $r_{11}(n), \cdots, r_{MM}(n)$ are complex numbers.

Then, the correlation matrix of all N frequency bins are saved in the following format by separating the real part and imaginary part of it.

Correlation Matrix File (Real part)

$$\begin{array}{cccccccc} \mathrm{Re}[r_{11}(1)] & \mathrm{Re}[r_{12}(1)] & \cdots & \mathrm{Re}[r_{1M}(1)] & \mathrm{Re}[r_{21}(1)] & \mathrm{Re}[r_{22}(1)] & \cdots & \mathrm{Re}[r_{MM}(1)] \\ \mathrm{Re}[r_{11}(2)] & \mathrm{Re}[r_{12}(2)] & \cdots & \mathrm{Re}[r_{1M}(2)] & \mathrm{Re}[r_{21}(2)] & \mathrm{Re}[r_{22}(2)] & \cdots & \mathrm{Re}[r_{MM}(2)] \\ \mathrm{Re}[r_{11}(N)] & \mathrm{Re}[r_{12}(N)] & \cdots & \mathrm{Re}[r_{1M}(N)] & \mathrm{Re}[r_{21}(N)] & \mathrm{Re}[r_{22}(N)] & \cdots & \mathrm{Re}[r_{MM}(N)] \end{array}$$

Correlation Matrix File (Imaginary part)

$$\begin{array}{cccccccc} \mathrm{Im}[r_{11}(1)] & \mathrm{Im}[r_{12}(1)] & \cdots & \mathrm{Im}[r_{1M}(1)] & \mathrm{Im}[r_{21}(1)] & \mathrm{Im}[r_{22}(1)] & \cdots & \mathrm{Im}[r_{MM}(1)] \\ \mathrm{Im}[r_{11}(2)] & \mathrm{Im}[r_{12}(2)] & \cdots & \mathrm{Im}[r_{1M}(2)] & \mathrm{Im}[r_{21}(2)] & \mathrm{Im}[r_{22}(2)] & \cdots & \mathrm{Im}[r_{MM}(2)] \\ \mathrm{Im}[r_{11}(N)] & \mathrm{Im}[r_{12}(N)] & \cdots & \mathrm{Im}[r_{1M}(N)] & \mathrm{Im}[r_{21}(N)] & \mathrm{Im}[r_{22}(N)] & \cdots & \mathrm{Im}[r_{MM}(N)] \end{array}$$

The file is saved as a space-separated values (ssv) file. The file has N rows and $M \times M$ columns. CMLoad recognizes the file as $\boldsymbol{R}(n)$.

Chapter 6

Node References

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

How to read data type

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

How to read file format

1. **Outline:** For what the format is used is described.
2. **Nodes that use files in this format:** For what nodes 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 node reference

1. **Outline of the node:** What function the node 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 node 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 nodes should be used. When the user somehow wishes to use the node concerned, it is recommended to just to try the example.
4. **Input-output and property of node:** Types and meaning of the input and output terminals of nodes 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 node:** Detail descriptions including theoretical background and implementation methods of nodes are described. When the user wishes to know about the node 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
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 node

This node takes in multichannel speech waveform data from a microphone array. The audio interface devices supported by this node are the RASP series manufactured by System In Frontier, Inc., TD-BD-16ADUSB manufactured by Tokyo Electron Device, and ALSA-based devices (e.g. The RME Hammerfall DSP Multiface series). Furthermore, this module can receive IEEE-float-formatted multi-channel raw audio stream through a TCP/IP socket connection. For an introduction to various devices, see Section 8.

Necessary file

No files are required.

How to use

When to use

This node is used when wishing to use speech waveform data from a microphone array as input to the HARK system.

Typical connection

Figure 6.1 shows an example usage of the AudioStreamFromMic node.

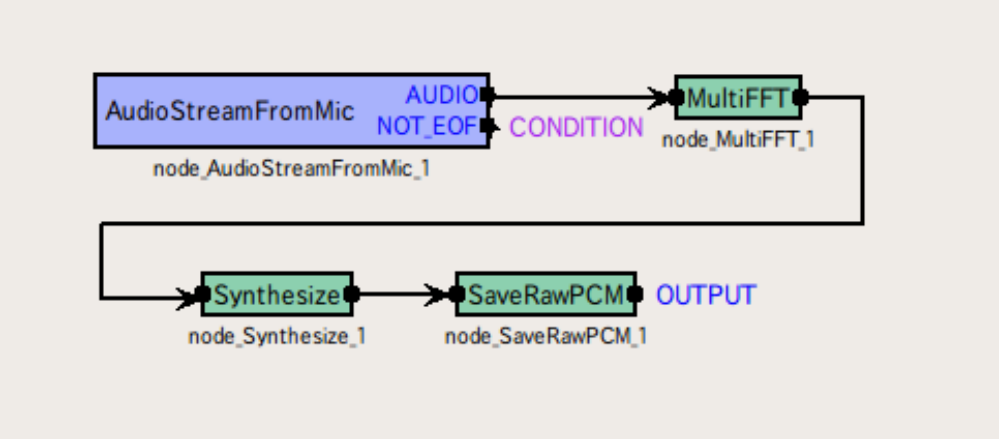


Figure 6.1: Connection example of AudioStreamFromMic

Overview of device

Among the devices that the AudioStreamFromMic node supports, the following are introduced with photos.

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

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

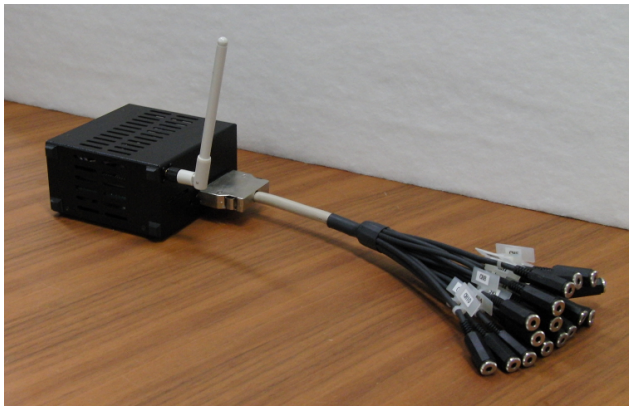


Figure 6.2: Wireless RASP

2. RME Hammerfall DSP Multiface series Figures 6.3 and 6.4 show the appearance of the RME Hammerfall DSP series Multiface. The device communicates with a host PC through a 32bit CardBus. Although a microphone can be connected to the device through a 6.3 mm TRS terminal, a microphone amplifier is used to ensure 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 a 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.

Input-output and property of node

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.
CHANNEL_COUNT	`int`	8	[ch]	Microphone input channel number of a device to use.
SAMPLING_RATE	`int`	16000	[Hz]	Sampling frequency of audio waveform data loaded.
DEVICETYPE	`string`	WS		Type of device to be used.
GAIN	`string`	0dB		Gain value used with RASP device.
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. Indexed, multichannel audio waveform data with rows as channels and columns as samples. Size of the column is equal to the parameter LENGTH.

NOT_EOF : `bool` type. This indicates whether there is still input from the waveform to be processed. Used as an ending flag when processing the waveforms in a loop. When it is `true`, waveforms are loaded, and when it is `false`, reading is complete. `true` is output continuously.

Figure 6.3: Front view of RME Hammerfall DSP Multiface

Figure 6.4: Back view of RME Hammerfall DSP Multiface

Parameter

LENGTH : `int` type. The default value is 512. Designates the frame length, which is a base unit of processing, in terms of number of samples. The higher the value, the higher the frequency resolution, but the lower the temporal resolution. It is known that length corresponding to 20 $\sim$ 40 [ms] is appropriate for the analysis of audio waveforms. The default value of 32 [ms] corresponds to the sampling frequency 16,000 [Hz].

ADVANCE : `int` type. The default value is 160. Designates the frame shift length in samples. The default value of frame frequency of 10 [ms] corresponds to the sampling frequency 16,000 [Hz].

CHANNEL_COUNT : `int` type. The number of channels of the device to be used.

SAMPLING_RATE : `int` type. The default value is 16000. Designates the sampling frequency – how often to sample per second – of the loaded waveforms. When frequencies up to ω [Hz] are needed for processing, set the sampling frequency to over 2ω [Hz]. When the sampling frequency is high, data generally increases and it makes it difficult to perform real-time processing.

DEVICETYPE : `string` type. Select from ALSA, RASP, WS, TDBD16ADUSB, RASP24-16, RASP24-32, RASP-LC. When a device supporting ALSA-based drivers is used, select ALSA. When RASP-2 is used, select RASP. When wireless RASP is used, select WS. When TD-BD-16ADUSB is used, select TDBD16ADUSB. When RASP-24 is used with the 16bit quantization bit rate, select RASP24-16. When RASP-24 is used with the 24bit quantization bit rate, select RASP24-24. When RASP-LC is used with the wireless connection to your PC, select RASP-LC. (If RASP-LC is directly connected to your PC, select ALSA.) When you want to receive IEEE-float-formatted raw audio stream via a TCP/IP socket connection, select NETWORK.

GAIN : `string` type. The default value is 0dB. This sets the microphone gain for the recording. Select from 0dB, 12dB, 24dB, 36dB, 48dB. This parameter is activated when RASP-24 is used.

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

Details of the node

HARK supports three audio devices as follows:

1. The following RASP series manufactured by System In Frontier, Inc.
   - RASP-2
   - Wireless RASP
   - RASP-24
   - RASP-LC
2. TD-BD-16ADUSB manufactured by Tokyo Electron Device Co., Ltd.
3. ALSA-based devices. The following devices are the examples.
   - Kinect Xbox (manufactured by Microsoft)
   - PlayStation Eye (manufactured by Sony)
   - Microcone (manufactured by Dev-Audio)
   - RME Hammerfall DSP series Multiface
4. Raw audio stream via TCP/IP socket connection (IEEE float wav format)

The following are parameter settings for each device.

RASP series:

- Parameter settings for using **RASP-2**

CHANNEL_COUNT	8
DEVICETYPE	WS
DEVICE	IP address of **RASP-2**

- Parameter settings for using **Wireless RASP**

CHANNEL_COUNT	16
DEVICETYPE	WS
DEVICE	IP address of **Wireless 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, ChannelSelector node needs to be connected to the AUDIO output of AudioStreamFromMic node and only the microphone input channel has to be selected.

- Parameter settings for using **RASP-24**

CHANNEL_COUNT	Miltiples of 9
DEVICETYPE	RASP24-16 or RASP24-32
DEVICE	IP address of **RASP-24**
Remarks	Set DEVICETYPE=RASP24-16 for the recording with the 16bit quantization bit rate. Set DEVICETYPE=RASP24-32 for the recording with the 24bit quantization bit rate. CHANNEL_COUNT should be the multiples of 9. The channels from 0th channel to 7th channel are microphone channels. The 8th channel is a line input. For microphone array processing, ChannelSelector node needs to be connected to the AUDIO output of AudioStreamFromMic node and only the microphone input channel has to be selected.

- Parameter settings for using **RASP-LC**

CHANNEL_COUNT	8
DEVICETYPE	ALSA or RASP-LC
DEVICE	If DEVICETYPE=ALSA, DEVICE parameter should be plughw:a,b. Please refer "Device corresponding to ALSA" for the detail of the parameter setting. If DEVICETYPE=RASP-LC,DEVICE parameter should be the IP address of **RASP-LC**.
Remarks	If the RASP-LC is connected directly to the USB interface of the PC, set DEVICETYPE=ALSA. If the RASP-LC is connected to the PC through the wireless LAN, set DEVICETYPE=RASP-LC. All the channels are microphone channels.

Devices manufactured by Tokyo Electron Device LTD.:

- Parameter settings for using **TD-BD-16ADUSB**

CHANNEL_COUNT	16
DEVICETYPE	TDBD16ADUSB
DEVICE	TDBD16ADUSB

Device corresponding to ALSA:

To use ALSA devices, designate plughw:a,b as the DEVICE parameter. Enter positive integers to a and b. Enter the card number indicated in arecord -l to a. When multiple audio input devices are connected, multiple card numbers are indicated. Enter card number to be used. Enter the subdevice number indicated in arecord -l to b. For a device that has multiple subdevices, enter the number of the subdevice to be used. Devices that have analog input and digital inputs are one of the examples of multiple subdevices.

- Parameter settings for using **Kinect Xbox**

CHANNEL_COUNT	4
DEVICETYPE	ALSA
DEVICE	plughw:a,b

- Parameter settings for using **PlayStation Eye**

CHANNEL_COUNT	4
DEVICETYPE	ALSA
DEVICE	plughw:a,b

- Parameter settings for using **Microcone**

CHANNEL_COUNT	7
DEVICETYPE	ALSA
DEVICE	plughw:a,b

- Parameter settings for using **RME Hammerfall DSP Multiface series**

CHANNEL_COUNT	8
DEVICETYPE	ALSA
DEVICE	plughw:a,b

Socket Connection (if DEVICETYPE=NETWORK is selected):

DEVICE should be the IP address of the machine that sends the audio stream. Othre parameters should be set depending on the setting of audio stream. If the audio has M channels and can be obtained T samples at once, you can send the audio strem by the program like the following pseudo code.

```
WHILE(1){
    X = Get_Audio_Stream (Suppose X is a T-by-M matrix.)
    FOR t = 1 to T
        FOR m = 1 to M
            DATA[M * t + m] = X[t][m]
        ENDFOR
    ENDFOR
    send(soket_id, (char*)DATA, M * T * sizeof(float), 0)
}
```

Here, X is IEEE-float-formated audio stream. Therefore, $-1 \leq X \leq 1$.

Device corresponding to DirectSound and ASIO on Windows OS:

This instruction is only for Windows OS users.

When using DirectSound- or ASIO-Devices, designate the device name as the DEVICE parameter.

You can check the device name by the Device Manager on Windows Control Panel. You can also check the name by "Sound Device List" found in [Start] → [Programs] → [HARK]. The Sound Device List lists up all the name of sound devices connected to your PC like Figure 6.5.

For example, if you want to use "ASIO Hammerfall DSP" displayed in Sound Device List, you should designate "ASIO" and "ASIO Hammerfall DSP" as the DEVICETYPE and DEVICE parameters, respectively. The DEVICE can be partially matched to "ASIO Hammerfall DSP", for instance "Hammerfall". Notice that the DEVICE does not support multi-byte characters.

Kinect Xbox, PlayStation Eye, and Microcone have the special DEVICE names described as follows.

- Parameter settings for using **Kinect Xbox** on Windows OS

CHANNEL_COUNT	4
DEVICETYPE	DS
DEVICE	kinect

- Parameter settings for using **PlayStation Eye** on Windows OS

CHANNEL_COUNT	4
DEVICETYPE	DS
DEVICE	pseye

- Parameter settings for using **Microcone** on Windows OS

CHANNEL_COUNT	7
DEVICETYPE	DS
DEVICE	microcone

- Parameter settings for using **RME Hammerfall DSP Multiface series** on Windows OS

CHANNEL_COUNT	8
DEVICETYPE	ASIO
DEVICE	ASIO Hammerfall DSP

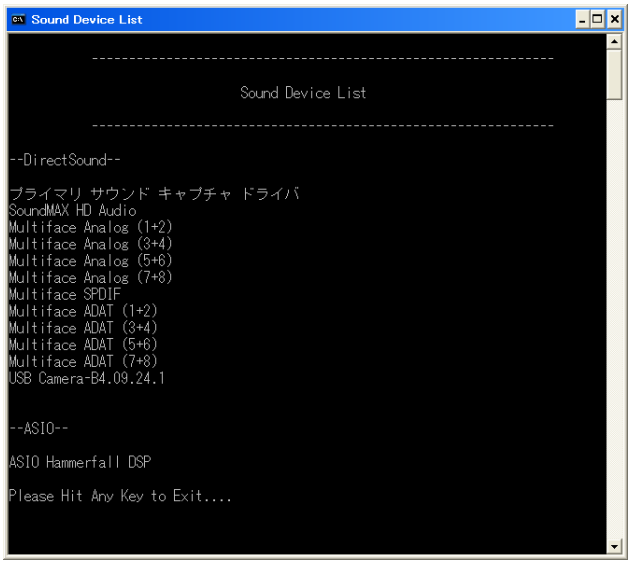


Figure 6.5: Confirmation of the device name

6.1.2 AudioStreamFromWave

Outline of the node

This node reads speech waveform data from a WAVE file. The waveform data is read into a `Matrix<float>` type: indexed, multichannel audio waveform data with rows as channels and columns as samples.

Necessary file

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

Usage

When to use

This node is used when wishing to use WAVE files as input to the HARK system

Typical connection

Figures 6.6 and 6.7 show an usage example of the AudioStreamFromWave node. Figure 6.6 shows an example of AudioStreamFromWave converting `Matrix<float>` type multichannel waveforms read from a file into frequency domain with the MultiFFT node. To read a file with AudioStreamFromWave , designate a filename in Constant node (Normal node FlowDesigner) and generate a file descriptor in the InputStream node as shown in Figure 6.7. Further, connect the output of the InputStream node to the iterator subnetwork(LOAD_WAVE in Figure 6.7), which contains networks of various nodes of HARK such as AudioStreamFromWave .

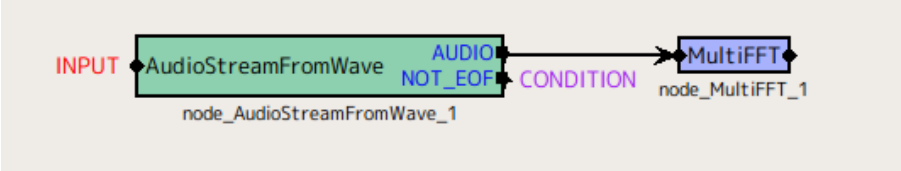


Figure 6.6: Connection example of AudioStreamFromWave 1

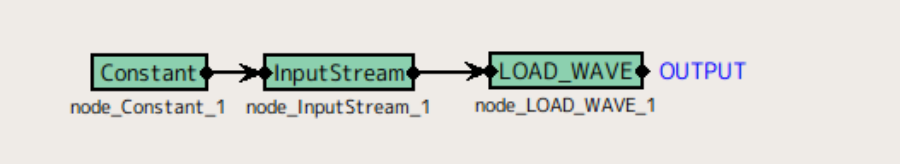


Figure 6.7: Connection example of AudioStreamFromWave 2

Input-output and property of node

Input

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

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.
USE_WAIT	bool	false		Designate if processing is performed in real time

AUDIO : Matrix<float> type. Indexed, multichannel audio waveform data with rows as channels and columns as samples. The number of columns is equal to the parameter LENGTH.

NOT_EOF : bool type. Indicate if the file can still be read. Used as an ending flag for loop processing of files. When reaching the end of file, its outputs false and outputs true in other cases.

Parameter

LENGTH : int type. The default value is 512. Designates the frame length, which is a base unit of processing, in terms of number of samples. The higher the value, the higher the frequency resolution, but the lower the temporal resolution. It is known that length corresponding to 20 ~ 40 [ms] is appropriate for the analysis of audio waveforms. The default value of 32 [ms] corresponds to the sampling frequency 16,000 [Hz].

ADVANCE : int type. The default value is 160. Designates the frame shift length in samples. The default value of 10 [ms] corresponds to the sampling frequency 16,000 [Hz].

USE_WAIT : bool type. The default value is false. Usually, acoustic processing of the HARK system proceeds faster than real time. This option can be used to add "wait time" 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 of real time.

Details of the node

Applicable file format: RIFF WAVE files can be read. The number of channels and quantization bit rate are read from headers of files. 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 processing, they should be set as parameters required by nodes (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: When the goal of processing is speech analysis (speech recognition), about 20 ~ 40 [ms] would be appropriate for LENGTH and 1/3 ~ 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 node

This node saves speech waveform data in the time domain as files. The outputted binary files are Raw PCM sound data, where sample points are recorded as 16 [bit] or 24 [bit] integer numbers. Depending on the input data type, a multichannel audio file, or multiple monaural audio files (one for each separated sound) are output.

Necessary file

No files are required.

Usage

When to use

This node is used when wishing to convert separated sound into waveforms with the Synthesize node to confirm a sound, or when wishing to record the sound from a microphone array by connecting it with the AudioStreamFrom-Mic node.

Typical connection

Figures 6.8 and 6.9 show a usage example of SaveRawPCM . Figure 6.8 shows an example of saving multichannel acoustic signals from AudioStreamFromMic into a file using the SaveRawPCM node. As shown in this example, select a channel to save to a file using the ChannelSelector node. Note that since SaveRawPCM accepts `Map<int, ObjectRef>` type inputs, the MatrixToMap node is used to convert from the `Matrix<float>` type into the `Map<int, ObjectRef>` type. Figure 6.9 shows an example for saving a separated sound using the SaveRawPCM node. Since the separated sound output from the GHDSS node or the PostFilter node, which suppresses noise after separation, is in the frequency domain, it is converted into a waveform in the time domain using the Synthesize node before it is input into the SaveRawPCM node. The WhiteNoiseAdder node is usually used for improving the speech recognition rate of the separated sound and is not essential for the use of SaveRawPCM .

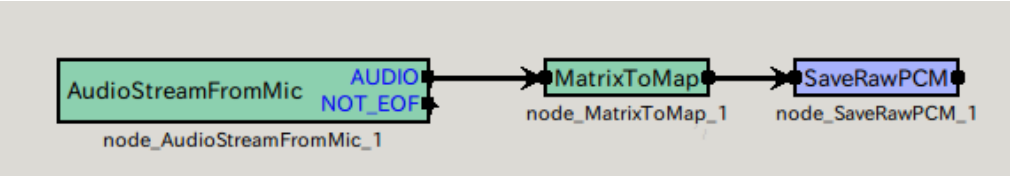


Figure 6.8: Connection example of SaveRawPCM 1

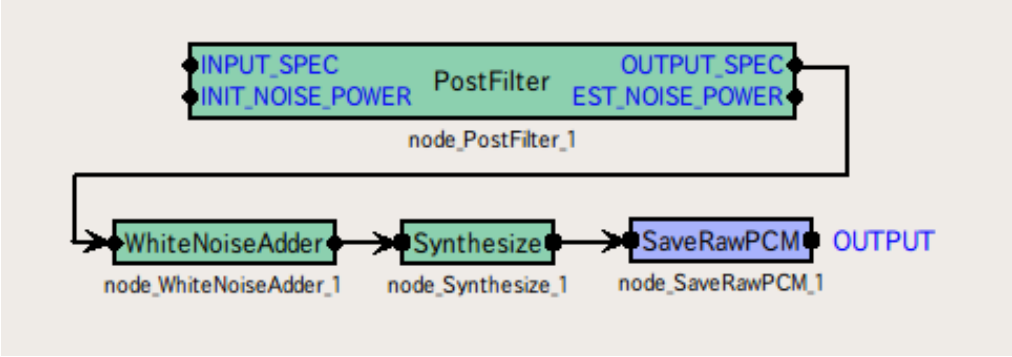


Figure 6.9: Connection example of SaveRawPCM 2

Input-output and property of the node

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 the analysis frame of the speech waveform to be saved in a file.
BITS	int	16	[bit]	Quantization bit rate of speech waveform to be saved in a file. Choose 16 or 24.

Input

INPUT : Map<int, ObjectRef> or Matrix<float> . In the case of Map<int, ObjectRef> , the object should be a Vector<complex<float> > that is an audio signal in frequency domain. Matrix<float> data contains a waveform in the time domain where each row corresponds to a channel.

Output

OUTPUT : Map<int, ObjectRef> .

Parameter

BASENAME : string type. By default, this designates the prefix of the filename as sep_. The filename output is "BASENAME_ID.sw" when a sound source ID is attached. In other words, when BASENAME is sep_, the filenames of separated sounds when separating a mixture of three sounds is sep_0.sw, sep_1.sw, sep_2.sw.

ADVANCE : int type. This must correspond to the values of ADVANCE of other nodes used.

BITS : int type. Quantization bit rate of speech waveform to be saved in a file. Select 16 or 24.

Details of the node

Format of the files saved: The files saved are recorded as Raw PCM sound data without header information. Therefore, when reading the files, users need to designate either 16 [bit] or 24 [bit] as the appropriate quantization bit rate, as well as sampling frequency and track quantity. Moreover, the written files vary depending on the type of input as follows.

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

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

6.1.4 SaveWavePCM

Outline of the node

This node saves speech waveform data in time domain as files. The difference between this and the SaveRawPCM node is only the format of the output files, that is, the wave file format has a header. Therefore, audacity or wavesurfer can easily read the output files of this node. If you want to read a waveform using the AudioStreamFromWave node, use this node instead of using the SaveRawPCM node.

Necessary files

No files are required.

Usage

When to use

The same as the SaveRawPCM node. This node is used when wishing to convert separated sound into waveforms in the Synthesize node to confirm the sound or when wishing to record sound from a microphone array by connecting it to the AudioStreamFromMic node.

Typical connection

The usage is almost the same as for the SaveRawPCM node. The only difference is the SAMPLING_RATE parameter. You can use this node by replacing the SaveRawPCM node with SaveWavePCM node in Fig. 6.8 and 6.9.

Input-output and property of the node

Table 6.5: Parameter list of SaveRawPCM

Parameter name	Type	Default value	Unit	Description
BASENAME	`string`	sep		Prefix of the name of the file to be saved.
ADVANCE	`int`	160	[pt]	Shift length of the analysis frame of the speech waveform to be saved in a file.
SAMPLING_RATE	`int`	16000	[Hz]	Sampling rate. This parameter is used to set its header.
BITS	`string`	int16	[bit]	Quantization bit rate of speech waveform to be saved in a file. Choose int16 or int24.

Input

INPUT : `Map<int, ObjectRef>` or `Matrix<float>` type. The former is a structure containing a sound source ID and waveform data (such as a separated sound) and the latter is a waveform data matrix of multiple channels.

Output

OUTPUT : `Map<int, ObjectRef>` or `Matrix<float>` type. The output data is the same as the input.

Parameter

BASENAME : `string` type. The default filename prefix is sep_. The output filename is "BASENAME_ID.wav" when a sound source ID is attached. For example, the file names of separated sounds as a result of separating three sound mixtures is `sep_0.wav`, `sep_1.wav`, `sep_2.wav` when BASENAME is sep_.

ADVANCE : `int` type. It is necessary to make this the same as the values of ADVANCE of other nodes.

SAMPLING_RATE : `int` type. It is necessary to make this the same as the values of SAMPLING_RATE of other nodes. This value is used only for the header and you cannot change the SAMPLING_RATE of the A/D converter.

BITS : `string` type. Quantization bit rate of the speech waveform to be saved in a file. Select int16 or int24.

Details of the node

Format of the files saved: The files saved are recorded as Wave PCM sound data that have header information. Therefore, when reading the files, users don't need to specify sampling frequency, track quantity and quantization bit rate. Moreover, the written files vary depending on the types of inputs as follows.

`Matrix<float>` type The file written is a multichannel audio file with the same number of channels as the number of rows in the input.

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

6.1.5 HarkDataStreamSender

Details of the node

This node sends the following acoustic signal results by socket communication.

- Acoustic signal
- Frequency spectrum after STFT
- Source information of source localization result
- Acoustic feature
- Missing Feature Mask

Necessary file

No files are required.

Usage

When to use

This node is used to send the above data to a system external to HARK using TCP/IP communication.

Typical connection

In the example in Figure 6.10, all input terminals are connected. It is also possible to leave input terminals open depending on the transmitted data. To learn about the relation between the connection of the input terminals and transmitted data, see “Details of the node”.

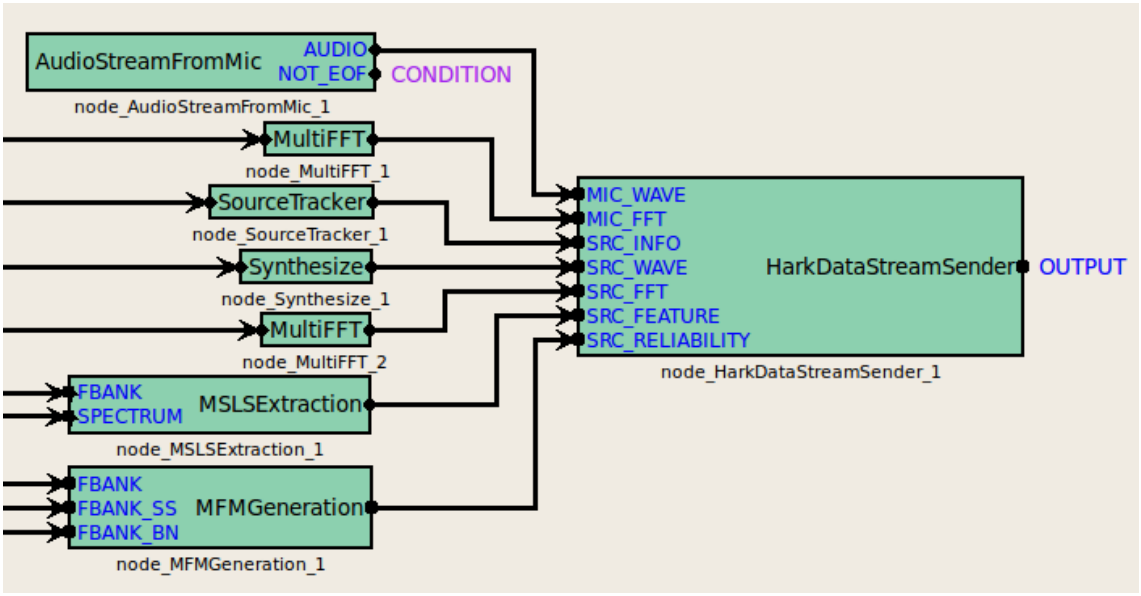


Figure 6.10: Connection example of HarkDataStreamSender

Table 6.6: Parameter list of HarkDataStreamSender

Parameter name	Type	Default value	Unit	Description
HOST	string	localhost		Host name /IP address of the server to which data is sent
PORT	int	8890		Port number for outbound network communication
ADVANCE	int	160	[pt]	Shift length of frame
BUFFER_SIZE	int	512		Size of allocated float-sized memory for socket communication
FRAMES_PER_SEND	int	1	[frm]	Frequency of socket communication in frame unit
DEBUG_PRINT	bool	false		ON/OFF for outputting debugging information
SOCKET_ENABLE	bool	true		Flag to determine whether or not to perform the socket output

Input-output and property of the node

Input

MIC_WAVE : Matrix<float> type. Acoustic signal (The number of channels × acoustic signal of window length size STFT in each channel)

MIC_FFT : Matrix<complex<float> > type. Frequency spectrum (The number of channels × spectrum of each channel)

SRC_INFO : Vector<ObjectRef> type. Source information on the source localization results of several sound sources

SRC_WAVE : Map<int, ObjectRef> type. A sound source ID and acoustic signal (Vector<float> type) data pair.

SRC_FFT : Map<int, ObjectRef> type. A sound source ID and frequency spectrum (Vector<complex<float> > type) data pair.

SRC_FEATURE : Map<int, ObjectRef> type. A sound source ID and acoustic feature (Vector<float> type) data pair.

SRC_RELIABILITY : Map<int, ObjectRef> type. A sound source ID and mask vector (Vector<float> type) data pair.

Output

OUTPUT : ObjectRef type. Same output as the input.

Parameter

HOST : string type. IP address of a host to which data is transmitted. It is invalid when SOCKET_ENABLED is set to false.

PORT : int type. Socket number. It is invalid when SOCKET_ENABLED is set to false.

ADVANCE : int type. Shift length of a frame. It must be equal to the value set in previous processing.

BUFFER_SIZE : int type. Buffer size secured for socket communication.

FRAMES_PER_SEND : int type. Frequency of socket communication in frame unit.

DEBUG_PRINT : bool type. ON/OFF of debug to standard output.

SOCKET_ENABLE : bool type. Data is transferred to the socket when true and not transferred when false.

Details of the node

(A) Description of the parameters

For HOST, designate a host name or an IP address of the host running an external program to transmit data. For PORT, designate a network port number for data transmission. ADVANCE is the shift length of a frame and must be equal to the value set in previous processing. BUFFER_SIZE is a buffer size to be secured for socket communication. A float type array of BUFFER_SIZE * 1024 is secured at the time of initialization. It must be greater than the transmitted data. FRAMES_PER_SEND is the frequency of socket communication in frame unit. The default value is 1 and sufficient for the most cases, which sends data in every frame. If you want to reduce the amount of socket communication, increase this value. DEBUG_PRINT indicates if debug to standard output should be displayed. This outputs some parts of the transmitted data. For more information, see "Debug" in Table 6.13. When SOCKET_ENABLED is set to `false`, data is not sent to external systems. This is used to perform a network operation check for HARK without operating an external program.

(B) Details of data transmission

(B-1) Structure for data transmission

Data transmission is performed for each frame, being divided into some parts. The structures defined for data transmission are listed as follows.

- **HD_Header**
  Description: A header that contains basic information on top of the transmitted data
  Data size: `3 * sizeof(int)`

Table 6.7: Member of HD_Header

Variable name	Type	Description
type	int	Bit flag that indicates the structure of the transmitted data. For relations between each bit and data to be transmitted, see Table 6.8.
advance	int	Shift length of a frame
count	int	Frame number of HARK

Table 6.8: Each bit and transmit data of the types of HD_Header

Number of digits	Related input terminal	Transmit data
The first column	MIC_WAVE	Acoustic signal
The second column	MIC_FFT	Frequency spectrum
The third column	SRC_INFO	Source localization result source information
The fourth column	SRC_INFO, SRC_WAVE	Source localization result source information + acoustic signal for each sound source ID
The fifth column	SRC_INFO, SRC_FFT	Source localization result source information + frequency spectrum for each sound source ID
The sixth column	SRC_INFO, SRC_FEATURE	Source localization result source information + acoustic feature for each sound source ID
The seventh column	SRC_INFO, SRC_RELIABILITY	Source localization result source information + missing feature mask for each sound source ID

In HarkDataStreamSender , The transmitted data differs depending on whether the input terminal can be opened. On the receiving end, the transmitted data can be interpreted according to their types. Examples are given below. Further details on transmitted data are given in (B-2).

Example 1) In the case that only the MIC_FFT input terminal is connected, the type is 0000010 in binary number. Moreover, the transmitted data becomes only a frequency spectrum for each microphone.

Example 2) In the case that the three input terminals of MIC_WAVE, SRC_INFO and SRC_FEATURE are connected, the type is 0100101 in binary. The data to be transmitted are acoustic signals for each microphone, source information of a source localization result and acoustic features for each sound source ID.

Note) For the four input terminals of SRC_WAVE, SRC_FFT, SRC_FEATURE and SRC_RELIABILITY, the data to be transmitted are information for each sound source ID and therefore information of SRC_INFO is required. Even if the above four input terminals are connected without connecting SRC_INFO, no data is transmitted. In such a case, the type is 0000000 in binary.

- **HDH_MicData**
  Description: Structural information on the array size for sending two-dimensional arrays
  Data size: `3 * sizeof(int)`

Table 6.9: Member of HDH_MicData

Variable name	Type	Description
nch	int	Number of microphone channels
length	int	Data length (number of columns of the two-dimensional array to be transmitted)
data_bytes	int	Number of bytes of data to be transmitted. In the case of a float type matrix, `nch * length * sizeof(float)`.

- **HDH_SrcInfo**
  Description: Source information of a source location result
  Data size: `1 * sizeof(int)+ 4 * sizeof(float)`

Table 6.10: Member of HDH_SrcInfo

Variable name	Type	Description
src_id	int	Sound source ID
x[3]	float	Three-dimensional position of sound source
power	float	Power of the MUSIC spectrum calculated in LocalizeMUSIC

- **HDH_SrcData**
  Description: Structural information on the array size for sending one-dimensional arrays
  Data size: `2 * sizeof(int)`

Table 6.11: Member of HDH_SrcData

Variable name	Type	Description
length	int	Data length (number of one-dimensional array elements to be transmitted)
data_bytes	int	Number of bytes of transmitted data. In the case of a float type vector, `length * sizeof(float)`.

(B-2) Transmitted data

Transmitted data is divided for each frame as shown in (a)-(q) of Tables 6.12 and 6.13. Table 6.12 shows the relation between the transmitted data (a)-(q) and the input terminal connected, and Table 6.13 describes the transmitted data.

Table 6.12: Data list in order of sending and connection input terminal (The data with the ○ symbol is transmitted. ○* indicates the data that are not transmitted when the SRC_INFO terminal is not connected)

Details of the transmitted data			Input terminal and transmitted data						
	Type	Size	MIC_WAVE	MIC_FFT	SRC_INFO	SRC_WAVE	SRC_FFT	SRC_FEATURE	SRC_RELIABILITY
(a)	HD_Header	`sizeof(HD_Header)`	○	○	○	○	○	○	○
(b)	HDH_MicData	`sizeof(HDH_MicData)`	○						
(c)	float[]	`HDH_MicData.data_bytes`	○						
(d)	HDH_MicData	`sizeof(HDH_MicData)`		○					
(e)	float[]	`HDH_MicData.data_bytes`		○					
(f)	float[]	`HDH_MicData.data_bytes`		○					
(g)	int	`1 * sizeof(int)`			○	○*	○*	○*	○*
(h)	HDH_SrcInfo	`sizeof(HDH_SrcInfo)`			○	○*	○*	○*	○*
(i)	HDH_SrcData	`sizeof(HDH_SrcData)`				○*			
(j)	short int[]	`HDH_SrcData.data_bytes`				○*			
(k)	HDH_SrcData	`sizeof(HD_SrcData)`					○*		
(l)	float[]	`HDH_SrcData.data_bytes`					○*		
(m)	float[]	`HDH_SrcData.data_bytes`					○*		
(n)	HDH_SrcData	`sizeof(HD_SrcData)`						○*	
(o)	float[]	`HDH_SrcData.data_bytes`						○*	
(p)	HDH_SrcData	`sizeof(HD_SrcData)`							○*
(q)	float[]	`HDH_SrcData.data_bytes`							○*

Table 6.13: Details of the transmitted data

	Description	Debug
(a)	Transmitted data header. See Table 6.7.	○
(b)	Structure of acoustic signals (number of microphones, frame length, byte count for transmission). See Table 6.9.	○
(c)	Acoustic signal (number of microphones × float type matrix of frame length)	
(d)	Structure of frequency spectra (number of microphones, number of frequency bins, byte count for transmission). See Table 6.9.	○
(e)	Real part of frequency spectrum (number of microphones × float type matrix of number of frequency bins)	
(f)	Imaginary part of frequency spectrum (number of microphones × float type matrix of number of frequency bins)	
(g)	Number of sound sources detected	○
(h)	Source of a source location result. See Table 6.10.	○
(i)	Structure that indicates that of acoustic signals for each sound source ID (frame length, byte count for transmission). See Table 6.11.	○
(j)	Acoustic signal for each sound source ID (float type linear array of frame length)	
(k)	Structure that indicates that of frequency spectra for each sound source ID (number of frequency bins, byte count for transmission). See Table 6.11.	○
(l)	Real part of a frequency spectrum for each sound source ID (float type linear array of number of frequency bins)	
(m)	Imaginary part of a frequency spectrum for each sound source ID (float type linear array of number of frequency bins)	
(n)	Structure that indicates that of acoustic features for each sound source ID (dimension number of features, byte count for transmission). See Table 6.11.	○
(o)	Acoustic feature for each sound source ID (float type linear array of dimension number of features)	
(p)	Structure that indicates that of MFM for each sound source ID (dimension number of features, byte count for transmission). See Table 6.11.	○
(q)	MFM for each sound source ID (float type linear array of dimension number of features)	

```
  calculate{
  Send (a)
  IF MIC_WAVE is connected
  Send (b)
  Send (c)
  ENDIF
  IF MIC_FFT is connected
  Send (d)
  Send (e)
  Send (f)
  ENDIF
  IF SRC_INFO is connected
  Send (g)
  (Let the number of sounds 'src_num'.)
FO  R i = 1 to src_num (This is a sound ID based routine.)
  Send (h)
  IF SRC_WAVE is connected
  Send (i)
  Send (j)
  ENDIF
  IF SRC_FFT is connected
  Send (k)
  Send (l)
  Send (m)
  ENDIF
  IF SRC_FEATURE is connected
  Send (n)
  Send (o)
  ENDIF
  IF SRC_RELIABILITY is connected
  Send (p)
  Send (q)
  ENDIF
  ENDFOR
  ENDIF}
```

(B-3) Transmission algorithm Some parts of the algorithm that operate in a loop when executing the HARK network file are shown above. Here, (a)-(q) in the code correspond to (a)-(q) in Tables 6.12 and 6.13.

6.2 Localization category

6.2.1 CMLoad

Module Overview

Reads the sound source correlation matrix from a file.

Requested Files

A file to save in CMSave format.

Usage

In what case is the node used?

Use when reading the correlation matrix of a sound source saved using CMSave .

Typical Examples

Figure 6.11 shows the usage example for the CMLoad node. FILENAMER and FILENAMEI are inouts with type string, indicating the files containing the real and imaginary parts respectively of the correlation matrix. OPERATION_FLAG is `int` type or `bool` type of input, and specifies when reading the correlation matrix is read. In the usage example, for all inputs of FILENAMER, FILENAMEI, OPERATION_FLAG, Constant nodes are connected. Parameters during run-time are constant, but the output of the constant node is dynamic and hence it is possible to change the correlation matrix used.

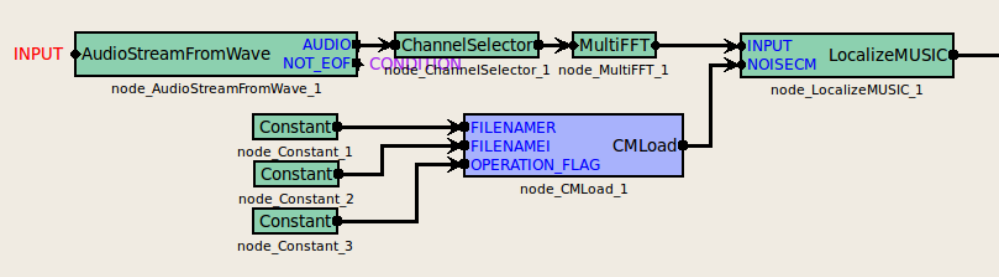


Figure 6.11: Network Example using CMLoad

I/O and property setting of the node

Table 6.14: Parameter list of CMLoad

Parameter name	Type	Default	Unit	Description
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

FILENAMER : `string` type. Name of the file containing the real part of the correlation matrix.

FILENAMEI : `string` type. Name of the file containing the imaginary part of the correlation matrix.

OPERATION_FLAG : `int` type or `bool` type. Only when this input terminal is 1 or when true, and the filename changes, the correlation matrix is read.

Output

OUTPUTCM : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. A M-th order complex square array with correlation matrix outputs $NFFT/2 + 1$ items. `Matrix<complex<float> >` contains rows corresponding to frequency ($NFFT/2 + 1$ rows), and columns containing the complex correlation matrix ($M * M$ columns across).

Parameter

ENABLE_DEBUG : `bool` type. Default value is false. When true, it is output to the standard output when reading the correlation matrix.

Module Description

M-th order complex square array for each frequency bin with the correlation matrix's real and imaginary parts is read as a `Matrix<float>` type from the respective files. If the number of frequency bins is k, and ($k = NFFT/2 + 1$), then each read file consists of k rows and M^2 columns. Reading is performed only when OPERATION_FLAG is 1 or true, and immediately after network operation, or when the read filename is changed.

6.2.2 CMSave

Module Overview

Saves the sound source correlation matrix to a file.

Requested Files

None.

Usage

In what case is the node used?

Use when saving the correlation matrix for a sound source, created from CMMakerFromFFT or CMMakerFromFFTwithFlag , etc.

Typical Examples

Figure. 6.12 shows a usage example of CMSave node.

INPUTCM input terminal is connected with a correlation matrix calculated from CMMakerFromFFT or CMMakerFromFFTwithFlag , etc. (type is `Matrix<complex<float> >` type, but to process correlation matrices, convert it from a three dimensional complex array to a two dimensional complex array and then output). FILENAMER and FILENAMEI are `string` type inputs, each indicating the saved files name containing the real and imaginary parts of the correlation matrix. OPERATION_FLAG is `int` type or `bool` type input, specifying when the correlation matrix is read. (Figure. 6.12 shows the connection of an Equal node as an example. However, if the node can output `int` type or `bool` type, then it does not matter at all).

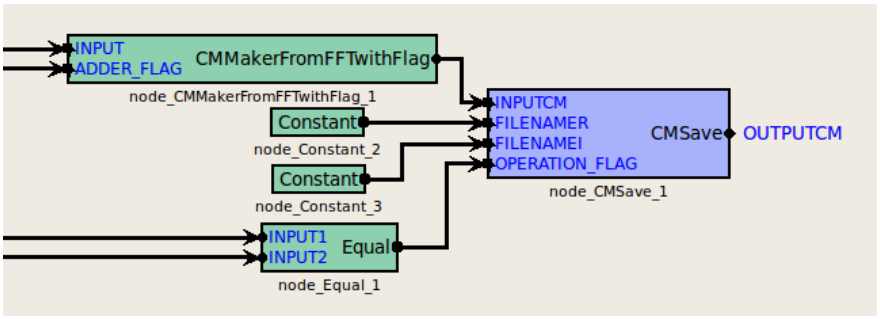


Figure 6.12: Network Example using CMSave

I/O and property setting of the node

Table 6.15: Parameter list of CMSave

Parameter	Type	Default	Unit	Description
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

INPUTCM : `Matrix<complex<float> >` type. Correlation matrix of each frequency bin. M-th order complex square array with correlation matrix outputs $NFFT/2+1$ items. `Matrix<complex<float> >` indicates the row of frequency ($NFFT/2+1$ rows), and column of complex correlation matrix ($M*M$ columns).

FILENAMER : `string` type. Name of file containing the real part of the correlation matrix.

FILENAMEI : `string` type. Name of file containing the imaginary part of the correlation matrix.

OPERATION_FLAG : `int` type or `bool` type. Only when this input terminal is 1 or when true, the correlation matrix is saved.

Output

OUTPUTCM : `Matrix<complex<float> >` type. Same as INPUTCM.

Parameter

ENABLE_DEBUG : `bool` type. Default value is false. When true, it is output to the standard output when saving the correlation matrix.

Module Description

Corrects the correlation matrix to `Matrix<float>` type to a M-th order complex square array for each frequency bin, and saves with the specified file name. Saved files are divided into real and imaginary parts of the correlation matrix. If the number of frequency bins is k, and ($k = NFFT/2 + 1$), then the saved file will store k rows and M^2 columns each. Saving is performed only when OPERATION_FLAG is 1 or true.

6.2.3 CMChannelSelector

Module Overview

From a multi-channel correlation matrix, get only the specified channel data in the specified sequence.

Requested Files

None.

Usage

In what case is the node used?

From the multi-channel correlation matrix that was input, using this when channels not required are to be deleted, or the channel sequence is to be exchanged, or when the channel is to be copied.

Typical Examples

Figure. 6.13 shows a usage example for the CMChannelSelector node. The input terminal is connected to a correlation matrix calculated from CMMakerFromFFT or CMMakerFromFFTwithFlag , etc. (type is `Matrix<complex<float> >` type, but to handled a correlation matrix, convert the three dimensional complex array to a two dimensional complex array and then output).

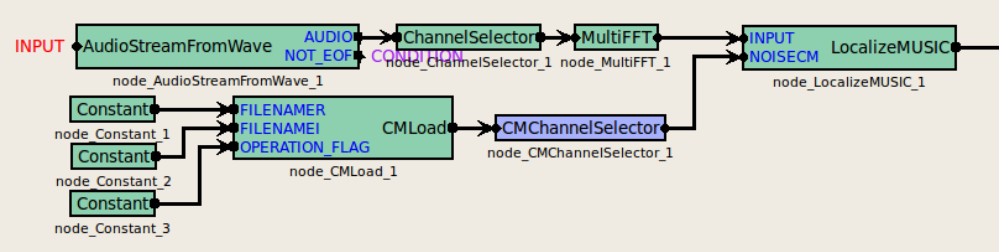


Figure 6.13: Network Example using CMChannelSelector

I/O and property setting of the node

Table 6.16: Parameter list of CMChannelSelector

Parameter	Type	Default	Unit	Description
SELECTOR	`Vector<int>`	`<Vector<int> >`		Specifies the channel numbers to be output

Input

INPUTCM : `Matrix<complex<float> >` type. Correlation matrix for each frequency bin. M-th order complex square array with correlation matrix inputs $NFFT/2+1$ items. `Matrix<complex<float> >` indicates the row of frequency ($NFFT/2+1$ rows), and column of complex correlation matrix ($M*M$ columns).

Output

OUTPUTCM : `Matrix<complex<float> >` type. Same as INPUTCM.

Parameter

SELECTOR : `Vector<int>` type. No default value (`<Vector<int> >`). Specifies the channel numbers to be used. Channel numbers start from 0.

Example: From 5 channels (0-4), specify `<Vector <int> 2 3 4>` when using only channels 2, 3, 4. When exchanging the 3rd channel and 4th channel, specify `<Vector <int> 0 1 2 4 3 5>`.

Module Description

From a correlation matrix containing a complex three-dimensional array of input data of size $k \times M \times M$, extract only the correlation matrix of the specified channel, and output the new complex three-dimensional array of data of size $k \times M \times M$. k is the number of frequency bins ($k = NFFT/2 + 1$), M is the number of input channels, and M' is the number of output channels.

6.2.4 CMMakerFromFFT

Module Overview

From the multi-channel complex spectrum that is output from the MultiFFT node, generate the sound source correlation matrix with a fixed period.

Requested Files

None.

Usage

In what case is the node used?

Given a sound source of LocalizeMUSIC node, in order to suppress a specific sound source like noise, etc., it is necessary to prepare a correlation matrix that includes noise information beforehand. This node generates the correlation matrix for a sound source, at fixed period, from a multi-channel complex spectrum that is output from the MultiFFT node. Suppressed sound source can be achieved by connecting the output of this node to the NOISECM input terminal of LocalizeMUSIC node, assuming that information before a fixed period is always noise.

Typical Examples

Figure. 6.14 shows the usage example of CMMakerFromFFT node.

INPUT The input terminal is connected to the complex spectrum of the input signal calculated from a MultiFFT node. The type is `Matrix<complex<float> >` type. This node calculates and outputs the correlation matrix between channels for each frequency bin from the complex spectrum of an input signal. The output type is `Matrix<complex<float> >` type, but to handle a correlation matrix, convert the three dimensional complex array to a two dimensional complex array and then output.

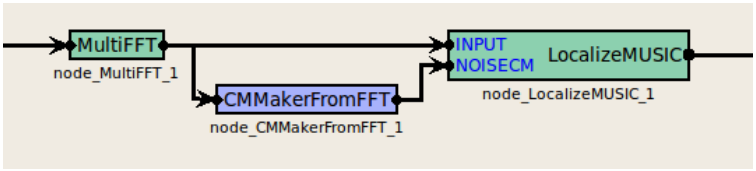


Figure 6.14: Network Example using CMMakerFromFFT

I/O and property setting of the node

Table 6.17: Parameter list of CMMakerFromFFT

Parameter	Type	Default	Unit	Description
WINDOW	`int`	50		Number of averaged frames for a CM
PERIOD	`int`	50		Frame rate for renewing the correlation matrix
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

INPUT : `Matrix<complex<float> >` type, the complex spectrum expression of an input signal $M \times (NFFT/2+1)$.

Output

OUTPUT : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. An M-th order complex square array with correlation matrix outputs $NFFT/2 + 1$ items. `Matrix<complex<float> >` indicates the rows corresponding to frequency ($NFFT/2 + 1$ rows), and columns containing the complex correlation matrix ($M * M$ columns across).

Parameter

WINDOW : `int` type. Default value is 50. Specifies the number of average smoothed frames when calculating the correlation-matrix. The node generates a correlation matrix for each frame from the complex spectrum of the input signal and outputs a new correlation matrix by averaging the frames that are specified in WINDOW. The correlation matrix calculated at the end is output between the PERIOD frames. If this value is increased, the correlation matrix is stabilized but the calculation cost becomes high.

PERIOD : `int` type. Default value is 50. Specifies the frame rate for renewing the correlation-matrix. The node generates a correlation matrix for each frame from the complex spectrum of the input signal and outputs a new correlation matrix by averaging the frames that are specified in WINDOW. The correlation matrix calculated at the end is output between the PERIOD frames. If this value is increased, the time resolution of correlation matrix is improved but the calculation cost becomes high.

ENABLE_DEBUG : `bool` type. Default value is false. When true, the frame number is output to the standard output while generating the correlation matrix.

Module Description

The complex spectrum of the input signal output from a MultiFFT node is represented as follows.

$$\boldsymbol{X}(\omega, f) = [X_1(\omega, f), X_2(\omega, f), X_3(\omega, f), \cdots, X_M(\omega, f)]^T \tag{6.1}$$

Here, ω is the frequency bin number, f is the frame number for use with HARK , M represents the number of input channels.

The correlation matrix of the input signal $\boldsymbol{X}(\omega, f)$ can be defined as follows for every frequency and frame.

$$\boldsymbol{R}(\omega, f) = \boldsymbol{X}(\omega, f)\boldsymbol{X}^*(\omega, f) \tag{6.2}$$

Here, $()^*$ denotes the complex conjugate transpose operator. There is no problem if this $\boldsymbol{R}(\omega, f)$ is used as it is in subsequent processing, but practically, in order to obtain a stable correlation matrix in HARK , it uses an average through time as shown below.

$$\boldsymbol{R}'(\omega, f) = \frac{1}{\mathrm{WINDOW}} \sum_{i=0}^{\mathrm{WINDOW}-1} \boldsymbol{R}(\omega, f + i) \tag{6.3}$$

$\boldsymbol{R}'(\omega, f)$ is output by every PERIOD frame from the OUTPUT terminal of CMMakerFromFFT node.

6.2.5 CMMakerFromFFTwithFlag

Module Overview

From the multi-channel complex spectrum that is output from the MultiFFT node, generate the correlation matrix for a sound source when specified by the input flag.

Requested Files

None.

Usage

In what case is the node used?

The scenario for usage is same as CMMakerFromFFT node; for details refer to the CMMakerFromFFT node. The main difference is in the calculation of the correlation matrix. In the CMMakerFromFFT node, the correlation matrix is updated at a fixed period (PERIOD), but in this node it is possible to generate a correlation matrix for a specified section according to the flag value obtained from the input terminal.

Typical Examples

Figure. 6.15 shows the usage example of CMMakerFromFFTwithFlag node. The INPUT input terminal is connected to the complex spectrum of the input signal calculated from a MultiFFT node. The type is `Matrix<complex<float> >` type. ADDER_FLAG is `int` type or `bool` type of input, and controls the events related to the correlation matrix calculation. Event control details are given in the Module details section. This node calculates and outputs the correlation matrix between channels for each frequency bin from the complex spectrum of the input signal. The output type is `Matrix<complex<float> >` type, but to handle a correlation matrix, convert the three dimensional complex array to a two dimensional complex array and then output.

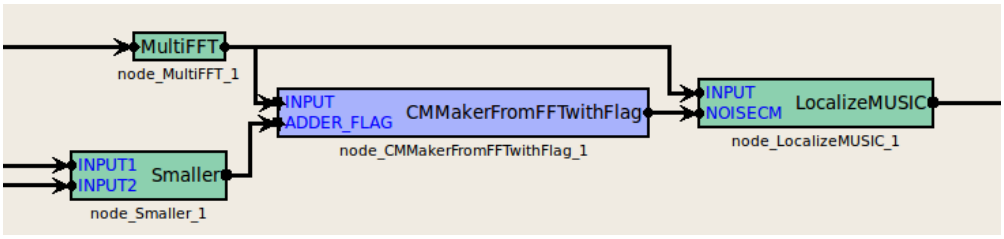


Figure 6.15: Network Example using CMMakerFromFFTwithFlag

I/O and property setting of the node

Table 6.18: Parameter list of CMMakerFromFFTwithFlag

Parameter	Type	Default	Unit	Description
DURATION_TYPE	`string`	FLAG_PERIOD		Flag-based or Frame-period-based generation
WINDOW	`int`	50		Number of averaged frames for a CM
PERIOD	`int`	50		Frame rate for renewing the correlation matrix
MAX_SUM_COUNT	`int`	100		Maximum number of normalized frames of a CM
ENABLE_ACCUM	`bool`	false		Enable averaging with past correlation matrix
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

INPUT : `Matrix<complex<float> >` type, the complex spectrum expression of an input signal with size $M \times (NFFT/2+1)$.

ADDER_FLAG : `int` type or `bool` type. Controls the events related to correlation matrix calculation. Refer to the Module Description section for event control details.

Output

OUTPUT : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. An M-th order complex square array correlation matrix outputs $NFFT/2+1$ items. `Matrix<complex<float> >` contains rows corresponding to frequency ($NFFT/2+1$ rows), and columns containing the complex correlation matrix ($M*M$ columns across).

Parameter

DURATION_TYPE : `string` type. Default value is FLAG_PERIOD. This changes the algorithm for renewing and averaging a correlation matrix. If DURATION_TYPE=FLAG_PERIOD, it renews/averages a correlation matrix based on the value of ADDER_FLAG. If DURATION_TYPE=FRAME_PERIOD, it renews/averages a correlation matrix based on a constant frame period. Refer to the Module Description section for event control details.

WINDOW : `int` type. Default value is 50. This parameter is active when DURATION_TYPE=FRAME_PERIOD. Specifies the number of average smoothed frames when calculating the correlation-matrix. The node generates a correlation matrix for each frame from the complex spectrum of the input signal and outputs a new correlation matrix by averaging the frames that are specified in WINDOW. The correlation matrix calculated at the end is output between the PERIOD frames. If this value is increased, the correlation matrix is stabilized but the calculation cost becomes high.

PERIOD : `int` type. Default value is 50. This parameter is active when DURATION_TYPE=FRAME_PERIOD. Specifies the frame rate for renewing the correlation-matrix. The node generates a correlation matrix for each frame from the complex spectrum of the input signal and outputs a new correlation matrix by averaging the frames that are specified in WINDOW. The correlation matrix calculated at the end is output between the PERIOD frames. If this value is increased, the time resolution of correlation matrix is improved but the calculation cost becomes high.

MAX_SUM_COUNT : `int` type. Default value is 100. This parameter is active when DURATION_TYPE=FLAG_PERIOD. Specifies the maximum number of average smoothed frames when calculating the correlation-matrix. This node can control the number of average smoothed frames of a correlation matrix by ADDER_FLAG. For this reason, if the ADDER_FLAG is always 1, only addition of correlation matrix is performed and there will be no output at all. Thus, when it reaches the maximum count of average smoothed frames by correctly setting the MAX_SUM_COUNT, the correlation matrix will be output forcefully. To turn OFF this feature specify MAX_SUM_COUNT = 0.

ENABLE_ACCUM : `bool` type. Default value is false. This parameter is active when DURATION_TYPE=FLAG_PERIOD. This enables averaging a corrlation matrix with current one and past one together.

ENABLE_DEBUG : `bool` type. Default value is false. When true, the frame number is output to the standard output at the time of generating the correlation matrix.

Module Description

The algorithm for the CMMakerFromFFT node and correlation matrix calculation is the same. Refer to the Module description of the CMMakerFromFFT node for details. The difference with CMMakerFromFFT node is that the average smoothed frames of a correlation matrix can be controlled with the ADDER_FLAG input terminal flag.

In the CMMakerFromFFT node, the correlation matrix was computed with the following formula with the number of frames specified by PERIOD.

$$\boldsymbol{R}'(\omega, f) = \frac{1}{\mathrm{PERIOD}} \sum_{i=0}^{\mathrm{PERIOD}-1} \boldsymbol{R}(\omega, f+i) \tag{6.4}$$

When DURATION_TYPE=FLAG_PERIOD, this node generates a correlation matrix based on the value of the ADDER_FLAG as follows.

A) When ADDER_FLAG changes from 0 (or false) to 1 (or true)

- The correlation matrix returns to zero matrix and PERIOD returns to 0.
  $\boldsymbol{R}'(\omega) = \boldsymbol{O}$
  PERIOD = 0
  where $\boldsymbol{O} \in \mathbb{C}^{(NFFT/2+1)\times M\times M}$ represents the zero matrix.

B) When ADDER_FLAG is 1 (or true)

- Add the correlation matrix.
  $\boldsymbol{R}'(\omega) = \boldsymbol{R}'(\omega) + \boldsymbol{R}(\omega, f + i)$
  PERIOD = PERIOD + 1

C) When ADDER_FLAG changes from 1 (or true) to 0 (or false)

- Take the average of the added correlation matrix and output it to OUTPUT.
  $\boldsymbol{R}_{out}(\omega, f) = \frac{1}{\text{PERIOD}}\boldsymbol{R}'(\omega)$

D) When ADDER_FLAG is 0 (or false)

- Keep the correlation matrix generated in the end.
  $\boldsymbol{R}_{out}(\omega, f)$

Here, $\boldsymbol{R}_{out}(\omega, f)$ is the correlation matrix that is output from the OUTPUT terminal. In other words, the new correlation matrix will be stored in $\boldsymbol{R}_{out}(\omega, f)$ in phase C).

When DURATION_TYPE=FRAME_PERIOD, this node renews the correlation matrix only when ADDER_FLAG is 1 (or true) based on eq. (6.4).

6.2.6 CMDivideEachElement

Module Overview

Performs component-wise division of two sound source correlation matrices.

Requested Files

None.

Usage

In what case is the node used?

The calculation node of the correlation matrix is the one created for the sound source from CMMakerFromFFT , CMMakerFromFFTwithFlag , and has the function of dividing each component.

Typical Examples

Figure. 6.16 shows the usage example of CMDivideEachElement node. The CMA input terminal is connected to a correlation matrix calculated from CMMakerFromFFT or CMMakerFromFFTwithFlag , etc. (type is `Matrix<complex<float> >` type, but to handle a correlation matrix, convert the three dimensional complex array to a two dimensional complex array and then output). Like CMA, the CMB input terminal also connects to the correlation matrix. At the time of division, CMA ./ CMB is calculated, where ./ shows component-wise division. OPERATION_FLAG is `int` type, or `bool` type input, and controls when the correlation matrix division is calculated.

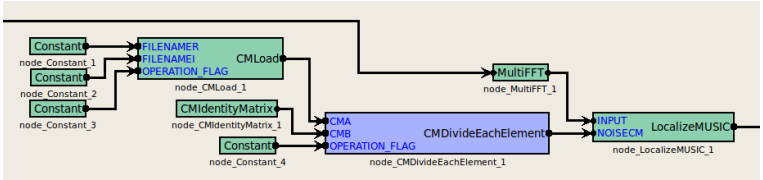


Figure 6.16: Network Example using CMDivideEachElement

I/O and property setting of the node

Table 6.19: Parameter list of CMDivideEachElement

Parameter	Type	Default	Unit	Description
FIRST_FRAME_EXECUTION	`bool`	false		Selection of first frame execution only
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

CMA : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. The M-th order complex square array correlation matrix inputs $NFFT/2 + 1$ items. `Matrix<complex<float> >` contains rows corresponding to frequency ($NFFT/2 + 1$ rows), and columns containing the complex correlation matrix ($M * M$ columns across).

CMB : `Matrix<complex<float> >` type. Same as CMA.

OPERATION_FLAG : `int` type or `bool` type. Only when this input terminal is 1 or when true, calculation of the correlation matrix is performed.

Output

OUTPUTCM : Matrix<complex<float> > type. The correlation matrix equivalent to CMA ./ CMB after division is output.

Parameter

FIRST_FRAME_EXECUTION : bool type. Default value is false. When true, OPERATION_FLAG is always 0. Even when it is false, the operation is performed only on the first frame.

ENABLE_DEBUG : bool type. Default value is false. When true, output the frame number calculated to the standard output, during dividing the correlation matrix division.

Module Description

Performs component-wise division of the two correlation matrices. Note that it is not a matrix division of the correlation matrix. The correlation matrix is a complex three-dimensional array of size $k \times M \times M$ and the division $k \times M \times M$ times is performed as follows. Here, k is the number of frequency bins ($k = NFFT/2 + 1$), and M is the number of channels in the input signal.

```
OUTPUTCM = zero_matrix(k,M,M)

calculate{

   IF OPERATION_FLAG
       FOR i = 1 to k
           FOR i = 1 to M
               FOR i = 1 to M

                   OUTPUTCM[i][j][k] = CMA[i][j][k] / CMB[i][j][k]

               ENDFOR
           ENDFOR
       ENDFOR
   ENDIF

}
```

The matrix that is output from the OUTPUTCM terminal is initialized as a zero matrix, and maintains the final operational result from this point onward.

6.2.7 CMMultiplyEachElement

Module Overview

Performs component-wise multiplication of two sound source correlation matrices.

Requested Files

None.

Usage

In what case is the node used?

The calculation node of the correlation matrix is the one created for the sound source from CMMakerFromFFT , CMMakerFromFFTwithFlag , and has the function of multiplying each component.

Typical Examples

Figure. 6.17 shows the usage example of CMMultiplyEachElement node.

The CMA input terminal is connected to a correlation matrix calculated from CMMakerFromFFT or CMMakerFromFFTwithFlag , etc. (type is `Matrix<complex<float> >` type, but to handle a correlation matrix, convert the three dimensional complex array to a two dimensional complex array and then output). Like CMA, the CMB input terminal also connects to the correlation matrix. At the time of multiplication, CMA .∗ CMB is calculated, where .∗ shows component-wise multiplication. OPERATION_FLAG is `int` type, or `bool` type input, and controls when the correlation matrix multiplication is calculated.

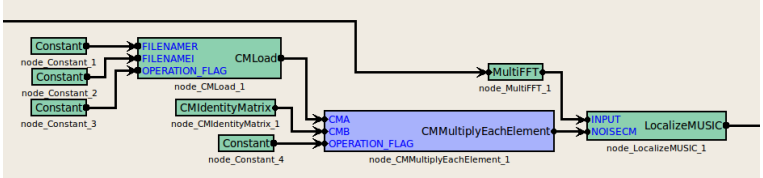


Figure 6.17: Network Example using CMMultiplyEachElement

I/O and property setting of the node

Table 6.20: Parameter list of CMMultiplyEachElement

Parameter	Type	Default	Unit	Description
FIRST_FRAME_EXECUTION	`bool`	false		Selection of the first frame execution
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

CMA : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. The M-th order complex square array correlation matrix inputs $NFFT/2 + 1$ items. `Matrix<complex<float> >` contains rows corresponding to frequencies ($NFFT/2 + 1$ rows), and columns containing the complex correlation matrix ($M * M$ columns across).

CMB : `Matrix<complex<float> >` type. Same as CMA.

OPERATION_FLAG : int type or bool type. Only when this input terminal is 1 or when true, calculation of the correlation matrix is performed.

Output

OUTPUTCM : Matrix<complex<float> > type. The correlation matrix equivalent to CMA .∗ CMB after multiplication is output.

Parameter

FIRST_FRAME_EXECUTION : bool type. Default value is false. When true, OPERATION_FLAG is always 0. Even when it is false, the operation is performed only on first frame.

ENABLE_DEBUG : bool type. Default value is false. When true, output the frame number calculated to the standard output,during correlation matrix division.

Module Description

Performs component-wise multiplication of each component of the two correlation matrices. Note that it is not a matrix multiplication of the correlation matrix (Refer to CMMultiplyMatrix for matrix multiplication). The correlation matrix is a complex three-dimensional array of size $k \times M \times M$ and the multiplication $k \times M \times M$ times is performed as follows. Here, k is the number of frequency bins ($k = NFFT/2 + 1$), and M is the number of channels in the input signal.

```
OUTPUTCM = zero_matrix(k,M,M)

calculate{

   IF OPERATION_FLAG
       FOR i = 1 to k
           FOR i = 1 to M
               FOR i = 1 to M

                   OUTPUTCM[i][j][k] = CMA[i][j][k] * CMB[i][j][k]

               ENDFOR
           ENDFOR
       ENDFOR
   ENDIF

}
```

The matrix that is output from the OUTPUTCM terminal is initialized as a zero matrix, and maintains the final operational result from this point onward.

6.2.8 CMInverseMatrix

Module Overview

Calculates the inverse of the sound source correlation matrix.

Requested Files

None.

Usage

In what case is the node used?

The calculation node of the correlation matrix is the one created for the sound source from CMMakerFromFFT , CMMakerFromFFTwithFlag , and has the function to calculate the inverse matrix of the correlation matrix.

Typical Examples

Figure. 6.18 shows a usage example of CMInverseMatrix node.

INPUTCM input terminal is connected to a correlation matrix calculated from CMMakerFromFFT or CMMakerFromFFTwithFlag , etc. (type is `Matrix<complex<float> >` type, but to handle a correlation matrix, convert the three dimensional complex array to a two dimensional complex array and then output). OPERATION_FLAG is `int` type or `bool` type input, and specifies when to calculate the inverse matrix of the correlation matrix.

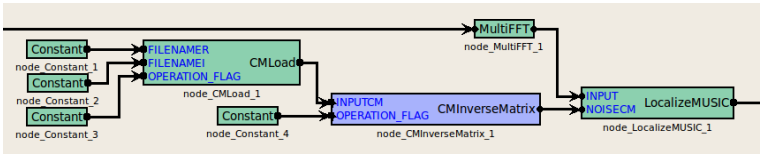


Figure 6.18: Network Example using CMInverseMatrix

I/O and property setting of the node

Table 6.21: Parameter list of CMInverseMatrix

Parameter	Type	Default	Unit	Description
FIRST_FRAME_EXECUTION	`bool`	false		Selection of the first frame execution
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

INPUTCM : `Matrix<complex<float> >` type. A correlation matrix of each frequency bin. The M-th order complex square array correlation matrix outputs $NFFT/2+1$ items. `Matrix<complex<float> >` contains rows corresponding to frequencies ($NFFT/2+1$ rows), and columns containing the complex correlation matrix ($M * M$ columns across).

OPERATION_FLAG : `int` type or `bool` type. Only when this input terminal is 1 or when true, the calculation of correlation matrix is performed.

Output

OUTPUTCM : Matrix<complex<float> > type. The correlation matrix after the inverse matrix calculation is output.

Parameter

FIRST_FRAME_EXECUTION : bool type. Default value is false. When true, OPERATION FLAG is always 0. Even when it is false, the operation is performed only on the first frame.

ENABLE_DEBUG : bool type. Default value is false. When true, output the frame number calculated to the standard output, during correlation matrix calculation.

Module Description

Calculates the inverse matrix of the correlation matrix. The correlation matrix is a complex three-dimensional array of size $k \times M \times M$ and the inverse matrix calculation k times is performed as follows. Here, k is the number of frequency bins ($k = NFFT/2 + 1$), and M is the number of channels in the input signal.

```
OUTPUTCM = zero_matrix(k,M,M)

calculate{

   IF OPERATION_FLAG
       FOR i = 1 to k
           OUTPUTCM[i] = inverse( INPUTCM[i] )
       ENDFOR
   ENDIF

}
```

The matrix that is output from OUTPUTCM terminal is initialized as a zero matrix, and maintains the final operational result from this point onward.

6.2.9 CMMultiplyMatrix

Module Overview

Multiply each frequency bin of the two sound source correlation matrices.

Requested Files

None.

Usage

In what case is the node used?

The calculation node of the sound source correlation matrix is the same as the one from CMMakerFromFFT , CMMakerFromFFTwithFlag , and has the function of multiplying the correlation matrix of each frequency bin.

Typical Examples

Figure. 6.19 shows the usage example of CMMultiplyMatrix node.

CMA input terminal is connected to the correlation matrix calculated from CMMakerFromFFT , CMMaker-FromFFTwithFlag , etc. (Type is `Matrix<complex<float> >` type, but to handle a correlation matrix, convert the three-dimensional complex array to a two-dimensional complex matrix and then output). The CMB input terminal, like CMA, connects to the same correlation matrix. At the time of multiplication, CMA $*$ CMB is calculated for each frequency bin. OPERATION_FLAG is an `int` type or `bool` type input, specifying when the correlation matrix is calculated.

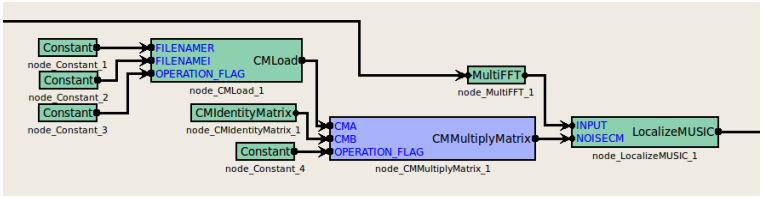


Figure 6.19: Network Example using CMMultiplyMatrix

I/O and property setting of the node

Table 6.22: Parameter list of CMMultiplyMatrix

Parameter	Type	Default	Unit	Description
FIRST_FRAME_EXECUTION	`bool`	false		Selection of the first frame execution
ENABLE_DEBUG	`bool`	false		ON/OFF of debugging information output

Input

CMA : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. The M-th order complex square array correlation matrix inputs $NFFT/2+1$ items. `Matrix<complex<float> >` contains rows corresponding to frequencies ($NFFT/2+1$ rows) and the columns contains the complex correlation matrix ($M*M$ columns across).

CMB : `Matrix<complex<float> >` type. Same as CMA.

OPERATION_FLAG : `int` type or `bool` type. Only when this input terminal is 1 or true, calculation of the correlation matrix is performed.

Output

OUTPUTCM : `Matrix<complex<float> >` type. Correlation matrix equivalent to CMA $*$ CMB, after multiplication, is output.

Parameter

FIRST_FRAME_EXECUTION : `bool` type. The default value is false. When true, OPERATION_FLAG is always 0. Even if false, perform only the calculation of first frame.

ENABLE_DEBUG : `bool` type. Default value is false. When true, output the frame number calculated to the standard output, during correlation matrix multiplication.

Module Description

Performs the multiplication of two correlation matrices for each frequency bin. The correlation matrix is a complex three-dimensional array of size $k \times M \times M$ and the multiplication k times is performed as follows. Here, k is the number of frequency bins ($k = NFFT/2 + 1$) and M is a number of channels in the input signal.

```
OUTPUTCM = zero_matrix(k,M,M)

calculate{

   IF OPERATION_FLAG
       FOR i = 1 to k

           OUTPUTCM[i] = CMA[i] * CMB[i]

       ENDFOR
   ENDIF

}
```

The matrix that is output from the OUTPUTCM terminal is initialized as a zero matrix, and maintains the final operational result from this point onward.

6.2.10 CMIdentityMatrix

Module Overview

Outputs a correlation matrix containing a unit matrix.

Requested Files

None.

Usage

In what case is the node used?

Connect this node to the NOISECM input terminal of the LocalizeMUSIC node to tuen OFF the noise suppression function of LocalizeMUSIC .

Typical Examples

Figure. 6.20 shows a usage example of CMIdentityMatrix node.

Since this node generates correlation matrix data with a unit matrix for all frequency bins, the input terminal does not exist. It outputs the generated correlation matrix from the output terminal.

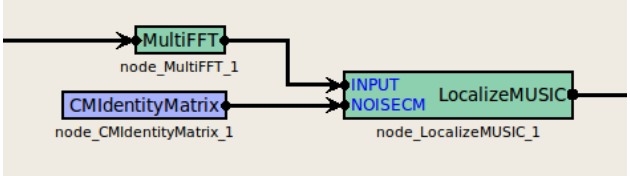


Figure 6.20: Network Example using CMIdentityMatrix

I/O and property setting of the node

Table 6.23: Parameter list of CMIdentityMatrix

Parameter	Type	Default	Unit	Description
NB_CHANNELS	`int`	8		Number of channels M of input signal
LENGTH	`int`	512		Frame length $NFFT$

Input

None.

Output

OUTPUTCM : `Matrix<complex<float> >` type. Correlation matrix for each frequency bin. the M-th order unit matrix correlation matrix outputs $NFFT/2+1$ items. `Matrix<complex<float> >` contains rows corresponding to the row of frequencies ($NFFT/2$ + 1 rows), and columns containing the complex correlation matrix ($M * M$ columns across).

Parameter

NB_CHANNELS : `int` type. Number of channels for input signal. Equivalent to the order of the correlation matrix. Must be matched with the order of the former correlation matrix used. Default value is 8.

LENGTH : `int` type. Default value is 512. FFT points at the time of Fourier transform. Must be matched till the former FFT length values.

Module Description

For an M-th order complex square array with a correlation matrix for each frequency bin, it stores the unit matrix, and corrects and outputs the result in `Matrix<complex<float> >` format.

6.2.11 ConstantLocalization

Outline of the node

A node that continuously outputs constant sound source localization results. There are four parameters used for this node, which are ANGLES, ELEVATIONS, POWER, and MIN_ID, and the user sets azimuths (ANGLES), elevation angles (ELEVATIONS), power (POWER), and IDs (MIN_ID) of the sound sources. Since each of these parameters is `Vector` , multiple localization results can be output.

Necessary file

No files are required.

Usage

When to use

This node 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 whether a problem is 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 the results of sound source separation.

Typical connection

Figure 6.21 shows a connection example. This network continuously displays constant localization results.

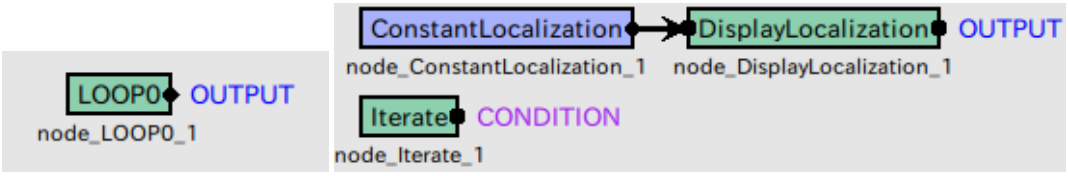


Figure 6.21: Connection example of ConstantLocalization

Input-output and property of the node

Input

No inputs.

Output

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

Parameter

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

Table 6.24: Parameter list of ConstantLocalization

Parameter name	Type	Default value	Unit	Description
ANGLES	Object	<Vector<float> >	[deg]	Azimuth (right or left) of the sound source
ELEVATIONS	Object	<Vector<float> >	[deg]	Elevation angle (up or down) of the sound source
POWER	Object	<Vector<float> >		Power of the sound sources
MIN_ID	int	0		IDs of the sound sources

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

POWER : Vector< float > type. This sets the power of sound sources. The unit is dB, same as the spatial spectrum calculated in LocalizeMUSIC . This parameter is not mandatory. If nothing defined, POWER is automatically set as 1.0.

MIN_ID : int type. This sets the minimum ID number allocated to each sound source. The IDs should be unique in order to distinguish each sound source. The IDs are allocated in order of elements specified in ANGLES and ELEVATIONS, whose number if started from MIN_ID. For instance, if `MIN_ID = 0` and `ANGLES = <Vector<float> 0 30>` are specified, the ID of 0° sound source will be zero, and the ID of 30° sound source will be one.

Details of the node

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) 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 ConstantLocalization actually outputs are values in the Cartesian coordinate system (x_i, y_i, z_i), which correspond to points on the unit ball. Conversion from the spherical coordinate system to the Cartesian coordinate system is performed based on the following equations.

$$x_i = \cos(a_i\pi/180)\cos(e_i\pi/180) \tag{6.5}$$
$$y_i = \sin(a_i\pi/180)\cos(e_i\pi/180) \tag{6.6}$$
$$z_i = \sin(e_i\pi/180) \tag{6.7}$$

Other than the coordinates of sound sources, ConstantLocalization also outputs the power (specified by POWER. If not set, fixed at 1.0) and ID (specified by MIN_ID + i) of the sound source.

6.2.12 DisplayLocalization

Outline of the node

This node displays sound source localization results using GTK.

Necessary file

No files are required.

Usage

When to use

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

Typical connection

This node is connected after localization nodes such as ConstantLocalization or LocalizeMUSIC . The example in Figure 6.22 continuously displays fixed localization results from ConstantLocalization .

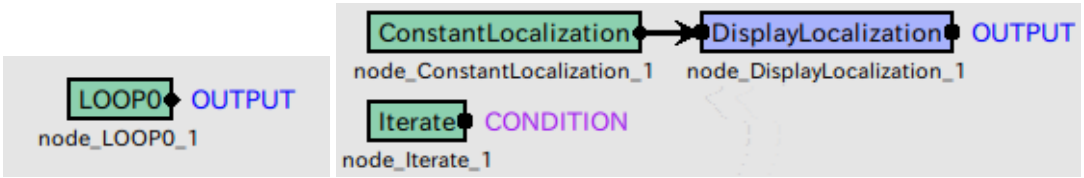


Figure 6.22: Connection example of DisplayLocalization

Input-output and property of the node

Input

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

Output

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

Parameter

WINDOW_NAME]: `string` type. Name of the GUI window.

WINDOW_LENGTH]: `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.

VERTICAL_RANGE : `Vector< int >` type. The plot range for the vertical axis. This is the vector of 2 elements. The first and second elements denote the minimum and maximum values of the data, respectively. In the default setting, namely `<Vector<int> -180 180>`, this module plots the azimuth data from -180[deg] to 180[deg].

Table 6.25: Parameter list of DisplayLocalization

Parameter name	Type	Default value	Unit	Description
WINDOW_NAME	string	Source Location		Name of the GUI window that displays
WINDOW_LENGTH	int	1000	Frame	Length of the GUI window that displays sound source localization results
VERTICAL_RANGE	Vector< int >	See below		Plot range for the vertical axis
PLOT_TYPE	string	AZIMUTH		Type of data for plotting

PLOT_TYPE : string type. The type of data for plotting. If AZIMUTH, this module plots the result of azimuth estimation. If ELEVATION, this module plots the result of elevation estimation.

Details of the node

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

6.2.13 LocalizeMUSIC

Outline of the node

From multichannel speech waveform data, direction-of-arrival (DOA) in the horizontal plane is estimated using the MUltiple SIgnal Classification (MUSIC) method. It is the main node for Sound Source Localization in HARK .

Necessary file

The transfer function file which consists of a steering vector is required. It is generated based on the positional relationship between the microphone and sound, or the transfer function for which measurement was performed.

Usage

This node estimates a sound's direction and amount of power using the MUSIC method. Detection of a direction with high power in each frame allows the system to know the direction of sound, the number of sound sources, the speech periods, etc. to some extent. The orientation result outputted from this node is used for post-processing such as tracking and source separation.

Typical connection

Figure 6.23 shows a typical connection example.

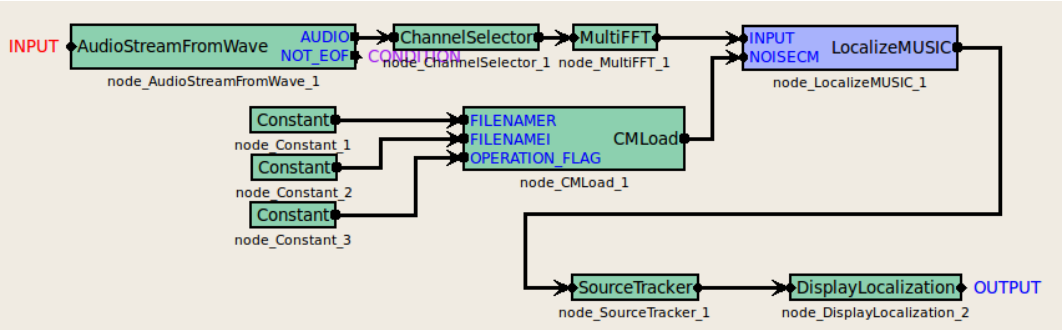


Figure 6.23: Connection example of LocalizeMUSIC

Input-output and property of the node

Input

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

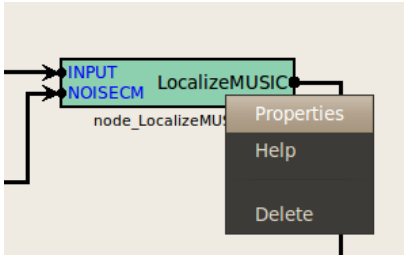
NOISECM : `Matrix<complex<float> >` type. The correlation matrix for each frequency bin. The $NFFT/2+1$ correlation matrices are inputted, corresponding to the M-th complex square matrix. The rows of `Matrix<complex<float> >` express frequency ($NFFT/2+1$ rows) and the columns express the complex correlation matrix ($M*M$ columns). This input terminal can also be left disconnected; then an identity matrix is used for the correlation matrix instead.

Output

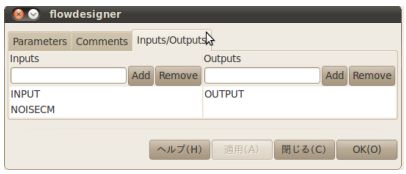
OUTPUT : Source position (direction) is expressed as `Vector<ObjectRef>` type. `ObjectRef` is a `Source` and is a structure which consists of the power of the MUSIC spectrum of the source and its direction. The element number of `Vector` is a sound number (N). Please refer to node details for the details of the MUSIC spectrum.

SPECTRUM : `Vector<float>` type. Power of the MUSIC spectrum for every direction. The output is equivalent to $\bar{P}(\theta)$ in Eq. (6.16). In case of three dimensional sound source localization, θ is a three dimensional vector, and $\bar{P}(\theta)$ becomes three dimensional data. Please refer to node details for the detail of the output format. This output terminal is not displayed by default.

Refer to Figure 6.24 for the addition method of hidden output.

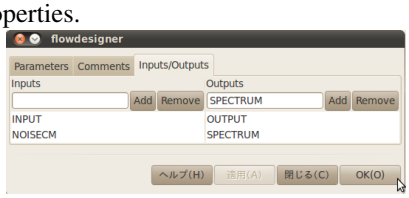


Step 1: Right-click LocalizeMUSIC and click Properties.

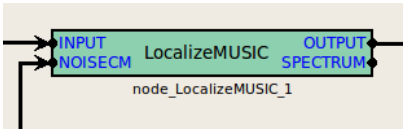


Step 2: Click Inputs/Outputs.

Step 3: Enter SPECTRUM in the input form of Outputs, click Add.



Step 4: Check that SPECTRUM has been added to the output list of Outputs, click O.K.



Step 5: The SPECTRUM output terminal is added to the node.

Figure 6.24: Usage example of hidden outputs : Display of SPECTRUM terminal

Table 6.26: Parameter list of LocalizeMUSIC

Parameter name	Type	Default value	Unit	description
MUSIC_ALGORITHM	`string`	SEVD		Algorithm of MUSIC
TF_CHANNEL_SELECTION	`Vector<int>`	See below.		Channel number used
LENGTH	`int`	512	[pt]	FFT points ($NFFT$)
SAMPLING_RATE	`int`	16000	[Hz]	Sampling rate
A_MATRIX	`string`			Transfer function file name
WINDOW	`int`	50	[frame]	Frames to normalize CM
WINDOW_TYPE	`string`	FUTURE		Frame selection to normalize CM
PERIOD	`int`	50	[frame]	The cycle to compute SSL
NUM_SOURCE	`int`	2		Number of sounds
MIN_DEG	`int`	-180	[deg]	Minimum azimuth
MAX_DEG	`int`	180	[deg]	Maximum azimuth
LOWER_BOUND_FREQUENCY	`int`	500	[Hz]	Lower bound frequency
UPPER_BOUND_FREQUENCY	`int`	2800	[Hz]	Upper bound frequency
SPECTRUM_WEIGHT_TYPE	`string`	Uniform		Type of frequency weight
A_CHAR_SCALING	`float`	1.0		Coefficient of weight
MANUAL_WEIGHT_SPLINE	`Matrix<float>`	See below.		Coefficient of spline weight
MANUAL_WEIGHT_SQUARE	`Matrix<float>`	See below.		Key point of rectangular weight
ENABLE_EIGENVALUE_WEIGHT	`bool`	`true`		Enable eigenvalue weight
DEBUG	`bool`	`false`		ON/OFF of debug output

Parameter

MUSIC_ALGORITHM : `string` type. Selection of algorithm used in order to calculate the signal subspace in the MUSIC method. SEVD represents standard eigenvalue decomposition, GEVD represents generalized eigenvalue decomposition, and GSVD represents generalized singular value decomposition. LocalizeMUSIC enters a correlation matrix with sound information from the NOISECM terminal, and possesses a function which can do SSL whitening of the noise (suppression). SEVD realizes SSL without the function. When SEVD is chosen, the input from NOISECM terminal is disregarded. Although both GEVD and GSVD have a function to whiten the noise inputted from the NOISECM terminal, GEVD has better noise suppression performance compared with GSVD. It has the problem that the calculation time takes approximately 4 times longer. Depending on the scene and computing environment, you can select one of the three algorithms. Please refer to node details for the details of algorithm.

TF_CHANNEL_SELECTION : `Vector<int>` type. Of steering vectors of multichannel stored in the transfer function file, it is parameters which chooses the steering vector of specified channel to use. The channel number begins from 0 like ChannelSelector . Signal processing of 8 channel is assumed by default and it is set as `<Vector<int> 0 1 2 3 4 5 6 7>` . It is necessary to align the number (M) of elements of the parameters with the channel number of incoming signals. Moreover, it is necessary to align the order of channel and the channel order of TF_CHANNEL_SELECTION to be inputted into INPUT terminal.

LENGTH : `int` type. 512 is the default value. FFT point in the case of fourier transform. It is necessary to align it with the FFT points to the preceding paragraph.

SAMPLING_RATE : `int` type. 16000 is the default value. Sampling frequency of input acoustic signal. It is necessary to align with other nodes like LENGTH.

A_MATRIX : `string` type. There is no default value. The file name of the transfer function file is designated. Both absolute path and relative path are supported. Refer to the harktool4 for the creation method of the transfer function file.

WINDOW : `int` type. 50 is the default value. The number of smoothing frames for correlation matrix calculation is designated. Within the node, the correlation matrix is generated for every frame from the complex spectrum of the input signal, and the addition mean is taken by the number of frames specified in WINDOW. Although the correlation matrix will be stabilized if this value is enlarged, time delays become long due to the long interval.

WINDOW_TYPE : `string` type. FUTURE is the default value. The selection of used smoothing frames for correlation matrix calculation. Let f be the current frame. If FUTURE, frames from f to $f + WINDOW - 1$ will be used for the normalization. If MIDDLW, frames from $f - (WINDOW/2)$ to $f + (WINDOW/2) + (WINDOW\%2) - 1$ will be used for the normalization. If PAST, frames from $f - WINDOW + 1$ to f will be used for the normalization.

PERIOD : `int` type. 50 is the default value. The cycle of SSL calculation is specified in frames number. If this value is large, the time interval for obtaining the orientation result becomes large, which will result in improper acquisition of the speech interval or bad tracking of the mobile sound. However, since the computational cost increases if it is small, tuning according to the computing environment is needed.

NUM_SOURCE : `int` type. 2 is the default value. It is the number of dimensions of the signal subspace in the MUSIC method, and can be practically interpreted as the number of desired sound sources to be emphasized in the peak detection. It is expressed as N_s in the following nodes details. It should be $1 \leq N_s \leq M - 1$. It is desirable to match the sound number of the desired sound, but, for example, in the case of the number of desired sound sources being 3, the interval that each sound pronounces is different, thus, it is sufficient to select a smaller value than it is practically.

MIN_DEG : `int` type. -180 is the default value. It is the minimum angle for peak search, and is expressed as θ_{min} in the node details. 0 degree is the robot front direction, negative values are the robot right hand direction, and positive values are the robot left-hand direction. Although the specified range is considered as ±180 degrees for convenience, since the surrounding range of 360 degrees or more is also supported, there is no particular limitation.

MAX_DEG : `int` type. 180 is the default value. It is the maximum angle for peak search, and is expressed as θ_{max} in the node details. Others are the same as that of MIN_DEG.

LOWER_BOUND_FREQUENCY : `int` type. 500 is the default value. It is the minimum of frequency bands which is taken into consideration for peak detection, and is expressed as ω_{min} in the node details. It should be $0 \leq \omega_{min} \leq$ SAMPLING_RATE/2.

UPPER_BOUND_FREQUENCY : `int` type. 2800 is the default value. It is the maximum of frequency bands Which is taken into consideration for peak detections, and, is expressed as ω_{max} below. It should be $\omega_{min} < \omega_{max} \leq$ SAMPLING_RATE/2.

SPECTRUM_WEIGHT_TYPE : `string` type. 'Uniform' is the default value. The distribution of weights against the frequency axial direction of the MUSIC spectrum used for peak detections is designated. 'Uniform' sets weights to OFF. 'A_Characteristic' gives the MUSIC spectrum weights imitating the sound pressure sensitivity of human hearing. 'Manual_Spline' gives the MUSIC spectrum weights suited to the Cubic spline curve for which the point specified in MANUAL_WEIGHT_SPLINE is considered as the interpolating point. 'Manual_Square' generates the rectangular weights suited to the frequency specified in MANUAL_WEIGHT_SQUARE, and gives it to MUSIC spectrum.

A_CHAR_SCALING : `float` type. 1.0 is the default value. This is scaling term which modifies the A characteristic weight on the frequency axis. Since the A characteristic weight imitates the sound pressure sensitivity of human's hearing, filtering to suppress sound outside of the speech frequency range is possible. Although the A characteristic weight has a standard, depending on the general sound environment, noise may enter the speech frequency range, and it may be unable to orientate well. Then, the A characteristic weight should be increased, causing more suppression, especially in the lower frequencies.

MANUAL_WEIGHT_SPLINE : `Matrix<float>` type.
`<Matrix<float> <rows 2> <cols 5> <data 0.0 2000.0 4000.0 6000.0 8000.0 1.0 1.0 1.0 1.0 1.0> >`
is the default value. It is designated with the float value 2-by-K matrix. K is equivalent to the number of interpolation points for spline interpolations. The first row specifies the frequency and the second row specifies the weight corresponding to it. Weighting is performed according to the spline curve which passes along the interpolated point. By default, the weights are all set to 1 for the frequency bands from 0 [Hz] to 8000 [Hz] .

MANUAL_WEIGHT_SQUARE : `Vector<float>` type. `<Vector<float> 0.0 2000.0 4000.0 6000.0 8000.0>`
is the default value. By the frequency specified in MANUAL_WEIGHT_SQUARE, the rectangular weight is generated and is given to MUSIC spectrum. For the frequency bands from the odd components of MANUAL_WEIGHT_SQUARE to the even components, the weight of 1 is given, and for the frequency bands from the even components to the odd components, the weight of 0 is given. By default, the MUSIC spectrum from 2000 [Hz] to 4000 [Hz] and 6000 [Hz] to 8000 [Hz] can be suppressed.

ENABLE_EIGENVALUE_WEIGHT : `bool` type. True is the default value. For true, in the case of calculation of the MUSIC spectrum, the weight is given as the square root of the maximum eigenvalue (or the maximum singular value) acquired from eigenvalue decomposition (or singular value decompositions) of the correlation matrix. Since this weight greatly changes depending on the eigenvalue of the correlation matrix inputted from NOISECM terminal when choosing GEVD and GSVD for MUSIC_ALGORITHM, it is good to choose false.

DEBUG : `bool` type. ON/OFF of the debug output and the format of the debug output are as follows. First, the set of index of sound, direction, and power is outputted in tab delimited for only several number of sound detected in frames. ID is the number given for convenience in order from 0 for every frame, though the number itself is meaningless. For direction [deg], an integer with rounded decimal is displayed. As for power, the power value of MUSIC spectrum $\bar{P}(\theta)$ of Eq. (6.16) is outputted as is. Next, "MUSIC spectrum:" is outputted after a line feed, and the value of $\bar{P}(\theta)$ of Eq. (6.16) is displayed for all θ.

Details of the node

The MUSIC method is the method of estimating the direction-of-arrival (DOA) utilizing the eigenvalue decomposition of the correlation matrix among input signal channels. The algorithm is summarized below.

Generation of transfer function :

In the MUSIC method, the transfer function from sound to each microphone is measured or calculated numerically and it is used as a priori information. If the transfer function in the frequency domain from sound $S(\theta)$ in direction θ in view of microphone array to the i-th microphone M_i is set to $h_i(\theta, \omega)$, the multichannel transfer function multichannel can be expressed as follows.

$$\boldsymbol{H}(\theta, \omega) = [h_1(\theta, \omega), \cdots, h_M(\theta, \omega)] \tag{6.8}$$

This transfer function vector is prepared for every suitable interval delta, $\Delta\theta$ (non-regular intervals are also possible) by calculation or measurement in advance. In HARK , harktool4 is offered as a tool which can generate the transfer function file also by numerical calculation and also by measurement. Please refer to the paragraph of harktool4 for the prepare a specific transfer function file (From harktool4, we can create the database of three dimensional transfer functions for three dimensional sound source localization). In the LocalizeMUSIC node, this a priori information file (transfer function file) is imported and used with the file name specified in A_MATRIX. Thus, since the transfer function is prepared for every direction of sound and is scanned to the direction using the direction vector (or transfer function, in the case of orientation), it is sometimes called 'steering vector'.

Calculation of correlation matrix between the inputs signal channels :

The operation by HARK begins from here. First, the signal vector in the frequency domain obtained by short-time fourier transform of the input acoustic signal in M channel is found as follows.

$$\boldsymbol{X}(\omega, f) = [X_1(\omega, f), X_2(\omega, f), X_3(\omega, f), \cdots, X_M(\omega, f)]^T, \tag{6.9}$$

where ω expresses frequency and f expresses frame index. In HARK , the process so far is performed by the MultiFFT node in the preceding paragraph.

The correlation matrix between channels of the incoming signal $\boldsymbol{X}(\omega, f)$ can be defined as follows for every frame and for every frequency .

$$\boldsymbol{R}(\omega, f) = \boldsymbol{X}(\omega, f)\boldsymbol{X}^*(\omega, f) \tag{6.10}$$

where $()^*$ represents the conjugate transpose operator. If this $\boldsymbol{R}(\omega, f)$ is utilized in following processing as is, theoretically, it will be satisfactory, but practically, in order to obtain the stable correlation matrix, those time averaging is used in HARK .

$$\boldsymbol{R}'(\omega, f) = \frac{1}{\mathrm{WINDOW}} \sum_{i=W_i}^{W_f} \boldsymbol{R}(\omega, f + i) \tag{6.11}$$

The frames used for the averaging can be changed by WINDOW_TYPE. If WINDOW_TYPE=FUTURE, $W_i = 0$ and $W_f = \mathrm{WINDOW} - 1$. If WINDOW_TYPE=MIDDLE, $W_i = \mathrm{WINDOW}/2$ and $W_f = \mathrm{WINDOW}/2 + \mathrm{WINDOW}\%2 - 1$. If WINDOW_TYPE=PAST, $W_i = -\mathrm{WINDOW} + 1$ and $W_f = 0$.

Decomposition to the signal and noise subspace :

In the MUSIC method, an eigenvalue decomposition or singular value decomposition of the correlation matrix $\boldsymbol{R}'(\omega, f)$ found in the Eq. (6.11) is performed and the M-th space is decomposed into the signal subspace and the other subspace.

Since the processing has high computational cost, it is designed to be calculated only once in several frames. In LocalizeMUSIC , this operation period can be specified in PERIOD.

In LocalizeMUSIC , the method for decomposing into subspace can be specified by MUSIC_ALGORITHM.

When MUSIC_ALGORITHM is specified for SEVD, the following standard eigenvalue decomposition is performed.

$$\boldsymbol{R}'(\omega, f) = \boldsymbol{E}(\omega, f)\boldsymbol{\Lambda}(\omega, f)\boldsymbol{E}^{-1}(\omega, f)\,, \tag{6.12}$$

where $\boldsymbol{E}(\omega, f)$ represents the matrix $\boldsymbol{E}(\omega, f) = [\boldsymbol{e}_1(\omega, f), \boldsymbol{e}_2(\omega, f), \cdots, \boldsymbol{e}_M(\omega, f)]$ which consists of singular vectors which perpendicularly intersect each other, and $\boldsymbol{\Lambda}(\omega)$ represents the diagonals matrix using the eigenvalue corresponding to individual eigenvectors as the diagonal component. In addition, the diagonal component of $\boldsymbol{\Lambda}(\omega)$, $[\lambda_1(\omega), \lambda_2(\omega), \ldots, \lambda_M(\omega)]$ is considered to have been sorted in descending order.

When MUSIC_ALGORITHM is specified for GEVD, the following generalized eigenvalue decomposition is performed.

$$\boldsymbol{K}^{-\frac{1}{2}}(\omega, f)\boldsymbol{R}'(\omega, f)\boldsymbol{K}^{-\frac{1}{2}}(\omega, f) = \boldsymbol{E}(\omega, f)\boldsymbol{\Lambda}(\omega, f)\boldsymbol{E}^{-1}(\omega, f)\,, \tag{6.13}$$

where $\boldsymbol{K}(\omega, f)$ represents the correlation matrix inputted from NOISECM terminal at the f-th frame. Since large eigenvalues from the noise sources included in $\boldsymbol{K}(\omega, f)$ can be whitened (surpressed) using generalized eigenvalue decomposition with $\boldsymbol{K}(\omega, f)$, SSL with suppressed noise is realizable.

When MUSIC_ALGORITHM is specified for GSVD, the following generalized singular value decomposition is performed.

$$\boldsymbol{K}^{-1}(\omega, f)\boldsymbol{R}'(\omega, f) = \boldsymbol{E}(\omega, f)\boldsymbol{\Lambda}(\omega, f)\boldsymbol{E}_r^{-1}(\omega, f)\ , \tag{6.14}$$

where $\boldsymbol{E}(\omega, f)$, $\boldsymbol{E}_r(\omega, f)$ represents the matrix which consists of left singular vector and right singular vector, respectively, and $\boldsymbol{\Lambda}(\omega)$ represents the diagonal matrix using each singular-value as the diagonal components.

Since the eigenvalue (or singular-value) corresponding to eigen vector space $\boldsymbol{E}(\omega, f)$ obtained by degradation has correlation with the power of sound, by taking eigen vector corresponding to the eigenvalue with the large value, only the subspace of loud desired sound with large power can be chosen. If the number of sounds to be considered is set to N_s, then eigen vector $[e_1(\omega), \cdots, e_{N_s}(\omega)]$ corresponds to the sound, are eigen vector $[e_{N_s+1}(\omega), \cdots, e_M(\omega)]$ corresponds to noise. In LocalizeMUSIC , N_s can be specified as NUM_SOURCE.

Calculation of MUSIC spectrum :

The MUSIC spectrum for SSL is calculated as follows using only noise-related eigen vectors.

$$P(\theta, \omega, f) = \frac{|\boldsymbol{H}^*(\theta, \omega)\boldsymbol{H}(\theta, \omega)|}{\sum_{i=N_s+1}^{M} |\boldsymbol{H}^*(\theta, \omega)e_i(\omega, f)|} \tag{6.15}$$

In the denominator in the right-hand side, the inner product of the noise-related eigen vector and steering vector is calculated. On the space spanned by the eigen vector, since the noise subspace corresponding to small eigenvalue and the signal subspace corresponding to a large eigenvalue intersect perpendicularly each other, if the transfer function is a vector corresponding to the desired sound, this inner product will be 0 theoretically. Therefore, $P(\theta, \omega, f)$ diverges infinitely. In fact, although it does not diverge infinitely under the effect of noise etc., a sharp peak is observed compared to beam forming. The right-hand side of the numerator is an normalization term.

Since $P(\theta, \omega, f)$ is MUSIC spectrum obtained for every frequency, broadband SSL is performed as follows.

$$\bar{P}(\theta, f) = \sum_{\omega=\omega_{min}}^{\omega_{max}} W_\Lambda(\omega, f)W_\omega(\omega, f)P(\theta, \omega, f)\ , \tag{6.16}$$

where ω_{min} and ω_{max} show the minimum and maximum of the frequency bands which are handled in the broadband integration of MUSIC spectrum, respectively, and they can be specified as LOWER_BOUND_FREQUENCY and UPPER_BOUND_FREQUENCY in LocalizeMUSIC , respectively.

Moreover, $W_\Lambda(\omega, f)$ is the eigen-value weight in the case of broadband integration and is square root of the maximum eigenvalue (or maximum singular-value).

In LocalizeMUSIC , the presence or absence of eigenvalue weight can be chosen by ENABLE_EIGENVALUE_WEIGHT, and in case of false, it is $W_\Lambda(\omega, f) = 1$ and in case of true, it is $W_\Lambda(\omega, f) = \sqrt{\lambda_1(\omega, f)}$. Moreover, $W_\omega(\omega, f)$ is the frequency weight in the case of broadband integration, and the type can be specified as follows by SPECTRUM_WEIGHT_TYPE in LocalizeMUSIC .

- In the case that SPECTRUM_WEIGHT_TYPE is Uniform
  weights become uniform and $W_\omega(\omega, f) = 1$ all frequency bins.

- In the case that SPECTRUM_WEIGHT_TYPE is A_Characteristic
  it will be A characteristic weight $\mathcal{W}(\omega)$ which the International Electrotechnical Commission standardizes. The frequency characteristics of A characteristic weight is shown in Figure 6.25. The horizontal axis is ω and the vertical axis is $\mathcal{W}(\omega)$. In LocalizeMUSIC , the scaling term A_CHAR_SCALING of frequency direction is introduced to the frequency characteristic. If A_CHAR_SCALING is set as α, then the frequency weight actually used can be expressed as $\mathcal{W}(\alpha\omega)$. In Figure 6.25, the case of $\alpha = 1$ and the case of $\alpha = 4$ are plotted as an example. The weight finally applied to the MUSIC spectrum is $W_\omega(\omega, f) = 10^{\frac{\mathcal{W}(\alpha\omega)}{20}}$. As an example, $W_\omega(\omega, f)$ when A_CHAR_SCALING=1 is shown in Figure 6.26.

- When SPECTRUM_WEIGHT_TYPE is Manual_Spline
  it is the frequency weight in line with the curve in which the spline interpolation was carried out for the interpolating point specified in MANUAL_WEIGHT_SPLINE. MANUAL_WEIGHT_SPLINE is specified with the

`Matrix<float>` type of 2-by-k matrix. The first row represents the frequency and the second row represents the weight for the frequency. The interpolation mark k may be any point. As an example, in the case that MANUAL_WEIGHT_SPLINE is
`<Matrix<float> <rows 2> <cols 3> <data 0.0 4000.0 8000.0 1.0 0.5 1.0> >`
the number of interpolation points is 3, and the spline curve to which the weight of 1, 0.5, and 1 is applied in three frequencies, 0, 4000, and 8000[Hz], on the frequency axis, respectively can be created. $W_\omega(\omega, f)$ at that time is shown in Figure 6.27.

- When SPECTRUM_WEIGHT_TYPE is Manual_Square
it is the frequency weight in line with the rectangular weight from which the rectangle changes at the frequency specified in MANUAL_WEIGHT_SQUARE. MANUAL_WEIGHT_SQUARE is specified in the k-th `Vector<float>` type, and expresses the frequency to switch the rectangle. The number k of the switching point is arbitrary. As an example, the rectangle weight $W_\omega(\omega, f)$ in the case that MANUAL_WEIGHT_SQUARE is considered as
`<Vector<float> 0.0 2000.0 4000.0 6000.0 8000.0>`
is shown in Figure 6.28. By using this weight, two or more frequency domains which cannot be specified with only UPPER_BOUND_FREQUENCY and LOWER_BOUND_FREQUENCY can be chosen.

The output port SPECTRUM outputs the result of $\bar{P}(\theta, f)$ in Eq. (6.16) as a one dimensional vector. In case of three dimensional sound source localization, $\bar{P}(\theta, f)$ becomes three dimensional data, and $\bar{P}(\theta, f)$ is converted to one dimensional vector and output from the port. Let `Ne`, `Nd`, and `Nr` denote the number of elevation, the number of azimuth, and the number of radius, respectively. Then, the conversion is described as follows.

```
FOR ie = 1 to Ne
  FOR id = 1 to Nd
    FOR ir = 1 to Nr
      SPECTRUM[ir + id * Nr + ie * Nr * Nd] = P[ir][id][ie]
    ENDFOR
  ENDFOR
ENDFOR
```

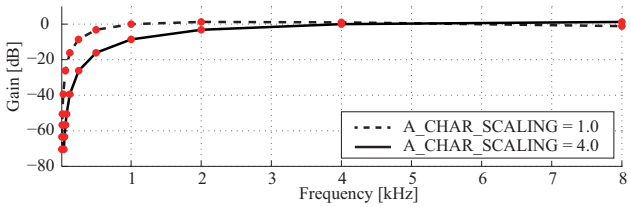


Figure 6.25: Frequency characteristic of characteristic weight when considering as SPECTRUM_WEIGHT_TYPE=A_Characteristic

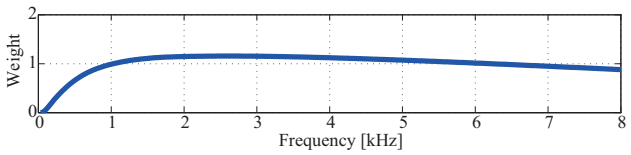


Figure 6.26: $W_\omega(\omega, f)$ in case of SPECTRUM_WEIGHT_TYPE=A_Charasteristic and A_CHAR_SCALING=1

Search of sound :

Next, the local peak is detected from the range in θ_{min} to θ_{max} for $\bar{P}(\theta)$ of Eq. (6.16), and the power of the MUSIC spectrum corresponding to DoA for the top N_s are outputted in descending order of the value. Moreover, the number of output sound sources may become below when the number of peaks does not reach to N_s. In LocalizeMUSIC , θ_{min}

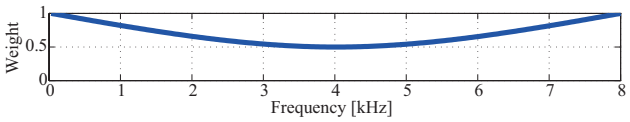


Figure 6.27: $W_\omega(\omega, f)$ in case of SPECTRUM_WEIGHT_TYPE=Manual Spline

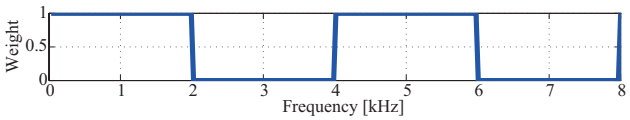


Figure 6.28: $W_\omega(\omega, f)$ in case of SPECTRUM_WEIGHT_TYPE=Manual_Square

and θ_{max} of azimuth can be specified in MIN_DEG and MAX_DEG, respectively. The module uses all elevation and radius for the sound source search.

Discussion :

Finally, we describe the effect that whitening (noise suppression0 has on MUSIC spectrum in Eq. (6.15) when choosing GEVD and GSVD for MUSIC_ALGORITHM.

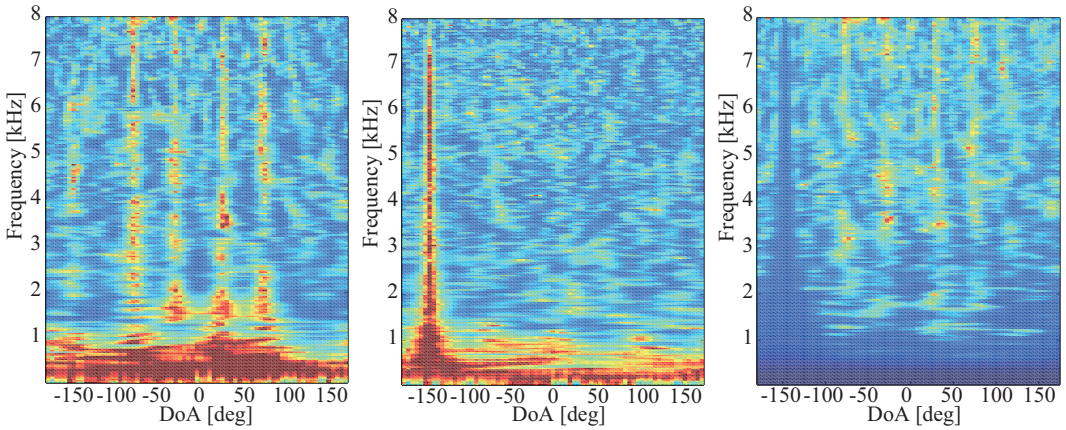
Here, as an example, consider the situation of four speakers (Directions = 75[deg], 25[deg], -25[deg], and -75[deg]) speaking simultaneously.

Figure 6.29(a) shows the result of choosing SEVD for MUSIC_ALGORITHM and not having whitened the noise. The horizontal axis is the azimuth, the vertical axis is frequency, and the value is $P(\theta, \omega, f)$ of the Eq. (6.15). As shown in the figure, there is diffusion noise in the low frequency domain and -150 degree direction, which reveals that the peak is not correctly detectable to only the direction of the 4 speakers.

Figure 6.29(b) shows the MUSIC spectrum in the interval in which SEVD is chosen for MUSIC_ALGORITHM and 4 speakers do not perform speech. The diffusion noise and the direction noise observed can be seen in Figure 6.29(a).

Figure 6.29(c) is the MUSIC spectrum when generating $\boldsymbol{K}(\omega, f)$ from the information on Figure 6.29(b), choosing GSVD for MUSIC_ALGORITHM as general sound information, and whitening the noise. As shown in the figure, it can be seen that the diffusion noise and the direction noise contained in $\boldsymbol{K}(\omega, f)$ are suppressed correctly and the strong peaks are only in the direction of the 4 speakers.

Thus, it is useful to use GEVD and GSVD for known noise.



(a) MUSIC spectrum when MUSIC_ALGORITHM=SEVD (four speakers)

(b) MUSIC spectrum of the noise by generating $\boldsymbol{K}(\omega, f)$ (zero speaker)

(c) MUSIC spectrum when MUSIC_ALGORITHM=GSVD (four speakers)

Figure 6.29: Comparison of MUSIC spectrum

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.

(3) K. Nakamura, K. Nakadai, F. Asano, Y. Hasegawa, and H. Tsujino, "Intelligent Sound Source Localization for Dynamic Environments", in *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS 2009)*, pp. 664–669, 2009.

6.2.14 LoadSourceLocation

Outline of the node

This node reads the source localization results stored in the SaveSourceLocation node.

Necessary file

Files in the formats saved from SaveSourceLocation .

Usage

When to use

This node is used when wishing to reuse a source localization result or to evaluate different sound source separation methods based on the exact same source localization result.

Typical connection

Figure 6.30 shows a network that reads and displays saved localization results.

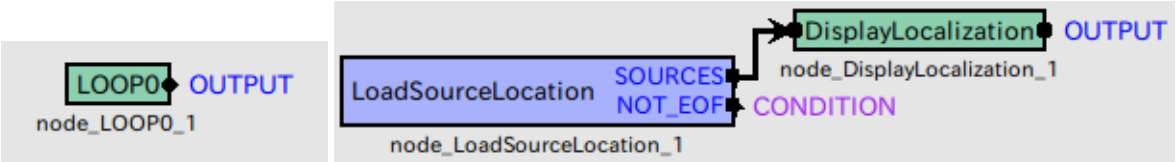


Figure 6.30: Connection example of LoadSourceLocation

Input-output and property of the node

Input

No inputs.

Output

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

NOT_EOF : `bool` type. Since this outputs `false` when the file has been read till the end, this terminal can be used as a termination condition of an Iteration subnetwork.

Parameter

Table 6.27: Parameter list of LoadSourceLocation

Parameter name	Type	Default value	Unit	Description
FILENAME	`string`			File name of the file to be read

FILENAME : `string` type. The name of the file to be read.

Details of the node

The source localization result output by this node is a `Vector` of objects containing 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.

6.2.15 SaveSourceLocation

Outline of the node

This nodes saves source localization results.

Necessary file

No files are required

Usage

When to use

This node is used when a localization result is saved in a file for analysis of the localization result later. To read the saved file, the LoadSourceLocation node is used.

Typical connection

Figure 6.31 shows a typical connection example. The network shown in the example saves fixed localization results in files. For the save format, see 5.3.4. Moreover, this node can be connected to nodes that output localization results (e.g. LocalizeMUSIC , ConstantLocalization or LoadSourceLocation).

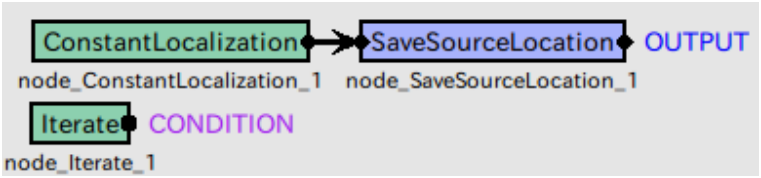


Figure 6.31: Connection example of SaveSourceLocation

Input-output and property of the node

Input

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

Output

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

Parameter

FILENAME : `string` type. No default values.

Table 6.28: Parameter list of SaveSourceLocation

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

Details of the nodes

Typical error messages and their reasons

`FILENAME is empty` Node parameter FILENAME is not assigned.

`Can't open file name` File open for write failed because, for example, the file is not writable.

6.2.16 SourceIntervalExtender

Outline of the node

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

Necessary file

No files are required

Usage

When to use

This node 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 beginning of the separated sound is cut by this delay time. This node is used to prevent this problem.

Typical connection

Figure 6.32 shows a typical connection example. As shown in the figure, when wishing to separate sound based on a localization result, the starting delay of sound source separation can be avoided by inserting the SourceIntervalExtender node in between.

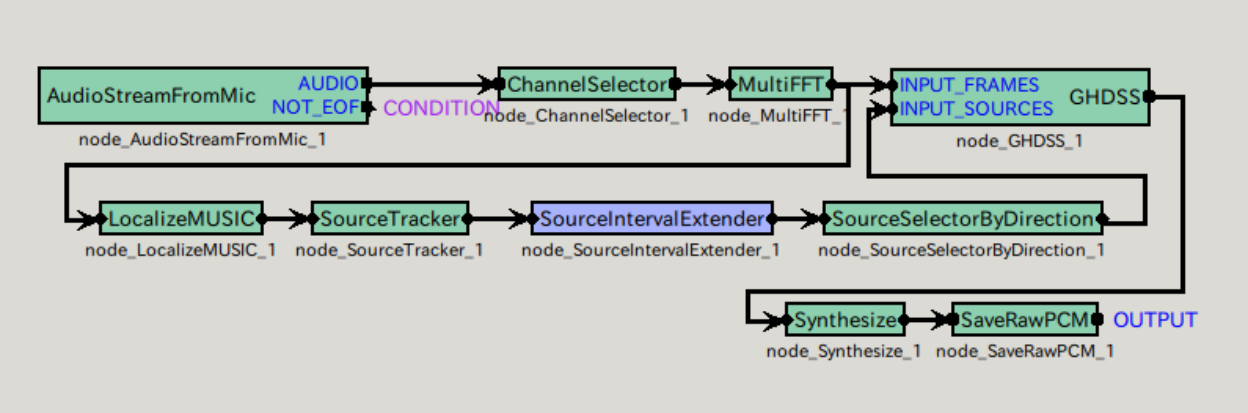


Figure 6.32: Connection example of SourceIntervalExtender

Input-output and property of the node

Input

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

Output

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

Parameter

Table 6.29: Parameter list of SourceIntervalExtender

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

PREROLL_LENGTH : `int` type. This value determines how much sooner the localization result is output. When this value is too small, the start of sound source separation is delayed and therefore users should set this value considering the delay of the source location method used in the following paragraph.

Details of the node

When sound source separation is performed based on a localization result without SourceIntervalExtender , the beginning part of the separated sound is cut by the processing time of the source localization as shown in Figure 6.33. In the case of speech recognition in particular, since the cut-off of the beginning part influences recognition performance, it is necessary to output a localization result beforehand with this node. This node reads ahead in each repeat for the length set at PREROLL_LENGTH. When a localization result is identified, the output of the localization result is started from the point in time when it was identified. (See Figure 6.34)

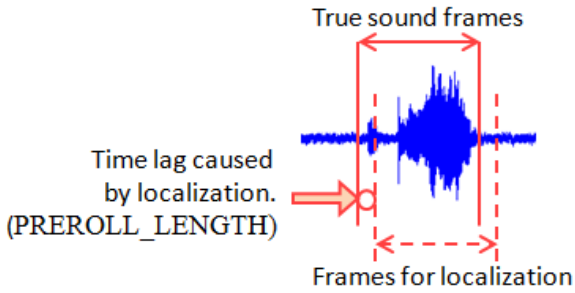


Figure 6.33: Necessity to output a source location result sooner than usual

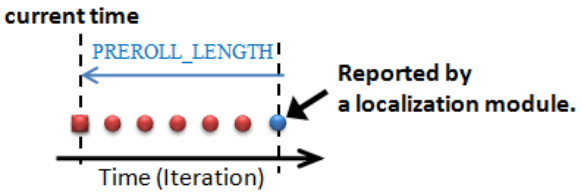


Figure 6.34: Operation of SourceIntervalExtender

6.2.17 SourceTracker

Outline of the node

For source localization results input in time series without IDs, this node gives the same ID to the source localization result obtained from an adjacent direction and a different ID to source localization results obtained from different directions. After running this node, 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 coming from the same sound source, it is necessary to track the source location results. SourceTracker uses an algorithm that gives the same ID to source localization results when 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 with this node, which enables to perform processing for each ID.

Typical connection

Usually, the outputs of source localization nodes such as ConstantLocalization or LocalizeMUSIC are connected to the input terminal of this node. Then an appropriate ID is added to a localization result so users can connect it to the sound source separation module GHDSS or the presentation node for source location results (DisplayLocalization), which are based on source localization. Figure 6.35 shows a connection example. Here, a fixed source location result is displayed through SourceTracker . In this case, if the localization result that ConstantLocalization outputs is close to another, they are output together in one sound source. When giving the following property to ConstantLocalization in the figure, the angle between the two sound sources is less than 20[deg], which is the default value of MIN_SRC_INTERVAL and therefore only one sound source is presented.

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

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

See ConstantLocalization for the meaning of the set points

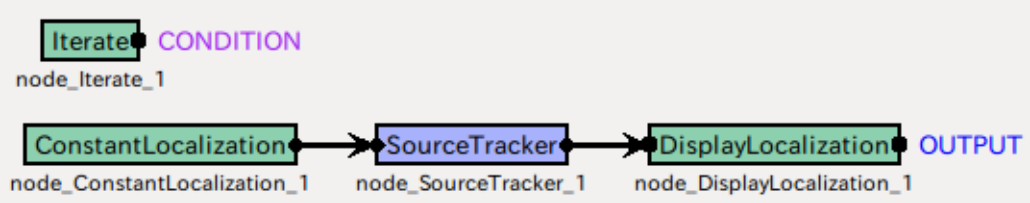


Figure 6.35: Connection example of SourceTracker

Input-output and property of the node

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 sound sources positioned near to another

Parameter

Table 6.30: Parameter list of SourceTracker

Parameter name	Type	Default value	Unit	Description
THRESH	`float`			To be ignored if the power of the sound source is smaller than THRESH.
PAUSE_LENGTH	`float`	800	[frame*10]	Length when assuming that the localized sound continues.
MIN_SRC_INTERVAL	`float`	20	[deg]	Threshold value of angular difference for judging that the sound source is same as another.
MIN_ID	`int`	0		
DEBUG	`bool`	false		

THRESH : `float` type. This parameter judges by the power whether the source localization result is noise to be ignored. The result is considered noise if the power is smaller than THRESH, and the localization result is not sent to the output. When THRESH is too small, noise is sent to output, 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.

PAUSE_LENGTH : `float` type. This parameter determines how long the sound source once output as a localization result. For a direction that is localized once, even though there are no valid source localization results after the first localization, localization results for that direction continue being output during a period of PAUSE_LENGTH / 10 [frame].

Since the default value is 800, for a direction that is localized once, localization results continue being output for 80 [frame] after the first localization.

MIN_SRC_INTERVAL : `float` type. If the source location result is smaller than MIN_SRC_INTERVAL, 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 node

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 p, and the Cartesian coordinates on a unit ball corresponding to the sound source direction are x, y, z. Assuming N is the number of sound sources that a node presently maintains and M is that of the sound sources newly input, they are distinguished by the subscripts ^{last} and ^{cur}. For example, power of the i th newly input sound source 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 of $\boldsymbol{q}_1 = (x_1, y_1, z_1)$ and $\boldsymbol{q}_2 = (x_2, y_2, z_2)$, the angle θ is expressed as follows.

$$\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 = |\boldsymbol{q}_1||\boldsymbol{q}_2| \cos\theta \tag{6.17}$$

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

$$\theta = \cos^{-1}\left(\frac{\boldsymbol{q}_1 \cdot \boldsymbol{q}_2}{|\boldsymbol{q}_1||\boldsymbol{q}_2|}\right) = \cos^{-1}\left(\frac{x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}}\right) \tag{6.18}$$

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 SourceTracker performs in sound source tracking is shown in Figure 6.36. 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 node 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 node already has, if they are sufficiently close ($=\theta_{ij}$ is below MIN_SRC_INTERVAL[deg]), they are integrated. The same ID is given to the integrated sound sources and the duration r^{last} is reset at PAUSE_LENGTH. 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 MIN_SRC_INTERVAL [deg] are judged as different sound sources. For the source positions that the node 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 have disappeared and is deleted. If the source position newly input is different from any of those that the node already has, a new ID is given to the sound source and r^{cur} is initialized at PAUSE_LENGTH.

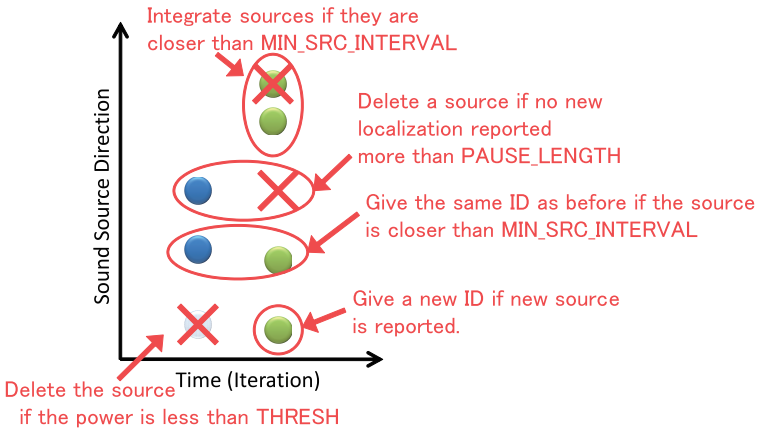


Figure 6.36: Sound source tracking method of SourceTracker

6.3 Separation category

6.3.1 BGNEstimator

Outline of the node

This node estimates stationary noise (or BackGround Noise) such as fan noise contained in signals, based on the power spectra of multichannel signals. The estimated stationary noise is used in the PostFilter node.

Necessary files

No files are required.

Usage

When to use

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

Typical connection example

A connection example of the BGNEstimator node is shown in Figure 6.37. As input, the user enters the power spectra that are obtained by converting speech waveforms into the frequency domain. The outputs are used in the PostFilter node.

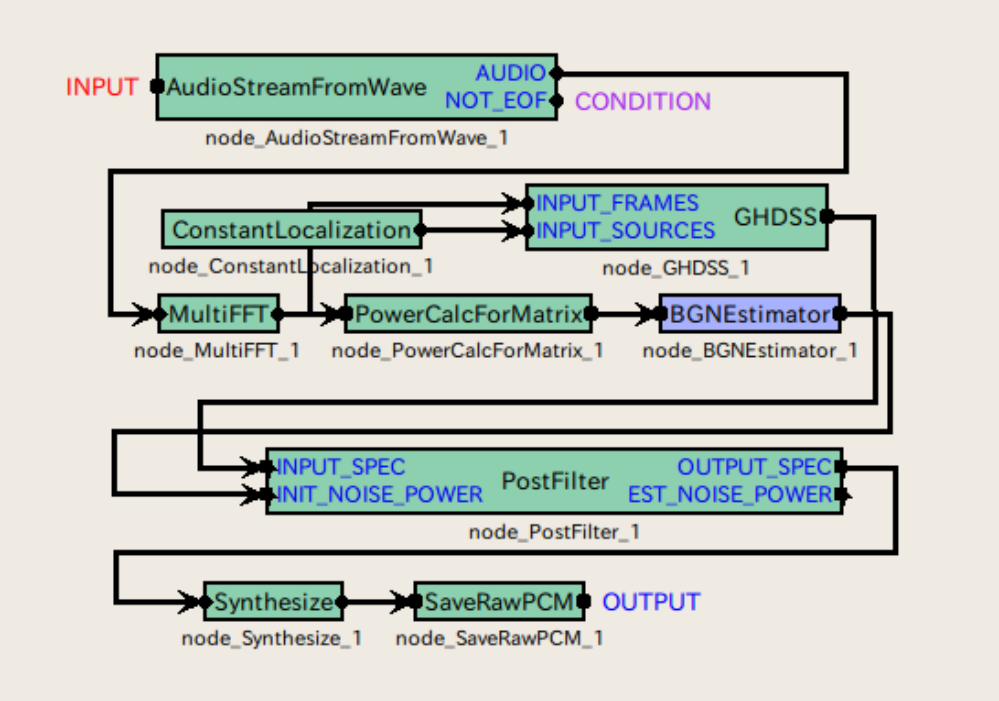


Figure 6.37: Connection example of BGNEstimator

Input-output and property of the node

Table 6.31: 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
ALPHA_S	float	0.7		Smoothing coefficient of input signal
NOISE_COMPENS	float	1.0		Mix rate of stationary noise
ALPHA_D_MIN	float	0.05		The minimum value of smoothing coefficient
NUM_INIT_FRAME	int	100	[frame]	Number of initialization frames

Input

INPUT_POWER : Matrix<float> type. Multichannel power spectrum

Output

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

Parameter

DELTA : float type. The default 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 : int type. The default value is 150. This is the amount of time to hold the minimum spectrum in history (stationary noise component), which is the criterion for determining the target sound. This parameter is designated in the AudioStreamFromWave node as the number of shifts for the parameter ADVANCE.

ALPHA_S : float type. The default value is 0.7. The coefficient when smoothing input signals in a temporal direction. The greater the value, the greater we weight the past frame value during smoothing.

NOISE_COMPENS : float type. 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).

ALPHA_D_MIN : float type. 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 the smoothing processing of stationary noise.

NUM_INIT_FRAME : int type. 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 node

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 6.38.

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 $\boldsymbol{\lambda}(f-1, k_i)$.

$$S_m^{smo,t}(f, k_i) = \alpha_s \lambda_m(f-1, k_i) + (1-\alpha_s) S_m(f, k_i) \tag{6.19}$$

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

$$S_m^{smo}(f, k_i) = 0.25 S_m^{smo}(f, k_{i-1}) + 0.5 S_m^{smo}(f, k_i) + 0.25 S_m^{smo}(f, k_{i+1}) \tag{6.20}$$

Table 6.32: Variable

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

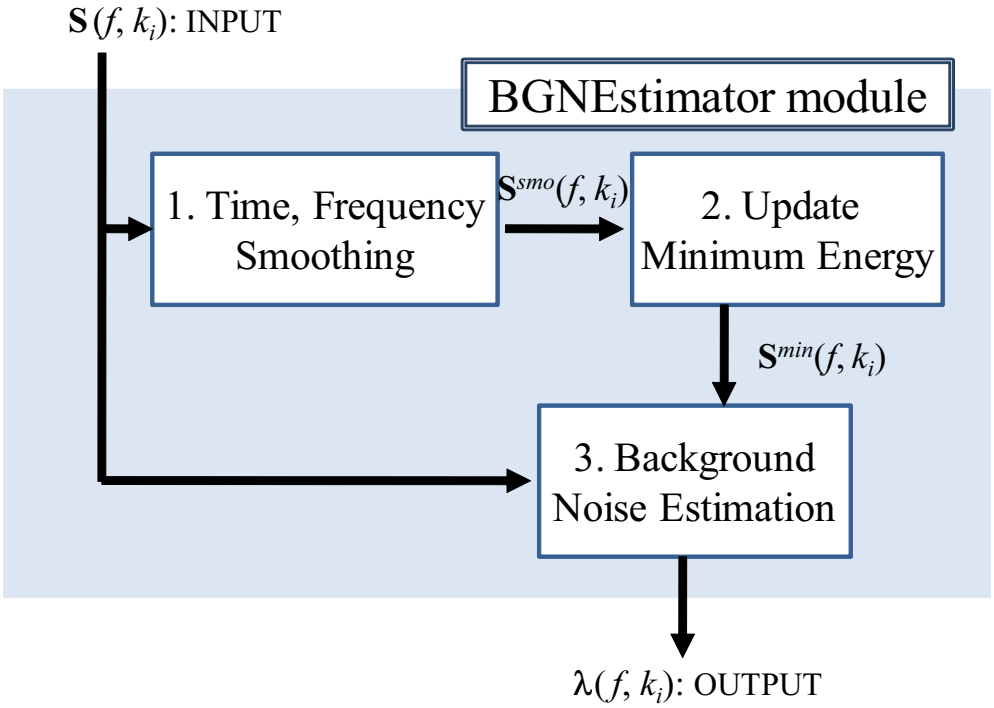


Figure 6.38: Flow of stationary noise estimation

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.

$$S_m^{tmp}(f,k_i) = \begin{cases} S_m^{smo}(f,k_i), & \text{if } f = nL \\ \min\{S_m^{tmp}(f-1,k_i), S_m^{smo}(f,k_i), & \text{if } f \neq nL \end{cases} \tag{6.21}$$

$$S_m^{min}(f,k_i) = \begin{cases} \min\{S_m^{tmp}(f-1,k_i), S_m^{smo}(f,k_i), & \text{if } f = nL \\ \min\{S_m^{min}(f-1,k_i), S_m^{smo}(f,k_i), & \text{if } f \neq nL \end{cases} \tag{6.22}$$

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.

$$S_m^{smo}(f,k_i) < \delta S_m^{min}(f,k_i) \text{or} \tag{6.23}$$
$$f < N \text{ or} \tag{6.24}$$
$$S_m^{smo}(f,k_i) < \lambda_m(f-1,k_i) \tag{6.25}$$

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

$$\alpha_d = \begin{cases} \frac{1}{f+1}, & \text{if } (\frac{1}{f+1} \geq \alpha_d^{min}) \\ \alpha_d^{min}, & \text{if } (\frac{1}{t+1} < \alpha_d^{min}) \\ 0 & \text{(When the target sound is contained)} \end{cases} \tag{6.26}$$

Steady noise is obtained by the following equations.

$$\lambda_m(f,k_i) = (1-\alpha_d)\lambda_m(f-1,k_i) + \alpha_d \theta S_m(f,k_i) \tag{6.27}$$

6.3.2 CalcSpecSubGain

Outline of the node

This node determines an optimum gain for how much of the power spectrum of estimated noise is to be removed from a power spectrum including signals + noise. Further, it outputs the probability of speech presence (See Section 6.3.7). However, this node constantly outputs 1 as the probability of speech presence. It outputs the difference between a separated sound's power spectrum and the estimated noise's power spectrum.

Necessary file

No files are required.

Usage

When to use

It is used when performing noise estimation with the HRLE node.

Typical connection

Figure 6.39 shows a connection example of CalcSpecSubGain . Inputs are power spectra after separation with GHDSS and those of the noise estimated in HRLE . Its outputs connect VOICE_PROB and GAIN to SpectralGain-Filter .

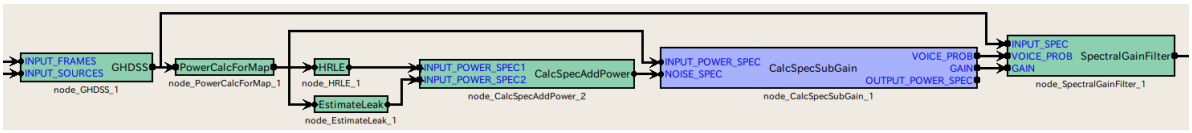


Figure 6.39: Connection example of CalcSpecSubGain

Input-output and property of the node

Table 6.33: Parameters of CalcSpecSubGain

Parameter name	Type	Default value	Unit	Description
ALPHA	`float`	1.0		Gain for spectral subtraction
BETA	`float`	0.0		Floor for GAIN
SS_METHOD	`int`	2		Selection of power/amplitude spectra

Input

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

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

Output

VOICE_PROB : `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.

OUTPUT_POWER_SPEC : `Map<int, ObjectRef>` type. `Vector<float>` type data pair of the sound source ID and the power spectrum of the separated sound with the estimated noise deducted.

Parameter

ALPHA : `float` type. Gain for spectral subtraction.

BETA : `float` type. Spectral floor.

SS_METHOD : `int` type. Selection of power/amplitude spectra for the spectral subtraction.

Details of the node

This node determines an optimum gain for how much of the estimated noise's 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 the probability of speech presence. (See Section 6.3.7.) However, this node constantly outputs 1 as the probability of speech presence. It outputs the difference of the power spectrum of the separated sound and that of the estimated noise. Assuming that the power spectrum from which noise was is $Y_n(k_i)$, the 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 OUTPUT_POWER_SPEC is expressed as follows.

$$Y_n(k_i) \quad = \quad X_n(k_i) - N_n(k_i) \tag{6.28}$$

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

$$G_n(k_i) \quad = \quad \begin{cases} \text{ALPHA}\frac{Y_n(k_i)}{X_n(k_i)}, & \text{if } Y_n(k_i) > \text{BETA}, \\ \text{BETA}, & \text{if otherwise.} \end{cases} \tag{6.29}$$

When processing simply with $Y_n(k_i)$, power can become negative. The purpose of this node 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 such a power spectrum in subsequent processing.

6.3.3 CalcSpecAddPower

Outline of the node

This node outputs the sum of two input spectra.

Necessary files

No files are required.

Usage

When to use

This node is used for noise estimation with the HRLE node. The power spectrum of noise estimated with the HRLE node is added to that of EstimateLeak so as to obtain a total noise power spectrum.

Typical connection

Figure 6.40 shows a connection example of CalcSpecAddPower . Inputs are the power spectrum of the noise estimated from HRLE and EstimateLeak . The output is connected with CalcSpecSubGain .

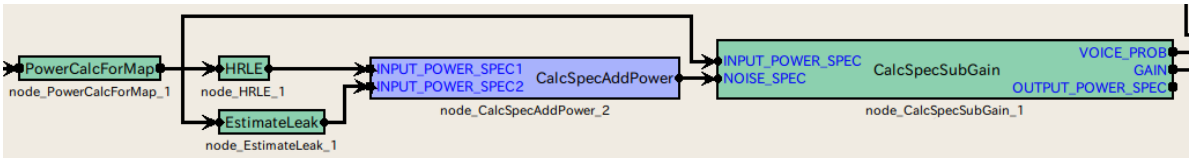


Figure 6.40: Connection example of CalcSpecAddPower

Input-output and property of the node

Input

INPUT_POWER_SPEC1 : `Map<int, ObjectRef>` type. `Vector<float>` type data pair of a sound source ID and power spectrum.

INPUT_POWER_SPEC2 : `Map<int, ObjectRef>` type. `Vector<float>` type data pair of a sound source ID and a power spectrum.

Output

OUTPUT_POWER_SPEC : `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 node

This node outputs the sum of two spectra.

6.3.4 EstimateLeak

Outline of the node

This node estimates a leakage component from the other channels.

Necessary files

No files are required.

Usage

When to use

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

Typical connection

Figure 6.41 shows a connection example of EstimateLeak . The input is a speech power spectrum output from GHDSS . The outputs are connected to CalcSpecAddPower .

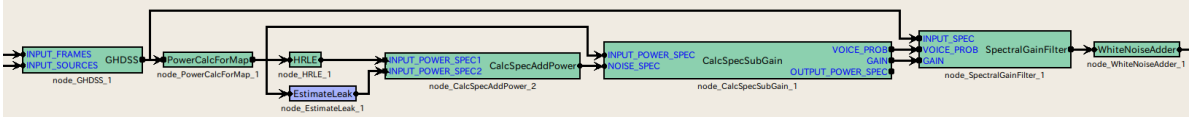


Figure 6.41: Connection example of EstimateLeak

Input-output and property of the node

Input

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

Output

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

Parameter

Table 6.34: Parameter list of EstimateLeak

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

Details of the node

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

6.3.5 GHDSS

Outline of the node

The GHDSS node 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 6.35: Necessary files

Corresponding parameter name	Description
TF_CONJ_FILENAME	Transfer function of microphone array
INITW_FILENAME	Initial value of separation matrix

Usage

When to use

Given a sound source direction, the node 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 6.42 shows a connection example of the GHDSS . The node has two inputs as follows:

1. INPUT_FRAMES takes a multi-channel complex spectrum containing the mixture of sounds,
2. INPUT_SOURCES takes 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 nodes:

1. The PostFilter node to suppress the inter-channel leak and diffusive noise caused by the source separation processing (shown in the upper right part in Fig.6.42).
2. PowerCalcForMap , HRLE , and SpectralGainFilter in cascade to suppress the inter-channel leakage and diffusive noise caused by source separation processing (this tuning would be easier than with PostFilter), or
3. PowerCalcForMap , MelFilterBank and MFMGeneration in cascade to generate missing feature masks so that the separated speech is recognized by a missing-feature-theory based automatic speech recognition system (shown in the lower right part of Fig.6.42).

Input-output and property of the node

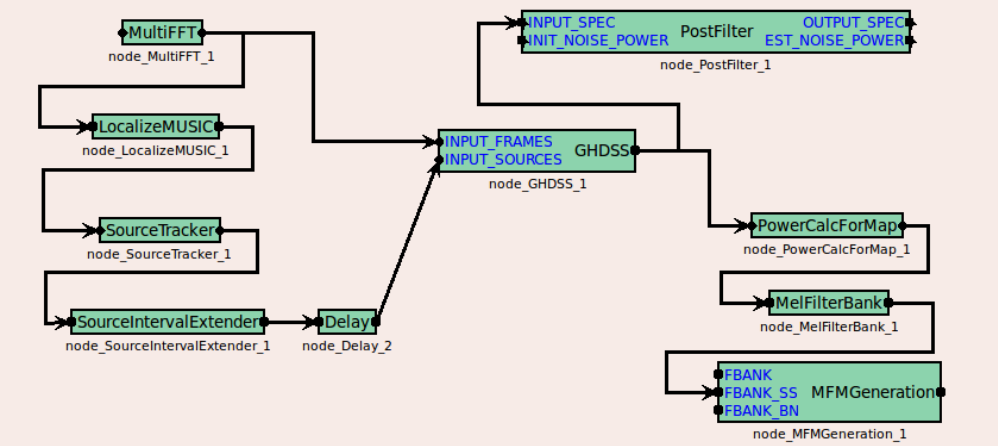


Figure 6.42: Example of Connections of GHDSS

Table 6.36: 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.
SAMPLING_RATE	int	16000	[Hz]	Sampling frequency.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	The minimum value of the frequency used for separation processing
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	The maximum value of the frequency used for separation processing
TF_CONJ_FILENAME	string			File name of transfer function database of your microphone array
INITW_FILENAME	string			A file name in which the initial value of the separation matrix is described.
SS_METHOD	string	ADAPTIVE		A stepsize calculation method based on higher-order decorrelation. Select FIX, LC_MYU or ADAPTIVE. FIX indicates fixed values, LC_MYU indicates the value that links with the stepsize based on geometric constraints and ADAPTIVE indicates automatic regulation.
SS_METHOD==FIX				The following is valid when FIX is chosen for SS_METHOD.
SS_MYU	float	0.001		A stepsize based on higher-order decorrelation at the time of updating a separation matrix
SS_SCAL	float	1.0		The scale factor in a higher-order correlation matrix computation
NOISE_FLOOR	float	0.0		The threshold value of the amplitude for judging the input signal as noise (upper limit)
LC_CONST	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.
LC_METHOD	string	ADAPTIVE		The stepsize calculation method based on geometric constraints. Select FIX or ADAPTIVE. FIX indicates fixed values and ADAPTIVE indicates automatic regulation.
LC_METHOD==FIX				
LC_MYU	float	0.001		The stepsize when updating a separation matrix based on higher-order decorrelation.

UPDATE_METHOD_TF_CONJ	string	POS		Designate a method to update transfer functions. Select POS or ID.
UPDATE_METHOD_W	string	ID		Designate a method to update separation matrixes. Select ID, POS or ID_POS.
UPDATE_SEARCH_AZIMUTH	float		[deg]	The range of azimuth for searching a sound source as identical to another in separation processing.
UPDATE_SEARCH_ELEVATION	float		[deg]	The range of elevation for searching a sound source as identical to another in separation processing.
UPDATE_ACCEPT_ANGLE	float	5.0	[deg]	The threshold value of angle difference for judging a sound source as identical to another in separation processing.
EXPORT_W	bool	false		Designate whether separation matrixes are to be written to files.
EXPORT_W==true				The following is valid when truefor EXPORT_W.
EXPORT_W_FILENAME	string			The name of the file to which the separation matrix is written.
UPDATE	string	STEP		The method to update separation matrixes. Select STEP or TOTAL. In STEP, separation matrixes are updated based on the geometric constraints after an update based on higher-order decorrelation. In TOTAL, separation matrixes are updated based on the geometric constraints and higher-order decorrelation at the same time.

Input

INPUT_FRAMES : Matrix<complex<float> > type. Multi-channel complex spectra. Rows correspond to channels, i.e., complex spectra of waveforms input from microphones, and columns correspond to frequency bins.

INPUT_SOURCES : Vector<ObjectRef> type. A Vector array of the Source type object in which Source localization results are stored. It is typically connected to the SourceTracker node and SourceIntervalExtender node and its outputs are used.

Output

OUTPUT : Map<int, ObjectRef> type. A pair containing the sound source ID of a 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. AudioStreamFromMic or the MultiFFT node).

ADVANCE : int type. Shift length of a frame, which must be equal to the values at a preceding stage value (e.g. AudioStreamFromMic or the MultiFFT node).

SAMPLING_RATE : int type. Sampling frequency of the input waveform.

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

UPPER_BOUND_FREQUENCY This parameter is the maximum frequency used when GHDSS processing is performed. Processing is not performed for frequencies above this value and the value of the output spectrum is zero then. LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY must be maintained.

TF_CONJ_FILENAME: `string` type. The file name in which the transfer function database of your microphone array is saved. Refer to Section 5.1.2 for the detail of the file format.

INITW_FILENAME : `string` type. The file name in which the initial value of a separation matrix is described. Initializing with a converged separation matrix through preliminary computation allows for separation with good precision from the beginning. The file given here must be ready beforehand by setting to `true`for EXPORT_W. For its format, see 5.1.3 .

SS_METHOD : `string` type. Select a stepsize calculation method based on higher-order decorrelation. When wishing to fix it at a designated value, select FIX. When wishing to set a stepsize based on geometric constraints, select LC_MYU. When wishing to perform automatic regulation, select ADAPTIVE.

1. When FIX is chosen: set SS_MYU.
   SS_MYU: `float` type. The default value is 0.01. Designate the stepsize to be used when updating a separation matrix based on higher-order decorrelation. By setting this value and LC_MYU to zero and passing a separation matrix of delay-and-sum beamformer type as INITW_FILENAME, processing equivalent to delay-and-sum beamforming is performed.

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

NOISE_FLOOR : `float` type. 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.

LC_CONST : `string` type. Select a method for geometric constraints. To establish geometric constraints to focus on the target sound source direction only, add DIAG to the focus of the target direction. 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.

LC_METHOD : `string` type. 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 LC_MYU.
   LC_MYU: `float` type. 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 LC_MYU to zero and passing the separation matrix of the beamformer of Delay and Sum type as INITW_FILENAME enables the processing equivalent to the beamformer of the Delay and Sum type.

UPDATE_METHOD_TF_CONJ : `string` type. Select ID or POS. The default value is POS. The user designates if updates of the complex conjugate TF_CONJ 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)

UPDATE_METHODW : `string` type. Select ID, POS or ID_POS. 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 changed. A separation matrix is saved along with its corresponding sound source ID and sound source direction angle for a given period of time. Even if the sound stops once, when a detected sound is judged to be from the same direction, 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 ID_POS is selected, if the sound source is judged not to be the same sound source using a sound source ID comparison, further judgment is performed by comparing the angles of the sound source direction.

UPDATE_SEARCH_AZIMUTH : `float` type. The default value is brank. The unit is [deg]. This specifies the range of searching a sound source as identical to another in separation processing. This parameter is valid when UPDATE_METHOD_TF_CONJ=POS/ID_POS, or UPDATE_METHOD_W=POS/ID_POS. If the value is x, sound sources of the next frame are searched within x degree from the sound sources of the previous frame. If this box is brank, the sound sources are searched in the whole range. The appropriate value of this would reduce the cost of sound source search.

UPDATE_SEARCH_ELEVATION : `float` type. This is almost the same parameter as UPDATE_SEARCH_AZIMUTH but works with the elevation of sound sources.

UPDATE_ACCEPT_ANGLE : `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 ID_POS are selected for UPDATE_METHOD_TF_CONJ and UPDATE_METHOD_W.

EXPORT_W : `bool` type. The default value is `false`. The user determines if the results of the separation matrix updated by GHDSS will be output. When `true`, select EXPORT_W_FILENAME.

EXPORT_W_FILENAME : `string` type. This parameter is valid when EXPORT_W is set to `true`. Designate the name of the file into which a separation matrix will be output. For its format, see Section 5.1.3.

Details of the node

Formulation of sound source separation: Table 6.37 shows symbols used for the formulation of the sound source separation problem. The meaning of the indices is in Table 6.1. Since the calculation is performed in the 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. 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.

Table 6.37: Definitions of the parameters

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

Mixture model The sound that is emitted from N sound sources is affected by the transfer function $H(k_i)$ in space and observed through M microphones as expressed by Equation (6.30).

$$X(k_i) = H(k_i)S(k_i) + N(k_i). \tag{6.30}$$

The transfer function $H(k_i)$ 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 $H_D(k_i)$ is calculated as expressed in Equation (6.31).

$$H_{Dm,n}(k_i) = \exp(-j2\pi l_i r_{m,n}) \tag{6.31}$$

$$l_i = \frac{2\pi\omega_i}{c}, \tag{6.32}$$

Here, c indicates the speed of sound 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 the difference between the reference point of the coordinate system (e.g. origin) to the sound source n. In other words, $H_D(k_i)$ is defined as the phase difference incurred by the difference in arrival time from the sound source to each microphone.

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

$$Y(k_i) = W(k_i)X(k_i) \tag{6.33}$$

The GHDSS algorithm estimates the separation matrix $W(k_i)$ so that $Y(k_i)$ closes to $S(k_i)$.

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 LocalizeMUSIC node estimates source location in HARK)
3. Microphone position
4. Transfer function of the direct sound component $H_D(k_i)$ (measurement or approximation by Equation (6.31))

As unknown information,

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

Update equation of separation matrix GHDSS estimates $W(k_i)$ so that the following conditions are satisfied.

1. Higher-order decorrelation of the separated signals
   In other words, the diagonal component of the higher-order matrix $R^{\phi(y)y}(k_i) = E[\phi(Y(k_i))Y^H(k_i)]$ of the separated sound $Y(k_i)$ 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 node.

$$\phi(Y) = [\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_N)]^T \tag{6.34}$$
$$\phi(Y_k) = \tanh(\sigma|Y_k|)\exp(j\angle(Y_k)) \tag{6.35}$$

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

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

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.

$$J(W) = \alpha J_1(W) + \beta J_2(W), \tag{6.36}$$
$$J_1(W) = \sum_{i \neq j} |R^{\phi(y)y}_{i,j}|^2, \tag{6.37}$$
$$J_2(W) = WH_D - I^2, \tag{6.38}$$

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

An update equation of the separation matrix to minimize Equation (6.36) is obtained by the gradient method that uses the complex gradient calculation $\frac{\partial}{\partial W^*}$.

$$W(k_i, f+1) = W(k_i, f) - \mu \frac{\partial J}{\partial W^*}(W(k_i, f)) \tag{6.39}$$

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 (6.39), multiple frame values are required for expectation value calculation such as $R^{xx} = E[XX^H]$ and $R^{yy} = E[YY^H]$. An autocorrelation matrix is not obtained in calculation of the GHDSS node. However, Equation (6.40), which uses only one frame, is used.

$$W(k_i, f+1) = W(k_i, f) - \left[\mu_{SS} \frac{\partial J_1}{\partial W^*}(W(k_i, f)) + \mu_{LC} \frac{\partial J_2}{\partial W^*}(W(k_i, f))\right], \tag{6.40}$$

$$\frac{\partial J_1}{\partial W^*}(W) = \left(\phi(Y)Y^H - \mathrm{diag}[\phi(Y)Y^H]\right)\tilde{\phi}(\mathrm{boldmath}WX)X^H, \tag{6.41}$$

$$\frac{\partial J_2}{\partial W^*}(W) = 2\left(WH_D - I\right)H_D^H, \tag{6.42}$$

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

$$\tilde{\phi}(Y) = [\phi(\tilde{Y}_1), \phi(\tilde{Y}_2), \ldots, \phi(\tilde{Y}_N)]^T \tag{6.43}$$

$$\tilde{\phi}(Y_k) = \phi(Y_k) + Y_k \frac{\partial \phi(Y_k)}{\partial Y_k} \tag{6.44}$$

Moreover, $\mu_{SS} = \mu\alpha$ and $\mu_{LC} = \mu\beta$, which are the stepsizes based on the higher-order decorrelation and geometric constraints. The stepsizes, which are automatically regulated, are calculated by the equations

$$\mu_{SS} = \frac{J_1(W)}{2\frac{\partial J_1}{\partial W}(W)^2} \tag{6.45}$$

$$\mu_{LC} = \frac{J_2(W)}{2\frac{\partial J_2}{\partial W}(W)^2} \tag{6.46}$$

The indices of each parameter in Equations (6.41, 6.42) are (k_i, f), which are abbreviated above. The initial values of the separation matrix are obtained as follows.

$$W(k_i) = H_D^H(k_i)/M, \tag{6.47}$$

Here, M indicates the number of microphones.

Processing flow The main processing for time frame f in the GHDSS node is shown in Figure 6.43. The detailed processing related to fixed noise is as follows:

1. Acquiring a transfer function (direct sound)
2. Estimating the separation matrix W
3. Performing sound source separation in accordance with Equation (6.33)
4. Writing of a separation matrix (When EXPORT_W is set to `true`)

Acquiring a transfer function: At the first frame, the transfer function, specified by the file name TF_CONJ_FILENAME, that is closest to the localization result is searched.

Processing after the second frame is as follows.

- UPDATE_METHOD_TF_CONJ decides whether the transfer function selected in the previous frame is used or not in the currenct frame. If not used, the new transfer function is loaded from the file.

UPDATE_METHOD_TF_CONJ is ID

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

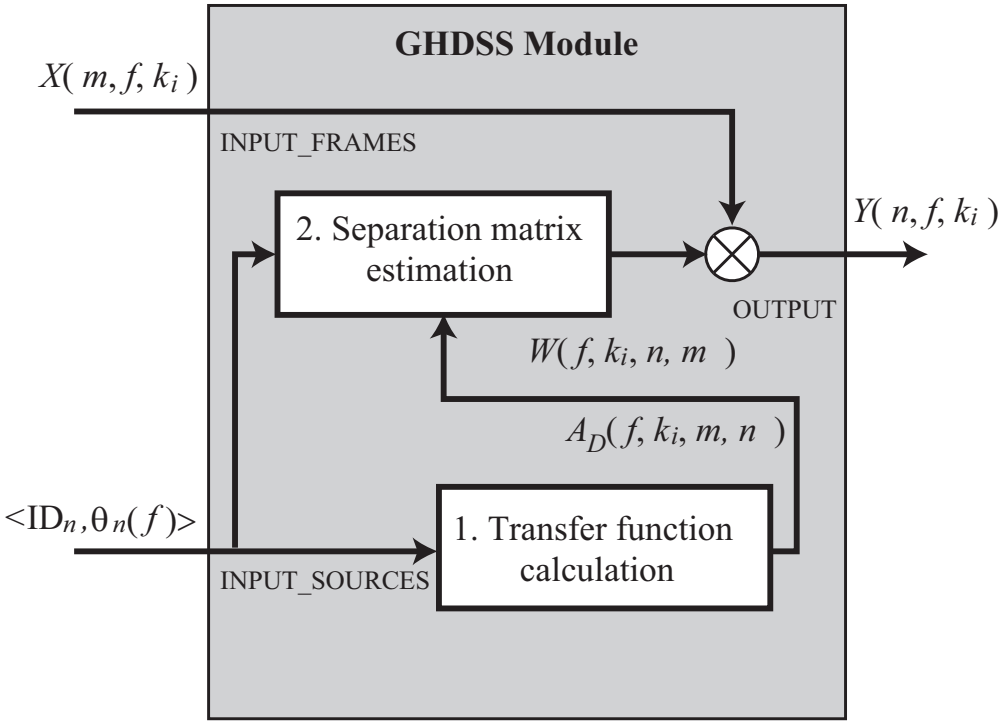


Figure 6.43: Flowchart of GHDSS

UPDATE_METHOD_TF_CONJ is POS

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

Estimating the separation matrix: The initial value of the separation matrix is different depending on if the user designates a value for the parameter INITW_FILENAME.
When the parameter INITW_FILENAME is not designated, the separation matrix W is calculated from the transfer function H_D.
When the parameter INITW_FILENAME is designated, the data most close to the direction of the input source localization result is searched for from the designated separation matrix.
Processing after the second frame is as follows. The flow for estimating the separation matrix is shown in Figure 6.44. Here, the previous frame is updated based on Equation (6.40) or an initial value of the separation matrix is derived by the transfer function based on Equation (6.47).

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

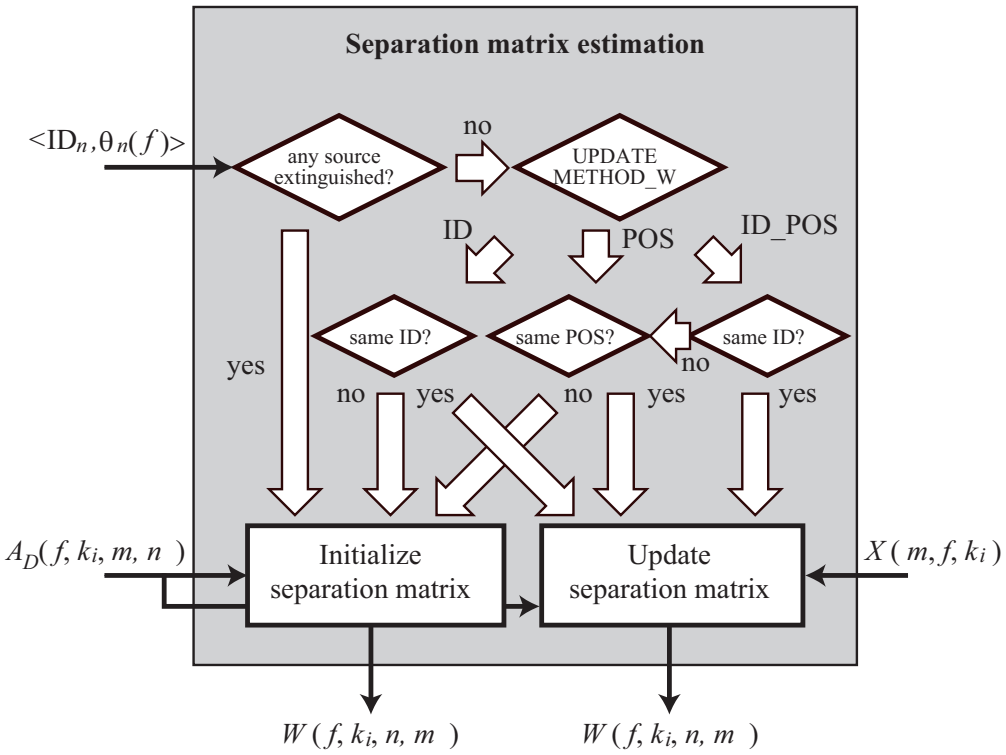


Figure 6.44: Flowchart of separation matrix estimation

UPDATE_METHOD_W is ID

1. Compare with the previous frame ID
   - Same: Update W
   - Different: Initialize W

UPDATE_METHOD_W is POS

1. Compare with the former frame localization direction
   - Error is less than UPDATE_ACCEPT_ANGLE: Update W
   - Error is more than UPDATE_ACCEPT_ANGLE: Initialize W

UPDATE_METHOD_W is ID_POS

1. Compare with the former frame ID
   - Same: Update W
2. Localization directions are compared when IDs are different
   - Error is less than UPDATE_ACCEPT_ANGLE: Update W
   - Error more than UPDATE_ACCEPT_ANGLE: Initialize W

Writing of a separation matrix (When EXPORT_W is set to `true`) When EXPORT_W is set to `true`, a converged separation matrix is output to a file designated for EXPORT_W_FILENAME.
When multiple sound sources are detected, all those separation matrices are output to one file. When a sound source

disappears, 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 UPDATE_ACCEPT_ANGLE: Overwrite and save W
   - Error more than UPDATE_ACCEPT_ANGLE: Save additionally W

6.3.6 HRLE

Outline of the node

This node estimates the stationary noise level using 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 with a previous input spectrum weighted with an exponent window, and the position of the exponent window is updated every frame.

Necessary files

No files are required.

Usage

When to use

This node is used when within to suppress noise using spectrum subtraction.

Typical connection

As shown in Figure 6.45, the input is connected after separation nodes such as GHDSS and the output is connected to the nodes that calculate an optimal gain such as CalcSpecSubGain . Figure 6.46 shows a connection example when EstimateLeak is used together.

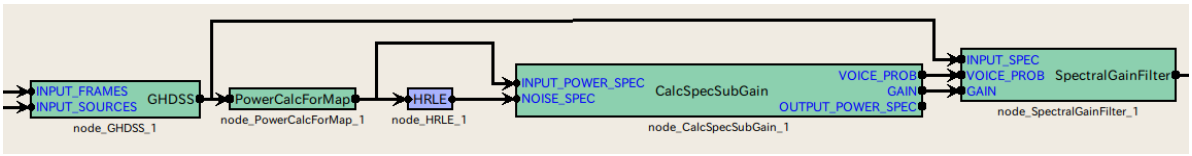


Figure 6.45: Connection example of HRLE 1

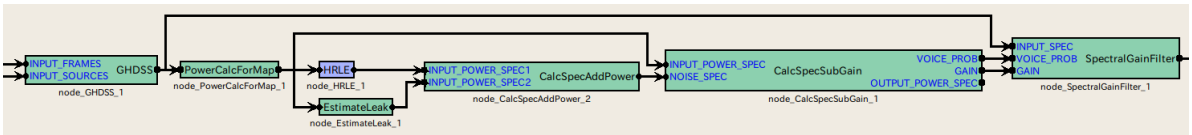


Figure 6.46: Connection example of HRLE 2

Input-output and property of the node

Table 6.38: Parameter list of HRLE

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

Input

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

Output

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

Parameters

LX : `float` type. 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 is designated.

TIME_CONSTANT : `float` type. The default value is 16000. A time constant (more than zero) is designated in time sample unit.

NUM_BIN : `float` type. The default value is 1000. Designate the number of bins of a histogram.

MIN_LEVEL : `float` type. The default value is -100. Designate the minimum level of a histogram in dB.

STEP_LEVEL : `float` type. The default value is 0.2. Designate a width of a bin of a histogram in dB.

DEBUG : `bool` The 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 multiple 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 future versions.) 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 NUM_BIN, MIN_LEVEL and STEP_LEVEL 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, MIN_LEVEL 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)]
ˆT
Matrix size = 1000 x 257
```

Details of the node

Figure 6.47 shows a processing flow of HRLE. HRLE obtains a level histogram from the input power and estimates the Lx level from the cumulative distribution. The Lx level, as shown in Figure 6.48, 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

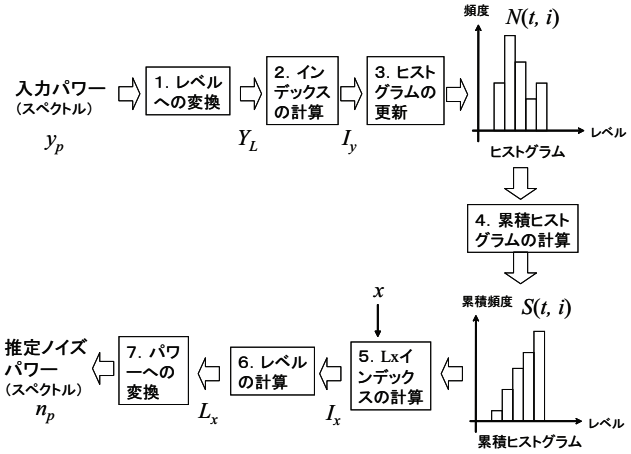


Figure 6.47: Processing flow of HRLE

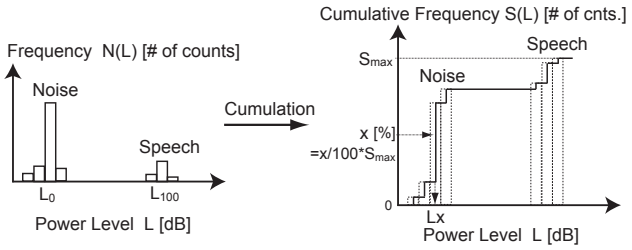


Figure 6.48: Estimation of Lx value

estimated in its processing. The details of the processing in HRLE are expressed by the following seven equations (corresponding to Figure 6.47). In the equations, t indicates time (frame), y_p indicates input power (INPUT_SPEC) and n_p indicates estimated noise power (NOISE_SPEC]). x, α, L_{min} and L_{step} are the parameters related to histograms and indicate normalization accumulation frequency (LX), time constant (TIME_CONSTANT), the minimum level (MIN_LEVEL) of a bin, and a level width (STEP_LEVEL) 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.

$$Y_L(t) = 10 \log_{10} y_p(t), \tag{6.48}$$

$$I_y(t) = \lfloor (Y_L(t) - L_{min})/L_{step} \rfloor, \tag{6.49}$$

$$N(t, l) = \alpha N(t-1, l) + (1-\alpha)\delta(l - I_y(t)), \tag{6.50}$$

$$S(t, l) = \sum_{k=0}^{l} N(t, k), \tag{6.51}$$

$$I_x(t) = \underset{I}{\operatorname{argmin}} \left[S(t, I_{max}) \frac{x}{100} - S(t, I) \right], \tag{6.52}$$

$$L_x(t) = L_{min} + L_{step} \cdot I_x(t), \tag{6.53}$$

$$n_p(t) = 10^{L_x(t)/10} \tag{6.54}$$

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 node

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

Necessary files

No files are required.

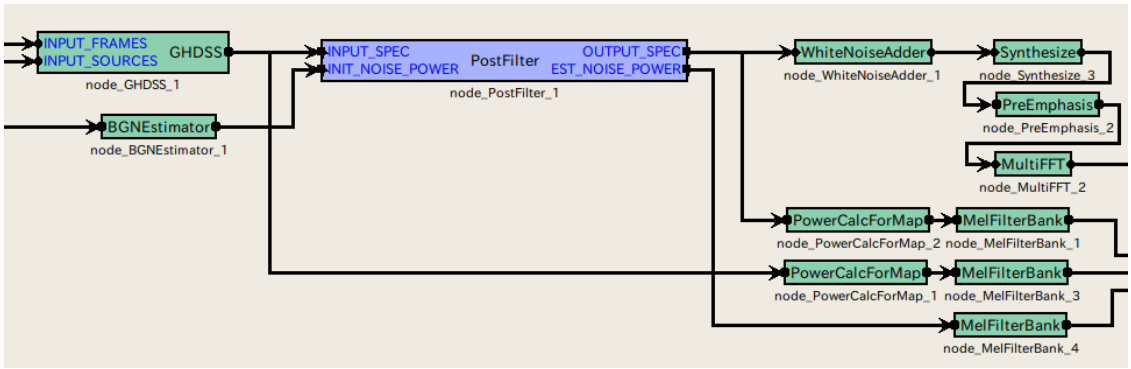
When to use

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

Typical connection

Figure 6.3.7 shows an example of a connection for the PostFilter node. The output of the GHDSS node is connected to the INPUT_SPEC input and the output of the BGNEstimator node is connected to the INIT_NOISE_POWER input. Figure 6.3.7 shows examples for typical output connections:

1. Speech feature extraction from separated sound (OUTPUT_SPEC) (MSLSExtraction node)
2. Generation of Missing Feature Masks from separated sound and power (EST_NOISE_POWER) of noise contained in it at the time of speech recognition (MFMGeneration node)



Input-output and property of the node

Input

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

INPUT_NOISE_POWER : `Matrix<float>` type. The power spectrum of the stationary noise estimated by the BGNEstimator node.

Output

OUTPUT_SPEC : `Map<int, ObjectRef>` type. The `Object` is the complex spectrum from the input INPUT_SPEC, with noise removed.

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

Parameter

Table 6.39: Parameter list of PostFilter (first half)

Parameter name	Type	Default value	Description
MCRA_SETTING	bool	false	When the user set parameters for the MCRA estimation, which is a noise removal method, select true.
MCRA_SETTING			The following are valid when MCRA_SETTING is set to true
STATIONARY_NOISE_FACTOR	float	1.2	Coefficient at the time of stationary noise estimation.
SPEC_SMOOTH_FACTOR	float	0.5	Smoothing coefficient of an input power spectrum.
AMP_LEAK_FACTOR	float	1.5	Leakage coefficient.
STATIONARY_NOISE_MIXTURE_FACTOR	float	0.98	Mixing ratio of stationary noise.
LEAK_FLOOR	float	0.1	Minimum value of leakage noise.
BLOCK_LENGTH	int	80	Detection time width.
VOICEP_THRESHOLD	int	3	Threshold value of speech presence judgment.
EST_LEAK_SETTING	bool	false	When the user sets parameters related to the leakage rate estimation, select true.
EST_LEAK_SETTING			The followings are valid when EST_LEAK_SETTING is set to true.
LEAK_FACTOR	float	0.25	Leakage rate.
OVER_CANCEL_FACTOR	float	1	Leakage rate weighting factor.
EST_REV_SETTING	bool	false	When the user sets parameters related to the component estimation, select true.
EST_REV_SETTING			The followings are valid when EST_REV_SETTING is set to true.
REVERB_DECAY_FACTOR	float	0.5	Damping coefficient of reverberant power.
DIRECT_DECAY_FACTOR	float	0.2	Damping coefficient of a separated spectrum.
EST_SN_SETTING	bool	false	When the user sets parameters related to the SN ratio estimation, select true.
EST_SN_SETTING			The followings are valid when EST_SN_SETTING is set to true.
PRIOR_SNR_FACTOR	float	0.8	Ratio of priori and posteriori SNRs.
VOICEP_PROB_FACTOR	float	0.9	Amplitude coefficient of the probability of speech presence.
MIN_VOICEP_PROB	float	0.05	Probability of the minimum speech presence.
MAX_PRIOR_SNR	float	100	Maximum value of preliminary SNR.
MAX_OPT_GAIN	float	20	Maximum value of the optimal gain intermediate variable v.
MIN_OPT_GAIN	float	6	Minimum value of the optimal gain intermediate variable v.

Table 6.40: Parameter list of PostFilter (latter half)

Parameter name	Type	Default value	Description
EST_VOICEP_SETTING	bool	false	When the user sets parameters related to the speech probability estimation, select true.
EST_VOICEP_SETTING			The following are valid when EST_VOICEP_SETTING is set to true.
PRIOR_SNR_SMOOTH_FACTOR	float	0.7	Time smoothing coefficient.

MIN_FRAME_SMOOTH_SNR	float	0.1	Minimum value of the frequency smoothing SNR (frame).
MAX_FRAME_SMOOTH_SNR	float	0.316	Maximum value of the frequency smoothing SNR (frame).
MIN_GLOBAL_SMOOTH_SNR	float	0.1	Minimum value of the frequency smoothing SNR (global).
MAX_GLOBAL_SMOOTH_SNR	float	0.316	Maximum value of the frequency smoothing SNR (global).
MIN_LOCAL_SMOOTH_SNR	float	0.1	Minimum value of the frequency smoothing SNR (local).
MAX_LOCAL_SMOOTH_SNR	float	0.316	Maximum value of the frequency smoothing SNR (local).
UPPER_SMOOTH_FREQ_INDEX	int	99	Frequency smoothing upper limit bin index.
LOWER_SMOOTH_FREQ_INDEX	int	8	The frequency smoothing lower limit bin index.
GLOBAL_SMOOTH_BANDWIDTH	int	29	Frequency smoothing band width (global).
LOCAL_SMOOTH_BANDWIDTH	int	5	The frequency smoothing band width (local).
FRAME_SMOOTH_SNR_THRESH	float	1.5	Threshold value of frequency smoothing SNR.
MIN_SMOOTH_PEAK_SNR	float	1.0	Minimum value of the frequency smoothing SNR peak.
MAX_SMOOTH_PEAK_SNR	float	10.0	Maximum value of the frequency smoothing SNR peak.
FRAME_VOICEP_PROB_FACTOR	float	0.7	Speech probability smoothing coefficient (frame).
GLOBAL_VOICEP_PROB_FACTOR	float	0.9	Speech probability smoothing coefficient (global).
LOCAL_VOICEP_PROB_FACTOR	float	0.9	Speech probability smoothing coefficient (local).
MIN_VOICE_PAUSE_PROB	float	0.02	Minimum value of speech quiescent probability.
MAX_VOICE_PAUSE_PROB	float	0.98	Maximum value of speech quiescent probability.

Details of the node

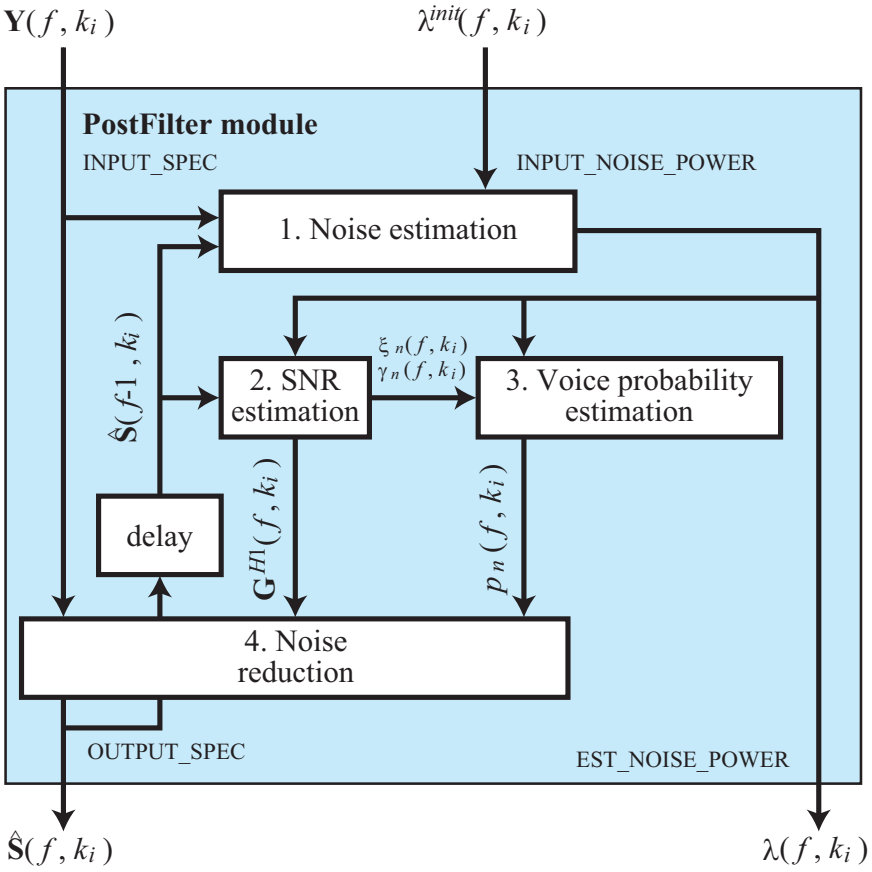


Figure 6.49: Flowchart of PostFilter

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 6.49 shows a flowchart of the PostFilter node.

A separated sound spectrum from the GHDSS node and a stationary noise power spectrum of the BGNEstimator node are obtained as inputs. Outputs are the separated sound spectrum for which the speech is emphasized, and a power spectrum of noise mixed with the separated sound. The processing flow is as follows.

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

1) Noise estimation: Figure 6.50 shows the processing flow of noise estimation . The three kinds of noise that the

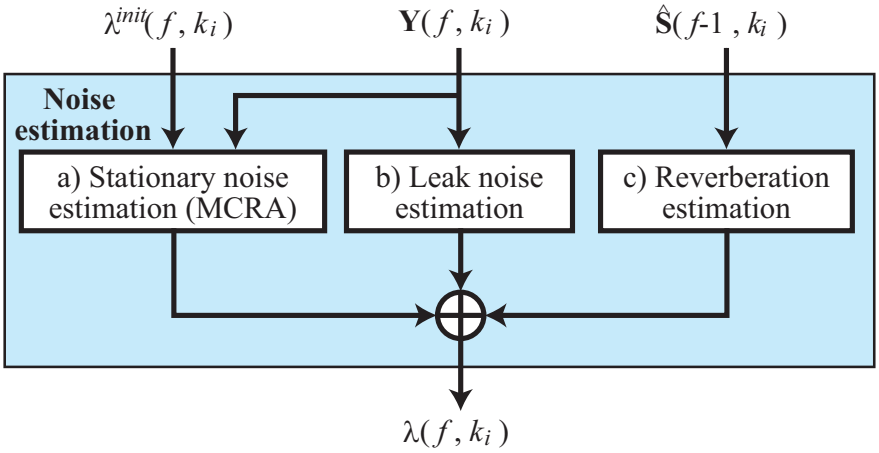


Figure 6.50: Procedure of noise estimation

PostFilter node processes are:
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) Reverberations from the previous frame.

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

$$\lambda(f, k_i) = \lambda^{sta}(f, k_i) + \lambda^{leak}(f, k_i) + \lambda^{rev}(f-1, k_i) \tag{6.55}$$

Here, $\lambda^{sta}(f, k_i)$, $\lambda^{leak}(f, k_i)$ and $\lambda^{rev}(f-1, k_i)$ indicate stationary noise, leakage noise and reverberation from the previous frame, respectively.

1-a) Stationary noise estimation by MCRA method The parameters used in 1-a) are based on Table 6.41. First, calculate the power spectrum for which the input spectrum is smoothed with the power from one frame before. $S(f, k_i) = [S_1(f, k_i), \ldots, S_N(f, k_i)]$.

$$S_n(f, k_i) = \alpha_s S_n(f-1, k_i) + (1-\alpha_s)|Y_n(k_i)|^2 \tag{6.56}$$

Table 6.41: Definition of variable

Parameter	Description, Corresponding parameter
$Y(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	Complex spectrum of separated sound corresponding to the frequency bin k_i
$\lambda^{init}(k_i) = \left[\lambda_1^{init}(k_i), \ldots, \lambda_N^{init}(k_i)\right]^T$	Initial value power spectrum used for the stationary noise estimation
$\lambda^{sta}(k_i) = \left[\lambda_1^{sta}(k_i), \ldots, \lambda_N^{sta}(k_i)\right]^T$	Estimated stationary noise power spectrum.
α_s	Smoothing coefficient of the input power spectrum. Parameter SPEC_SMOOTH_FACTOR. The default value is 0.5
$S^{tmp}(k_i) = \left[S_1^{tmp}(k_i), \ldots, S_N^{tmp}(k_i)\right]$	Temporary parameter for minimum power calculation.
$S^{min}(k_i) = \left[S_1^{min}(k_i), \ldots, S_N^{min}(k_i)\right]$	The parameter that maintains the minimum power.
L	Maintained frame numbers of S_{tmp}. Parameter BLOCK_LENGTH. The default value is 80
δ	Threshold value of speech presence judgment. Parameter VOICEP_THRESHOLD. The default value is 3.0
α_d	Mixing ratio of estimated stationary noise. Parameter STATIONARY_NOISE_MIXTURE_FACTOR. The default value is 0.98
$Y^{leak}(k_i)$	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 AMP_LEAK_FACTOR. The default value is 1.5.
S_{floor}	Minimum value of leakage noise. Parameter LEAK_FLOOR. The default value is 0.1.
r	Coefficient at the time of stationary noise estimation. Parameter STATIONARY_NOISE_FACTOR. The default value is 1.2

Next, update S^{tmp}, S^{min}.

$$S_n^{min}(f, k_i) = \begin{cases} \min\{S_n^{min}(f-1, k_i), S_n(f, k_i) & \text{if } f \neq nL \\ \min\{S_n^{tmp}(f-1, k_i), S_n(f, k_i) & \text{if } f = nL \end{cases}, \tag{6.57}$$

$$S_n^{min}(f, k_i) = \begin{cases} \min\{S_n^{tmp}(f-1, k_i), S_n(f, k_i) & \text{if } f \neq nL \\ S_n(f, k_i) & \text{if } f = nL \end{cases}, \tag{6.58}$$

Here, n indicates an arbitrary integer. S^{min} maintains the minimum power after the noise estimation begins S^{tmp} maintains an extremely small power of a recent frame. S^{tmp} is updated every L frames. Next, judge if the frame contains speech based on the power ratio of the minimum power and the input separated sound.

$$S_n^r(k_i) = \frac{S_n(k_i)}{S^{min}(k_i)}, \tag{6.59}$$

$$I_n(k_i) = \begin{cases} 1 & \text{if } S_n^r(k_i) > \delta \\ 0 & \text{if } S_n^r(k_i) \leq \delta \end{cases} \tag{6.60}$$

When speech is included, $I_n(k_i)$ is 1 and when it is not included, it is 0. Based on this result, we determine the mixing ratio $\alpha_{d,n}^C(k_i)$ of the frame's estimated stationary noise.

$$\alpha_{d,n}^C(k_i) = (\alpha_d - 1) I_n(k_i) + 1. \tag{6.61}$$

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

$$S_n^{leak}(k_i) = \sum_{p=1}^{N} |Y_p(k_i)|^2 - |Y_n(k_i)|^2, \tag{6.62}$$

$$S_n^0(k_i) = |Y_n(k_i)|^2 - q S_n^{leak}(k_i), \tag{6.63}$$

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

$$S_n^0(k_i) = S_{floor} \tag{6.64}$$

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 $\lambda^{sta}(f-1, k_i)$ or $bf\lambda^{init}(f, k_i)$, which is the output from BGNEstimator .

$$\lambda_n^{sta}(f, k_i) = \begin{cases} \alpha_{d,n}^C(k_i)\lambda_n^{sta}(f-1, k_i) + (1-\alpha_{d,n}^C(k_i)rS_n^0(f, k_i) & nochangeinsourceposition \\ \alpha_{d,n}^C(k_i)\lambda_n^{init}(f, k_i) + (1-\alpha_{d,n}^C(k_i)rS_n^0(f, k_i) & \text{ifChange in source position} \end{cases} \tag{6.65}$$

1-b)Leakage noise estimation The variables used in 1-b) are based on Table 6.42.

Table 6.42: Definition of variable

Variable	Description, Corresponding parameter
$\lambda^{leak}(k_i)$	Power spectrum of leakage noise. Vector comprising elements of each separated sound.
α^{leak}	Leakage rate for the total of separated sound power. LEAK_FACTOR × OVER_CANCEL_FACTOR
$S_n(f, k_i)$	Smoothing power spectrum obtained by Equation (6.56)

Some parameters are calculated as follows.

$$\beta = -\frac{\alpha^{leak}}{1-(\alpha^{leak})^2 + \alpha^{leak}(1-\alpha^{leak})(N-2)} \tag{6.66}$$

$$\alpha = 1-(N-1)\alpha^{leak}\beta \tag{6.67}$$

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 (6.62).

$$Z_n(k_i) = \alpha S_n(k_i) + \beta S_n^{leak}(k_i), \tag{6.68}$$

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

$$\lambda_n^{leak} = \alpha^{leak}\left(\sum_{n' \neq n} Z_{n'}(k_i)\right) \tag{6.69}$$

1-c) Reverberant estimation The variables used in 1-c) are based on Table 6.43.

$$\lambda_n^{leak} = \alpha^{leak}\left(\sum_{n' \neq n} Z_{n'}(k_i)\right) \tag{6.70}$$

$$\lambda_n^{rev}(f, k_i) = \gamma\left(\lambda_n^{rev}(f-1, k_i) + \Delta|\hat{S}_n(f-1, k_i)|^2\right) \tag{6.71}$$

2)SNR estimation: Figure 6.51 shows the flow of the SNR estimation. The SNR estimation consists of the followings

Table 6.43: Definition of variable

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

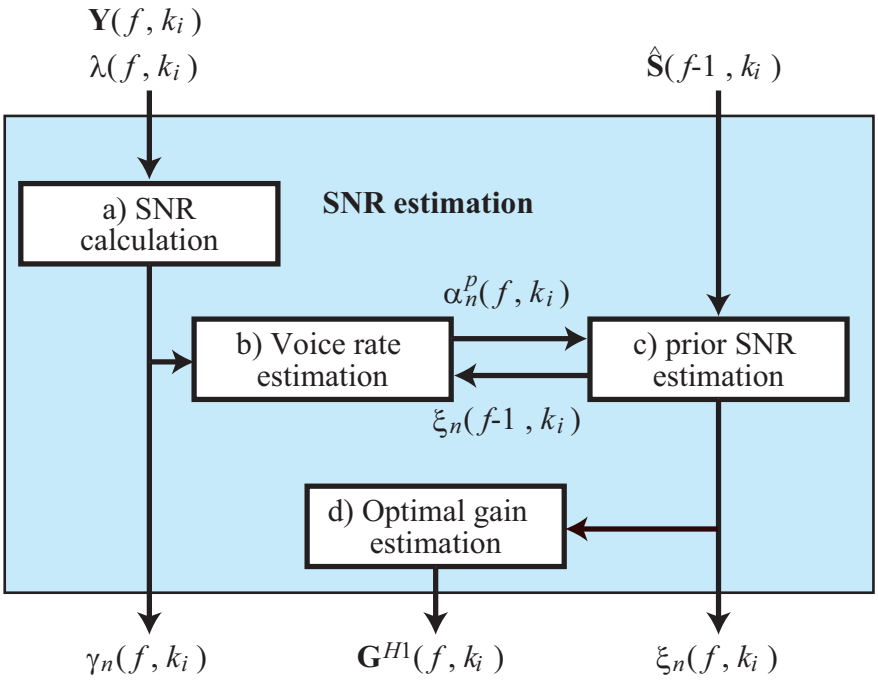


Figure 6.51: Procedure of SNR estimation

Table 6.44: Definition of major variable

Variable	Description, corresponding parameter
$Y(k_i)$	Complex spectra of the separated sound, which is an input of the PostFilter node
$\hat{S}(k_i)$	Complex spectra of the formed separated sound, which is an output of the PostFilter node
$\lambda(k_i)$	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
$G^{H1}(k_i)$	Optimal gain to improve SNR of the separated sound

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 6.44 indicate value of each separated sound.

2-a) Calculation of SNR The variables used in 2-a) are based on Table 6.44. Here, SNR $\gamma_n(k_i)$ is calculated based on the complex spectra $Y(k_i)$ of the input and the power spectrum of the noise estimated above.

$$\gamma_n(k_i) = \frac{|Y_n(k_i)|^2}{\lambda_n(k_i)} \tag{6.72}$$

$$\gamma_n^C(k_i) = \begin{cases} \gamma_n(k_i) & \text{if } \gamma_n(k_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.73}$$

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

2-b)Estimation of speech content rate The variables used in 2-b) are based on Table 6.45. The speech content rate

Table 6.45: Definition of variable

Variable	Description, corresponding parameter
α^{p}_{mag}	Preliminary SNR coefficient. Parameter VOICEP_PROB_FACTOR. The default value is 0.9.
α^{p}_{min}	Minimum speech content rate. Parameter MIN_VOICEP_PROB. The default value is 0.05.

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

$$\alpha^{p}_{n}(f, k_i) = \alpha^{p}_{mag}\left(\frac{\xi_n(f-1, k_i)}{\xi_n(f-1, k_i)+1}\right)^2 + \alpha^{p}_{min} \tag{6.74}$$

2-c) Preliminary SNR estimation before noise mixture The variables used in 2-c) are based on Table 6.46. The

Table 6.46: Definition of variable

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

preliminary SNR $\xi_n(k_i)$ is calculated as follows.

$$\xi_n(k_i) = \left(1-\alpha^{p}_{n}(k_i)\right)\xi_{tmp} + \alpha^{p}_{n}(k_i)\gamma^{C}_{n}(k_i) \tag{6.75}$$

$$\xi_{tmp} = a\frac{|\hat{S}_n(f-1, k_i)|^2}{\lambda_n(f-1, k_i)} + (1-a)\xi_n(f-1, k_i) \tag{6.76}$$

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, change 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 6.47. Prior to calculating an optimal

Table 6.47: Definition of variable

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

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

$$v_n(k_i) = \frac{\xi_n(k_i)}{1+\xi_n(k_i)}\gamma_n(k_i) \tag{6.77}$$

When $v_n(k_i) > \theta^{max}$ is satisfied, $v_n(k_i) = \theta^{max}$. The optimal gain $G^{H1}(k_i) = [G^{H1}_1(k_i), \ldots, G^{H1}_N(k_i)]$ when speech exists is obtained as sollows.

$$G^{H1}_{n}(k_i) = \frac{\xi_n(k_i)}{1+\xi_n(k_i)}\exp\left\{\frac{1}{2}\int_{v_n(k_i)}^{\inf}\frac{e^{-t}}{t}\mathrm{d}t\right\} \tag{6.78}$$

Here,

$$\begin{aligned} &G^{H1}_{n}(k_i) = 1 \quad \text{if} v_n(k_i) < \theta^{min} \\ &G^{H1}_{n}(k_i) = 1 \quad \text{if} G^{H1}_{n}(k_i) > 1. \end{aligned} \tag{6.79}$$

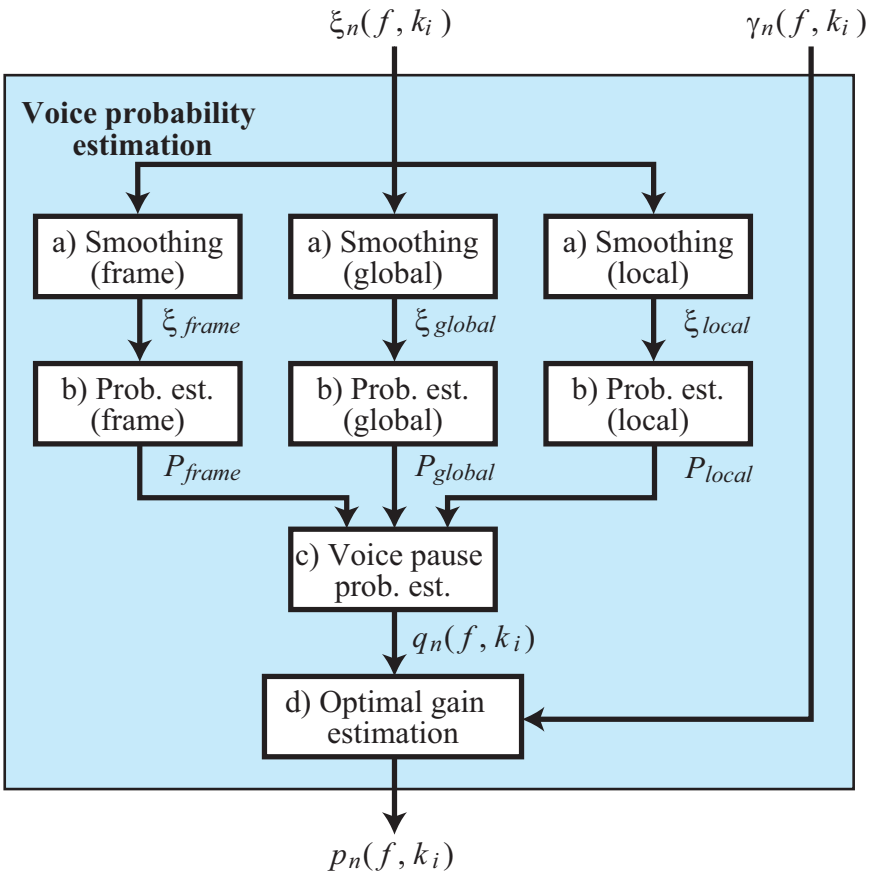


Figure 6.52: Procedure for estimation of probability of speech presence:

3) Estimation of probability of speech presence Figure 6.52 shows the flow of estimation of probability of speech presence. Estimation of the probability of speech presence consists of:
a) Smoothing of the preliminary SNR for each of the 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 6.48. First, temporally-

Table 6.48: 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 PRIOR_SNR_SMOOTH_FACTOR. The default value is 0.7
F_{st}	Parameter LOWER_SMOOTH_FREQ_INDEX. The default value is 8
F_{en}	Parameter UPPER_SMOOTH_FREQ_INDEX. The default value is 99
G	Parameter GLOBAL_SMOOTH_BANDWIDTH. The default value is 29
L	Parameter LOCAL_SMOOTH_BANDWIDTH. The default value is 5

smoothing is performed with the preliminary SNR $\xi_n(f, k_i)$ calculated by Equation (6.75) and the temporally-smoothed preliminary SNR $\zeta_n(f-1, k_i)$ of the former frame.

$$\zeta_n(f, k_i) = b\zeta_n(f-1, k_i) + (1-b)\xi_n(f, k_i) \tag{6.80}$$

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 $F_{st} \sim F_{en}$.

$$\zeta_n^f(k_i) = \frac{1}{F_{en} - F_{st} + 1} \sum_{k_j=F_{st}}^{F_{en}} \zeta_n(k_j) \tag{6.81}$$

- Global frequency smoothing in global
  Frequency smoothing with a hamming window in width G is performed globally.

$$\zeta_n^g(k_i) = \sum_{j=-(G-1)/2}^{(G-1)/2} w_{han}(j + (G-1)/2)\zeta_n(k_{i+j}), \tag{6.82}$$

$$w_{han}(j) = \frac{1}{C}\left(0.5 - 0.5\cos\left(\frac{2\pi j}{G}\right)\right), \tag{6.83}$$

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

- Local frequency smoothing
  Frequency smoothing with a triangle window in width F is performed locally.

$$\zeta_n^l(k_i) = 0.25\zeta_n(k_i - 1) + 0.5\zeta_n(k_i) + 0.25(k_i + 1) \tag{6.84}$$

3-b Estimation of the probability of provisional speech The variables used in 3-b) are shown in Table 6.49.

Table 6.49: 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 MIN_SMOOTH_PEAK_SNR. The default value is 1.
Z_{max}^{peak}	Parameter MAX_SMOOTH_PEAK_SNR. The default value is 10.
Z_{thres}	FRAME_SMOOTH_SNR_THRESH. The default value is 1.5.
$Z_{min}^{f,g,l}$	Parameter MIN_FRAME_SMOOTH_SNR, MIN_GLOBAL_SMOOTH_SNR, MIN_LOCAL_SMOOTH_SNR. The default value is 0.1.
$Z_{max}^{f,g,l}$	Parameter MAX_FRAME_SMOOTH_SNRF, MAX_GLOBAL_SMOOTH_SNR, MAX_LOCAL_SMOOTH_SNR. 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.

$$\zeta_n^{peak}(f, k_i) = \begin{cases} \zeta_n^f(f, k_i), & \text{if } \zeta_n^f(f, k_i) > Z_{thres}\zeta_n^f(f-1, k_i) \\ \zeta_n^{peak}(f-1, k_i), & \text{if otherwise.} \end{cases} \tag{6.85}$$

  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,

$$\zeta_n^{peak}(k_i) = \begin{cases} Z_{min}^{peak}, & \text{if } \zeta_n^{peak}(k_i) < Z_{min}^{peak} \\ Z_{max}^{peak}, & \text{if } \zeta_n^{peak}(k_i) > Z_{max}^{peak} \end{cases} \tag{6.86}$$

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

$$P_n^f(k_i) = \begin{cases} 0, & \text{if } \zeta_n^f(k_i) < \zeta_n^{peak}(k_i) Z_{min}^f \\ 1, & \text{if } \zeta_n^f(k_i) > \zeta_n^{peak}(k_i) Z_{max}^f \\ \frac{\log\left(\zeta_n^f(k_i)/\zeta_n^{peak}(k_i) Z_{min}^f\right)}{\log\left(Z_{max}^f/Z_{min}^f\right)}, & \text{otherwise} \end{cases} \tag{6.87}$$

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

$$P_n^g(k_i) = \begin{cases} 0, & \text{if } \zeta_n^g(k_i) < Z_{min}^g \\ 1, & \text{if } \zeta_n^g(k_i) > Z_{max}^g \\ \frac{\log\left(\zeta_n^g(k_i)/Z_{min}^g\right)}{\log\left(Z_{max}^g/Z_{min}^g\right)}, & \text{otherwise} \end{cases} \tag{6.88}$$

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

$$P_n^l(k_i) = \begin{cases} 0, & \text{if } \zeta_n^l(k_i) < Z_{min}^l \\ 1, & \text{if } \zeta_n^l(k_i) > Z_{max}^l \\ \frac{\log\left(\zeta_n^l(k_i)/Z_{min}^l\right)}{\log\left(Z_{max}^l/Z_{min}^l\right)}, & \text{otherwise} \end{cases} \tag{6.89}$$

3-c) Estimation of the probability of speech pause The variables used in 3-c) are shown in Table 6.50. As shown

Table 6.50: Definition of variable

Variable	description, a corresponding parameter
$q_n(k_i)$	Probability of speech pause.
a^f	FRAME_VOICEP_PROB_FACTOR. The default value is 0.7.
a^g	GLOBAL_VOICEP_PROB_FACTOR. The default value is 0.9.
a^l	LOCAL_VOICEP_PROB_FACTOR. The default value is 0.9.
q_{min}	MIN_VOICE_PAUSE_PROB. The default value is 0.02.
q_{max}	MAX_VOICE_PAUSE_PROB. The default value is 0.98.

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 $P_n^{f,g,l}(k_i)$.

$$q_n(k_i) = 1 - \left(1 - a^l + a^l P_n^l(k_i)\right)\left(1 - a^g + a^g P_n^g(k_i)\right)\left(1 - a^f + a^f P_n^f(k_i)\right), \tag{6.90}$$

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 (6.77).

$$p_n(k_i) = \left\{1 + \frac{q_n(k_i)}{1 - q_n(k_i)} (1 + \zeta_n(k_i)) \exp\left(-v_n(k_i)\right)\right\}^{-1} \tag{6.91}$$

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 $Y_n(k_i)$.

$$\hat{S}_n(k_i) = Y_n(k_i) G_n^{H1}(k_i) p_n(k_i) \tag{6.92}$$

6.3.8 SpectralGainFilter

Outline of the node

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

Necessary files

No files are required.

Usage

When to use

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

Typical connection

Figure 6.53 shows a connection example of SpectralGainFilter . Inputs are 1) separation spectra output from GHDSS 2) the optimal gain, and 3) the probability of speech presence, output from CalcSpecSubGain for example. In the figure, an audio file is created with this node being connected to Synthesize and SaveRawPCM as an example of output.

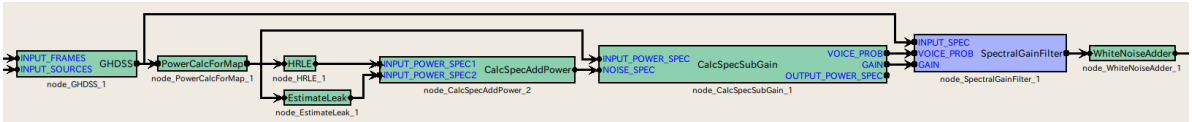


Figure 6.53: Connection example of SpectralGainFilter

Input-output and property of the node

Input

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

VOICE_PROB : `Map<int, ObjectRef>` type. A pair containing a 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

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

Parameter

No parameters

Details of the node

This node multiplies the input speech spectrum by the optimal gain and the probability of speech presence and outputs a separated sound spectrum with emphasized speech. If the separated sound spectrum input is $X_n(k_i)$, 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.

$$Y_n(k_i) = X_n(k_i)G_n(k_i)p_n(k_i) \tag{6.93}$$

6.4 FeatureExtraction category

6.4.1 Delta

Outline of the node

This node acquires dynamic feature vectors from static feature vectors. It is usually connected to the posterior half of MSLSExtraction and MFCCExtraction , which are feature extraction nodes. These feature-extracting nodes acquire static feature vectors while reserving the regions where dynamic features are saved. The dynamical feature of this time is set to 0. The Delta node 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 node is used for obtaining dynamic features from static features. It is usually used after MFCCExtraction and MSLSExtraction .

Typical connection

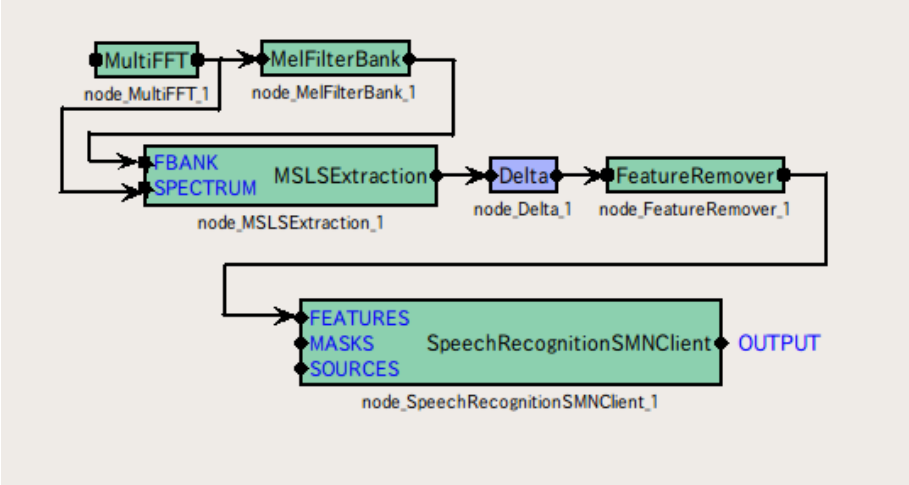


Figure 6.54: Typical connection example for Delta

Input-output and property of the node

Table 6.51: Parameter list of Delta

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	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

FBANK_COUNT : `int` type. Dimension numbers of features to be processed. Its range is positive integers. When connecting it just after the feature extract node, the same FBANK_COUNT used in feature extraction. However, in the case that `true` is selected for the option used for the power term in feature extraction, set FBANK_COUNT +1.

Details of the node

This node 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 FBANK_COUNT are static features. Dynamic features are added to the dimension elements higher than FBANK_COUNT. The input feature vector at the frame time f is expressed as follows.

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

Here, P is FBANK_COUNT.

$$y(f) = [x(f,0), x(f,1), \ldots, x(f,2P-1)]^T \tag{6.95}$$

Each output vector element is expressed as,

$$y(f,p) = \begin{cases} x(f,p), & \text{if } p = 0,\ldots,P-1, \\ w\sum_{\tau=-2}^{2} \tau \cdot x(f+\tau,p), & \text{if } p = P,\ldots,2P-1, \end{cases} \tag{6.96}$$

Here, $w = \frac{1}{\sum_{\tau=-2}^{\tau=2}\tau^2}$. Figure 6.55 shows the input-output flow of Delta .

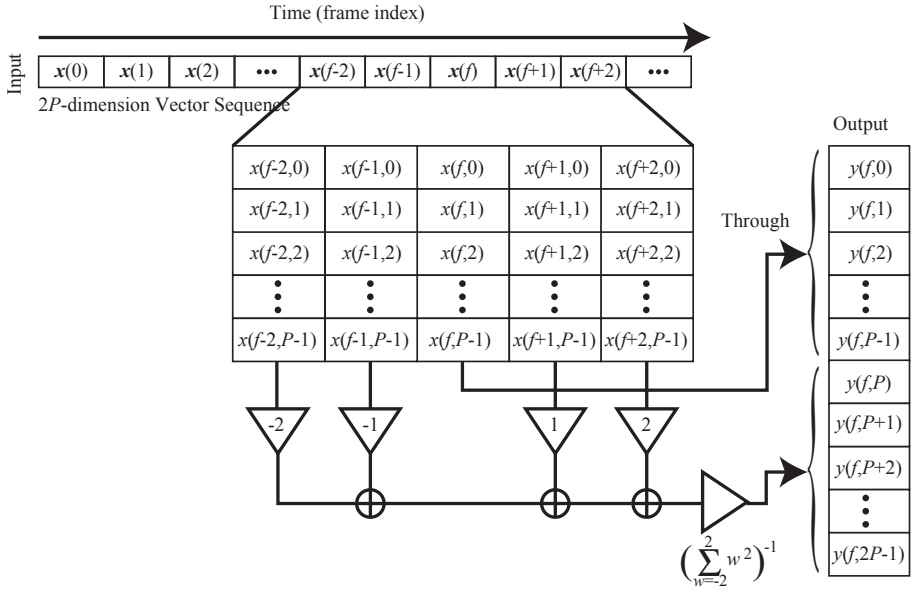


Figure 6.55: Input-output flow of Delta .

6.4.2 FeatureRemover

Outline of the node

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

Necessary file

No files are required.

Usage

When to use

This node 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 node 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 MSLSExtraction and MFCCExtraction and 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 the acoustic feature values, including the logarithmic power. This node is usually connected with the posterior half of Delta and used to remove logarithmic power terms.

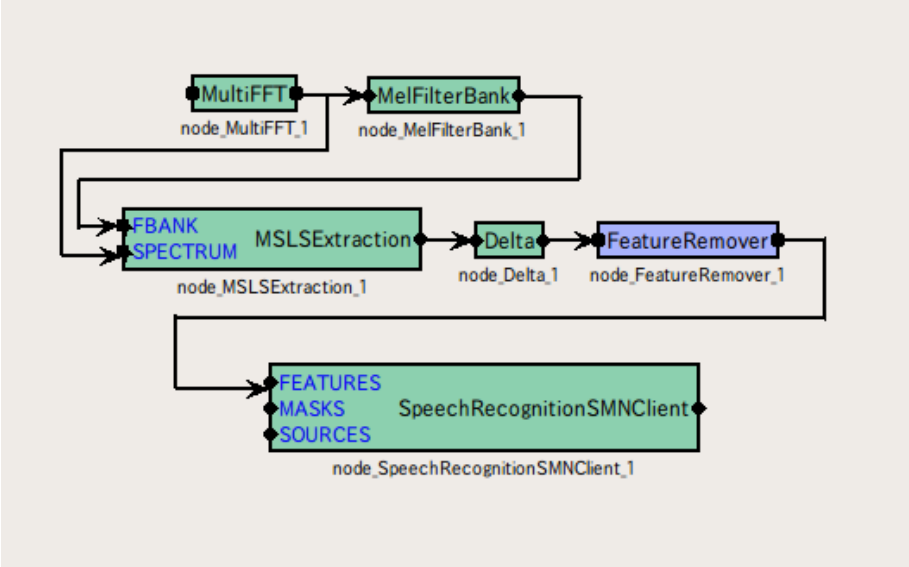


Figure 6.56: Typical connection example of FeatureRemover

Input-output and property of the node

Input

Table 6.52: Parameter list of FeatureRemover

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

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

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 node

This node 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 node

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

Necessary file

No files are required.

Usage

When to use

This node is used as preprocessing for acquiring acoustic features. It is used just after MultiFFT , PowerCalcForMap or PreEmphasis . It is used before MFCCExtraction or MSLSExtraction .

Typical connection

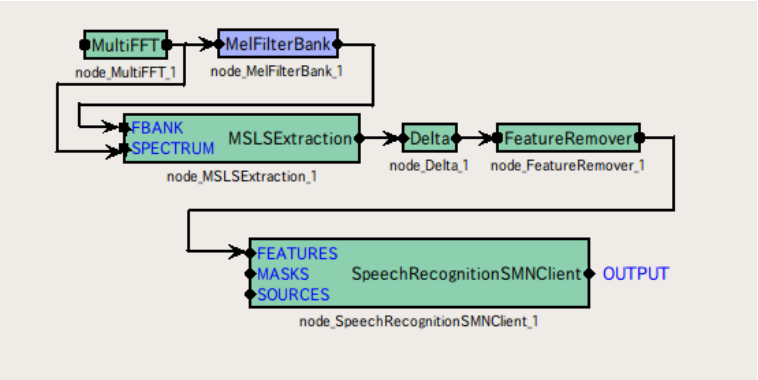


Figure 6.57: Connection example of MelFilterBank

Input-output and property of the node

Table 6.53: Parameter list of MelFilterBank

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	Analysis frame length
SAMPLING_RATE	int	16000	[Hz]	Sampling frequency
CUT_OFF	int	8000	[Hz]	Cut-off frequency of lowpass filter
MIN_FREQUENCY	int	63	[Hz]	Lower cut-off frequency of filter bank
MAX_FREQUENCY	int	8000	[Hz]	Upper limit frequency of filter bank
FBANK_COUNT	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 FBANK_COUNT. Output energy of the filter bank is in the range from 0 to FBANK_COUNT-1 and 0 is in the range from FBANK_COUNT to 2 *FBANK_COUNT-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 integers.

SAMPLING_RATE : `int` type. Sampling frequency. Its range is positive integers.

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

MIN_FREQUENCY : `int` type. Lower cut-off frequency of the filter bank. Its range is positive integers and less than CUT_OFF.

MAX_FREQUENCY : `int` type. Upper limit frequency of the filter bank. Its range is positive integers and less than CUT_OFF.

FBANK_COUNT : `int` type. The number of filter banks. Its range is positive integers.

Details of the node

This node 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 FBANK_COUNT division from the minimum frequency bin SAMPLING_RATE/LENGTH to SAMPLING_RATECUT_OFF/LENGTH Transformation of the linear scale and mel scale is expressed as follows.

$$m = 1127.01048 \log(1.0 + \frac{\lambda}{700.0}) \tag{6.97}$$

However, expression on the linear scale is λ (Hz) and that on the mel scale is m. Figure 6.58 shows an example of the transformation by 8000 Hz. The red points indicate the center frequency of each bank when SAMPLING_RATE is 16000Hz, CUT_OFF is 8000Hz and FBANK_COUNT is 13. The figure shows that the center frequency of each bank is at regular intervals on the mel scale. Figure 6.59 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 6.60 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 6.60 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.

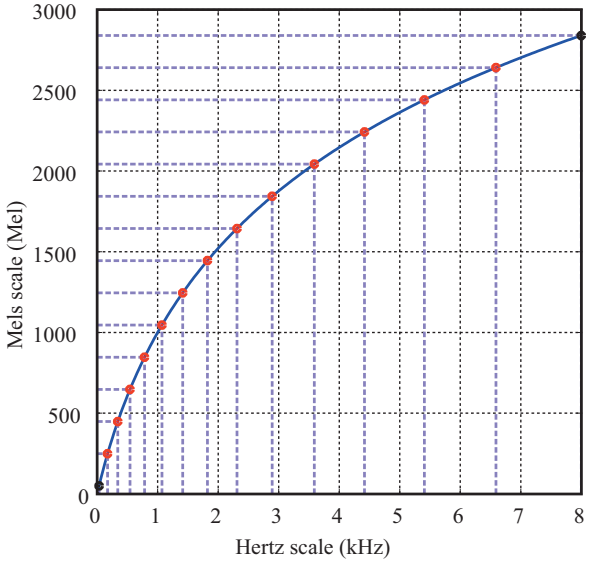


Figure 6.58: Correspondence between linear scale and a mel scale

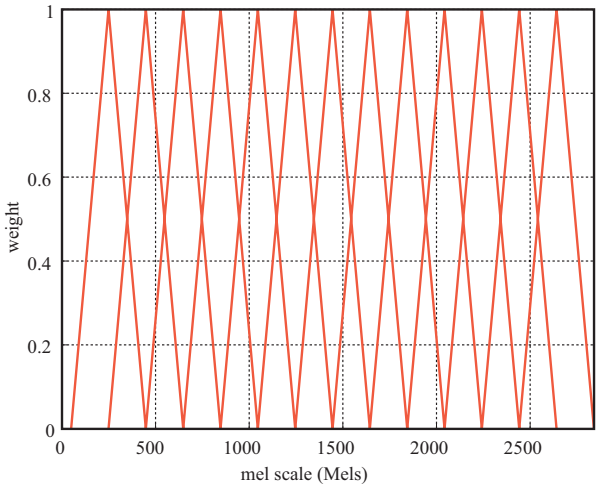


Figure 6.59: Window function on mel scale

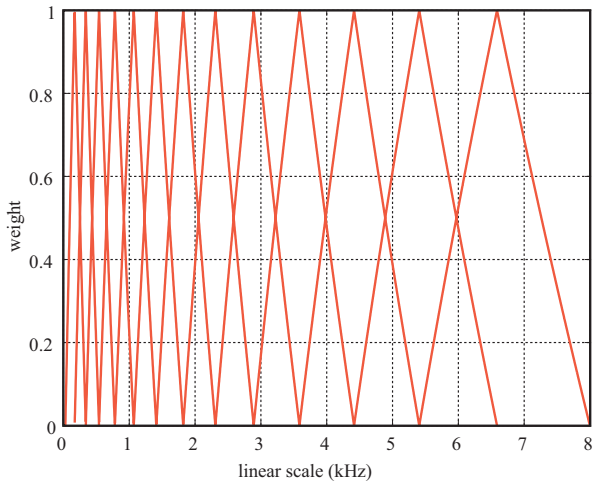


Figure 6.60: Window function on linear scale

6.4.4 MFCCExtraction

Outline of the node

This node acquires mel-cepstrum coefficients (MFCC), which are a type of acoustic feature. It generates acoustic feature vectors consisting of mel-cepstrum coefficients and logarithmic spectrum power as elements.

Necessary file

No files are required.

Usage

When to use

This node 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 node to identify phonemes and speakers.

Typical connection

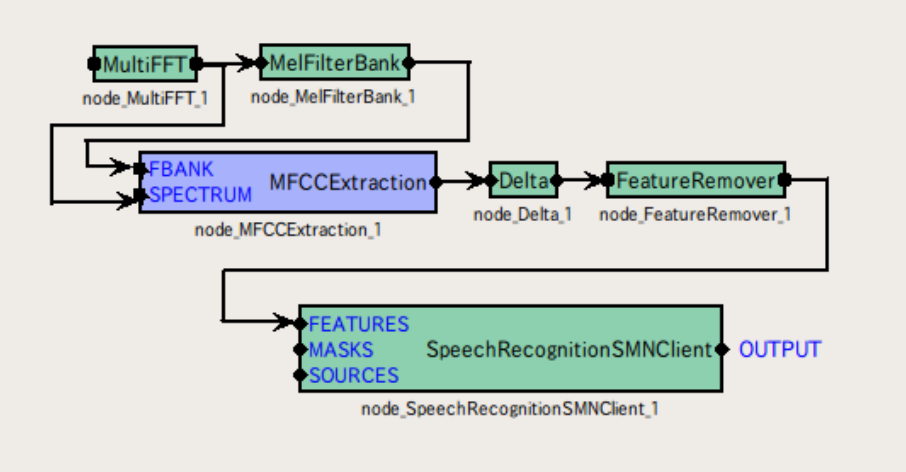


Figure 6.61: Typical connection example of MFCCExtraction

Input-output and property of the node

Table 6.54: 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<complex<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

FBANK_COUNT : `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 FBANK_COUNT. Precise expression is not necessarily optimal for speech recognition and it depends on acoustic environment of the utterances.

NUM_CEP : `int` type. The number of cepstrum coefficients on which to perform liftering. The default value is 12. The range is positive integers. When raising the value, the dimension of acoustic features increases. Acoustic features that express finer spectral changes are obtained.

USE_POWER : 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 node

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

$$x(f) = [x(f,0), x(f,1), \cdots, x(f,P-1)]^T \tag{6.98}$$

Here, P indicates FBANK_COUNT 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 first to Pth dimensions are for mel-cepstrum coefficients and the dimension $P+1$ is for the power term. The output vector of this node is expressed as;

$$y(f) = [y(f,0), y(f,1), \ldots, y(f,P-1), E]^T \tag{6.99}$$

$$y(f,p) = L(p) \cdot \sqrt{\frac{2}{P}} \cdot \sum_{q=0}^{P-1} \left\{ \log(x(q)) \cos\left(\frac{\pi(p+1)(q+0.5)}{P} \right) \right\} \tag{6.100}$$

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

$$L(p) = 1.0 + \frac{Q}{2} \sin\left(\frac{\pi(p+1)}{Q} \right), \tag{6.101}$$

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

$$s = [s(0), \ldots, s(K-1)]^T, \tag{6.102}$$

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

$$E = \log\left(\frac{1}{K} \sum_{k=0}^{K-1} s(k) \right) \tag{6.103}$$

6.4.5 MSLSExtraction

Outline of the node

This node acquires mel-scale logarithmic spectra (MSLS), which are a type of acoustic feature, 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 node. This node is used to generate acoustic feature vectors. For example, acoustic feature vectors are input to the speech recognition node to identify phonemes and speakers.

Typical connection

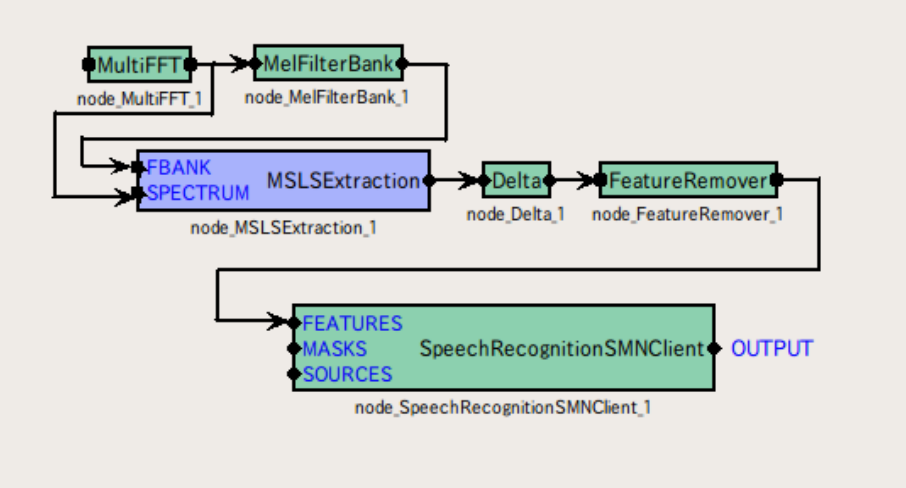


Figure 6.62: Typical connection example of MSLSExtraction

Input-output and property of the node

Table 6.55: Parameter list of MSLSExtraction

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	int	13		The number of filter banks for input spectrum. The implementation
NORMALIZATION_MODE	string	CEPSTRAL		Feature normalization method
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<complex<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 node obtains static features of MSLS, though vectors containing a dynamic feature part are output. The dynamic feature part is set to zero. Figure 6.63 shows the operation.

Parameter

FBANK_COUNT : `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. The current implementation is optimized for the value set at 13. We strongly recommend you use the defalt value. Acoustic features are expressed more finely by setting greater FBANK_COUNT. Precise expression is not necessarily optimal for speech recognition and it depends on the acoustic environment of the utterances.

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

USE_POWER : 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 node

This node 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 node. The calculation method of the MSLS to be output differs depending on the normalization method the user designates. The following are the calculation methods for the output vectors of this node for each normalization method.

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

$$x = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.104}$$

Here, P indicates FBANK_COUNT 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 node is expressed as;

$$y = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.105}$$

$$y(p) = \frac{1}{P}\sum_{q=0}^{P-1}\left\{L(q)\cdot\sum_{r=0}^{P-1}\left\{\log(x(r))\cos\left(\frac{\pi q(r+0.5)}{P}\right)\right\}\cos\left(\frac{\pi q(p+0.5)}{P}\right)\right\} \tag{6.106}$$

Here, the liftering coefficient is expressed as;

$$L(p) = \begin{cases} 1.0, & (p = 0, \ldots, P-1), \\ 0.0, & (p = P, \ldots, 2P-1), \end{cases} \tag{6.107}$$

Here, $Q = 22$.

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

$$x = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.108}$$

Here, P indicates FBANK_COUNT 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 first to the Pth dimensions are for mel-cepstrum coefficients and the dimension $P+1$ is for the power term. The output vector of this node is expressed as:

$$y = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.109}$$

$$y(p) = \begin{cases} (\log(x(p)) - \mu) - 0.9(\log(x(p-1)) - \mu), & if \;\; p = 1, \ldots, P-1 \\ \log(x(p)), & if \;\; p = 0, \end{cases} \tag{6.110}$$

$$\mu = \frac{1}{P}\sum_{q=0}^{P-1} \log(x(q)), \tag{6.111}$$

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:

$$s = [s(0), s(1), \ldots, s(N-1)]^T \tag{6.112}$$

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

$$p = \log\left(\frac{1}{N}\sum_{n=0}^{N-1} s(n)\right) \tag{6.113}$$

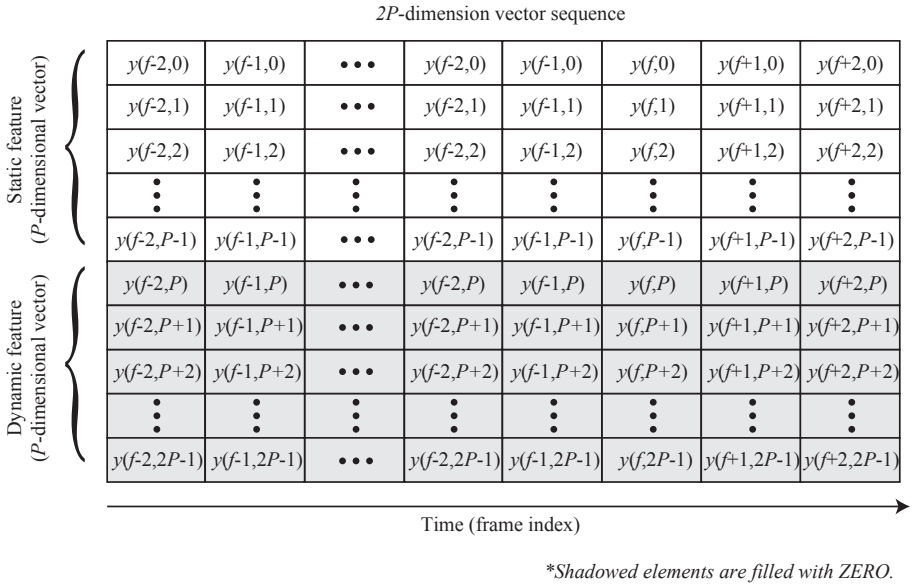


Figure 6.63: Output parameter of MSLSExtraction

6.4.6 PreEmphasis

Outline of the node

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

Necessary files

No files are required.

Usage

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

Typical connection

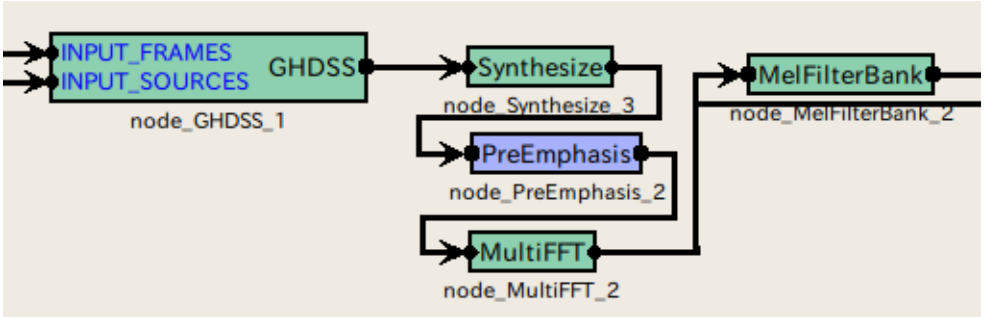


Figure 6.64: Connection example of PreEmphasis

Input-output and property of the node

Table 6.56: Parameter list of PreEmphasis

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

Input

INPUT : `Map<int, ObjectRef>` , When input signals are time domain waveforms, `ObjectRef` points to a `Vector<float>` . If the signals are in the frequency domain, it points to a `Vector<complex<float> >` .

Output

OUTPUT : `Map<int, ObjectRef>` , Signals for which the upper frequency is emphasized. The output corresponds to the type of input; `ObjectRef` refers to `Vector<float>` for time domain waveforms and to `Vector<complex<float> >` for frequency domain signals.

Parameter

LENGTH When INPUT_TYPE is SPECTRUM, LENGTH indicates FFT length and must be equal to the value set in previous nodes. When INPUT_TYPE is WAV, it indicates the length of the signal contained in one frame and must be equal to the value set in previous nodes. Typically the signal length is same as FFT length.

SAMPLING_RATE Similar to LENGTH, it is necessary to make this equal to the value in other nodes.

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

INPUT_TYPE 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 node

The necessity and effects of pre-emphasis on common speech recognition are described in various books and theses. Although it is commonly said that this processing makes the system robust to noise, not much performance difference is obtained with this processing with HARK. This is probably because microphone array processing is performed with HARK. It is necessary to make the audio data parameters equal to those used for the speech recognition acoustic model. In other words, when pre-emphasis is performed for the data used for learning acoustic model, the performance is improved by performing pre-emphasis also for input data. Concretely, PreEmphasis consists of two types of processing depending on the type of input signal.

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.

$$p[t] = \begin{cases} s[t] - c_p \cdot s[t-1] & t > 0 \\ (1 - c_p) \cdot s[0] & t = 0 \end{cases} \tag{6.114}$$

Upper frequency emphasis in frequency domain:

In order to realize a frequency domain filter equivalent to the time domain filter, a frequency domain spectral filter equivalent to the time domain $p[t]$ is used. 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 node

This node saves feature vectors in files.

Necessary file

No files are required.

Usage

When to use

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

Typical connection

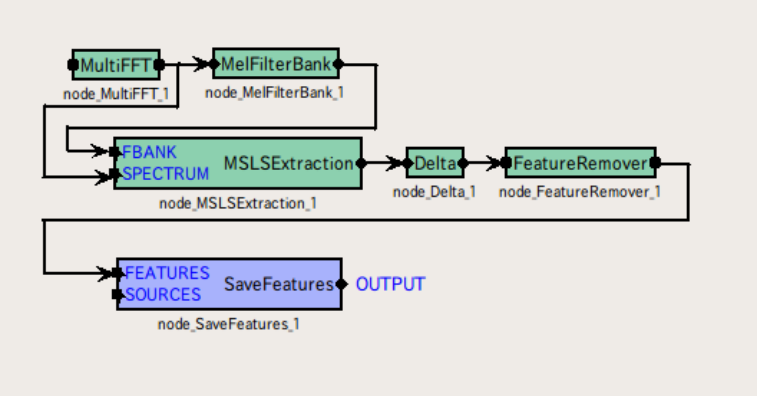


Figure 6.65: Connection example of SaveFeatures

Input-output and property of the node

Table 6.57: 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. The value is a feature vector (`Vector<float>`).

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 node

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

6.4.8 SaveHTKFeatures

Outline of the node

This node 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 node 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

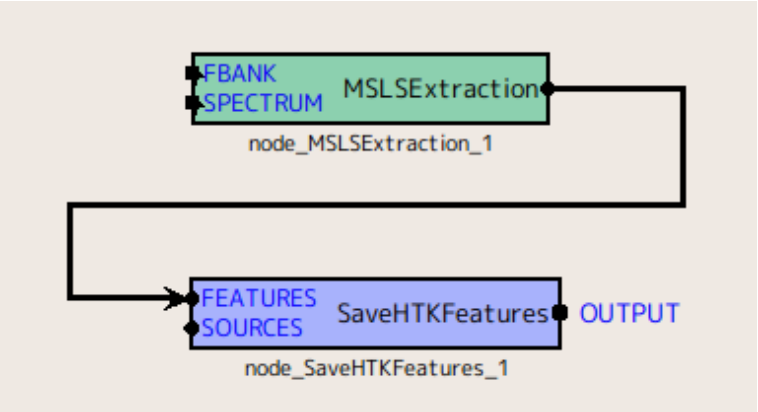


Figure 6.66: Connection example of SaveHTKFeatures

Input-output and property of the node

Table 6.58: Parameter list of SaveHTKFeatures

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

Input

FEATURES : `Map<int, ObjectRef>` 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.

HTK_PERIOD : 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.

FEATURE_TYPE : Designation of type of features to be treated in HTK. Follow 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 the HTKbook for the details.

Details of the node

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

6.4.9 SpectralMeanNormalization

Outline of the node

This node 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 node is used to subtract the mean of acoustic features. This node can remove 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 creation 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. Differences in the properties of microphones appears as a mismatch of the acoustic features of the recording sound, which causes recognition performance degradation. The difference in properties of microphones does not change with time and appears as a 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

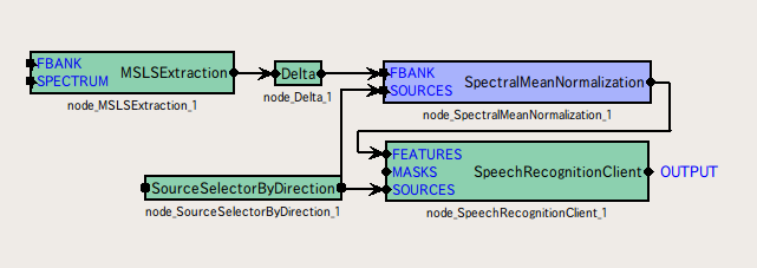


Figure 6.67: Connection example of SpectralMeanNormalization

Input-output and property of the node

Table 6.59: Parameter list of SpectralMeanNormalization

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	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

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

Details of the node

This node 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 is maintained in memory as the mean for the direction of the utterance concerned for subsequent mean subtraction. 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 node

This node obtains dynamic missing feature mask vectors from static missing feature mask vectors. It generates mask vectors consisting of the missing feature mask vectors of static and dynamic features.

Necessary file

No files are required.

Usage

When to use

This node 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

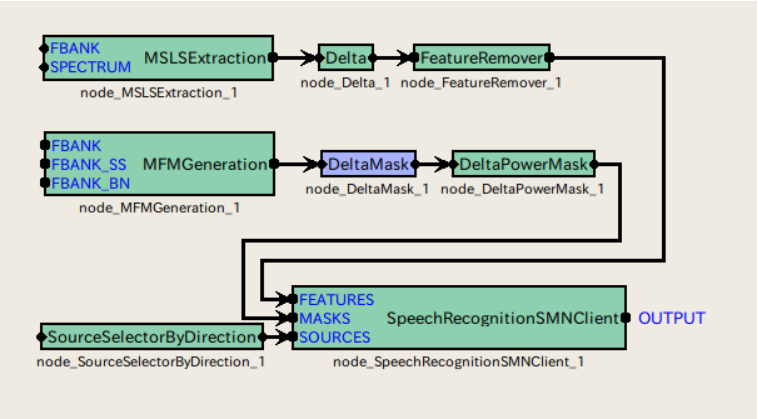


Figure 6.68: Typical connection example of DeltaMask

Input-output and property of the node

Table 6.60: Parameter list of DeltaMask

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

Input

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

FBANK_COUNT : `int` type. The number of feature dimensions to process. Its range is positive integer.

Details of the node

This node obtains missing feature mask vectors of dynamic features from those of static features and generates mask 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;

$$m(f) = [m(f,0), m(f,1), \ldots, m(f,2P-1)]^T \tag{6.115}$$

Here, P indicates the number of dimensions of static features among the input mask vectors and is given in FBANK_COUNT. The mask values for the dynamic features are obtained from those of the static features: the output is substituted into the dimensional elements from P to $2P-1$. The output vector $m'(f)$ is expressed as follows.

$$y'(f) = [m'(f,0), m'(f,1), \ldots, m'(f,2P-1)]^T \tag{6.116}$$

$$m'(f,p) = \begin{cases} m(f,p), & \text{if } p = 0, \ldots, P-1, \\ \prod_{\tau=-2}^{2} m(f+\tau, p), & \text{if } p = P, \ldots, 2P-1, \end{cases} \tag{6.117}$$

Figure 6.69 shows an input-output flow of DeltaMask .

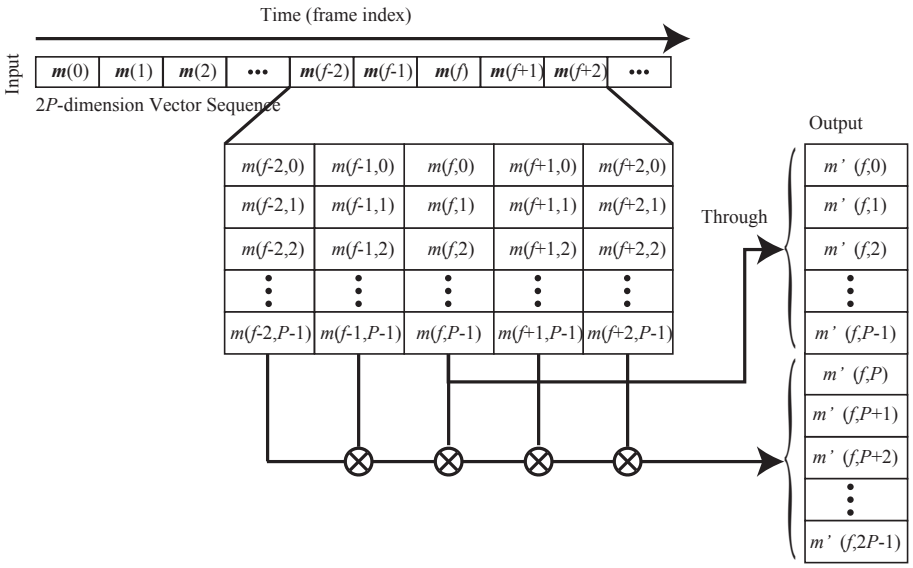


Figure 6.69: Input-output flow of DeltaMask .

6.5.2 DeltaPowerMask

Outline of the node

This node generates mask values for dynamic logarithmic power, which is a kind of acoustic feature. 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 node is used to perform speech recognition by masking features depending on reliability based on missing feature theory. It is usually used for the posterior half of DeltaMask .

Typical connection

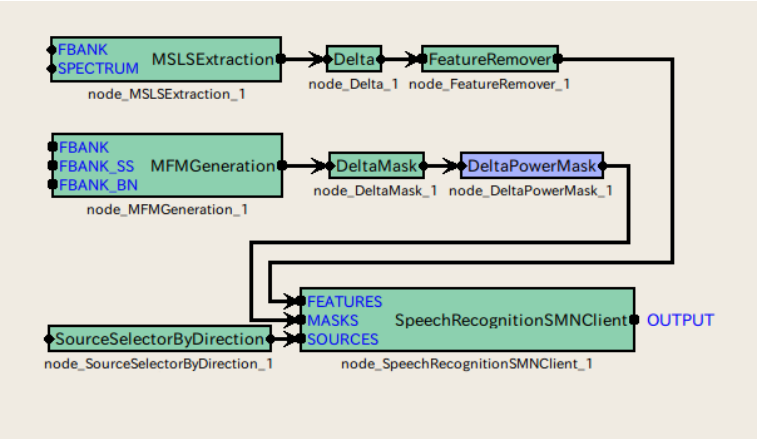


Figure 6.70: Typical connection example of DeltaPowerMask

Input-output and property of the node

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. The dimension size is one more than the input dimension.

Parameter

Details of the node

This node 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 of the output mask is the mask's dimension+1.

6.5.3 MFMGeneration

Details of the node

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

Necessary file

No files are required.

When to use

This node is used for performing speech recognition based on the missing feature theory. MFMGeneration generates Missing Feature Masks from the outputs of PostFilter and GHDSS . Therefore, PostFilter and GHDSS are used as a prerequisite.

Typical connection

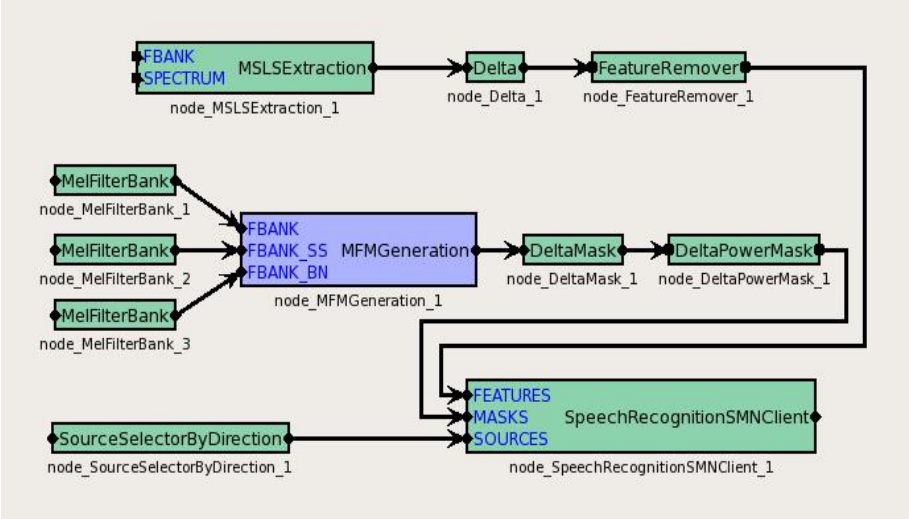


Figure 6.71: Connection example of MFMGeneration

Input-output and property of the node

Table 6.61: Parameter list of MFMGeneration

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	`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 data pair consisting of the sound source ID and a vector of mel filter bank output energy (`Vector<float>` type) obtained from the output of PostFilter .

FBANK_GSS : `Map<int, ObjectRef>` type. A data pair consisting of the sound source ID and a vector of mel filter bank output energy (`Vector<float>` type) obtained from the output of GHDSS .

FBANK_BN : `Map<int, ObjectRef>` type. A data pair consisting of the sound source ID and a vector of mel filter bank output energy (`Vector<float>` type) obtained from the output of BGNEstimator .

Output

OUTPUT : `Map<int, ObjectRef>` type. A data pair consisting of the sound source ID and a missing feature vector of type `Vector<float>` . Vector elements are 0.0 (not reliable) or 1.0 (reliable). The output vector is of dimension 2*FBANK_COUNT, and dimension elements greater than FBANK_COUNT are all 0. These elements are placeholders, which will later store the dynamic information of the Missing Feature Masks.

Parameter

FBANK_COUNT : `int` type. The 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 and it becomes equivalent to normal speech recognition processing.

Details of the node

This node 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 , GHDSS and BGNEstimator . Here, the mask vector of the frame number f is expressed as:

$$m(f) = [m(f,0), m(f,1), \ldots, m(f,P-1)]^T \tag{6.118}$$

$$m(f,p) = \begin{cases} 1.0, & r(p) > \text{THRESHOLD} \\ 0.0, & r(p) \leq \text{THRESHOLD} \end{cases}, \tag{6.119}$$

$$r(p) = \min(1.0, (f(p) + 1.4 * b(p))/(fg(p) + 1.0)), \tag{6.120}$$

$$\tag{6.121}$$

Here, P is the dimension number of the input feature vector and is a positive integer designated in FBANK_COUNT. The dimension number of the vector actually output is 2*FBANK_COUNT. Dimension elements more than FBANK_COUNT are filled up with 0. This is a placeholder for dynamic feature values. Figure 6.72 shows a schematic view of an output vector sequence.

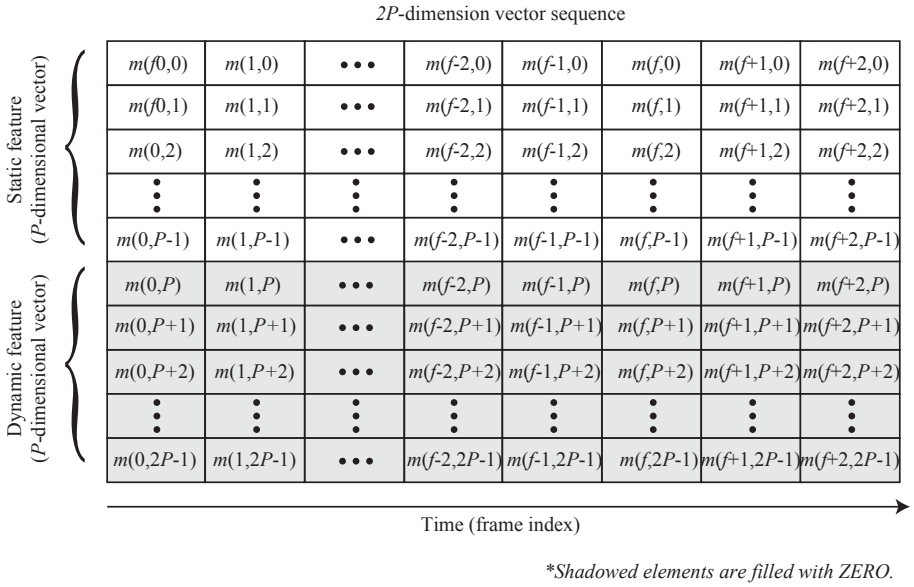


Figure 6.72: Output vector sequence of MFMGeneration

6.6 ASRIF category

6.6.1 SpeechRecognitionClient

Outline of the node

This node sends acoustic features to a speech recognition node via a network connection.

Necessary file

No files are required.

When to use

This node 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)] to perform speech recognition.

Typical connection

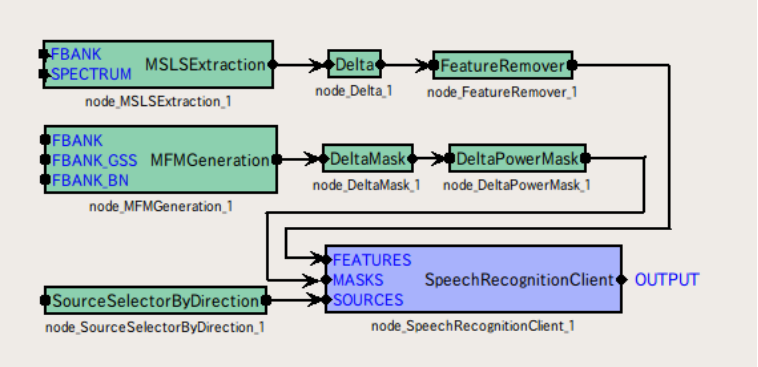


Figure 6.73: Connection example of SpeechRecognitionClient

Input-output and property of the node

Table 6.62: Parameter list of SpeechRecognitionClient

Parameter name	Type	Default value	Unit	Description
MFM_ENABLED	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 on which Julius/Julian is running
PORT	int	5530		Port number for sending out to network
SOCKET_ENABLED	bool	true		The flag that determines whether or not to output to the socket

Input

FEATURES : `Map<int, ObjectRef>` type. A data pair consisting of a sound source ID and feature vector of type `Vector<float>` .

MASKS : `Map<int, ObjectRef>` type. A data pair consisting of a sound source ID and mask vector of type `Vector<float>` .

SOURCES : `Vector<ObjectRef>` type.

Output

OUTPUT : `Vector<ObjectRef>` type.

Parameter

MFM_ENABLED : `bool` type. When `true` is selected, MASKS is transmitted. When `false` is selected, MASKS input is ignored; a mask of all 1's is transmitted.

HOST : `string` type. IP address of a host that transmits acoustic parameters. When SOCKET_ENABLED is set to `false`, it is not used.

PORT : `int` type. The socket number to transfer acoustic parameters. When SOCKET_ENABLED is set to `false`, it is not used.

SOCKET_ENABLED : `bool` type. When `true`, acoustic parameters are transmitted to the socket and when `false`, they are not transmitted.

Details of the node

When MFM_ENABLED is set to `true` and SOCKET_ENABLED, this node sends acoustic features and mask vectors to the speech recognition module via the network port. When `false` is selected for MFM_ENABLED, normal speech recognition not based on the missing feature theory is performed. In practice, mask vectors are sent out with all elements set to 1, all acoustic features as reliable in other words. When `false` is selected for SOCKET_ENABLED, the features are not sent to the speech recognition node. This is used to perform checks of the HARK network file without running the external speech recognition engine. For HOST, designate the IP address of HOST on which the external program that sends vectors runs. For PORT, designate a network port number to send the vector.

References:

(1) http://julius.sourceforge.jp/en_index.php

6.6.2 SpeechRecognitionSMNClient

Outline of the node

This node sends acoustic features to the speech recognition node via a network connection. The main difference from SpeechRecognitionClient is that this node 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 node.

Necessary file

No files are required.

Usage

When to use

This node is used to send acoustic features to software outside of HARK. For example, it sends them to the large vocabulary continuous speech recognition software Julius [(1)] to perform speech recognition.

Typical connection

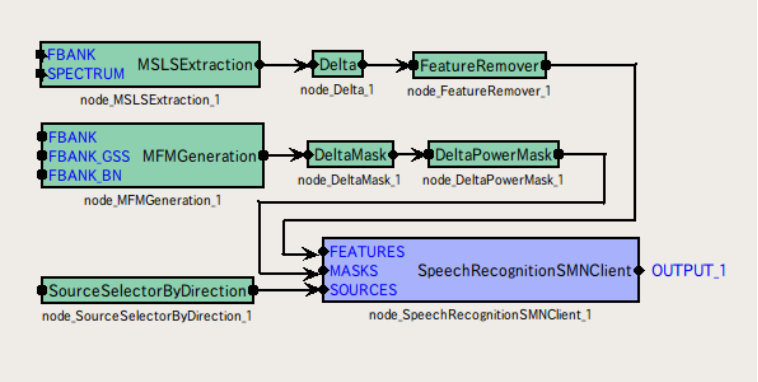


Figure 6.74: Connection example of SpeechRecognitionSMNClient

Input-output and property of the node

Table 6.63: Parameter list of SpeechRecognitionSMNClient

Parameter name	Type	Default value	Unit	Description
MFM_ENABLED	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 on which Julius/Julian is running
PORT	int	5530		Port number for sending out to network
SOCKET_ENABLED	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 : `Vector<ObjectRef>` type.

Output

OUTPUT : `Vector<ObjectRef>` type.

Parameter

MFM_ENABLED : `bool` type. When `true` is selected, MASKS is transmitted. When `false` is selected, MASKS input is ignored, a mask of all 1's is transmitted.

HOST : `string` type. The IP address of a host that transmits acoustic parameters. When SOCKET_ENABLED is set to `false`, it is not used.

PORT : `int` type. The socket number to transfer acoustic parameters.When SOCKET_ENABLED is set to `false`, it is not used.

SOCKET_ENABLED : `bool` type. When `true`, acoustic parameters are transmitted to the socket and when `false`, they are not transmitted.

When MFM_ENABLED is set to `true` and SOCKET_ENABLED, this node sends acoustic features and mask vectors to the speech recognition node via the network port. When `false`is selected for MFM_ENABLED, speech recognition not based on 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 SOCKET_ENABLED, the features are not sent to the speech recognition node. This node 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 is running. For PORT, designate a network port number to send the vector.

References:

(1) http://julius.sourceforge.jp/en_index.php

6.7 MISC category

6.7.1 ChannelSelector

Outline of the node

This node extracts data from specified channels, in specific order, from multichannel speech waveforms or multichannel complex spectrum.

Necessary files

No files are required.

Usage

When to use

This node is used to delete unnecessary channels from multichannel speech input waveforms/spectrum, changing the alignment of channels and copying channels.

Typical connection

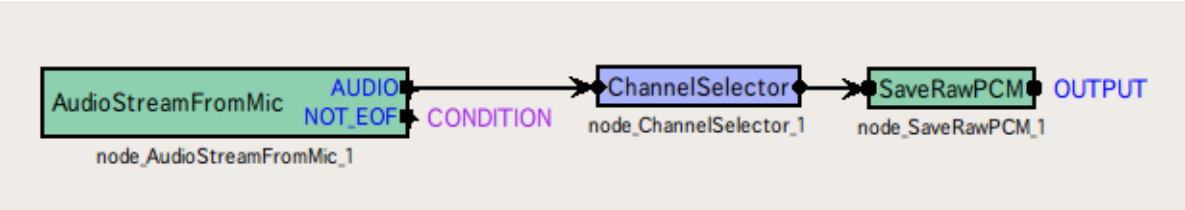


Figure 6.75: Example of a typical connection of ChannelSelector

Figure 6.75 shows an example of a typical connection. Several, but not all, channels will be extracted from the multichannel audio files in this network. The main input sources are AudioStreamFromMic , AudioStreamFromWave and MultiFFT , and the main output sources are SaveRawPCM , MultiFFT .

Input-output and property of the node

Input

INPUT : `Matrix<float>` or `Matrix<complex<float> >` type. Multichannel speech waveform/spectrum data.

Output

OUTPUT : `Matrix<float>` or `Matrix<complex<float> >` type. Multichannel speech waveform/spectrum data.

Parameter

Parameters of ChannelSelector

Parameter name	Type	Default value	Unit	Description
SELECTOR	`Object`	`Vector< int >`		Designate the number of channels to output

SELECTOR : `Object` type. No default value. Designate the numbers of the channels to be used. Channel numbers begin with 0. For example, to use only Channels 2, 3 and 4 of the five channels (0-4), use the values shown in (1) below; when swapping Channels 3 and 4, use the values shown in (2).

(1) `<Vector<int> 2 3 4>`

(2) `<Vector<int> 2 4 3>`

Details of the node

This node extracts only the speech waveform/spectrum data from the channels designated in the $N \times M$ type matrix (`Matrix`) of input and output data as a new $N' \times M$ type matrix. Here, N indicates the number of input channels and N' indicates the number of output channels.

6.7.2 CombineSource

Outline of the node

This node combines two sound source localization results coming from, for instance, ConstantLocalization or LocalizeMUSIC .

Necessary files

No files are required.

Usage

When to use

This node is useful if you want to use sound source localization results coming from more than two nodes in the latter processes such as in GHDSS , etc. The followings are paticular examples of the usage.

- Combination of localization results from two LocalizeMUSIC
- Combination of localization results from both ConstantLocalization and LocalizeMUSIC

Typical connection

This node takes results of two sound source localization modules as inputs. The candidates of preceding modules are ConstantLocalization , LoadSourceLocation , and LocalizeMUSIC . Since the output is also the combination of the two input sound source results, the post-modules of this node can be for example GHDSS and SpeechRecognitionClient , which needs sound source localization results.

Figure 6.76 shows the network example that combines the two sound source localization results from LocalizeMUSIC and ConstantLocalization .

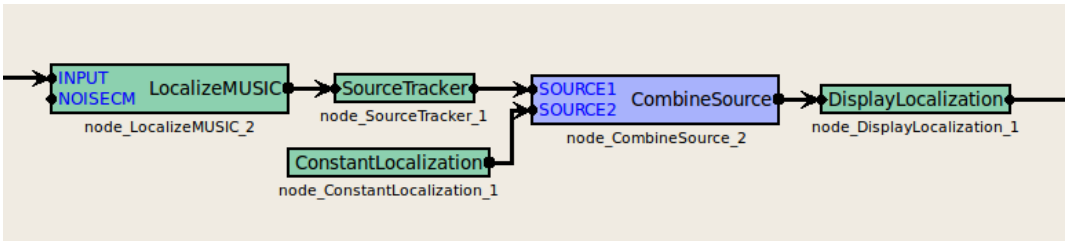


Figure 6.76: Example of a connection of CombineSource

Input-output and properties of the node

Input

SOURCES1 : `Vector<ObjectRef>` type. The first source location result combined by this node. `ObjectRef` refers to the type of `Source` data.

SOURCES2 : `Vector<ObjectRef>` type. The second source location result combined by this node. `ObjectRef` refers to the type of `Source` data.

Output

SRCOUT : `Vector<ObjectRef>` type. Source location results after combination. `ObjectRef` refers to the type of `Source` data.

Parameter

No parameters.

Details of the node

This node combines two sound source localization results from the input ports and output as one result. The azimuth, elevation, power, and IDs of the sound sources are inherited.

6.7.3 DataLogger

Outline of the node

This node provides specific labels to parameters of input and output data as standard outputs or to files.

Necessary files

No files are required.

Usage

When to use

This node is used to debug nodes or when saving outputs of a node and using them for experiments and analyses.

Typical connection

When outputting features of each sound source in text and analyzing them, connect the node as shown in Figure 6.77.

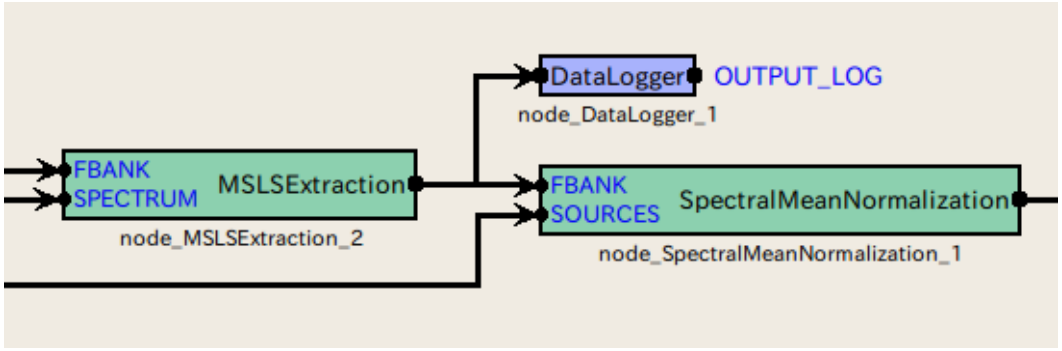


Figure 6.77: Connection example of DataLogger

Input-output and property of the node

Table 6.64: Parameter list of DataLogger

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 the character strings to be given to output data so that the user can determine the DataLogger outputs the result when multiple DataLogger s are used.

Details of the node

This node provides specific labels to the parameters of input and output data to standard outputs or to les.

This node gives a label specified by LABEL to a input data, and outputs to standard output or to a file. Currently, the node supports `Map` type whose key is Source ID, which is the most frequently used type.

The output format is: `Label Frame-count key1 value1 key2 value2 $\dots$` For each frame, the node outputs one line as the format shown above. Label and Frame-count denote the label specified by LABEL and the index of the frame. keys and values are the same as `Map` . These values are separated by a space.

6.7.4 MatrixToMap

Outline of the node

This node converts `Matrix<float>` type and `Matrix<complex<float> >` type data into `Map<int, ObjectRef>` type data.

Necessary files

No files are required.

Usage

When to use

This node is used when the input is connected to nodes that accept only `Map<int, ObjectRef>` type, such as PreEmphasis , MelFilterBank or SaveRawPCM .

Typical connection

Figures 6.78 and 6.79 show connection examples for the MatrixToMap node. In Figure 6.78, the speech waveform data is taken from microphones in the AudioStreamFromMic node, necessary channels are sorted in the ChannelSelector node and `Matrix<float>` type data are converted into the `Map<int, ObjectRef>` type using the MatrixToMap node.

The output is connected to the SaveRawPCM node, and waveforms are saved as files. Figure 6.79 shows the usage of `Map<int, ObjectRef>` type to obtain waveform spectra as a `Map<int, ObjectRef>` . The MultiFFT node can be connected to either side of MatrixToMap .

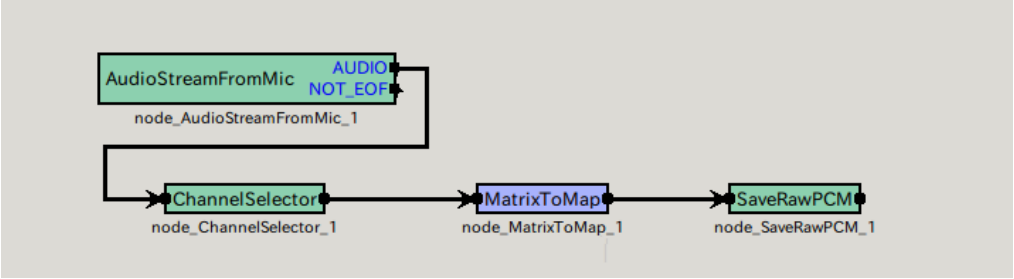


Figure 6.78: Example of a connection to MatrixToMap

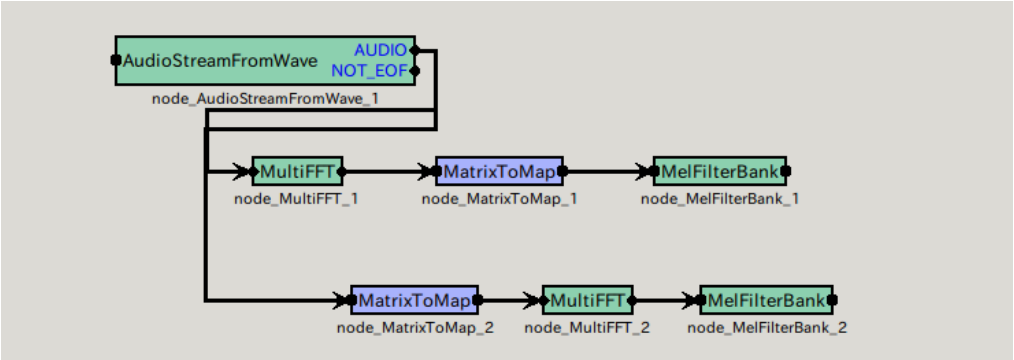


Figure 6.79: Example of a connection of MatrixToMap to MultiFFT

Input-output and property of the node

Input

INPUT : `Matrix<float>` or `Matrix<complex<float> >` types.

Output

OUTPUT : `Map<int, ObjectRef>` type. The structure of the same ID for input data.

Parameter No parameters.

Details of the node

ID: Values of ID are always 0.

6.7.5 MultiDownSampler

Outline of the node

This node performs downsampling of input signals and outputs their results. The window method is used for low-pass filters and its window function is Kaiser window.

Necessary files

No files are required.

Usage

When to use

This node is used when the sampling frequency of input signals is not 16 kHz. For the HARK nodes, the default sampling frequency is 16kHz. If, for example, the input signals are 48 kHz, downsampling is required to reduce the sampling frequency to 16 kHz. **Note 1 (Range of ADVANCE):** To make processing more convenient, it is necessary to limit the parameter settings of input nodes that are connected before nodes such as AudioStreamFromMic and AudioStreamFromWave . Differences in the parameters LENGTH and ADVANCE: OVERLAP = LENGTH - ADVANCE must be sufficiently large. More concretely, the differences must be greater than the low-pass filter length N of this node. Values over 120 are sufficient for the default setting of this node 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 (Setting of ADVANCE):** The ADVANCE value of this node must be SAMPLING_RATE_IN / SAMPLING_RATE_OUT times as great as the ADVANCE value of the node connected afterward (e.g. GHDSS). Since this is a specification, its operation is not guaranteed with values other than those above. For example, if ADVANCE = 160 and of SAMPLING_RATE_IN / SAMPLING_RATE_OUT is 3 for the node connected later, it is necessary to set the ADVANCE of this node and that connected before to 480. **Note 3 (Requirements for the LENGTH value of the node connected before this node):** The LENGTH value of the node connected before this node (e.g. AudioStreamFromMic) must be SAMPLING_RATE_IN / SAMPLING_RATE_OUT times as great as the ADVANCE value of the node connected afterward (e.g. GHDSS). For example, if SAMPLING_RATE_IN / SAMPLING_RATE_OUT is 3, and LENGTH is 512 and ADVANCE is 160 for GHDSS , then LENGTH should be 1536 and ADVANCE should be 480 for AudioStreamFromMic .

Typical connection

Examples of typical connections are shown below. This network file reads Wave file inputs, performs downsampling and saves files as Raw files. Wave file input is achieved by connecting Constant , InputStream and AudioStreamFromMic . This is followed by downsampling with MultiDownSampler , with output waveforms saved in SaveRawPCM .

Input-output and property of the node

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 a Kaiser window, are mostly set for the parameters. Figure 6.82 shows the relationships between symbols and filter properties. Note the correspondence when reading them.

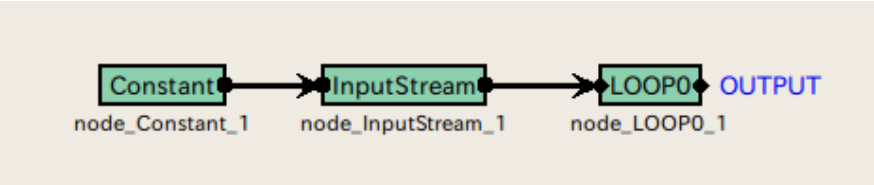


Figure 6.80: Example of a connection of MultiDownSampler : Main network

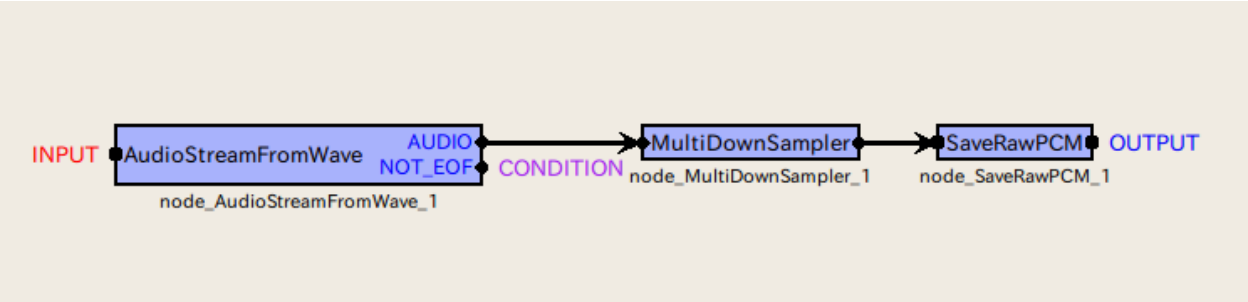


Figure 6.81: Example of a connection of MultiDownSampler : Iteration (LOOP0) Network

Table 6.65: 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.
SAMPLING_RATE_IN	int	48000	[Hz]	Sampling frequency of INPUT signals.
SAMPLING_RATE_OUT	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 : int type. The default value is 480. For processing frames for speech waveforms, designate the shift width on waveforms in sampling numbers. Here, use the values of the nodes connected prior to INPUT. **Note:** This value must be SAMPLING_RATE_IN / SAMPLING_RATE_OUT times as great as the ADVANCE value set for after OUTPUT.

SAMPLING_RATE_IN : int type. The default value is 48000. Designate sampling frequency for input waveforms.

SAMPLING_RATE_OUT : int type. The default value is 16000. Designate sampling frequency for output waveforms. Values that can be used are 1 / integer of SAMPLING_RATE_IN.

Wp : float type. 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 : float type. 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 tilter begin to be stable from around $48000 * 0.34 = 16320$ [Hz].

As : float type. The default value is 50. Designate the value indicating 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.

When Wp, Ws and As are set at their default values, Wp and Ws will be around the cutoff frequency Ws. For example, the accuracy of the frequency response characteristic of Kaiser window will improve. However, the dimensions of the low-pass filter and the processing time will be increased. This relationship is considered a trade off.

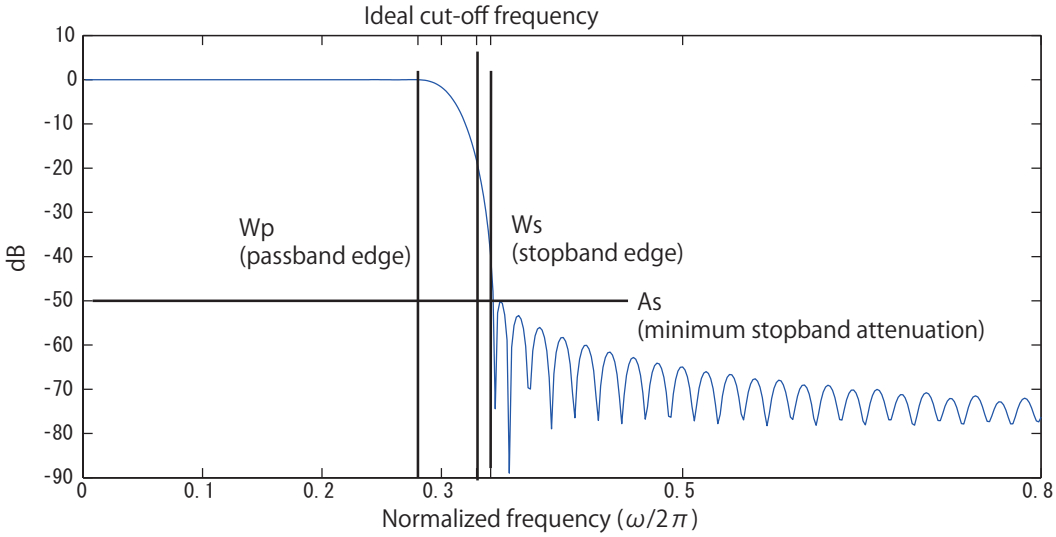


Figure 6.82: Filter properties of default settings.

Details of the node

MultiDownSampler is the node that uses the low-pass filter for band limiting, using the Kaiser window for multi-channel signals and downsampling. This node downsamples SAMPLING_RATE_OUT/SAMPLING_RATE_IN after creating / executing an FIR low-pass filter by synthesizing 1) a Kaiser window and 2) ideal low-pass responses.

FIR filter: Filtering with a finite impulse response $h(n)$ is performed based on the equation

$$s_{\text{out}}(t) = \sum_{i=0}^{N} h(n) s_{\text{in}}(t-n). \tag{6.122}$$

Here, $s_{\text{out}}(t)$ indicates output signals and $s_{\text{in}}(t)$ indicates input signals. For multichannel signals, the signals of each channel are filtered independently. The same finite impulse response $h(n)$ is used here. **Ideal low-pass response:** The

ideal low-pass response with a cutoff frequency of ω_c is obtained using the equation

$$H_i(e^{j\omega}) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \text{otherwise} \end{cases} \tag{6.123}$$

This impulse response is expressed as

$$h_i(n) = \frac{\omega_c}{\pi}\left(\frac{sin(\omega n)}{\omega n}\right), \quad -\infty \le n \le \infty \tag{6.124}$$

This impulse response does not satisfy the acausal and bounded input-bounded output (BIBO) stability conditions. It is therefore necessary to cut off the impulse response in the middle to obtain the FIR filter from this ideal filter.

$$h(n) = \begin{cases} h_i(n), & |n| \le \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.125}$$

Here, N indicates a dimension of the filter. In this filter, cutoff of the impulse response results in ripples in the pass band and stopband. Moreover, the minimum attenuation in stopband As remains around 21dB and sufficient attenuation is not obtained.

Low-pass filter by the window method with Kaiser window: To improve the properties of the above cutoff method, an impulse response, in which the ideal impulse response $h_i(n)$ is multiplied by the window function $v(n)$, is used instead.

$$h(n) = h_i(n)v(n) \tag{6.126}$$

Here, the low-pass filter is designed with the Kaiser window. The Kaiser window is defined by the equation

$$v(n) = \begin{cases} \frac{I_0\left(\beta\sqrt{1-(nN/2)^2}\right)}{I_0(\beta)}, & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.127}$$

Here, β indicates the parameter determining the shape of the window and $I_0(x)$ indicates the modified Bessel function of 0th order. The Kaiser window is obtained using the equation

$$I_0(x) = 1 + \sum_{k=1}^{\infty}\left(\frac{(0.5x)^k}{k!}\right) \tag{6.128}$$

The parameter β is determined by the attenuation obtained by the low-pass filter. Here, it is determined by the index,

$$\beta = \begin{cases} 0.1102(As - 8.7) & As > 50, \\ 0,5842(As - 21)^{0.4} + 0.07886(As - 21) & 21 < As < 50, \\ 0 & As < 21 \end{cases} \tag{6.129}$$

If the cutoff frequency ω_c and the filter order have been determined, the low-pass filter can be determined by the window method. The filter order N can be estimated with the minimum attenuation in stopband As and the transition region $\Delta f = (Ws - Wp)/(2\pi)$ as,

$$N \approx \frac{As - 7.95}{14.36\Delta f} \tag{6.130}$$

Moreover, the cutoff frequency ω_c is set to $0.5(Wp + Ws)$. **Downsampling:** Downsampling is realized by thinning

the sample points of SAMPLING_RATE_IN / SAMPLING_RATE_OUT from the signals that pass the low-pass filter. For example, 48000/16000 = 3 in the default setting; therefore, input samples taken once every 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.

6.7.6 MultiFFT

Outline of the node

This node performs Fast Fourier Transforms (FFT) on multichannel speech waveform data.

Necessary files

No files are required.

Usage

When to use

This node is used to convert multichannel speech waveform data into spectra and analyze the spectra with time frequency domains. It is often used as preprocessing of feature extractions for speech recognition.

Typical connection

Figure 6.83 shows an example, in which inputs of `Matrix<float>` and `Map<int, ObjectRef>` types are provided to the MultiFFT node. The path in Figure 6.83 receives multichannel acoustic signals of `Matrix<float>` type from the AudioStreamFromWave node. The signals are converted into `Matrix<complex<float> >` type complex spectra in the MultiFFT node and input to the LocalizeMUSIC node.

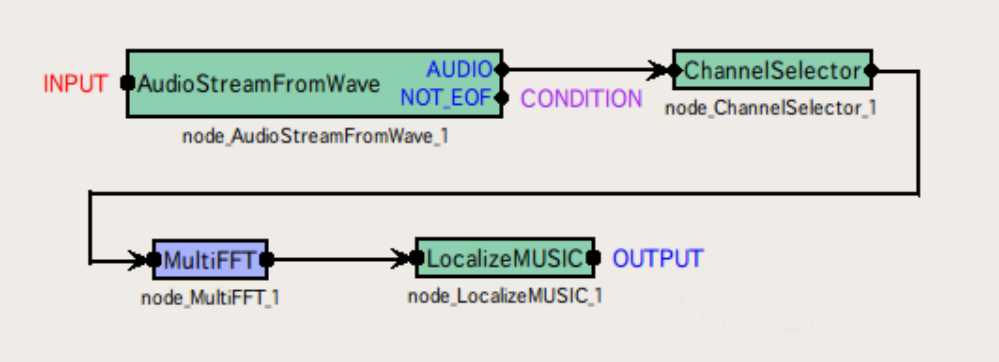


Figure 6.83: Example of a connection of MultiFFT

Input-output and property of the node

Table 6.66: 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.
WINDOW_LENGTH	`int`	512	[pt]	Length of window function when performing Fourier transform.

Input

INPUT : `Matrix<float>` or `Map<int, ObjectRef>` types. Multichannel speech waveform data. If 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 spectra corresponding to inputs. When the inputs are `Matrix<float>` type, the outputs are `Matrix<complex<float> >` type; 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. It must be expressed in powers of 2 to meet the properties of the algorithm. Moreover, it must be greater than WINDOW_LENGTH.

WINDOW : `string` type. The default value is CONJ. Select from CONJ, HAMMING and RECTANGLE, which indicate complex, hamming and rectangular windows, respectively. HAMMING windows are often used for audio signal analyses.

WINDOW_LENGTH : `int` type. The default value is 512. Designate the length of a window function. If this value increases, so does the frequency resolution, while the temporal resolution decreases. Intuitively, an increase in window length makes it more sensitive to differences in the pitch of sound while becoming less sensitive to changes in pitch.

Details of the node

Rough estimates of LENGTH and WINDOW_LENGTH: It is appropriate to analyze audio signals with frame length of 20 ~ 40 [ms]. If the sampling frequency is f_s [Hz] and the temporal length of a window is x [ms], the frame length L [pt] can be expressed as

$$L = \frac{f_s x}{1000}$$

For example, if the sampling frequency is 16 [kHz], the default value 512 [pt] will correspond to 32 [ms]. Powers of 2 are suited for the parameter LENGTH of Fast Fourier Transform. Select 512. WINDOW_LENGTH, the window function, is set at 400 [pt], corresponding to 25 [ms] when the sampling frequency is 16 [kHz] in some cases to designate a frame length more suited for acoustic analyses.

Shape of each window function: The shape of each window function $w(k)$ is defined by k, the index of a sample; L, the length of a window function; and $NFFT$, the FFT length. k moves within a range of $0 \leq k < L$. When FFT length is greater than window length, the window function for $NFFT \leq k < L$ is 0.

CONJ, Complex window:

$$w(k) = \begin{cases} 0.5 - 0.5\cos\frac{4kC}{L}, & \text{if } 0 \leq k < L/4 \\ \sqrt{\left(1.5 - 0.5\cos 2C - \frac{4kC}{L}\right)\left(0.5 + 0.5\cos 2C - \frac{4kC}{L}\right)}, & \text{if } L/4 \leq k < 2L/4 \\ \sqrt{\left(1.5 - 0.5\cos\frac{4kC}{L} - 2C\right)\left(0.5 + 0.5\cos\frac{4kC}{L} - 2C\right)}, & \text{if } 2L/4 \leq k < 3L/4 \\ 0.5 - 0.5\cos\left(4C - \frac{4kC}{L}\right), & \text{if } 3L/4 \leq k < L \\ 0, & \text{if } NFFT \leq k < L \end{cases},$$

$$w(k) = \begin{cases} 0.5 - 0.5\cos\left(\frac{4k}{L}C\right), & \text{if} 0 \leq k < L/4 \\ \sqrt{1 - \left\{0.5 - 0.5\cos\left(\frac{2L-4k}{L}C\right)\right\}^2}, & \text{if} L/4 \leq k < 2L/4 \\ \sqrt{1 - \left\{0.5 - 0.5\cos\left(\frac{4k-2L}{L}C\right)\right\}^2}, & \text{if} 2L/4 \leq k < 3L/4 \\ 0.5 - 0.5\cos\left(\frac{4L-4k}{L}C\right), & \text{if} 3L/4 \leq k < L \\ 0, & \text{if} NFFT \leq k < L \end{cases},$$

Here, $C = 1.9979$. Figures 6.84 and 6.85 show the shape and frequency responses of the complex window function.

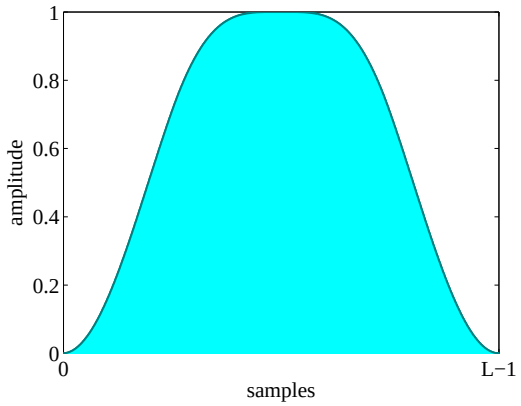


Figure 6.84: Shape of complex window function

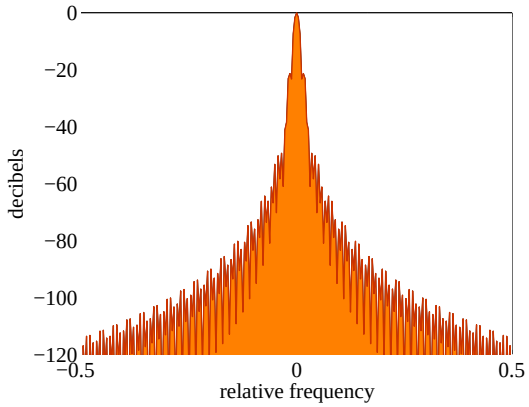


Figure 6.85: Frequency response of complex window function

The horizontal axis in Figure 6.85 indicates the mean relative sampling frequency. Generally, frequency responses of a window function are better if the peak at 0 in the horizontal axis is sharper. Components outside the center of the frequency response indicate the amount of power of other frequency components that leak to a certain frequency bin when performing Fourier transformation. The vertical axis shows the power of other frequencies components that leak into a certain frequency bin when performing Fourier transformation.

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 a circular constant. Figures 6.86 and 6.86 show the shape and frequency responses of the hamming

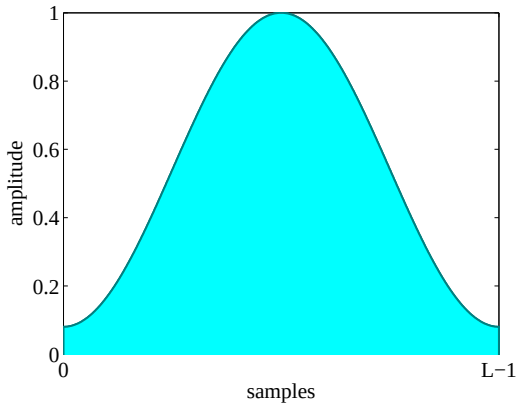


Figure 6.86: Shape of hamming window function

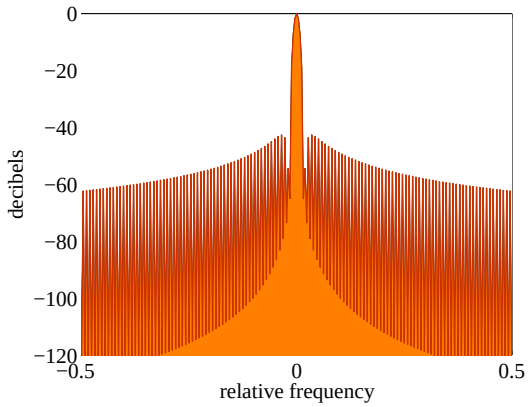


Figure 6.87: Frequency response of hamming window function

window function, respectively. **RECTANGLE, Rectangle window:**

$$w(k) = \begin{cases} 1, & \text{if } 0 \le k < L \\ 0, & \text{if } L \le k < NFFT \end{cases}$$

Figures 6.88 and 6.88 show the shape and frequency responses of the rectangular window function, respectively.

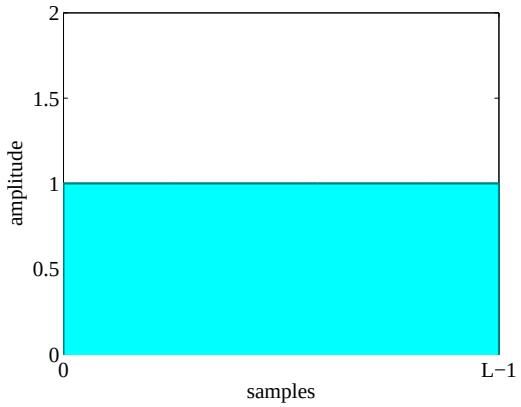


Figure 6.88: Shape of rectangle window function

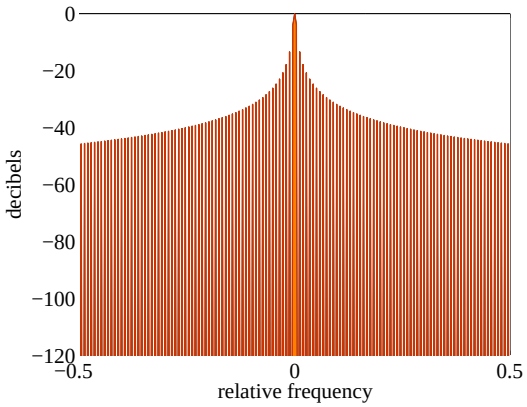


Figure 6.89: Frequency response of rectangle window function

6.7.7 MultiGain

Outline of the node

This node regulates gains of input signals.

Necessary files

No files are required.

Usage

When to use

This node is used to amplify input signals or modify them so that they cannot be clipped. For example, when speech waveform data quantified by 24 bits are used for inputs on a system built for 16 bits, the MultiGain node is used to decrease the gains by 8 bits.

Typical connection

This node is usually positioned just after AudioStreamFromMic or AudioStreamFromWave , or with ChannelSelector in between.

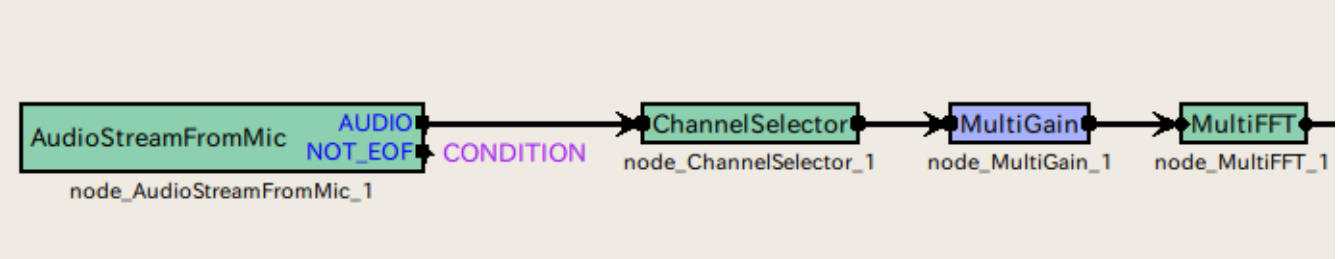


Figure 6.90: Example of a connection of MultiGain

Input-output and properties of the node

Table 6.67: Parameters 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.

Parameters

GAIN : `float` type. A gain parameter. Inputting 1.0 corresponds to outputting the inputs without change.

Details of the node

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.

$$20 \log x = -40 \tag{6.131}$$

$$x = 0.01 \tag{6.132}$$

6.7.8 PowerCalcForMap

Outline of the node

This node converts multichannel complex spectra of `Map<int, ObjectRef>` type IDs into real power/amplitude spectra.

Necessary files

No files are required.

Usage

When to use

This node is used to convert complex spectra into real power/amplitude spectra when inputs are `Map<int, ObjectRef>` type. When inputs are of `Matrix<complex<float> >` type, use the PowerCalcForMatrix node.

Typical connection

Figure 6.91 shows an example of the usage of the PowerCalcForMap node. A `Map<int, ObjectRef>` type complex spectrum obtained from the MultiFFT node is converted into `Map<int, ObjectRef>` type power spectrum and input to the MelFilterBank node.

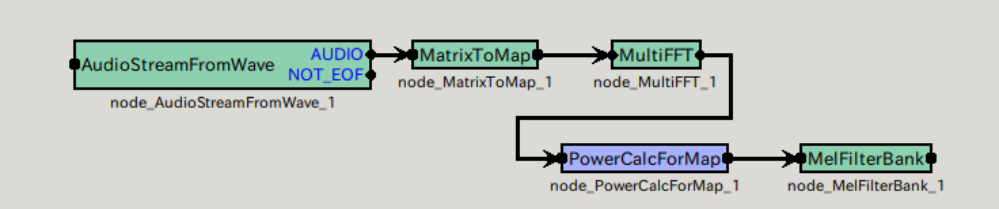


Figure 6.91: Example of a connection of PowerCalcForMap

Input-output and properties of the node

Table 6.68: Parameters of PowerCalcForMap

Parameter name	Type	Default value	Unit	Description
POWER_TYPE	`string`	POW		Selection of power/amplitude spectra

Input

INPUT : `Map<int, ObjectRef>` type. Complex matrices of `Matrix<complex<float> >` type are stored in the `ObjectRef` part.

Output

OUTPUT : `Map<int, ObjectRef>` type. Real matrices of power/absolute values are stored in the `ObjectRef` part for each element of the complex matrices of the inputs.

Parameter

POWER_TYPE : `string` type. Selection of power (POW) or amplitude (MAG) spectra for the output.

Details of the node

The real matrix $N_{i,j}$ of the output for the complex matrix $M_{i,j}$ of an input (i, j indicating the rows and columns of the index, respectively) is obtained as:

$$\begin{aligned} N_{i,j} &= M_{i,j}M_{i,j}^{*} \text{ (if POWER\_TYPE=POW),} \\ N_{i,j} &= abs(M_{i,j}) \text{ (if POWER\_TYPE=MAG),} \end{aligned}$$

Here, $M_{i,j}^{*}$ indicates the complex conjugate of $M_{i,j}$.

6.7.9 PowerCalcForMatrix

Outline of the node

This node converts `Matrix<complex<float> >` type multichannel complex spectra into the real power/amplitude spectra.

Necessary files

No files are required.

Usage

When to use

This node is used to convert complex spectra input as `Matrix<complex<float> >` type into real power/amplitude spectra. When inputs are of `Map<int, ObjectRef>` type, use the PowerCalcForMap node.

Typical connection

Figure 6.92 shows an example of the usage of a PowerCalcForMatrix node. A `Matrix<complex<float> >` type complex spectrum obtained from the MultiFFT node is converted into a `Matrix<float>` type power spectrum and input to the BGNEstimator node.

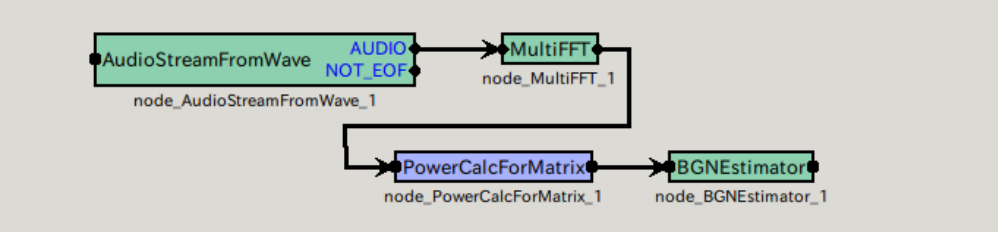


Figure 6.92: Example of a connection of PowerCalcForMatrix

Input-output and properties of the node

Table 6.69: Parameters of PowerCalcForMatrix

Parameter name	Type	Default value	Unit	Description
POWER_TYPE	`string`	POW		Selection of power/amplitude spectra

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 power/absolute values of each element of the input.

Parameter

POWER_TYPE : `string` type. Selection of power (POW) or amplitude (MAG) spectra for the output.

Details of the node

The real matrix $N_{i,j}$ of the output of the input complex matrix $M_{i,j}$ (i, j indicating the rows and columns of the index, respectively) can be obtained as follows.

$$\begin{aligned} N_{i,j} &= M_{i,j}M^*_{i,j} \text{ (if POWER\_TYPE=POW),} \\ N_{i,j} &= abs(M_{i,j}) \text{ (if POWER\_TYPE=MAG),} \end{aligned}$$

Here, $M^*_{i,j}$ indicates the complex conjugate of $M_{i,j}$.

6.7.10 SegmentAudioStreamByID

Outline of the node

This node extracts acoustic streams that use ID information and output them while adding ID information to them.

Necessary files

No files are required.

Usage

When to use

This node is useful in extracting and processing specific parts of acoustic streams, such as audio parts, not in processing all entire acoustic signals as one stream. Since extractions are performed with IDs as keys, ID information is essential for inputs. This node extracts sections in which the same ID is continuously seen as a signal and outputs these sections as `Map` data in one channel, adding ID information to it.

Usage

When to use

We presume that the input stream is a mixture of sounds from two audio signals. When comparing the signals of the original sound mixture separated by GHDSS from a temporally similar signal, the acoustic stream of 1 channel is input, as is the sound source detected by sound source localization to this node. In this case, the outputs are in a format (`Map`) identical to those of sounds separated by GHDSS and PostFilter .

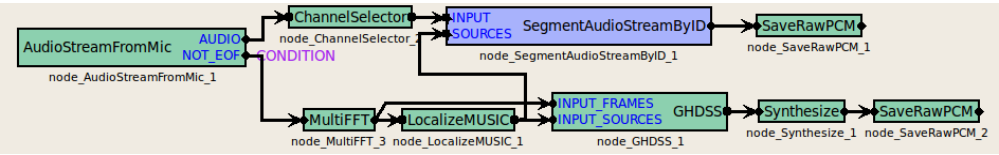


Figure 6.93: Example of a connection of SegmentAudioStreamByID

Input-output and properties of the node

Input

INPUT : `any` , `Matrix<complex<float> >` or `Matrix<float>` type.

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 always be designated.

Output

OUTPUT : `Map<int, ObjectRef>` and `ObjectRef` are slender pointers to `Vector<float>` and `Vector<complex<float> >` .

Details of the node

This node extracts acoustic streams using ID information and outputs them while adding ID information. Note that this node converts `Matrix<complex<float> >` and `Matrix<float>` into `Map<int, ObjectRef>` while adding IDs, although at present supports data from only 1 channel as input.

6.7.11 SourceSelectorByDirection

Outline of the node

This node is a filtering node that filters input source localization results specified angle range in a horizontal direction.

Necessary files

No files are required.

Usage

When to use

This node is used obtain localization results only for directions in which prior information is available (e.g. when it is known that the sound sources are only in the front). Alternatively, when the direction of noise source is known, this node can subtract noise localization results by designating all other directions.

Typical connection

Source location results such as ConstantLocalization , LoadSourceLocation and LocalizeMUSIC are mainly connected. The example of connection in Figure 6.94 is to a network to extract only those directions specified from the log files of the source localization results.

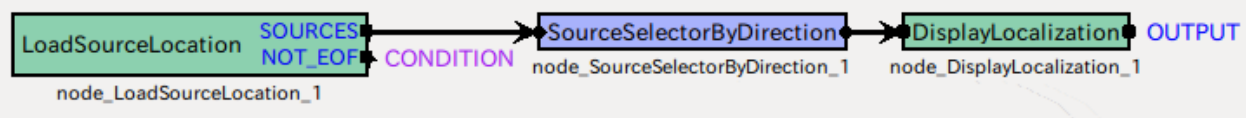


Figure 6.94: Example of a connection of SourceSelectorByDirection

Input-output and properties of the node

Input

SOURCES : `Vector<ObjectRef>` type. Source location results are input. `ObjectRef` refers to the type of `Source` data.

Output

OUTPUT : `Vector<ObjectRef>` type. Source location results after filtering. `ObjectRef` refers to the type of `Source` data.

Parameter

MIN_AZIMUTH，MAX_AZIMUTH : `float` type. The angles indicate right and left directions (azimuth angle) of the sound source to be passed.

Details of the node

In contrast to the beamformer, this node does not perform spatial filtering by microphone array signal processing. Rather SourceSelectorByDirection filters based on sound source directions of localization results.

Table 6.70: Parameters of SourceSelectorByDirection

Parameter name	Type	Default value	Unit	Description
MIN_AZIMUTH	float	-20.0	[deg]	Minimum value of a sound source direction to be passed
MAX_AZIMUTH	float	20.0	[deg]	Maximum value of a sound source direction to be passed

6.7.12 SourceSelectorByID

Outline of the node

This node is used to output only sound source separation results with IDs over a designated value among multiple sound source separation results. In particular, when setting the FIXED_NOISE property of the GHDSS node to `true`, negative IDs are given to stationary noise separation results. This node is used to filter the sounds other than these.

Necessary files

No files are required.

Usage

When to use

The sound source separation node GHDSS switches on the robot although it does not move it. When performing sound source separation under conditions in which the noise (e.g. the sound of the fan) is both stationary and known, an ID of the separation result of the noise is output as -1. Thus, when connecting SourceSelectorByID after the GHDSS node and setting the threshold value to 0, the separated sound of the stationary noise can be ignored in subsequent processing.

Typical connection

Figure 6.95 shows a example of a connection, in which the node is connected to the posterior half of GHDSS .

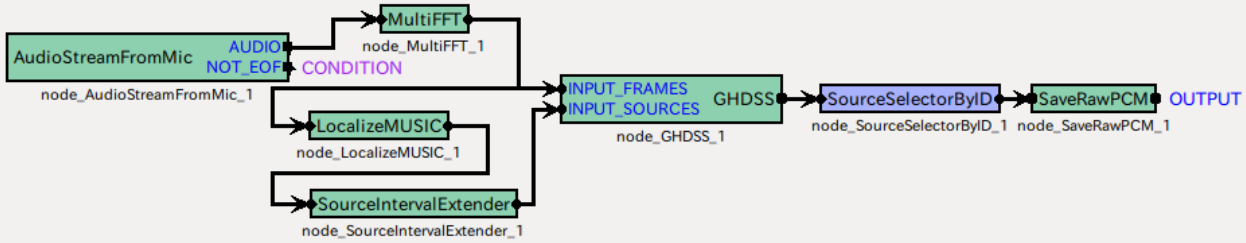


Figure 6.95: Example of connection of SourceSelectorByID

Input-output and properties of the node

Input

INPUT : `Map<int, ObjectRef>` type. Since this node is usually connected after the sound source separation node is connected, sound source IDs correspond to `int` , the key of `Map` . `ObjectRef` is of `Vector<float>` type (power spectra) or `Vector<complex<float> >` (complex spectra), indicating a separation .

Output

OUTPUT : `Map<int, ObjectRef>` type. Only data with sound source IDs greater than MIN_ID are extracted as output. The content of `Map` is same as that of INPUT.

Parameter

Table 6.71: Parameters of SourceSelectorByID

Parameter name	Type	Default value	Unit	Description
MIN_ID	int	0		Sound sources with IDs greater than this value are passed.

MIN_ID : int type. Separated sounds with sound source IDs greater than this parameter value are passed. The default value is 0. When this node is connected after GHDSS , no changes to the default value are required.

6.7.13 Synthesize

Outline of the node

This node converts signals of frequency domain into waveforms of time domain.

Necessary files

No files are required.

Usage

This node is used to convert signals of frequency domain into waveforms of time domain.

When to use

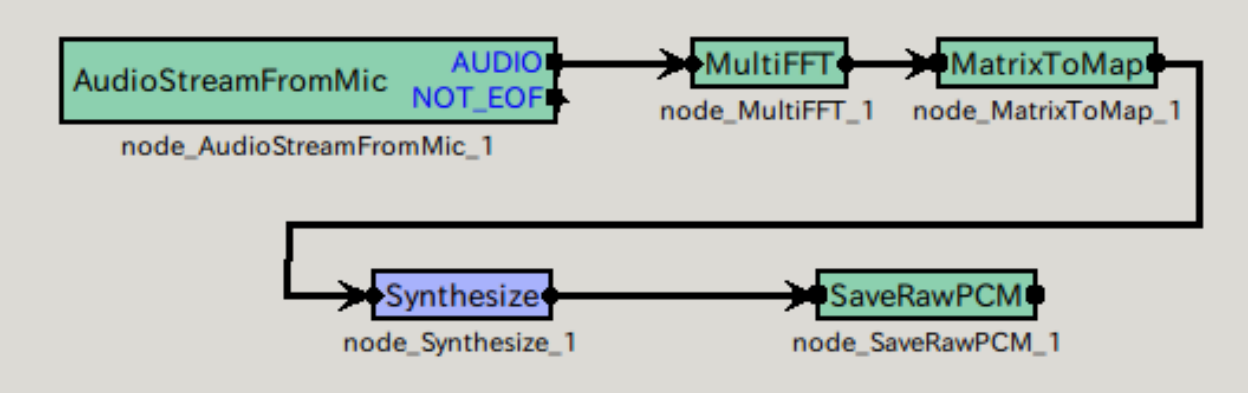


Figure 6.96: Example of a connection of Synthesize

Input-output and properties of the node

Table 6.72: Parameters of Synthesize

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	FFT length
ADVANCE	int	160	[pt]	Shift length
SAMPLING_RATE	int	16000	[Hz]	Sampling rate
MIN_FREQUENCY	int	125	[Hz]	Minimum frequency
MAX_FREQUENCY	int	7900	[Hz]	Maximum frequency
WINDOW	string	HAMMING		Window function
OUTPUT_GAIN	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; must be equal to the other node (MultiFFT).

ADVANCE Shift length; must be equal to the other node (MultiFFT).

SAMPLING_RATE Sampling rate; must be equal to the other nodes.

MIN_FREQUENCY The minimum frequency used for waveform generation

MAX_FREQUENCY The maximum frequency used for waveform generation

WINDOW Window function. Select HAMMING, RECTANGLE or CONJ

OUTPUT_GAIN Output gain

Details of the node

Inverse FFT is performed on four low bands of the input signals of the frequency domain by substituting 0 for frequency bins over $\omega_s/2 - 100$ [Hz]. A designated window is adopted for overlap-add processing, a method of reducing the influence of a window by performing inverse transformation for every frame, adding the signals for which the time domains are shifted back while shifting them. For details, see the web pages in References. Finally, the temporal waveforms are multiplied by the output gain and output. Further, subsequent frames must be read for overlap-add processing; therefore, this node delays the entire processor. The length of the delay can be calculated as:

$$delay = \begin{cases} |\text{LENGTH}/\text{ADVANCE}| - 1, & \text{if LENGTH } mod \text{ ADVANCE} \neq 0, \\ \text{LENGTH}/\text{ADVANCE}, & \text{otherwise.} \end{cases} \tag{6.133}$$

Since the default values for HARK are LENGTH = 512 and ADVANCE = 160, the delay is three frames, which resulting in a 30 [ms] delay for the entire system.

References

(1) http://en.wikipedia.org/wiki/Overlap-add

6.7.14 WhiteNoiseAdder

Outline of the node

This node adds white noise to input signals.

Necessary files

No files are required.

Usage

This node is used to add white noise to input signals, thus reducing the influence of non-linear distortions after separation. For example, since PostFilter performs nonlinear processing, it is difficult to avoid musical noise, which may greatly influence the speech recognition performance. The influence of such noise can be reduced by adding an appropriate amount of known white noise.

When to use

An example is shown in the figure.

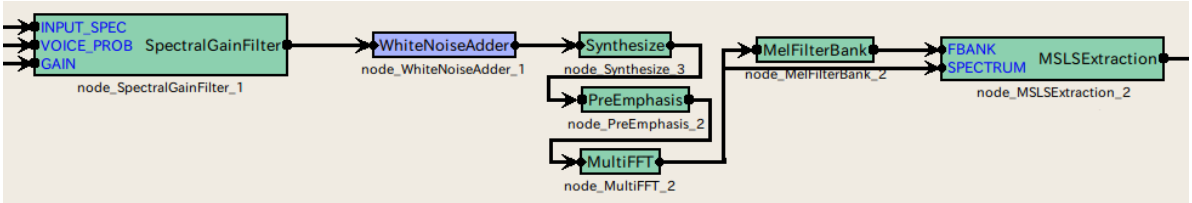


Figure 6.97: Example of connection of WhiteNoiseAdder

Input-output and properties of the node

Table 6.73: Parameters of WhiteNoiseAdder

Parameter name	Type	Default value	Unit	Description
LENGTH	int	512	[pt]	FFT length
WN_LEVEL	float	0		Additional noise level

Input

INPUT : `Map<int, ObjectRef>` type. Since `ObjectRef` is of `Vector<complex<float> >` type, signals of the frequency domain are input.

Output

OUTPUT : `Map<int, ObjectRef>` type. Since `ObjectRef` is of `Vector<complex<float> >` type, signals to which white noise has been added are output.

Parameter

LENGTH : FFT length; must be equal to the other nodes.

WN_LEVEL : Additional noise level. Designate the maximum amplitude in the time domain.

Details of the node

The following is added to each frequency bin of input signals.

$$\frac{\sqrt{\text{LENGTH}}}{2} \cdot \text{WN\_LEVEL} \cdot e^{2\pi jR} \tag{6.134}$$

R indicates a random number of $0 \leq R \leq 1$ (different for each frequency bin). $\sqrt{\text{LENGTH}}/2$ is used to revise distortions of scaling between the time and frequency domains that occur during frequency analysis by FFT.

6.8 Modules independent of FlowDesigner

6.8.1 JuliusMFT

Outline

JuliusMFT is the speech recognition module obtained by remodeling the large glossary speech recognition system Julius for 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 JuliusMFT of 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 Julius 4.0 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 JuliusMFT is performed as follows when assuming the setting file name as julius.conf for example.

```
> julius_mft -C julius.jconf
> julius_mft.exe -C julius.jconf (for Windows OS)
```

In HARK, after starting JuliusMFT , the socket connection with JuliusMFT is 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/desc_option.html and the users are recommended to refer to the website. The minimum required setting is the following seven items.

- -notypecheck
- -plugindir /usr/lib/julius_plugin
- -input mfcnet
- -gprune add_mask_to_safe
- -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 Julius though 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/julius_plugin when apt-get is installed and /usr/local/lib/julius_plugin 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 add_mask_to_safe. Note that all extensive plug in files in this path are read entirely in execution. For Windows OS, this option must be set even when the input option is not mfcnet. If mfcnet is disabled, the directory name can be arbitrary.

[1] http://www.furui.cs.titech.ac.jp/mband_julius/

- **-input mfcnet** -input itself is an option implemented in the original Julius and 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 julius_mft(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_r* (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 julius_mft(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 JuliusMFT as mentioned above. In such a case, JuliusMFT acts as a TCP/IP communications server and receives features. Moreover, SpeechRecognitionClient and SpeechRecognitionSMNClient , which are modules of HARK, work as a client to transmit acoustic features and Missing Feature Mask to JuliusMFT . The client connects to JuliusMFT for 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 6.74 only once just after the socket connection. The sound source information is expressed in a SourceInfo structure (Table 6.75) 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 6.74: 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 6.75: SourceInfo structure

Member variable name	Type	Description
source_id	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 6.76 as one

frame. It is assumed inside the JuliusMFT that the dimension number of feature vectors and mask vectors are same.

Table 6.76: 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 6.77) that indicate completion are transmitted and the socket is closed.

Table 6.77: 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, JuliusMFT functions 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 JuliusMFT is 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 Julius is 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 JuliusMFT are 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 Julius is made packaged in Ubuntu, in the case that the original Julius is 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 julius_4.1.3_plugin 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 Julius is 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 julius_4.1.3_plugin directory and execute the following command.

     ```
     > export JULIUS_SOURCE_DIR=../julius_4.1.4-hark;
     make;
     sudo make install
     ```

Designate the path of the source of julius_4.1.4-hark in JULIUS_SOURCE_DIR. Here, the example shows the case of developing the sources of Julius and plug-ins to the same directory. Now the installation is completed.

5. Confirm if there are plug in files under /usr/lib/julius_plugin properly.

   ```
   > ls /usr/lib/julius_plugin
   calcmix_beam.jpi calcmix_none.jpi mfcnet.jpi calcmix_heu.jpi calcmix_safe.jpi
   >
   ```

   If five plug in files are indicated as above, installation is completed normally.

- For the method in Windows OS, please refer to Section 3.2.

Chapter 7

Support Tools

7.1 HARKTOOL

7.1.1 Overview

HARKTOOL is a tool for generating and visualizing separation transfer function files used by GHDSS and the localization transfer function files used by LocalizeMUSIC. There are two ways of generating these files, the one is by using the GUI (see below), and the other is by using Commas alone (see 7.1.11).

The functions of HARKTOOL are as follows:

- Generating impulse response list files
- Generating TSP response list files
- Generating microphone position files
- Generating noise position files
- Generating localization transfer files
- Generating separation transfer function files
- Displaying graph of generated files

Files needed to generate a transfer file function differ, depending on whether an impulse recording file is used or TSP recording file is used, as follows:

When using an impulse response recorded file:

1. Impulse response list file (see 7.1.5 Generation of the impulse response list file)
2. Microphone position file (see 7.1.7 Generation of the microphone location information file)
3. Impulse response recording file

When using a TSP response recording file:

1. TSP response list file (see 7.1.6 Generation of the TSP response list file)
2. Microphone position file (see 7.1.7 Generation of the microphone location information file)
3. TSP response recording file

By specifying files through the window, transfer functions are generated.

Transfer function files can be generated in simulations using HARKTOOL. In this case, a transfer function is generated using only a microphone position file, and the transfer function is generated on the assumption that the microphones are actually set are microphones are arranged in free space, so echoes from the object on which microphones are actually set are ignored (e.g. reflection of the robot head).

7.1.2 Installation

If the tool's distribution/version is supported by HARK, it can be installed using apt-get. As to the registration of repository, see HARK's webpage.

```
>sudo apt-get install harktool4
```

7.1.3 Start up

In order to start up HARKTOOL, execute the following command.

- Standard:

  ```
  >harktool4
  ```

- In order to start up HARKTOOL with GUI written in English:

  ```
  >export LC_ALL=C
  >harktool4
  ```

- In order to start up HARKTOOL with GUI written in Japanese:

  ```
  >export LC_ALL="ja_JP.utf-8"
  >harktool4
  ```

After start up command is executed, the initial screen opens as shown in Fig 7.1, and after a few seconds, the working screen is displayed.

Figure 7.1: Start up screen

7.1.4 Working Screen

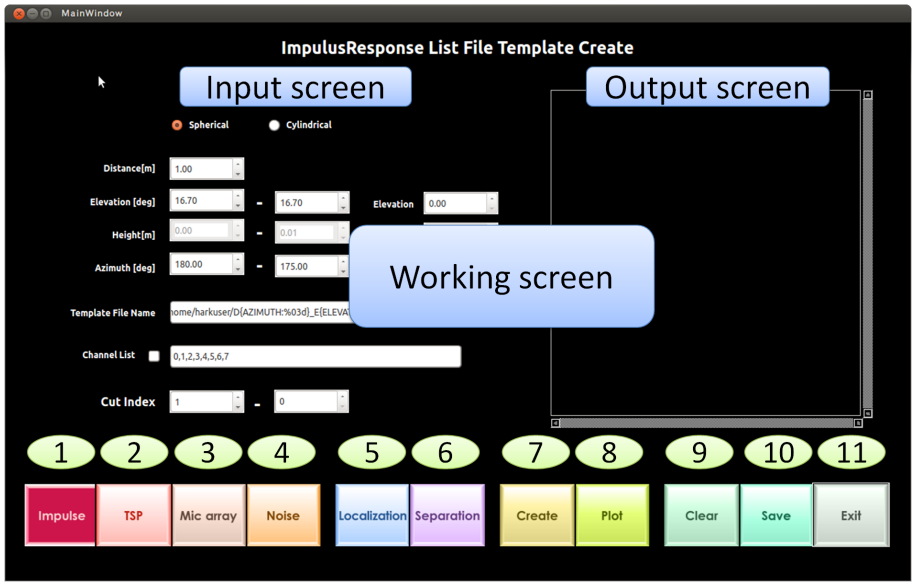


Figure 7.2: Working Screen

■ Buttons of Working Screen

1. Impulse
   This command creates an impulse response list file.
   This command also displays the graph of the generated impulse response.

2. TSP
   This command creates a TSP response list file.
   This command also displays the graph of the generated TSP response.

3. Mic array
   This command creates a microphone location information file.
   This command also displays the graph of the generated microphone location information file.

4. Noise
   This command creates a noise location information file.
   This command also displays the graph of the generated noise location information file.

5. Localization
   This command creates a localization transfer function file.

6. Separation
   This command creates a separation transfer function file.

7. Create
   This command creates a template file and a transfer function.

8. Plot
   This command displays a graph.

9. Clear
   This command resets the input parameters to the initial values.

10. Save
    This command saves the created file.

11. Exit
    This command exits the HARKTOOL4.

■ Menu Items of the FILE tab of the Menu bar

12. open
    Menu **open** opens the created file.

13. save
    Menu **save** saves the created file.

■ Menu Items of the **MENU** tab of the Menu bar

13. Impulse
    Menu **Impulse** is the same as the **Impulse** button.

14. tsp
    Menu **tsp** is the same as the **TSP** button.

15. MicArray
    Menu **MicArray** is the same as the **Mic array** button.

16. noise
    Menu **noise** is the same as the **Noise** button.

17. localization
    Menu **localization** is the same as the **Localization** button.

18. separation
    Menu **separation** is the same as the **Separation** button.

19. create
    Menu **create** is the same as the **Create** button.

20. plot
    Menu **plot** is the same as the **Plot** button.

21. clear
    Menu **clear** is the same as the **Clear** button.

22. Exit
    Menu **Exit** is the same as the **Exit** button.

7.1.5 Generation of the impulse response list file

- The impulse response file is a text file showing the correspondence between impulse response recording files and measurement locations.
- This file must be created according to the 5.3.1 Source position list information (srcinf) format.
- The method to create this file using template generation function of HARKTOOL is as follows:

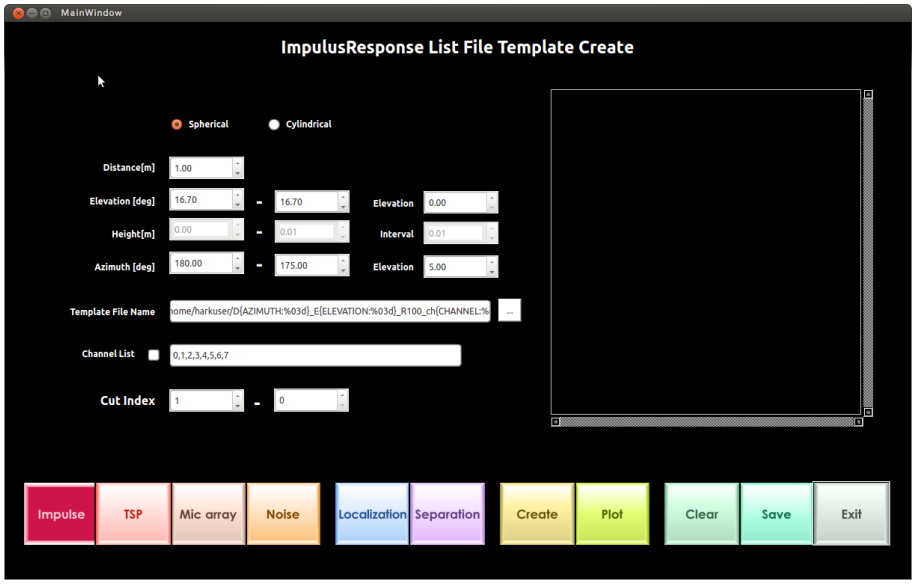


Figure 7.3: ImpulseResponse ListFile generation screen

Generation Process

1. Click the **Impulse** button on the **working screen**
2. Input the parameters into the parameter input screen on the left side.

Overview of setting items

- **Spherical, Cylindrical**
  Specify the coordinate system to use from the spherical coordinates or the cylindrical coordinates.
  If the spherical coordinate system is chosen, the elevation can be entered. If the cylindrical coordinate system is chosen, the height can be entered.
- **Distance [m]**
  The horizontal distance between this speaker and the microphone.
- **Elevation [deg]**
  The elevation angle seen from the microphone in the direction of the speaker.
  From the starting height, each elevation (whose interval is specified in the next item) is added to the template.
- **Height [m]**
  The height seen from the microphone in the direction of the speaker.
  From the starting height, each height (whose interval is specified in the next item) is added to the template.

- **Azimuth [deg]**
  Ending angle (left), starting angle (right).
  From the starting angle, in the counterclockwise direction, each angle (whose interval is specified in the next item) is added to the template.
- **Template File Name**
  The place/name of the impulse response recorded file (flt format).
  When the spherical coordinate is chosen, it contains the strings "{AZIMUTH:%03d}", "{ELEVATION:%03d}", and "{CHANNEL:%02d}" in the file name. When the cylindrical coordinate is chosen, it contains the strings "{AZIMUTH:%03d}", "{ELEVATION:%03d}", and "{CHANNEL:%02d}" in the file name. These strings are replaced to the correspond values.
- **Channel List**
  The channel number to use. If you select the channel to use, check the box and specify the number in comma-delimited format. If you use all channels, the box need not be checked.
- **Cut Index [sample]**
  The starting and ending positions of sample used for generating transfer functions.
  Default of the start position need not be changed.
  The ending position is used to ignore (delete) the reflection, for example, in the case of the refraction being louder than the direct sound.

3. Pushing the **Create** button on the bottom of the screen generates the Impulse Response List File and displays the contents of this file on the right side.

4. Pushing the bottom-right **Plot** button displays the graph according to the input parameters.

5. To save the created file, use the **Save** button on the bottom of the screen.

7.1.6 Generation of the TSP response list file

- The TSP response list file is a file showing the corresponding between TSP signal recording files and measurement locations.
- The file must be created according to the 5.3.1 Source position list information format (srcinf).
- the method to create this file using template generation function of HARKTOOL is as follows:

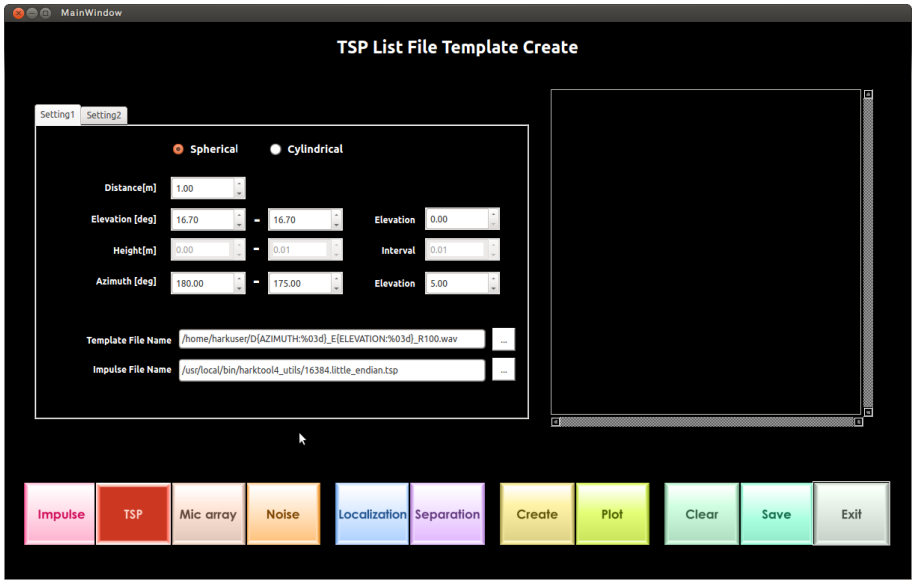


Figure 7.4: TSP ListFile generation screen (Setting1 tab)

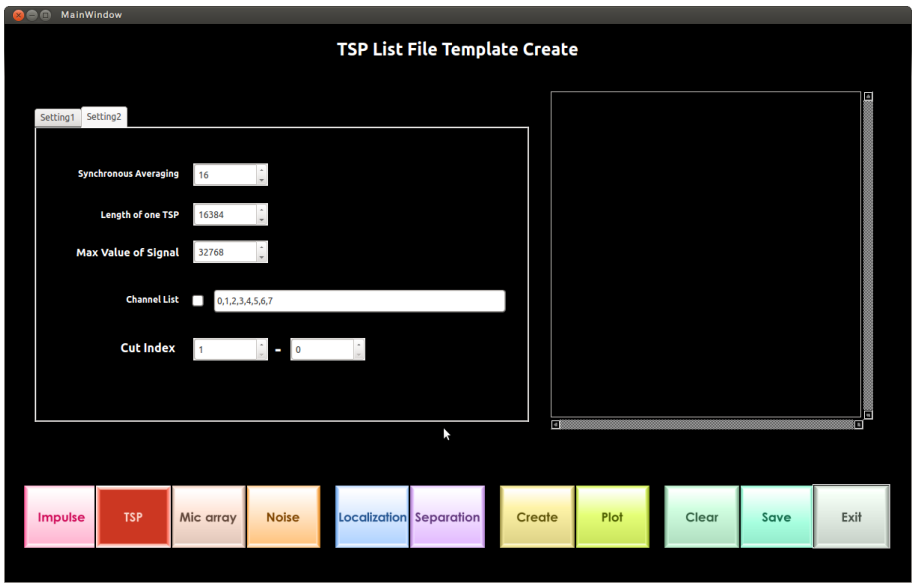


Figure 7.5: TSP ListFile generation screen (Setting2 tab)

Generation process

1. Click the **TSP** button on the **working space**.

2. Input the parameters into the parameter input screen on the left side.

Overview of setting items

■ Setting1

- **Spherical, Cylindrical**
  Specify the coordinate system to use from the spherical coordinates or the Cylindrical coordinates.
  If the spherical coordinate system is chosen, the elevation can be entered. If the cylindrical coordinate system is chosen, the height can be entered.
- **Distance [m]**
  The horizontal distance between the speaker and the microphone.
- **Elevation [deg]**
  The elevation angle seen from the microphone in the direction of the speaker.
  From the starting height, each elevation (whose interval is specified in the next item) is added to the template.
- **Height [m]**
  The height seen from the microphone in the direction of the speaker.
  From the starting height, each height (whose interval is specified in the next item) is added to the template.
- **Azimuth [deg]**
  Ending angle (left), starting angle (right).
  From the starting angle, in the counterclockwise direction, each angle (whose interval is specified in the next item) is added to the template.
- **Template File Name**
  The place/name of the TSP response recorded file (flt format).
  When the spherical coordinate is chosen, it contains the strings "{AZIMUTH:%03d}" and "{ELEVATION:%03d}" in the file name. When the cylindrical coordinate is chosen, it contains the strings "{AZIMUTH:%03d}" and "{ELEVATION:%03d}" in the file name. These strings are replaced to the correspond values.
- **Impulse File Name**
  The filename of the **TSP signal** of one cycle used for recording.

■ Setting2

- **Synchronous Averaging**
  The count of consecutive replays of TSP signal at the time of TSP signal recording
- **Length of one TSP**
  The number of samples of TSP signal used for recording. Default setting need not be changed.
- **Max Value of Signal**
  The value of the maximum amplitude. Default setting need not be changed.
- **Channel list**
  The channel number to use. If you select the channel to use, check the box and specify the number in comma-delimited format. If you use all channels, the box need not be checked.
- **Cut Index [sample]**
  The starting and ending positions of sample used for generating transfer functions.
  Default of the start position need not be changed.
  The ending position is used to ignore (delete) the reflection, for example, in the case of the refraction being louder than the direct sound.

3. Pushing the **Create** button on the bottom of the screen generates the TSP Response List File and displays the contents of the right side.

4. Pushing the bottom-right **Plot** button displays the graph according to the input parameters.
5. To save the created file use **Save** button on the bottom of the screen.

7.1.7 Generation of the microphone location information file

- The microphone location information file is a text file describing the microphone locations.
- This file must be created according to the 5.2.1 Microphone position text format.
- The microphone location information file is necessary to generate transfer function files used for the LocalizatioonMUSIC module or GHDSS module.
- The method to create this file using template generation function

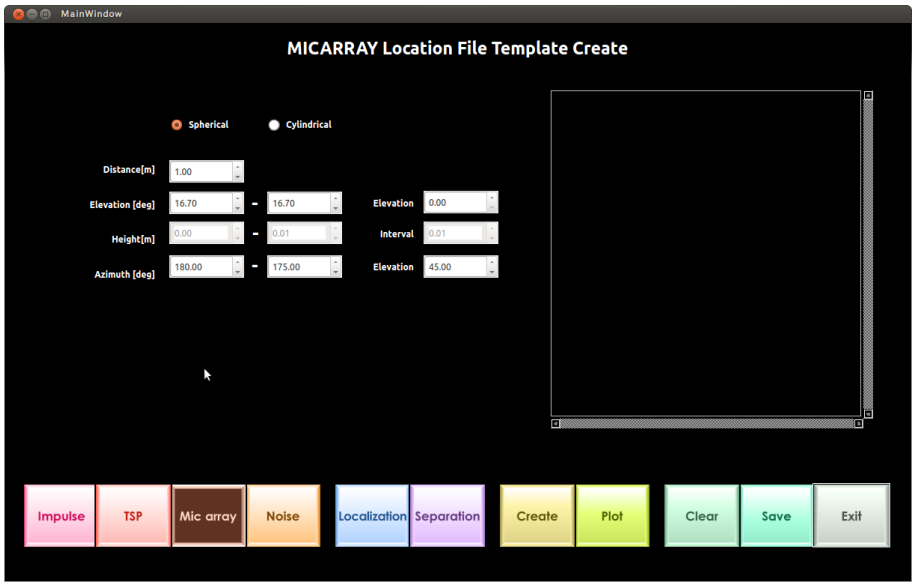


Figure 7.6: MICARY-LocationFile Edit

Generation Process

1. Click the **MICARRAY** button on the **working screen**.
2. Input the parameters into the parameter input screen on the left side.

   Overview of setting items

   - **Spherical, Cylindrical**
     Specify the coordinate system to use from the spherical coordinates or the cylindrical coordinates. If the spherical coordinate system is chosen, the elevation can be entered. If the cylindrical coordinate system is chosen, the height can be entered.
   - **Distance [m]**
     The horizontal distance between the speaker and the microphone.
   - **Elevation [deg]**
     Input the start, end, and interval elevation angles of the speaker and microphone.
   - **Height [m]**
     Input the start, end, and interval heights seen from the microphone in the direction of the speaker.
   - **Azimuth [deg]**
     Ending angle (left), starting angle(right). Input the starting, ending, and interval angles. These values are added to the template at every interval (specified in the next item), from the starting angle, in the counterclockwise direction.

3. Pushing the **Create** button on the bottom of the screen generates the microphone location information file and displays the contents of the right side.

4. Pushing the bottom-right **Plot** button displays the graph according to the input parameters.

5. To save the created file, use **Save** button on the bottom of the screen.

7.1.8 Generation of the noise location information file

- The noise location information file is a text file describing the noise location
- This file is generated using the template preparation function of HARKTOOL as follows:

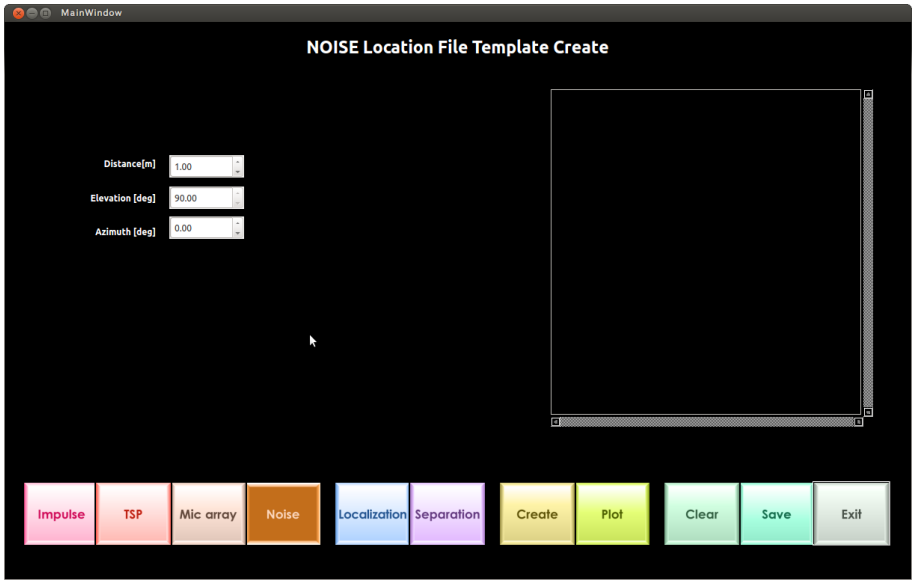


Figure 7.7: noise-LocationFile Edit

Generation Process

1. Click the **noise** button on the working screen.
2. Input the parameters into the parameter input screen on the left side.

   Overview of setting items

   - **Distance [m]**
     The horizontal distances of the speaker and microphone.
   - **Elevation [deg]**
     The elevation angle seen from the microphone in the direction of the speaker.
   - **Azimuth [deg]**
     The azimuth angle seen from the microphone in the direction of the speaker.
3. Pushing the **Create** button on the bottom of the screen generates the noise location information file and displays the contents of this file on the right side.
4. Pushing the bottom-right **Plot** button displays the graph according to the input parameters.
5. To save the created file, use **Save** button on the bottom of the screen.

7.1.9 Generation of the localization transfer function file

- The localization transfer function file is a configuration file used in the LocalizeMUSIC module.
- This file is generated from the ImpulseResponse-ListFile or the TSP-ListFile, and from the MICARY-LocationFile.

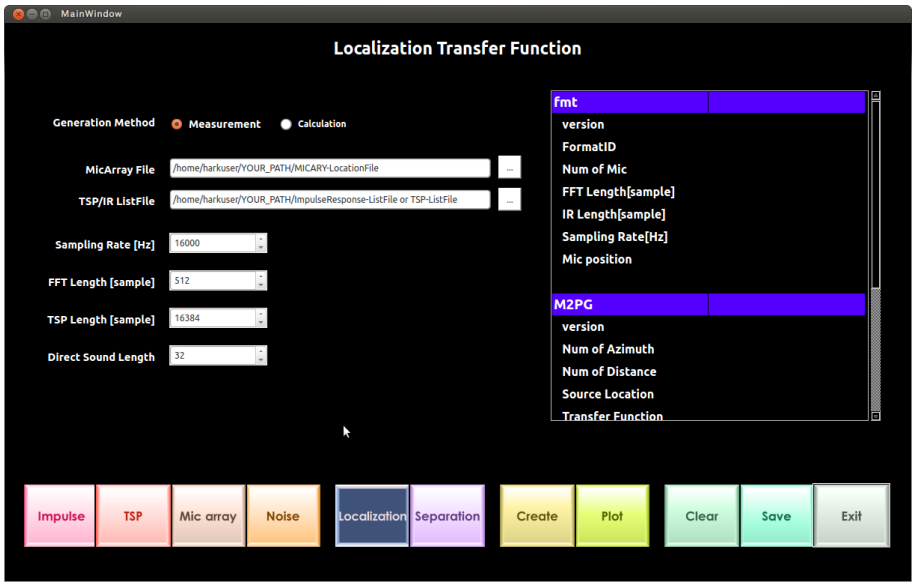


Figure 7.8: Generation of the localization transfer function file

1. Click the **Localization** button on the working screen.
2. Input the parameters into the parameter input screen on the left side.

   Overview of setting items

   - **Creation method**
     Measurement-based method or geometry calculation method can be selected.
     If actually measured, measurement-based method should be selected.
     If geometry calculation method is selected, the transfer function is generated in simulations.
   - **MicArray File**
     Microphone location information file.
     Specify the file generated in section 7.1.7.
   - **TSP/IR ListFile**
     Specify the TSP response list file (see 7.1.6) or the Impulse response list file (see 7.1.5).
   - **Sampling Rate [Hz]**
     Sampling frequency of the transfer function.
   - **FFT Length [sample]**
     The number of bins in the discrete frequency expression of the transfer function.
   - **TSP Length [sample]**
     Specify the length of one recorded TSP signal, or the impulse response length.
   - **Direct Sound Length [sample]**
     The number of samples used in generating the transfer function.a
3. Pushing the **Create** button on the bottom of the screen generates the localization transfer function file and displays the contents of this file on the right side.

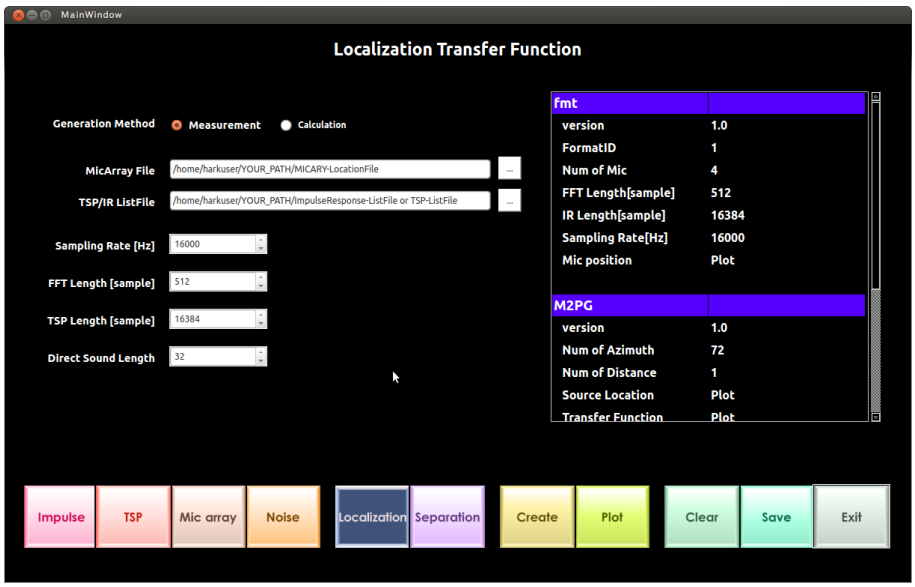


Figure 7.9: Localization Transfer Function Display

Output Screen

Graphic representation can be displayed by double-clicking the **plot** on the output screen.

4. Graphic representation of the **Mic Position** Double-click the **Plot** next to the **Mic Position** in the frame on the right side.

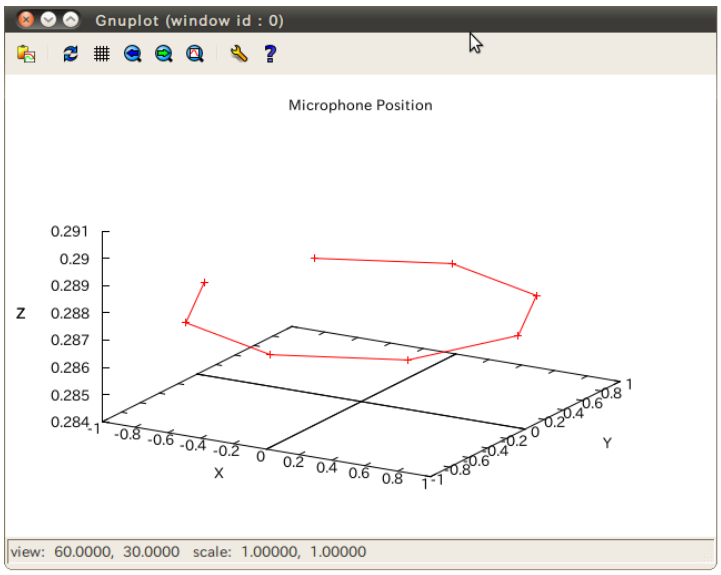


Figure 7.10: Graphic representation of Mic Position

5. Graphic representation of the **Source Location**
   Double-click the **Plot** next to the **Source Location** in the frame on the right side.

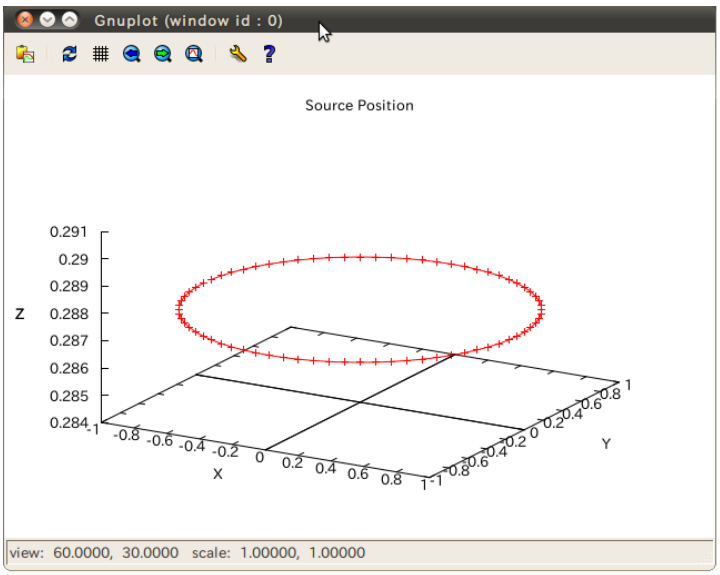


Figure 7.11: Graphical representation of Source Location

6. Graphic representation of the **Transfer Function**
   If you double-click the **Plot** next to the **Transfer Function** in the frame on the right side, graph representation setting screen (see Figure 7.12) appears.
   Set the values using **Overview of setting items** below as a guide. Clicking the bottom-right **PLOT** button displays the graph.

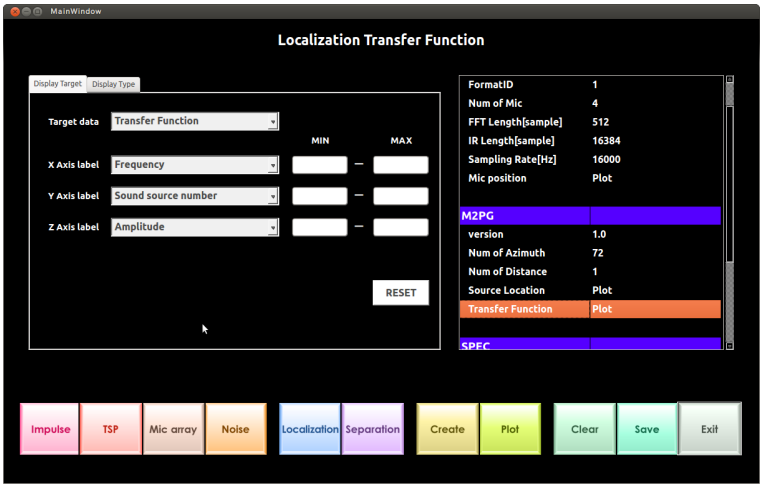


Figure 7.12: Transfer Function setting screen (Display Target tab)

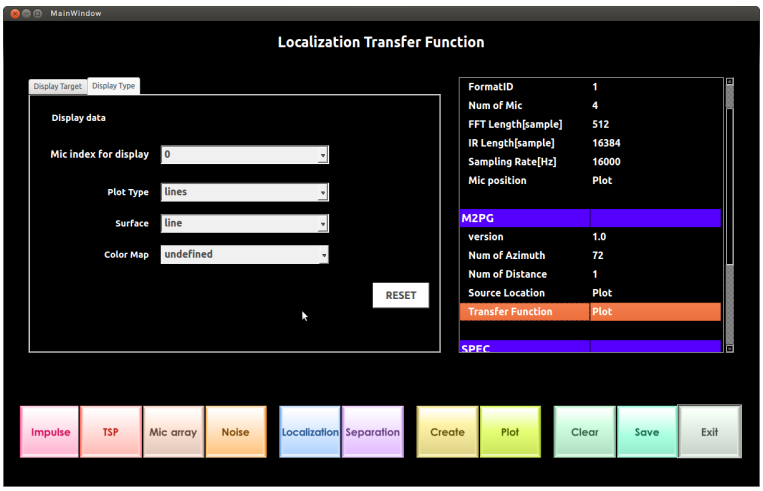


Figure 7.13: Transfer Function setting screen (Display Type tab)

Overview of setting items

■ Display Target tab

- Target data
  The type of graph to be displayed, choose one of the following three types:
  - Transfer function
  - Inverse Fourier Transform (Inverse FFT of Transfer function)
- X-axis label
  Choose the X-axis label from among the following labels.
  - Frequency (When the target data is ”Transfer function”)
  - Time (When the target data is ”Inverse Fourier Transform”)
  - Sound Source Number
  - Num of Mic

  Display range of X-axis can be specified by Min and Max.
  If every value is to be displayed, these items need not be specified.
  Min : Minimum value to be displayed on X-axis
  Max : Maximum value to be displayed on X-axis
- Y-axis label
  Choose the Y-axis label from among the following labels.
  - Frequency (When the target data is ”Transfer function”)
  - Time (When the target data is ”Inverse Fourier Transform”)
  - Sound Source Number
  - Num of Mic

  Display range of Y-axis can be specified by Min and Max.
  If every value is to be displayed, these item need not be specified.
  Min : Minimum value to be displayed on Y-axis
  Max : Maximum value to be displayed on Y-axis
- Z-axis label
  Choose the Z-axis label from among the following labels.
  - Amplitude
  - dB
  - Phase

Display range of Z-axis can be specified by Min and Max.
If every value is to be displayed, these item need not be specified.
Min : Minimum value to be displayed on Z-axis
Max : Maximum value to be displayed on Z-axis

■ Display Type tab

- Mic(or SRC or FREQ) index for display
  Select the number of microphone or source or frequency to be displayed.

- Plot Parameter
  Select the style of the graph to be displayed.

  – Plot type (undefined, dots, points, lines, lines_points)
  – Surface (undefined, line, mesh)
  – Color Map (undefined, pm3d, pm3d map, pm3d at b)

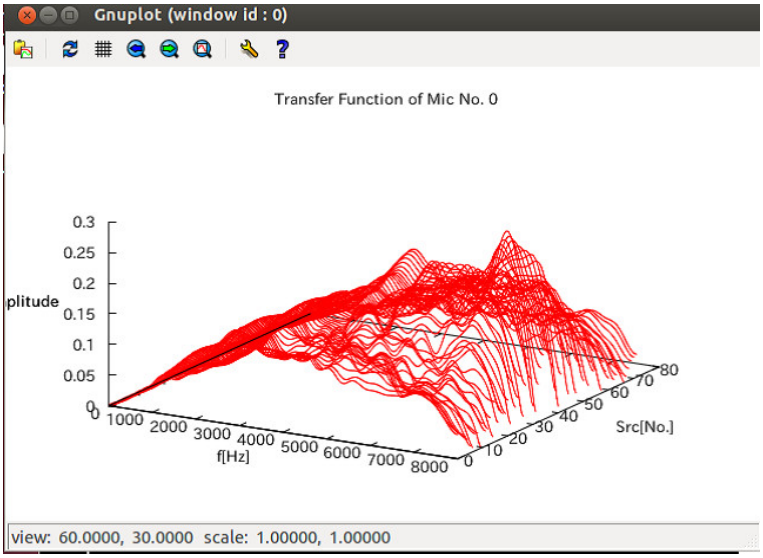


Figure 7.14: Transfer Function Graphic representation

7. To save the created file, use **Save** button on the bottom of screen.

7.1.10 Generation of the separation transfer function file

- The separation transfer function file is a file used in the GHDSS module.
- The separation transfer function file is generated from ImpulseResponse-ListFile or TSP-ListFile, and from MICARY-LocationFile.

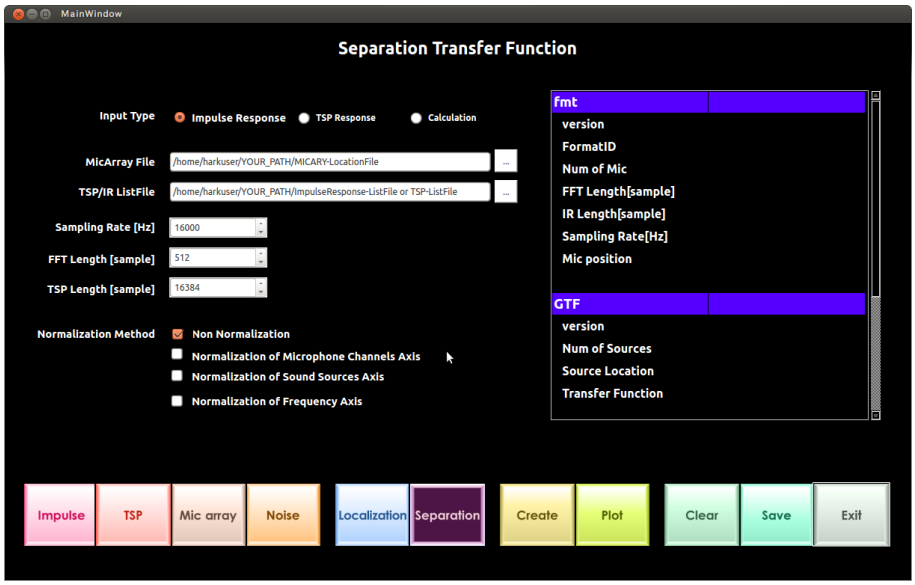


Figure 7.15: Generation of the separation transfer function file

1. Click the **Separation** button on the working screen.
2. Input the parameters into the parameter input screen on the left side.

   Overview of setting items

   - **Input Type**
     Select impulse response or TSP response or geometry calculation.
     Impulse Response: Select this option if you obtain the transfer function from the recorded impulse response.
     TSP Response: Select this option if you obtain the transfer function from the recorded TSP signal.
     Calculation: Select this option if you obtain the transfer function in simulations.

   - **MicArray File**
     Filename of microphone location setting file.
     Specify the file generated in section 7.1.7.
   - **TSP/IR ListFile**
     Specify the TSP response list file (see 7.1.6) or the Impulse response list file(see 7.1.5).
   - **Sampling Rate [Hz]**
     Sampling frequency of the transfer function.
   - **FFT Length [sample]**
     The number of bins in the discrete frequency expression of the transfer function.
   - **TSP Length [sample]**
     Specify the length of one recorded TSP signal, or the impulse response length.
   - **Normalization Method**
     Specify one of the following normalization axis:

- Non Normalization
- Normalization of Microphone Channel Axis
- Normalization of Sound Source Axis
- Normalization of Frequency Axis

3. Pushing the **Create** button on the bottom of the screen generates the separation transfer function file and displays the contents of the file on the right side.

4. Check by graphic representation
   Graphic representation can be displayed for the item with the **PLOT** on the output screen.
   If you double-click the **Plot**, graphic representation appears and you can check the results.
   (See Figure 7.16)

5. Graphic representation of **Mic Position**
   Double-click the **Plot** next to the **Mic position**.

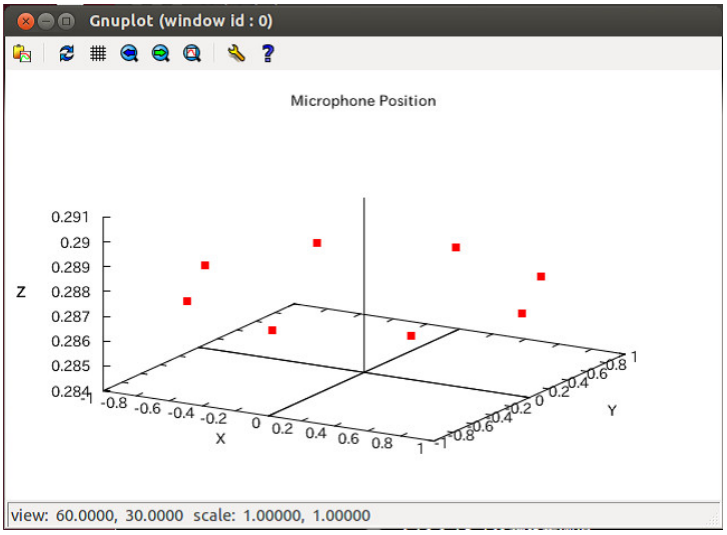


Figure 7.16: Graphic representation of the Mic Position

6. Graphic representation of the **Source Location**
   Double-click the **Plot** next to the **Source Location**.

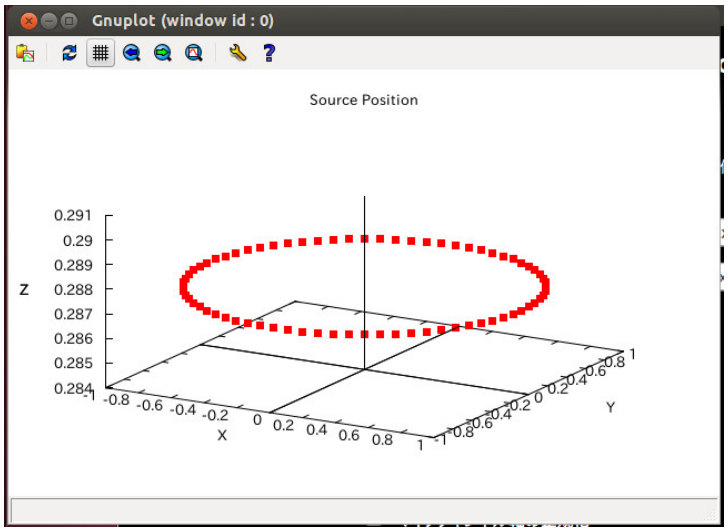


Figure 7.17: Graphical representation of the Source Location

7. Graphic representation of the **Transfer function**
   If you double-click the **Plot** next to the **Transfer function**, graph representation setting screen (see Figure 7.18) appears.
   Set the values using **Overview of setting items** below as a guide. Clicking the bottom-right **Plot** button displays the graph.
   (e.g. see Figure 7.20)

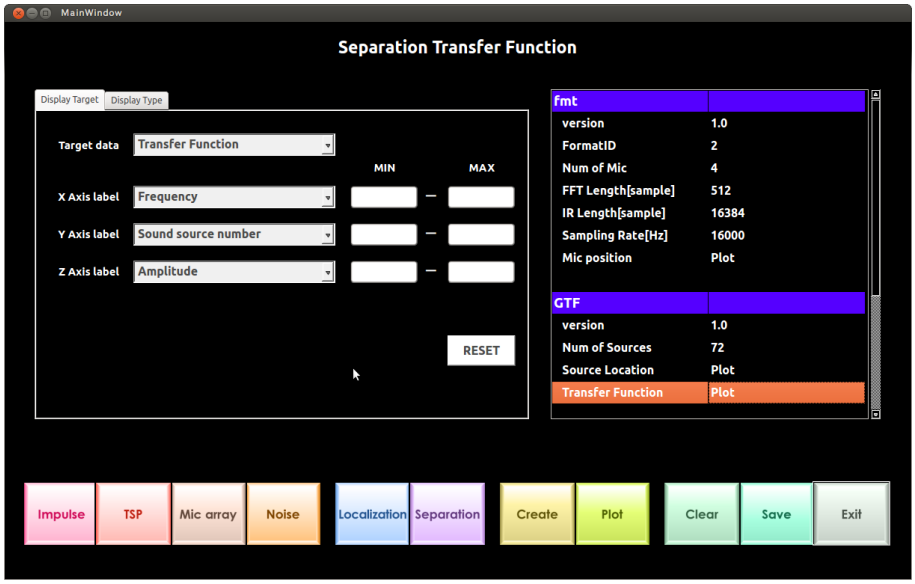


Figure 7.18: Transfer Function setting screen (Display Target tab)

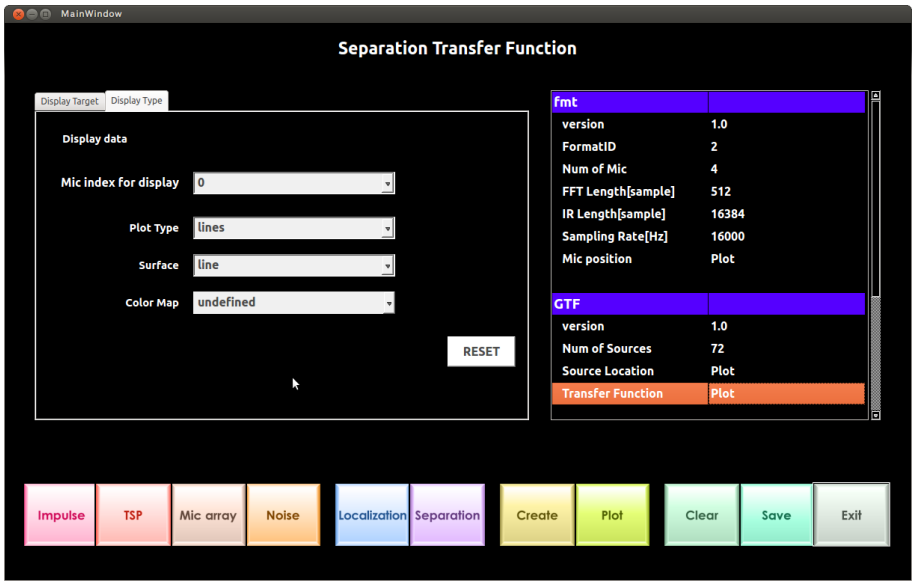


Figure 7.19: Transfer Function setting screen (Display Type tab)

Overview of setting items

- Target data
  The type of graph to be displayed, choose one of the following three types:
  - Transfer function

  - Inverse Fourier Transform (Inverse FFT of Transfer function)

- X-axis label
  Choose the X-axis label from among the following labels.
  - Frequency (When the target data is ”Transfer function”)
  - Time (When the target data is ”Inverse Fourier Transform”)
  - Sound Source Number
  - Num of Mic

  Display range of X-axis can be specified by Min and Max.
  If every value is to be displayed, these items need not be specified.
  Min : Minimum value to be displayed on X-axis
  Max : Maximum value to be displayed on X-axis

- Y-axis label
  Choose the Y-axis label from among the following labels.
  - Frequency (When the target data is ”Transfer function”)
  - Time (When the target data is ”Inverse Fourier Transform”)
  - Sound Source Number
  - Num of Mic

  Display range of Y-axis can be specified by Min and Max.
  If every value is to be displayed, these item need not be specified.
  Min : Minimum value to be displayed on Y-axis
  Max : Maximum value to be displayed on Y-axis

- Z-axis label
  Choose the Z-axis label from among the following labels.
  - Amplitude
  - dB
  - Phase

  Display range of Z-axis can be specified by Min and Max.
  If every value is to be displayed, these item need not be specified.
  Min : Minimum value to be displayed on Z-axis
  Max : Maximum value to be displayed on Z-axis

■ Display Type tab

- Mic(or SRC or FREQ) index for display
  Select the number of microphone or source or frequency to be displayed.

- Plot Parameter
  Select the style of the graph to be displayed.

  – Plot type (undefined, dots, points, lines, lines_points)
  – Surface (undefined, line, mesh)
  – Color Map (undefined, pm3d, pm3d map, pm3d at b)

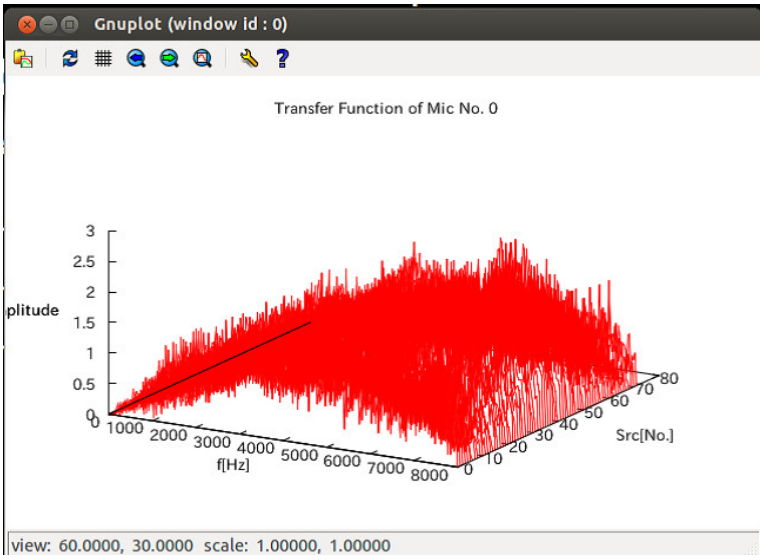


Figure 7.20: Graphical representation of the Transfer Function

8. To save the created file, use **Save** button on the bottom of screen.

7.1.11 Command Format

The way of executing harktool4 using only the commands is as follows:
Following features are available.

- Generating impulse response list files
- Generating TSP response list files
- Generating microphone position files
- Generating noise position files
- Generating localization transfer function files
- Generating separation transfer function files

1. Installation
   If the tool's distribution is supported by HARK, it can be installed using apt-get

   ```
   sudo apt-get install harktool4_cui
   ```

2. Usage

Enter the following command from the terminal.

```
 harktool4_cui [options] input-files

Example: Generating the template file of the ImpulseResponseListFile

Range of azimuth: -180〜180 at 1 interval
Height: 16.7 deg
Radius from the center to the sound source: 1m


harktool4_cui
--mode template          # template mode
--output-file ImpulseResponseListFile.xml # output file format
--output-format xml      # XML format
--output-type ir         # ImpulseResponse type
--azifrom -180.000000   # start of the azimuth range: -180 deg
--aziint 5.000000       # interval of the azimuth: 1 deg
--numazi 72             # number of azimuth  72
--elvfrom 16.700000     # start of the height range 16.7 deg
--elvint 0.000000       # no interval
--numelv 1              # number of intervals: 1
--radius 1.000000       # radius: 1m
--cut-start 1           # start index: from 1
--cut-end 0             # end index: to the last
--filepath /home/username/D{AZIMUTH:%03d}_E{ELEVATION:%03d}_R100_ch{CHANNEL:%02d}.flt
```

[Common Option:]
–help help output
-m [–mode] arg mode [template / tf]
-o [–output-file] arg output (default is STDOUT)
-f [–output-format] arg output format [xml / binary]
-t [–output-type] arg output type
(1)template mode [mic / noise / ir / tsp]
(2)transfer functino mode [m2pg / gtf]

[Option:]
Values in parentheses are initial values.

- –nummic arg (=8) Number of microphones
- –azifrom arg (=0) Start of the azimuth range
- –aziint arg (=5) Interval of the azimuth
- –numazi arg (=72) Number of the azimuths
- –elvfrom arg (=0) Start of the height range (vertical angle of the sound source; -90 to 90 deg)
- –elvint arg (=5) Interval of height
- –numelv arg (=1) Number of intervals
- –heightfrom arg (=0) Start of the height range (m)

- –heightint arg (=0.01) Interval of the height (m)
- –numheight arg (=1) Number of heights
- –radius arg (=1) Radius from the center to the microphone or sound source
- –synchronous-average arg (=16) Number of synchronous averages
- –original-impulse-file arg(=original-impulse-file.tsp) Original impulse file
- –tsp-offset arg (=16384) Impulse file in the sample
- –tsp-length arg (=16384) Length of a TSP in the sample
- –signal-max arg (=32768) Maximum width of the TSP signal
- –cut-start arg (=1) Cut-start index of the floating point vector
- –cut-end arg (=0) Cut-end index of the floating point vector
- –mic-channels arg Comma-separated list of the microphone channels.
- –filepath arg Template string of file path
- –nfft arg (=512) Number of FFT points
- –normalize-src Normalization of the sound source axis
- –normalize-mic Normalization of the microphone channel axis
- –normalize-freq Normalization of the frequency axis
- –geometry-calculus arg (=0) Geometry calculation flag (0: Geometry calculation is not appliedm 1: applied)
- –sampling-freq arg (=16000) Sampling frequency
- –direct-length arg (=32) Length of the direct sound

7.2 wios

7.2.1 What is this tool?

wios is a tool for recording, playing, and synchronized recording and playing. wios supports three kinds of devices: (1) ALSA devices, (2) the RASP series, and (3) TD-BD-16ADUSB. See 8 for details about the devices.

wios can be used to measure impulse responses, because it supports the synchronized recording and playing. The impulse responses are required for sound source localization and separation nodes.

7.2.2 Installation

If you are using HARK-supported distributions / devices, you can install wios using :`apt-get install wios`. See the HARK web page for details about repository registration.

7.2.3 How to use

You can run wios with no options to see the complete help.0 wios accepts three types of options: mode, device, and general, which can be freely combined. For example, if you want to record 2 seconds from an ALSA device at a sampling rate 44.1kHz and store it into voice.wav, the command is:

```
wios -x 0 -f 44100 -t 2 -o voice.wav
```

See the recipe "Recording multichannel sound" of the HARK cookbook for other examples.

The descriptions of the options for each category are:

Mode option

The three modes for wios are.

Record : The option is `-r` or `--rec`. You can specify the wave file name to record name using `-o`. The default file name is `da.wav`

Play : The option is `-p` or `--play`. You can specify the wave file name to play using `-i`. The default file name is `ad.wav`

Synchronized playing and recording : The option is `-s` or `--sync`. You can specify the wave file name to play using `-i` and the file to record using `-o`. wios will then play the specified file and simultaneously record to another specified file.

Device options

You can specify a device using two options, `-x` and `-d`.

ALSA : The option is `-x 0`. You can specify the device name using `-d`. The default device name is `plughw:0,0`.

TDBD : The option is `-x 1`. You can specify the device file using `-d`. The default path to the device file is `/dev/sinichusb0`.

RASP : The option is `-x 2`. You can specify the IP address using `-d`. The default IP address is `192.168.33.24`.

In addition to thees options, you can specify other device dependent options such as gains; see help for wios for details.

General options

- `-t`: Recording/playing duration.
- `-e`: Quantization bit rate. The default is 16 bits.
- `-f`: Sampling frequency. The default is 16000 Hz.
- `-c`: Number of channels. The default is 1ch.

Chapter 8

HARK-compatible Multi-Channel AD Devices and Their Installation

Users can perform microphone array processing by a connecting multichannel A/D device to HARK. Users who need to perform microphone array processing sets requiring A/D devices should refer to this chapter. Three types of A/D devices are currently supported by HARK.

1. The RASP series (System In Frontier, Inc.)
2. ALSA-based devices (e.g. the RME Hammerfall DSP Multiface series),
3. TD-BD-16ADUSB (Tokyo Electron device).

The methods for installing and setting up these devices are described here.

To use TD-BD-16ADUSB, the third party version of HARK was used with HARK 1.0. However, this third party version was integrated into HARK by default. Therefore, if you have installed HARK 1.1, you can use TD-BD-16ADUSB without the third party version.

8.1 System In Frontier, Inc. the RASP series

This section describes the usage of RASP series (System In Frontier, Inc), multichannel A/D converters supported by HARK. Especially, we introduce two of them: Wireless RASP and RASP 24. See the recording samples section in HARK cookbook for sample files.

8.1.1 Wireless RASP

Figure 8.1: Wireless RASP

Figure 8.1 shows an overview of the Wireless RASP. Radio RASP is a multichannel A/D, D/A conversion device that performs only 16ch AD and 2ch D/A conversions. It transmits and receives commands via TCP/IP and transmits and receives settings and data of A/D and D/A conversion processing from a host PC. It is a good filter of sampling frequencies and its analog input parts can be changed easily using software.

Network Configuration of Wireless RASP

Radio RASP consists of a single box. Since a host PC is connected to a wireless LAN, a router corresponding to a wireless LAN is required. For Wireless RASP, it is necessary to set an IP address beforehand and to register an SSID of a radio router. The IP address set at the time of shipping can be used, although it is essential to register an SSID. Complete the setting by referring to the attached manual.

Installing and setting up software for Wireless RASP

To operate Wireless RASP, it is necessary to install a library and a sample program in a host PC. Copy files from the bundled CD to Wireless RASP and install them. Refer to the attached materials for details. It is necessary to at least prepare an environment in which the setting of the analog filter for Wireless RASP can be initialized to use HARK. Installation of the command `ws_fpaa_config` is required. `ws_fpaa_config` is generated by installing a library and compiling a sample program. Since the set value of `ws_fpaa_config` is deleted when turning off Wireless RASP, make sure to execute it right after turning it off.

Recording test in HARK with Wireless RASP

First, check you have all equipments for recording shown in Table 8.1.

For a speech recording test with HARK, use the network file test.n shown in Figure 8.2. The connection route is from MatrixToMap to SaveRawPCM , from AudioStreamFromMic via ChannelSelector . Table 8.2 shows the setting for each property.

```
./test.n
```

Table 8.1: List of devices 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

By setting the above, the speech of the analog input 1 of Wireless RASP is recorded for three seconds. Speech of PCM 16-bit quantization and 16kHz sampling is saved in the little endian RAW format.

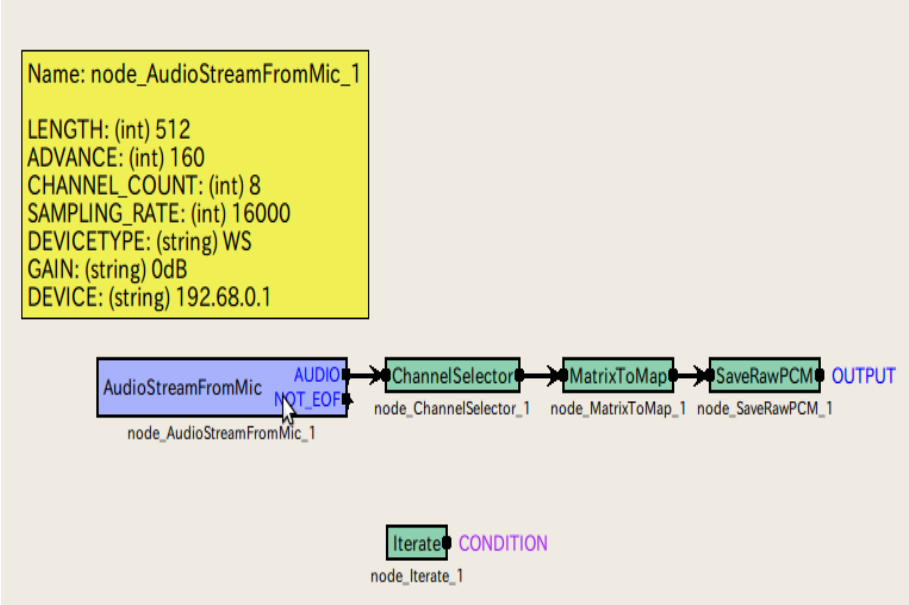


Figure 8.2: Network (test.n)

Table 8.2: Properties of a speech recording network

Module name	Property name	Type of value	Set value
AudioStreamFromMic	LENGTH	int	512
	ADVANCE	int	160
	CHANNEL_COUNT	int	16
	SAMPLING_RATE	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	MAX_ITER	int	300

8.1.2 RASP-24

Figure 8.3 shows the picture of RASP-24. Similarly to Wireless RASP, RASP-24 is a A/D converter that records 8 or 16 channel microphones simultaneously in various sampling rates. In addition, RASP-24 supports 2ch D/A converting (playing a wave file). Since the D/A can be synchronized with A/D converting, you can simultaneously play a wave file and record them. The difference from Wireless RASP is that RASP-24 can record sound at 24 bit (Wireless RASP records at 16 bit). Since RASP-24 has larger quantization bits, it can record with less quantization error.

You can connect your computer to RASP-24 via TCP/IP. RASP-24 supports wired and wireless recording: if you use a LAN port embedded on RASP-24, you can record via wired network. if you add a wireless LAN device on the USB port embedded on RASP-24, you can record via wireless network. See the product manual for details.

Basically, the usage is the same as Wireless RASP: (1) network configuration, and (2) recording using AudioStreamFromMic . The difference from Wireless RASP is that RASP-24 do not require firmware initialization after power on. Therefore, you do not need to run `ws_fpaa_config` at the beginning. See the chapter of samples in HARK Cookbook for sample network files.

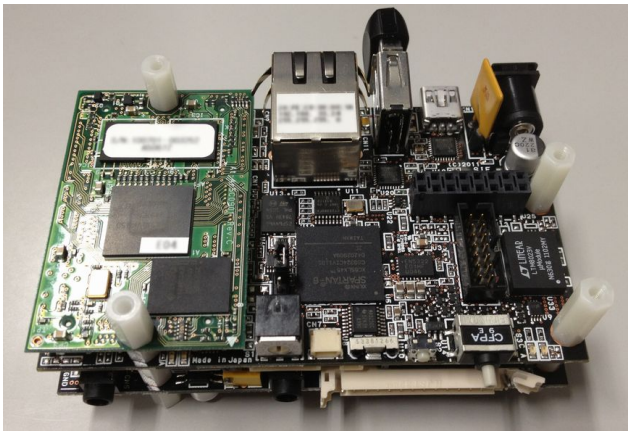


Figure 8.3: RASP-24

8.2 RME Hammerfall DSP series Multiface AE

One of the multichannel A/D devices checked by the HARK development team is the Hammerfall DSP Multiface series of RME (Multiface). This chapter describes the procedure for using Multiface in HARK. Multiface is an A/D, D/A conversion device that performs only 10ch AD and 10ch D/A conversions. Of the 10ch, 8 are analog and 2 are SPDIF. Multiface is an ALSA compliant device, a standard interface in Linux. This chapter introduces the method of setting up Multiface 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, similar to Multiface. Although ALSA compliant devices except for Multiface are out of the scope of this chapter, this chapter should be helpful in using other A/D, D/A devices that accept ALSA. The methods for installing Multiface and for recording speech from microphones connected to Multiface on HARK are introduced here. Description and operation checks are performed in Ubuntu 9.10 and 10.04, respectively.

8.2.1 Connection of Multiface to a PC

Each 24bit/96kHz multiple-channel A/D device made by RME consists of three devices.

- **RME HDSP I/O box**: The RME HDSP Multiface Avena unit,
- **RME HDSP interface**: Either an RME HDSP PCI card or an RME HDSP CardBus card
  Operation of these cards has not been confirmed, and are therefore outside the scope of our support.
- **Preamplifier for microphone**: RME OctaMic-II RME OctaMic and Yamaha HA -8 have been used, although both models have been discontinued.

However, a preamplifier for microphones is not necessarily needed. If the recording level is good, it will not be necessary to connect the device to a preamplifier. Connect these hardware devices to a PC by 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, to install firmware and software for Multiface and to set their parameters. Users are strongly recommended to install these devices as described by their manufacturers. Although compiling and installing from the source can be performed, it may cause conflicts with other software. This may result in the failure of recording/replaying during actual operations even if compiling and installing seemed to be completed successfully. Note that conflicts with pulseAudio often occur. Detailed descriptions strongly depend on individual system environments and they are therefore omitted here. Rather, this chapter describes the method of installing a general-purpose driver for ALSA devices, followed by installation work to enable the hdsp command. This is required for the initialization of Multiface followed by settings of the hdsp command. Finally, an example of the recording network with HARK is described.

Installation of a general-purpose driver for ALSA devices

General-purpose drivers (alsa-lib, alsa-driver) for ALSA devices are often already installed. Input the following bold-faced parts to confirm that they have been installed. > indicates a 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 during actual operations. In this test environment, use version 1.0.24.

```
> cat /proc/asound/version
 Advanced Linux Sound Architecture Drive Version 1. 0. 24.
```

If they are not installed, they will be installed automatically by the following operation. Therefore proceed to the next operation in the chapter.

Installation to enable command required for the initialization of Multiface

Two packages are required.

- alsa-firmware-loaders
- alsa-tools-gui

To install these packages, use either of the items below.

- Synaptic package manager
- apt-get

Installing these packages enable three commands required for setting Multiface.

- hdsploader (Package : alsa-firmware-loaders)
- hdspconf (Package : alsa-tools-gui)
- hdspmixer (Package : alsa-tools-gui)

An example of an installation with apt-get is shown below.

```
> sudo apt-get update
> sudo apt-get install alsa-firmware-loaders
> sudo apt-get install alsa-tools-gui
```

Finally, download the source of alsa-firmware from the alsa website to acquire multiface_firmware_rev11.bin. (http://www.alsa-project.org/main/index.php/Main_Page). The latest available version, dated December 13, 2011, is version 1.0.24. Download alsa-firmware-1.0.24.tar.bz2 and save the file in the "location" "download" of the system menu. It will be easier to operate if copied to "location" "home". Copy the downloaded file alsa-firmware-1.0.24.tar.bz2 to an appropriate directory. Develop a file in bunzip2. Select "application" "accessories" "terminal". If the downloaded file is copied to "location" "home", execute the following commands in the opened window. After copying the file to another directory, move to that directory using the cd command and then start compilations. Concrete procedures include.

```
> bunzip2 -d alsa-firmware-1.0.24.tar.bz2
> tar vfx alsa-firmware-1.0.24.tar
    alsa-firmware-1.0.24/
    alsa-firmware-1.0.24/multisound/
          (skipped)
    config. status: creating aica/Makefile
    config. status: executing depfiles commands
> make
> sudo mkdir -p /lib/firmware/hdsploader
> sudo cp hdsploader/multiface_firmware_rev11.bin /lib/firmware/hdsploader/
> sudo cp hdsploader/multiface_firmware_rev11.dat /lib/firmware/hdsploader/
```

The hdsp command is installed during the above operation, enabling the setting and operation of Multiface.

Setting hdsp commands

The necessary three hdsp commands are hdsploader, hdspconf and hdspmixer.

- **hdsploader**
  hdsploader is a command to upload the firmware program (multiface_firmware_rev11.dat) to initialize the 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 should be ignored, since it is not a problem here. This error message is displayed when executing more than twice on the same system. This error message is always indicated during this operation in a system 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,with the sampling frequency set in this window. For other items, refer to the documents for Multiface. Figure 8.4 shows an example of a setting in this window. The settable sampling rates are 32kHz, 44.1kHz, 48kHz, 64kHz, 88.2kHz and 96kHz. In this example, 32kHz was selected.

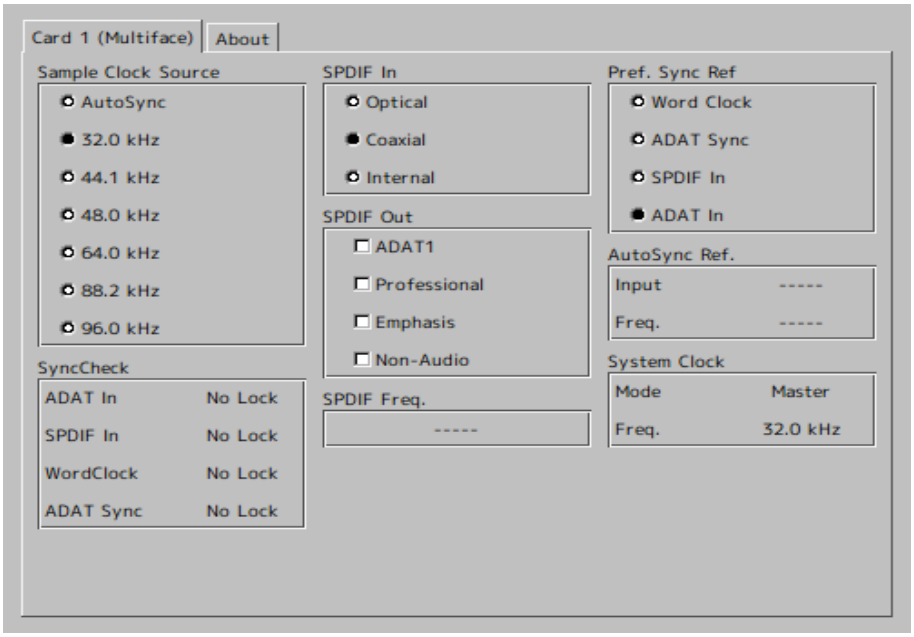


Figure 8.4: Parameter setting window

- **hdspmixer**
  GUI of the mixer is displayed when executing hdspmixer. Input and output levels can be controlled and confirmed. Figure 8.5 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 the devices to be used. For the sound recording test

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

for HARK, change only the property of each module with the network file test.n, similar to that shown in Figure 8.2. Table 8.4 shows the setting of each property.

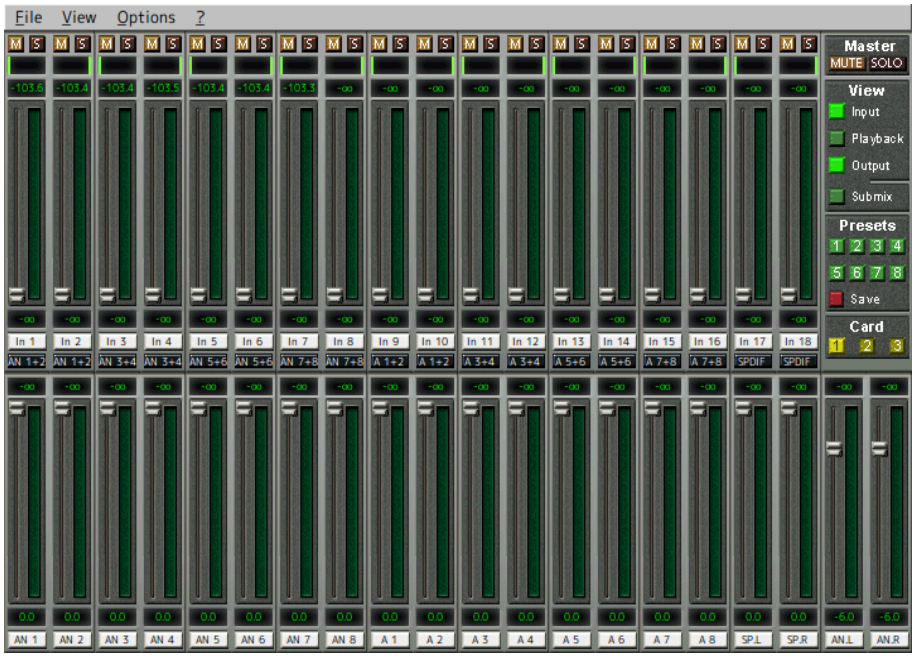


Figure 8.5: Parameter setting window

Figure 8.6: RME Hammerfall DSP Multiface (front)

Figure 8.7: RME Hammerfall DSP Multiface (Rear)

```
> ./test.n
```

Entering the above, the speech of the analog input 1 of Multiface is recorded for three seconds. The speech of PCM 16-bit quantization and 32kHz sampling are saved in the little endian RAW format.

Figure 8.8: PCMCIA CardBus Interface for Hammerfall DSP System

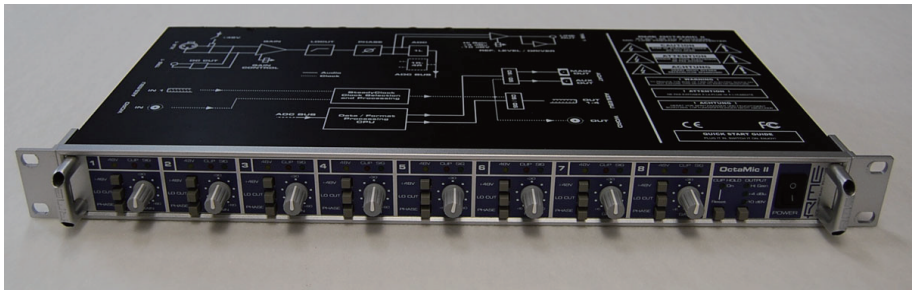


Figure 8.9: RME OctaMic (Front)

Figure 8.10: RME OctaMic (Rear)

Table 8.4: Properties of the sound recording network

Module name	Property name	Type of value	Set value
AudioStreamFromMic	LENGTH	int	1024
	ADVANCE	int	320
	CHANNEL_COUNT	int	8
	SAMPLING_RATE	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	MAX_ITER	int	300

8.3 Tokyo Electron Device TD-BD-16ADUSB

Another multichannel A/D device checked by the HARK development team is TD-BD-16ADUSB, one of the Inrevium series made by Tokyo Electron Devices (16ADUSB). This chapter describes the setting procedure for the use of 16ADUSB in HARK. The 16ADUSB is an A/D, D/A conversion device that solely performs 16ch AD 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. To connect to microphones, this device can be plugged into power supply, allowing condenser microphones for a plug in power supply to be connected without modifications. The methods of installing the 16ADUSB and of recording sound the microphones connected to the 16ADUSB on HARK are described. Description and operation checks are performed in Ubuntu 9.10.

8.3.1 Connection of 16ADUSB to a PC

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 the hardware to a PC by referring to their attached manuals.

8.3.2 Installation and set-up of software for 16ADUSB

It is necessary to install a kernel mode driver in a host PC to operate the 16ADUSB. Please check whether installation is okay or not by using the sample program included with the 16ADUSB.

8.3.3 Sound recording test of the 16ADUSB in HARK

Prior to sound recording, confirm the devices to be used (Table 8.5) are operable. 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 the same as that shown in Figure 8.2. Table 8.6 shows the settings for each property.

```
> ./test.n
```

By entering the above, speech of the analog input 1 of 16ADUSB is recorded for three seconds. The speech of the PCM 16-bit quantization and 16 kHz sampling are saved in the little endian RAW format.

Table 8.6: Property of sound recording network

Module name	Property name	Type of value	Set value
AudioStreamFromMic	LENGTH	int	512
	ADVANCE	int	160
	CHANNEL_COUNT	int	16
	SAMPLING_RATE	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	MAX_ITER	int	300