

HARK Document
Version 3.2.0. (Revision: 9448)

HARK development team

[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 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<https://www.hark.jp/>

We have continuously released and updated HARK under the above design concepts. Basically, users' feedback which was sent to HARK Forum ² plays a great role for improvements, and bug-fix.

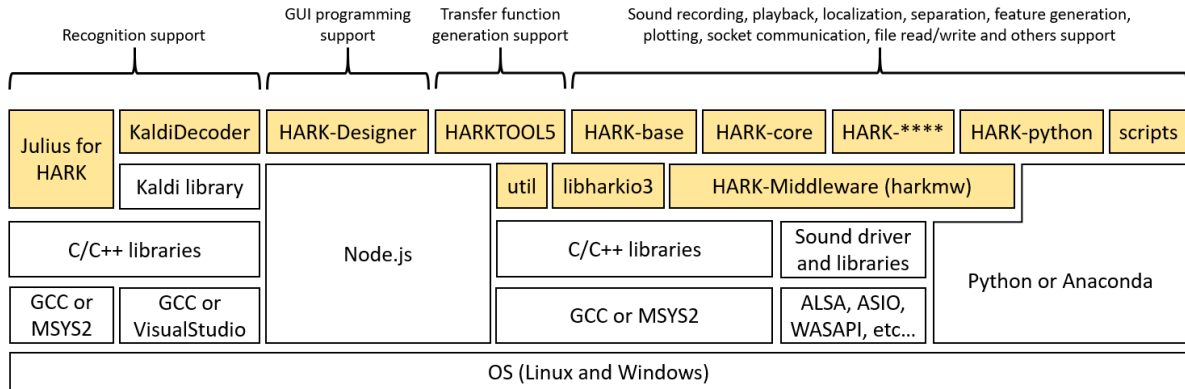


Figure 1.1: Relation between the robot audition software HARK and the middleware FlowDesigner and OS

HARK uses HARK middleware as middleware except for the speech recognition part (Julius, Kaldi) and support tools. (Up to ver 2.5, We were used FlowDesigner [?].)

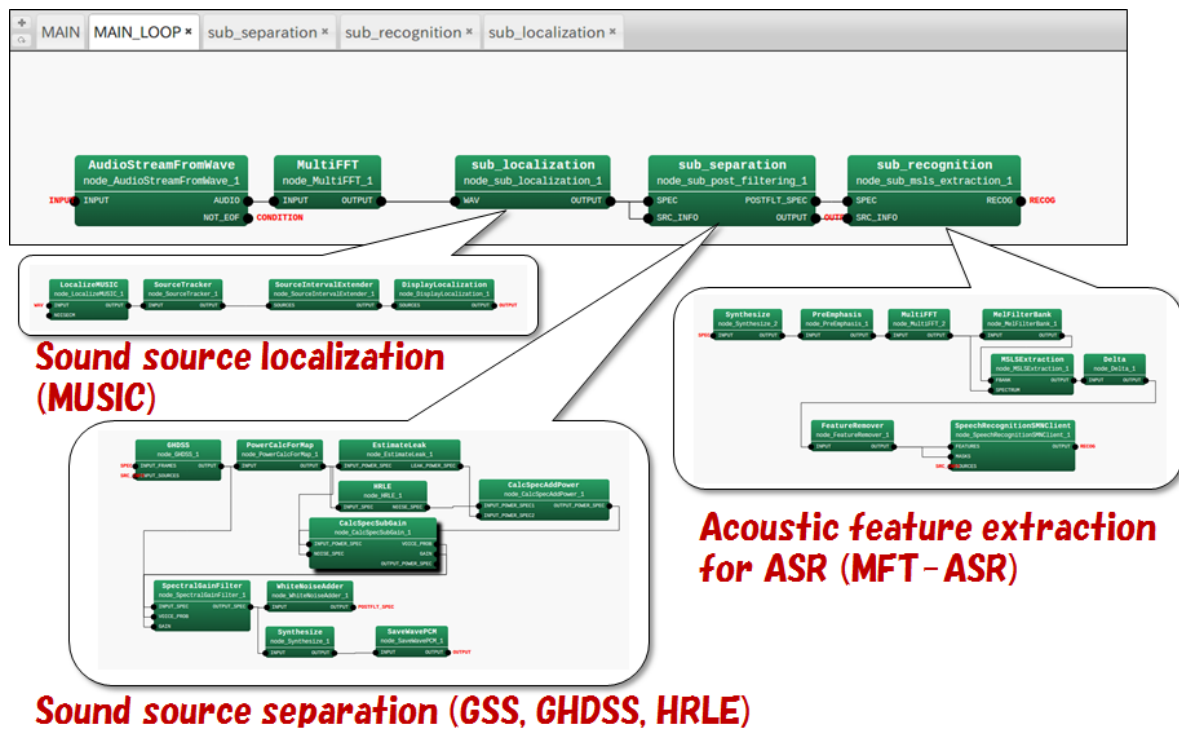


Figure 1.2: Module construction for typical robot audition with HARK

Middleware HARK Designer

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

²<https://wp.hark.jp/forums/>

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 developed data flow oriented middleware HARK middleware. HARK middleware is an improved version of FlowDesigner [?] that was used as middleware until then, but the contents are reimplemented from scratch. As with conventional FlowDesigner, HARK middleware is faster and lighter in processing than general-purpose module integration framework such as ROS and CORBA.

FlowDesigner is a framework that realizes high-speed, lightweight module integration on the premise of use within a single computer^{3 4}, but HARK middleware also considered distributed processing using multiple computers It is implementation. In FlowDesigner, the implementation language is C++ only, whereas HARK middleware is implemented with a combination of C++ and Python. Compared to FlowDesigner, the overhead is increasing, but it is implemented in consideration of minimizing the overhead. Also, regarding module implementation, it is fully compatible with FlowDesigner, and these classes are realized as C++ classes that inherit common superclasses (if you use HARK-Python, you can write modules with python It is also possible). For this reason, interfaces 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 HARK middleware.

Figure 1.3: Attribute setting screen of GHDSS

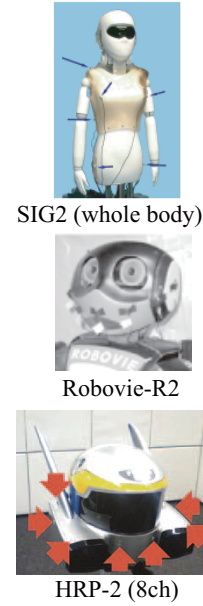


Figure 1.4: Three types of ear for robot (microphone layout).

?? shows a network of HARK middleware 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 (Figure?? shows an example of the attribute setting screen of GHDSS). Table ?? shows HARK modules and external tools that are currently provided for HARK. In the

³Connecting across computers can be realized by creating a module for network connection like a connection with speech recognition in HARK.

⁴The original version of FlowDesigner and function-improved version of FlowDesigner 0.9.0 are available at <http://flowdesigner.sourceforge.net/> and <https://www.hark.jp/>, respectively.

following section, outlines of each module are described with the design strategy.

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 [?] 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) [?], 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,

Table 1.1: Nodes and Tools provided by HARK 3.2.0

Function	Category name	Module name	Description
Voice input output	AudioIO	AudioStreamFromMic AudioStreamFromWave SaveRawPCM SaveWavePCM SaveWavePCM2 HarkDataStreamSender	Acquire sound from microphone Acquire sound from file Save sound in file Save sound in wav-formatted file Save sound in wav-formatted file Socket-based data communication
Sound source Localization / tracking	Localization	ConstantLocalization DisplayLocalization LocalizeMUSIC LoadSourceLocation NormalizeMUSIC SaveSourceLocation SourceIntervalExtender SourceTracker SourceTrackerPF CSP LocalizeBFDS CMLoad CMSave CMChannelSelector CMMakerFromFFT CMMakerFromFFTwithFlag CMDivideEachElement CMMultiplyEachElement CMConjEachElement CMInverseMatrix CMMultiplyMatrix CMIdentityMatrix	Output constant localized value Display localization result Localize sound source Load localization information from file Normalize the MUSIC spectrum from LocalizeMUSIC Save source location information in file Extend forward the tracking result Source tracking Source tracking Estimate sound source direction by CSP Estimate sound source direction BFDS Load a Correlation Matrix (CM) file Save a CM file Channel selection for CM Create a CM Create a CM Division of each element of CM Multiplication of each element of CM Conjugate of CM Inverse of CM Multiplication of CM Output identity CM
Sound source separation	Separation	BGNEstimator BeamForming CalcSpecSubGain CalcSpecAddPower EstimateLeak GHDSS ML MSNR MVDR HRLE PostFilter SemiBlindICA SpectralGainFilter	Estimate background noise Sound source separation Subtract noise spectrum subtraction and estimate optimum g Add power spectrum Estimate inter-channel leak noise Separate sound source by GHDSS Separate sound source by ML Separate sound source by MSNR Separate sound source by MVDR Estimate noise spectrum Perform post-filtering after sound source separation Separate sound source by ICA with prior information Estimate voice spectrum
Feature extract	FeatureExtraction	Delta FeatureRemover MelFilterBank MFCCExtraction MSLSExtraction PreEmphasis SaveFeatures SaveHTKFeatures SpectralMeanNormalization SpectralMeanNormalizationIncremental	Calculate Δ term Remove term Perform mel-scale filter bank processing Extract MFCC Extract MSLS Perform pre-emphasis Save features Save features in the HTK form Normalize spectrum mean Normalize spectrum mean
Missing Feature Mask	MFM	DeltaMask DeltaPowerMask MFMMGeneration	Calculate Δ mask term Calculate Δ power mask term Generate MFM
Communication with ASR	ASRIF	SpeechRecognitionClient SpeechRecognitionSMNClient	Send feature to ASR Same as above, with the feature SMN
Others	MISC	ChannelSelector CombineSource DataLogger HarkParamsDynReconf LoadMapFrames LoadMatrixFrames LoadVectorFrames MapIDOffset MapMatrixValueOverwrite MapOperator MapOperatorD MapOperatorD2	Select channel Combine the localization result Generate log output of data Dynamic parameter tuning via network Load data saved by SaveMapFrames Load data saved by SaveMatrixFrames Load data saved by SaveVectorFrames Shift the key of Map<int, ObjectRef> Overwrite Matrix<ObjectRef> of Map<int, ObjectRef> Mathematical operations on the ObjectRef Select channel

2. Combination of ManyEars Geometric Source Separation (GSS) and PostFilter [?], which was supported as an external module with HARK 0.1.7,
3. Combination of GSS and PostFilter as an original design [?] employed for in 1.0.0 HARK prerelease,
4. Combination of GHDSS and HRLE employed for HARK 1.0.0 [?, ?].

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

The spectral distortion caused by various factors such as sound mixture or separation is beyond those that are assumed in the conventional speech recognition community. In order to deal with it, it is necessary to connect more closely the sound source separation and speech recognition. In HARK, it is dealt with the speech recognition based on the missing feature theory (MFT-ASR) [?]. The concept of MFT-ASR is shown in Figure ?? . 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 ??(a)). In MTF-ASR, influences of the distortion are ignored by masking the distorted points with Missing Feature Mask (MFM) (Figure ??(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 [?]. 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

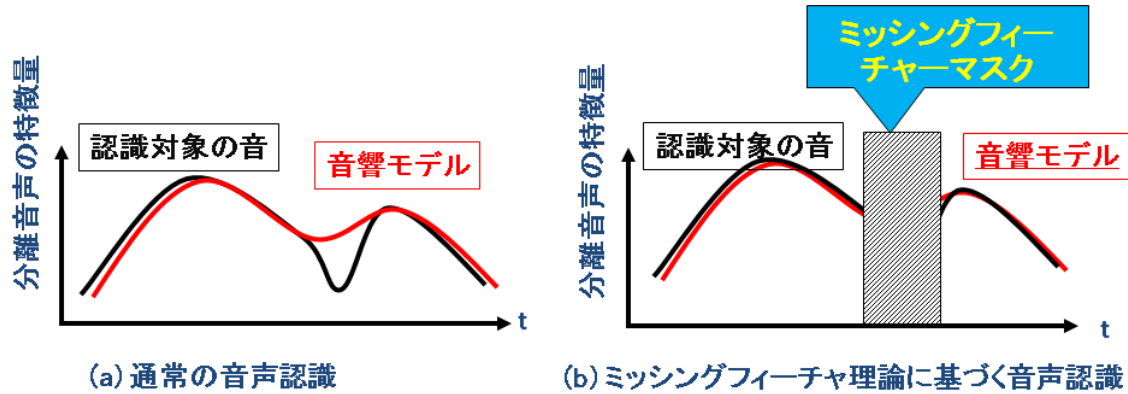


Figure 1.5: Schematic diagram of speech recognition using Missing Feature Theory

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) [?] 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 [?]. 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 [?] 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 [?]. 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 [?]. 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 [?, ?].

1.3.1 Three-speaker simultaneous speech recognition



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

The three-speaker simultaneous speech recognition system returns speech recognition results for each speaker through 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 ???. 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 through 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.⁵ 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.

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 [?]. 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

⁵Demonstration is available at <http://winnie.kuis.kyoto-u.ac.jp/SIG/>

a draw and tells the result. When the game is not finished, the robot orders the players to play the game again. The details of this system are described in the paper of ICRA-2008 [?]. Please read it if you are interested in it.

1.3.3 CASA 3D Visualizer

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

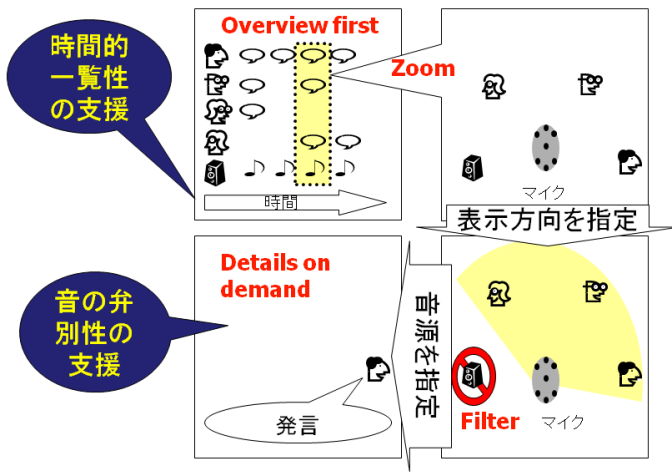


Figure 1.7: CASA 3D Visualizer: CASA 3D Visualizer: Visual Information-Seeking Matra Visualization of outputs from HARK in accordance with “Overview first, zoom and filter, then details on demand”

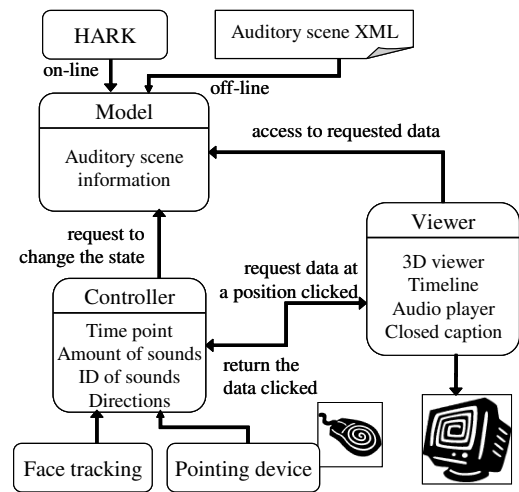


Figure 1.8: Implementation with the MVC (Model-View-Control) model of CASA 3D Visualizer

HARK, and designed and implemented a three-dimensional sound environment visualization system that supports sound environment comprehension [?, ?]. For GUI, the principle of information visualization that Schneiderman proposed (overview first, zoom and filter, then details on demand (Figure ??) 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 ??). Information to be provided from HARK is converted into AuditoryScene XML first. Next, the 3D visualization system displays the AuditoryScene XML. Figure ?? shows display screen. Scaling and rotation can be performed in the three-dimensional space information display. At the time of playing a sound, a

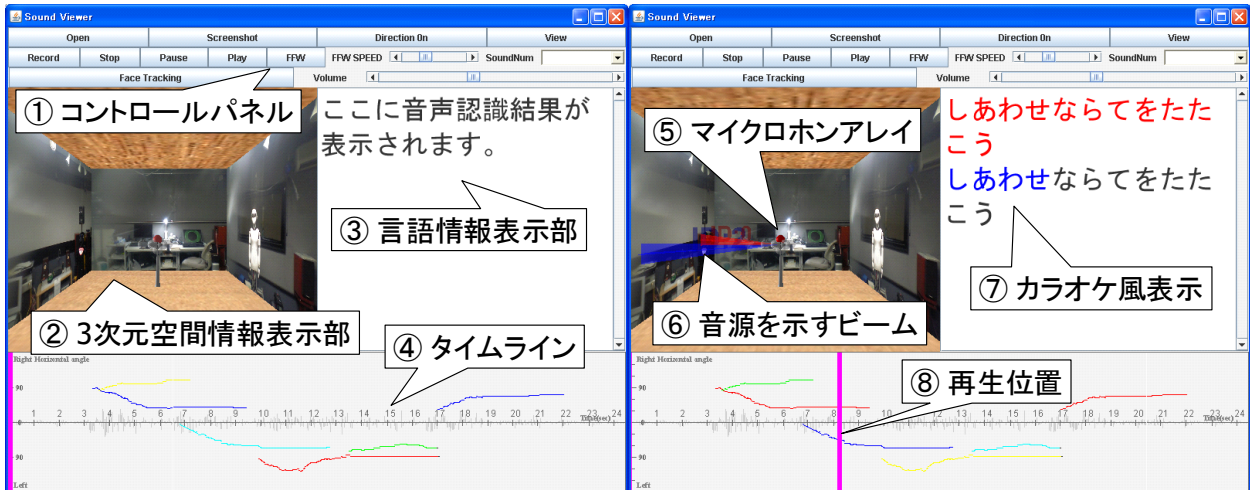


Figure 1.9: GUI of CASA 3D Visualizer

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

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

The GUI explained above is a mode of an external observer that gives bird's eye views of the 3D sound environment. While on the other hand, the first application is provision of an immersion mode that puts the observer in the middle of the 3D sound environment. Taking 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 direction⁶. The design of acoustic information presentation in a telepresence robot is based on the past experience "Auditory awareness is the key technology" described above. Details on transplant of HARK and development of an related modules of HARK for Texai are separated to the following two processes.

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

⁶<http://www.willowgarage.com/blog/2010/03/25/hark-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.

2. Implementation of HARK interfaces and modules to ROS (Robot Operating System), which Texai control programs runs on.

Figure ?? 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 ?? 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 ??, 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 ?. The brown part in the figure is the HARK system. The modules developed here are available at the website of ROS. These series of works including processing of the head, measurement of impulse response, pretests and design of GUI and control command groups were successfully completed in one week. We presume that the high modularity of HARK and ROS contributed to the improvement of productivity.

1.4 Conclusion

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

8 microphones
are embedded.

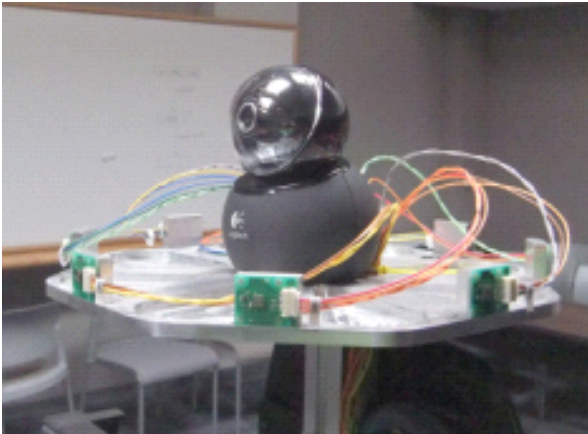


Figure 1.11: Closeup of the first head of Texai: Eight MEMS microphones are embedded on a disk

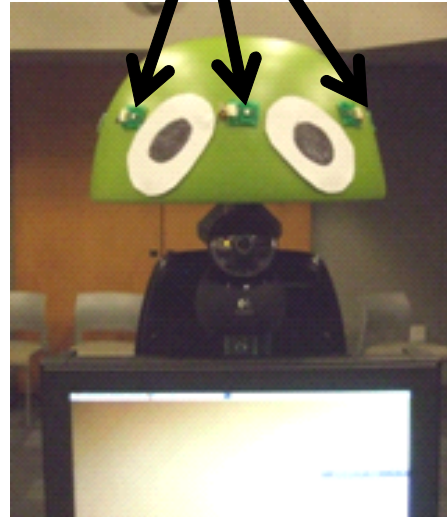


Figure 1.12: Closeup of the head of Texai: Eight MEMS microphones are embedded in the shape of circle

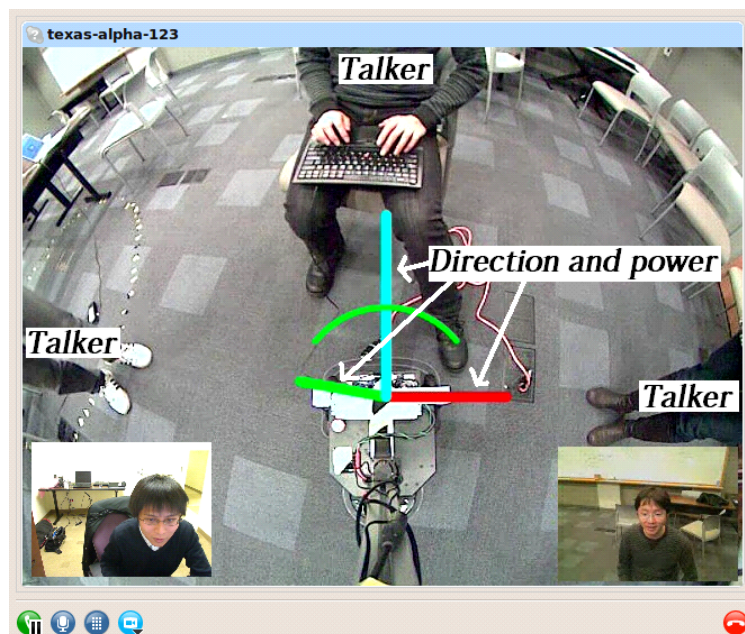


Figure 1.13: Image seen by the remote operator through Texai

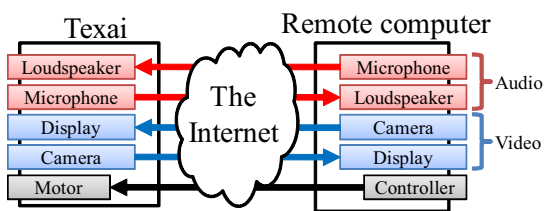


Figure 1.14: Teleoperation of Texai

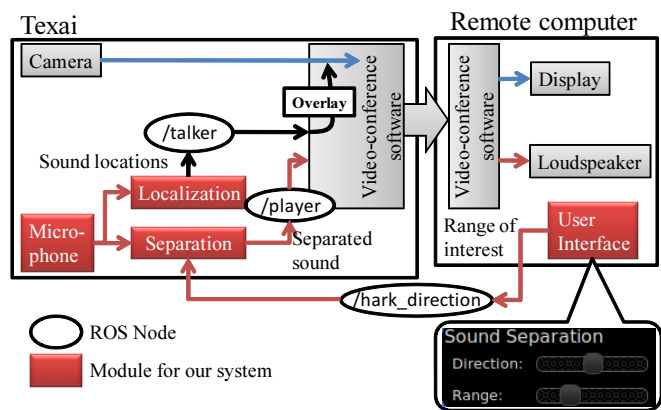


Figure 1.15: Built-in of HARK in Texai

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) [?], 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

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

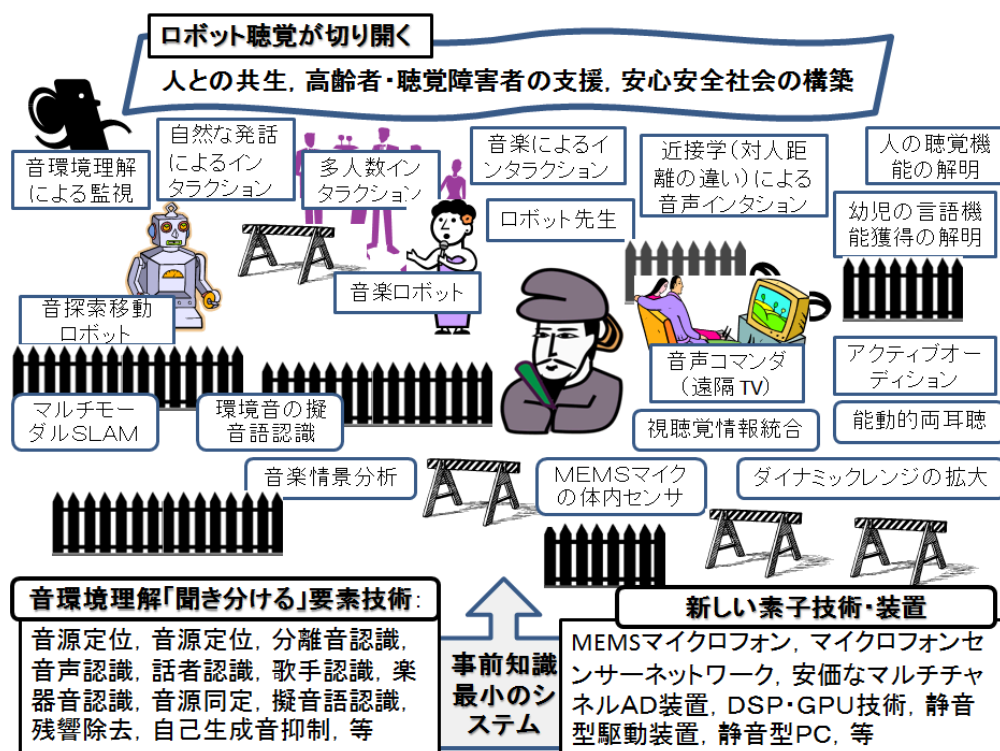


Figure 2.1: Development of robot audition based on sound environment understanding

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[?, ?]. For acquiring pair of reference points, the epipolar geometry is used in stereovision and harmonic structures are used in our method. In sound source localization from sound mixture by two microphones, localization becomes unstable and wobbles in many cases and it is difficult to distinguish especially the sound sources in front and back. Nakadai has realized stable source localization by audio visual information integration while he has realized a robot named SIG, which turns around when it is called [?, ?, ?]. “Seeing is believing” is for solving ambiguity resolution in the front and back problem. Kim and Okuno have realized a system that cancels the ambiguity in source localization by moving the head for a robot named SIG2. The ambiguity is cleared not 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,

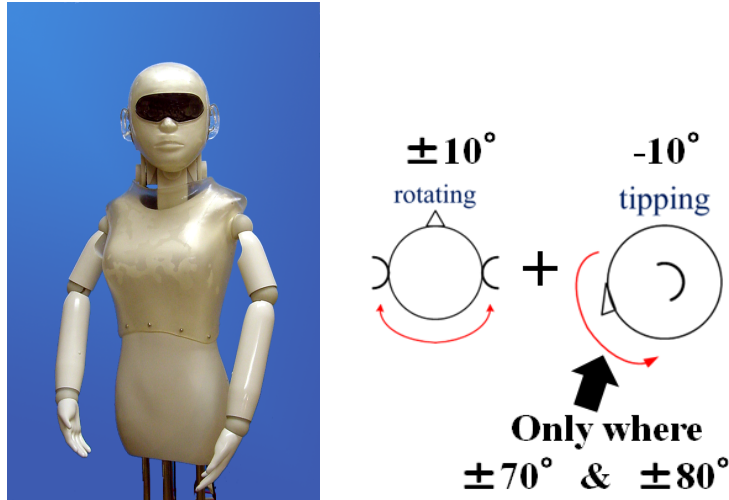


Figure 2.2: Active audition of SIG2: For the surrounding sound, ambiguity of front and back is removed by moving the neck right, left and down.

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

2.4 Function of suppress self-generated sound

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

Self-generated sound suppression based on model

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

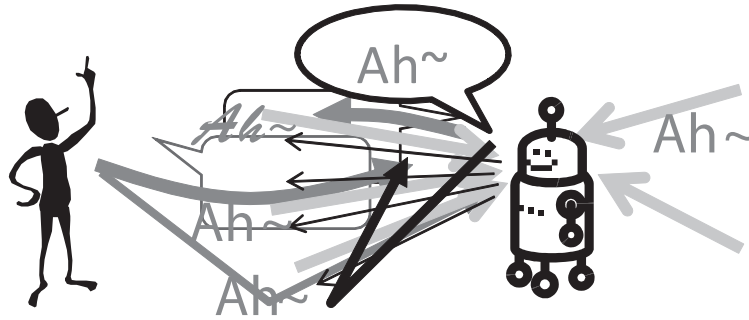


Figure 2.3: The robot's own voice enters its own ears accompanied with reverberations and another person's speech (called barge-in)

so as to suppress self-generated sound [?]. For interactions between a robot and human, it is essential to develop a “strategy for better hearing” such as moving to the position where the robot can hear the sound best or turning the body in consideration for influences of the self-generated sound and environment on the sound.

Self-generated sound suppression by semi-blind separation

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

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 symbiosis 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 [?]. Moreover, it will be necessary to accept new systems such as multichannel A/D systems or MEMS microphones for cellular phones.

Music robot

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

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

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

Audio visual integration type SLAM

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

2.7 Conclusion

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

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 Separation 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-Impaired 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 of Japan, 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 Society 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 and install HARK software, and explains the basic operations for first-time HARK users.

3.1 Software Acquisition

The HARK software installation instructions can be found in HARK'S website. (<https://www.hark.jp/>) The installers for Windows can be downloaded directly from the website. For Ubuntu, follow the instructions in the website on how to register the package's repository, and how to proceed with the installation. HARK's source code can also be downloaded as long as the repositories are registered.

For first-time users, it is strongly recommended to use the Windows package. The installation from source code on the other hand, which is for advanced users, will not be covered in this document.

3.2 Software Installation

3.2.1 For Linux Users

The commands needed for the installation are shown throughout this section. The **>** in the first line indicates a command prompt. The bold and the italic texts in the command indicate the input from the user and the output from the system, respectively. The following is an example of the command prompt.

```
> echo Hello World!  
Hello World!
```

The display of the command prompt may vary depending on the user's working environment (e.g. %, \$). The bold text in the first line of the command prompt is the input from the user. Based from the example above, this is the seventeen letters (including the spaces) of **echo Hello World!**. Enter key is then pressed at the end-of-line. The italic text in the second line is the output from the system. It is displayed after pressing the Enter key at the end of the first line. A part of the user's input and the system's output contains the version and the release numbers. Thus, alteration may be needed depending on the actual version and release to be installed. In addition, system messages shown may differ depending on the availability of the libraries, which can be installed optionally. Even if the messages are not completely identical to the one shown in this manual, the user may proceed as long as there is no error message displayed.

For Ubuntu

Ubuntu users can install using HARK Debian packages. Before proceeding with the installation, the user must first register the HARK repository. Refer to this link on how to register HARK repository: [HARK installation instructions](#)

Then, install the package using the following commands:

```
> sudo apt update
> sudo apt install hark-base harkmw hark-core
> sudo apt install hark-designer
> sudo apt install harktool5 harktool5-gui
> sudo apt install kaldidecoder-hark
```

Source compilation is required when installing HARK to a different environment. See [HARK installation instructions](#) on how to get, compile and install from the source code.

3.2.2 For Windows Users

First, install all the required software for HARK listed here: [HARK installation instructions](#). Then, download the HARK for Windows installer from HARK website. This will install the following software:

1. HARK(hark-base,harkmw,hark-core)
2. HARK-Designer
3. HARKTOOL5
4. HARK-Python3

Next, run the HARK for Windows installer. The installation will begin when the InstallShield Wizard appears. After reading the License Agreement, choose *I accept the terms in the license agreement*, then click *Next* to proceed. In the Setup Type screen, click *Complete* to install all of the program features, or click *Custom* to choose only which program features to install. Upon choosing *Custom* installation, the check on the programs that will not be installed should be removed first before clicking the *Install* button.



Figure 3.1: End-User License Agreement

3.2.3 Uninstalling HARK in Windows

There are 3 ways to uninstall HARK:

- Start menu → HARK → Uninstall

- Control Panel → Uninstall a program → HARK for Windows
- Execute Installer

3.3 HARK Designer

From HARK version 1.9.9, a new web-based GUI with the same system design called **HARK Designer** was developed to replace FlowDesigner. The operation of the GUI is basically the same as FlowDesigner. Please refer to another document on how to operate HARK Designer.

3.3.1 HARK Designer for Ubuntu

An overview of HARK Designer is shown in Figure??. Follow the command below to launch HARK Designer.

```
> hark_designer
```

3.3.2 HARK Designer for Windows

There are 2 ways to launch HARK Designer:

double click the HARK Designer icon on Desktop

or

Click Start menu -> Programs -> HARK -> HARK Designer

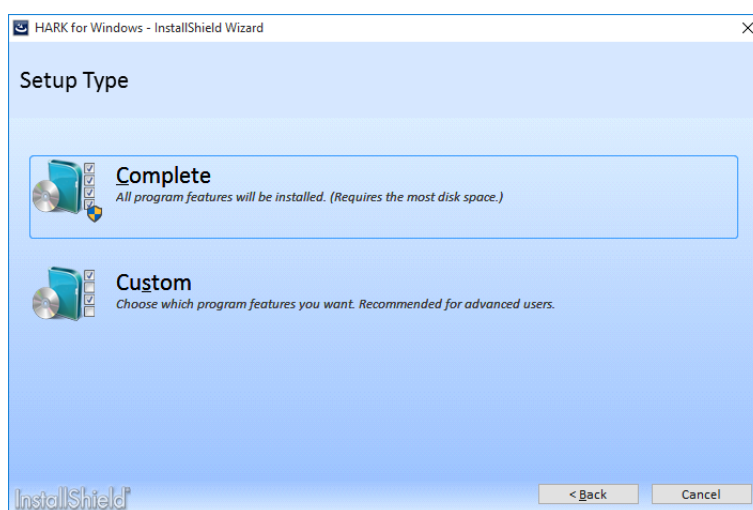


Figure 3.2: Select install type

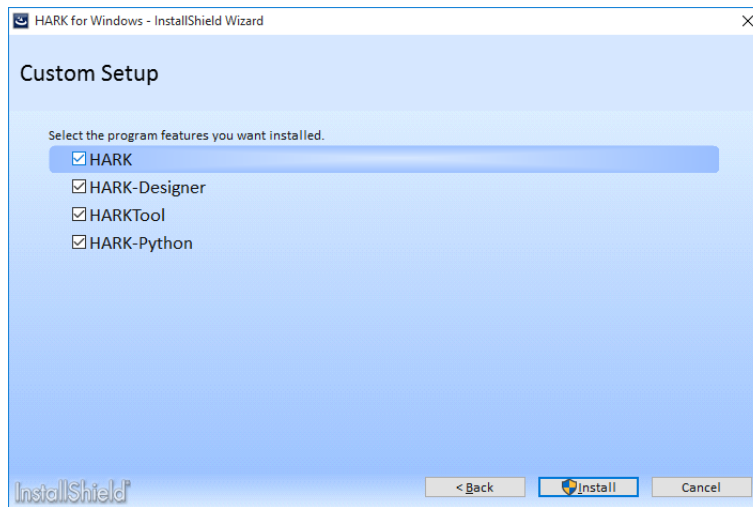


Figure 3.3: Select modules

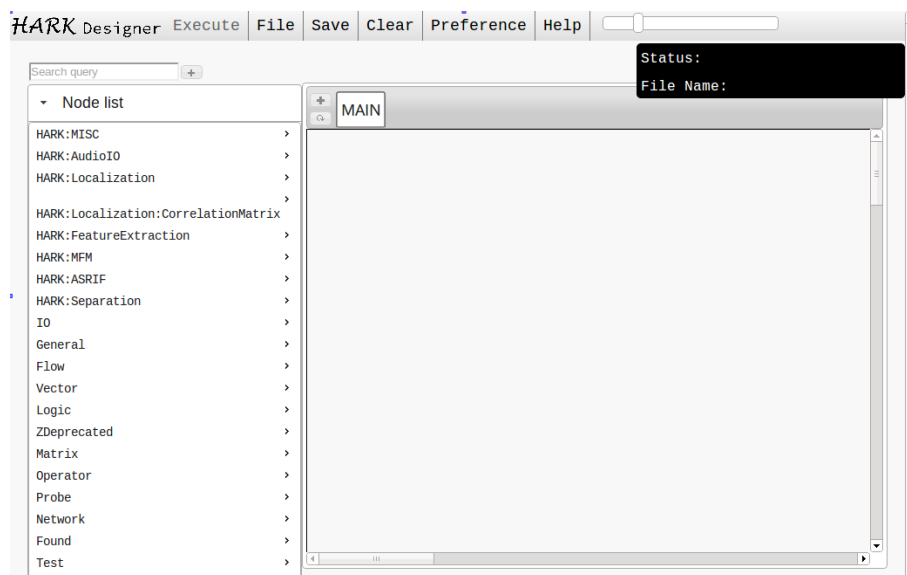


Figure 3.4: Overview of HARK Designer

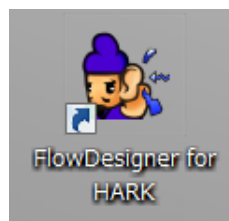


Figure 3.5: The icon of HARK Designer

Chapter 4

Data Types

This chapter describes the different data types used in HARK . The user needs to know about data types because of the following reasons:

- Setting the properties of a node
- Connecting one node to another (inter-node communication)

Data types used in the node's property settings

There are currently 6 data types that can be set in the properties of a node.

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	HARK -specific type
subnet_param	Subnet parameter type	HARK -specific type

The `int` , `float` , `string` and `bool` data types have the same usage and specification as C++. `Object` and `subnet_param` are HARK -specific data types. `Object` inherits the `Object` data type defined in the classes within HARK . The `Vector` and `Matrix` data types, which are non-primitive data types and will be discussed later, inherits the `Object` data type of HARK and can be set as `Object` in the node's property as long as its values can be written in text. Primitive data types can also be set as `Object` as long as it inherits the `Object` data type (example: `<Int 1>`). `subnet_param` is a special data type for setting a common parameter between nodes by assigning a value to a variable.

Data types used for connections between nodes

Different nodes can be connected (inter-node communication) by placing a line in between the terminals of the nodes (shown by a black line in the left or right side of a node) in the GUI of HARK Designer. The data types used for connecting nodes are the following.

Type	Meaning	Data type level
any	Any type	HARK -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	String type	Primitive type
bool	Logical type	Primitive type
Vector	Array type	HARK object type
Matrix	Matrix type	HARK object type
Int	Integral type	HARK object type
Float	Single-precision floating point real type	HARK object type
String	String type	HARK object type
Complex	Complex type	HARK object type
TrueObject	Logical type (true)	HARK object type
FalseObject	Logical type (false)	HARK object type
nilObject	object type (nil)	HARK object type
ObjectRef	Object reference type	HARK -specific type
Map	Map type	HARK -specific type
Source	Sound source information type	HARK -specific type
TransferFunction	Transfer function information type	HARK -specific type

any is a HARK -specific data type that is generic and can handle all data types. int , float , double , complex<float> , complex<double> , char , string and bool has the same specification as C++ primitive types. When a primitive type (besides string and complex) is used as an Object , it will automatically be converted into GenericType<T> . For example, in data types like Int and Float , the first letter of the data type name is replaced by a capital letter which means that it is handled as Object . For String and Complex , although not defined as GenericType, but as std:string and std:complex, respectively, are also handled as Object . This is also applicable to TrueObject , FalseObject and nilObject for true, false and nil, respectively. Primitive types that inherited HARK 's Object class is called "HARK Object types".

The most commonly used HARK Object Types are the Vector , Matrix and Map . These inherited the C++ STL (Standard Template Library) and basically conforms to the specification of C++ STL. ObjectRef is a HARK -specific data type that acts as a smart pointer to the Object type. This is often used as an element for Vector , Matrix and Map data types. Source is a HARK -specific data type that is used to define sound source information.

TransferFunction is a HARK -specific data type that is used to define transfer function.

Node's Terminal Type In HARK Designer, the nodes can successfully connect when the terminals, both the sender and the receiver, has the same data type. The connection is shown by the black line connecting the terminals. Terminals that did not satisfy the conditions will cause an error and will fail to connect. The next section will discuss each of the primitive types, HARK Object types and HARK specific types.

4.1 Primitive type

As discussed in the previous section, `int` , `float` , `double` , `bool` , `char` , `string` and `complex` (`complex<float>` , `complex<double>`) are data types adopted from C++. In HARK , whole numbers that can be identified as integers (sound source count, FFT window length, etc.) uses the `int` data type. Other number values (such as angles) uses `float` . flags or boolean that can only be assigned 2 values (either true or false) uses `bool` . A `string` is used for a sequence of characters such as filenames, etc.. Since HARK is always processing spectrum units and time blocks (frames), primitive types are not directly used as the data type for the node's terminal. Instead, it normally uses `Matrix` , `Vector` or `Map` . The same for `complex<float>` , it is not used individually, but normally used with `Vector` or `Map` in order to express the spectrum. Double precision floating point (`double`) is only used in `Source` .

4.2 HARK Object Type (FlowDesigner compatible)

`Int` , `Float` , `String` and `Complex` are the `Object` type equivalent of `int` , `float` , `string` and `complex` , respectively. `TrueObject` and `FalseObject` are the `Object` type of `bool` to support `true` and `false` conditions, while `nilObject` is the `Object` type of `nil` to support `nil` condition. The details of these data types will not be discussed in this section. HARK adapts to the C++ Standard Template Library (STL), where data types are re-declared as `Object` type. Re-declared `Object` types include `Vector` and `Matrix` , which will be described in details below.

4.2.1 Vector

`Vector` is a data type used to store an array of data. No matter what data is stored in `Vector` , it generally uses delegates such as `Vector< Obj >` with `ObjectRef` as an element, and `Vector< int >` , `Vector< float >` that has an element with values (`int` , `float`). Since the values are always paired before being used, assigning angles in `ConstantLocalization` , or expressing the output of localization results in `LocalizeMUSIC` can be executed. Below is the definition of a `Vector` data type. `BaseVector` is a data type that implements the method for HARK . As shown below, the `Vector` inherited `vector` type of STL.

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

The `ToVect` node in the `Conversion` category accepts input in `int` , `float` , etc., and then converts and output the value into `Vector` . In addition, to use `Vector` as a parameter of a node, the data type of the parameter must be set to `Object` , and then the values must be set to text in the same pattern below. For example, to use two `int` elements 3 and 4 as a parameter, the text should be entered as shown in Figure ?? . However, one should be careful not to put space after the first `<` before writing `Vector` .

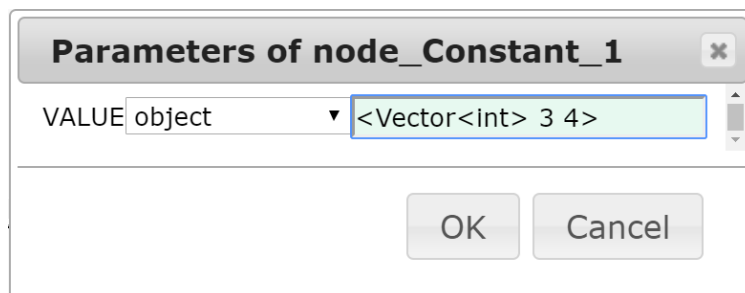


Figure 4.1: Input example of `Vector`

4.2.2 Matrix

A `Matrix` is used to describe rows and columns. It is delegated by data types `Matrix<complex<float> >` type and `Matrix<float>` type, which respectively holds complex numbers matrix and real numbers matrix as elements. Below is the definition of `Matrix` data type. `BaseMatrix` is a class that implements the methods for HARK .

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

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

Matrix is generally used for inter-node communication such as **MultiFFT** (frequency analysis) and **LocalizeMUSIC** (sound source localization). However, in a robot auditory system that uses **HARK** for sound source localization, source tracking, separation and speech recognition, an ID is assigned to each sound source in the source tracking process (**SourceTracker**). Before this process, **Matrix** is used for inter-node communication. But, from the source tracking process and beyond, **Map** is usually used.

4.3 HARK-specific Types (FlowDesigner compatible)

HARK-specific types include `any`, `ObjectRef`, `Object` and `subnet_param`.

4.3.1 any

`any` is a generic term for every data type. When the terminal of a node is set to `any`, no matter what the data type of the receiving node's terminal may be, terminals can be connected (using the black line) without any errors. However, since establishing actual communication depends on the implementation of both of the nodes internally, it is recommended not to use this data type as much as possible. The use of any data type is only limited to generally used nodes such as `MultiFFT`, `DataLogger`, `SaveRawPCM` and `MatrixToMap`.

4.3.2 ObjectRef

This is used as a reference to the data type that has `Object` type as its parent class. Basically, it acts as a smart pointer to the `Object` type. `ObjectRef` can also be used as a reference to HARK Object types and HARK specific types since they also have `Object` as the parent class. Primitive types, as mentioned previously, can be delegated as an `ObjectRef`, since it will be converted to `NetCType<T>`, and then become a subclass of `Object`.

4.3.3 Object

This data type is mainly used in the property settings of a node. It is used by nodes such as `ChannelSelector`, for example. Basically, it enables other data types aside from the already established ones such as `int`, `float`, `string`, `bool` and `subnet_param` to be set as a property. As discussed in Section ??, Because primitive types are also included in data types that can be used as `Object`, in principle, all data types can be declared as an `Object`. However, since the input should be in text, it becomes limited only to data types where the input can be set as text. Currently, `Vector` and `Matrix` can be set as text, but `Map` cannot. So, `Map` cannot be declared as `Object` yet.

Use the Tables ??, ??, ?? below as reference of the format in setting parameter values.

Table 4.1: Primitive Object Sample

Class	Value	Input	Remarks
Char	a	<Char a> <Char a >	HARK does not normally use Char class
Int	1	<Int 1> <Int 1 >	The same with setting the parameter type to <code>int</code> with a value of 1
Float	1.0	<Float 1.0> <Float 1.0 >	The same with setting the parameter type to <code>float</code> with a value of 1.0
Double	1.0	<Double 1.0> <Double 1.0 >	HARK does not normally use Double class
Complex <float >	1.0 + 2.0i	<Complex<float> (1.0, 2.0)> <Complex<float> (1.0, 2.0) >	
Complex<double>	1.0 + 2.0i	<Complex<double> (1.0, 2.0)> <Complex<double> (1.0, 2.0) >	HARK does not normally use <code>Complex<double></code> class
Bool	false true	<Bool 0> <Bool 0 > <Bool 1> <Bool 1 >	The same with setting the parameter type to <code>bool</code> with a value of false The same with setting the parameter type to <code>bool</code> with a value of true
String	"Hello World!" "Hello World! "	<String Hello World!> <String Hello World! >	The same with setting the parameter type to with a value "Hello World!" Note that in this case the trailing space is not ignored
nilObject		<NilObject >	

Table 4.2: Vector Object Sample

Class	Value	Input	Remarks
Vector<int>	[1,2,3,4,5,6]	<Vector<int> 1 2 3 4 5 6> <Vector<int> 1 2 3 4 5 6 >	
Vector<float>	[1.1,2.2,3.3 4.4,5.5,6.6]	<Vector<float> 1.1 2.2 3.3 4.4 5.5 6.6> <Vector<float> 1.1 2.2 3.3 4.4 5.5 6.6 > <Vector 1.1 2.2 3.3 4.4 5.5 6.6> <Vector 1.1 2.2 3.3 4.4 5.5 6.6 >	<float> can be omitted in case the type is float
Vector<Char>	['a', 'b']	<Vector<ObjectRef> <Char a > <Char b > >	
Vector<Int>	[1, 2, 3]	<Vector<ObjectRef> <Int 1> <Int 2> <Int 3> >	
Vector<Float>	[1.1,2.2,3.3]	<Vector<ObjectRef> <Float 1.1> <Float 2.2> <Float 3.3> >	
Vector<Double>	[1.1,2.2,3.3]	<Vector<ObjectRef> <Double 1.1> <Double 2.2> <Double 3.3> >	
Vector<Complex<float> >	[1.0 + 2.0i + 3.0 + 4.0i + 5.0 + 6.0i]	<Vector<ObjectRef> <Complex<float> (1.0,2.0)> <Complex<float> (3.0,4.0)> <Complex<float> (5.0,6.0)> >	
Vector<Bool>	[false, true false, true]	<Vector<ObjectRef> <Bool 0> <Bool 1> <Bool 0> <Bool 1> >	

Table 4.3: Matrix Object Sample

Class	Value	Input	Remarks
Matrix<int>	[[1, 2, 3], [4, 5, 6]]	<Matrix<int> <rows 2> <col 3> <data 1 2 3 4 5 6 > >	
Matrix<float>	[[1.1, 2.2, 3.3], [4.4, 5.5, 6.6]]	<Matrix<float> <rows 2> <col 3> <data 1.1 2.2 3.3 4.4 5.5 6.6 > > <Matrix <rows 2> <col 3> <data 1.1 2.2 3.3 4.4 5.5 6.6 > >	<float> can be omitted in case the type is float
Matrix<Char>	[['a','b'], ['c','d']]	<Matrix<ObjectRef> <rows 2> <col 2> <data <Char a> <Char b> <Char c> <Char d> > >	
Matrix<Int>	[[1, 2, 3], [4, 5, 6]]	<Matrix<ObjectRef> <rows 2> <col 3> <data <Int 1> <Int 2> <Int 3> <Int 4> <Int 5> <Int 6> > >	
Matrix<Float>	[[1.1, 2.2, 3.3], [4.4, 5.5, 6.6]]	<Matrix<ObjectRef> <rows 2> <col 3> <data <Float 1.1> <Float 2.2> <Float 3.3> <Float 4.4> <Float 5.5> <Float 6.6> > >	
Matrix<Double>	[[1.1, 2.2, 3.3], [4.4, 5.5, 6.6]]	<Matrix<ObjectRef> <rows 2> <col 3> <data <Double 1.1> <Double 2.2> <Double 3.3> <Double 4.4> <Double 5.5> <Double 6.6> > >	
Matrix<Complex<float> >	[[1.1, 2.2i, 3.3, 4.4i], [5.5, 6.6i, 7.7, 8.8i]]	<Matrix<ObjectRef> <rows 2> <col 3> <data <Complex<Float> (1.1, 2.2)> <Complex<Float> (3.3, 4.4)> <Complex<Float> (5.5, 6.6)> <Complex<Float> (7.7, 8.8)> > >	

4.3.4 subnet_param

This data type is used in the property settings of a node. When a common parameter in the property of multiple subnet nodes is assigned to a variable declared as `subnet_param`, its value can be changed in the `MAIN(subnet)`, and all nodes assigned with this variable can be modified at the same time. For example, when creating an Iterator network (names `LOOP0`) which has nodes that uses sampling frequency called `LocalizeMUSIC` and `GHDSS`, set the `SAMPLING_RATE` in the property settings of the node as `subnet_param`, then name it as “`SAMPLING_RATE`”. Then, when `LOOP0` is placed in the `MAIN(subnet)`, “`SAMPLING_RATE`”. will be shown in its property. Set this property to `int`, then its value to 16000. This way, it is guaranteed that the `SAMPLING_RATE` property of `LocalizeMUSIC` and `GHDSS` will always have the same value. To show another example, set the property of the node in the `MAIN(subnet)` as `subnet_param`, the set the name as `ARGx` (x is a variable number). Doing so will set its parameters as variables during batch execution (For example, when a `subnet_param` names `ARG1` is created, it will be set as the first variable during batch file execution).

4.4 HARK-specific type

Data types that are originally defined by HARK are `Map` , `Source` and `TransferFunction` .

4.4.1 Map

`Map` is a data type that is composed of a key and `ObjectRef` . Each uttered word can be distinguished by HARK through its speech recognition function. In order to do this, speech ID (sound source ID) is assigned as the key in `Map<int, ObjectRef>` , where `ObjectRef` is a pointer to `Object` data types, such as `Matrix` , `Vector` and `Source` . As an example, the output of GHDSS (sound source separation) is a `Map<int, ObjectRef>` , where the key is the speech ID, and the `ObjectRef` is the pointer to `Vector< complex >` which contains the spectrum of the separated speech.

`MatrixToMap` is used for connection between nodes that are placed before (`Matrix`) and after (`Map`) the sound tracking process for inter-node communication.

4.4.2 Source

`Source` is a data type that is used to express the sound source localization information. In HARK , `LocalizeMUSIC` (output), `SourceTracker` (input-output), `GHDSS` (input), a sequence of nodes such as this, from sound source localization to speech recognition, will set `ObjectRef` of `Map<int, ObjectRef>` to point to the `Source` .

`Source` has the following information below:

1. **ID:** `int` type. Sound source ID
2. **Power:** `float` type. Power of localized direction.
3. **Coordinate:** `float` type, 3-dimensional array. The Cartesian coordinate of a sphere unit that corresponds to the sound source direction.
4. **Duration:** `double` type. The number of frames before a localized sound source is terminated. This value decreases when the sound is not detected. When it reaches 0, the sound source will be terminated. This is an internal variable only used by the `SourceTracker` node.
5. **TF index:** `int` type. Indicates the index of the localized direction within the Transfer Function file.

4.4.3 TransferFunction

`TransferFunction` is a data type that is used to express the transfer function information. In HARK , `EstimateTF` (output), `LocalizeMUSIC` (input), `GHDSS` (input), a sequence of nodes such as this will apply the estimated transfer function to the localization and separation.

`TransferFunction` has the following information below:

1. **Informationtype:** `string` type. The kind of the file. "transfer function" means that the file includes all information. "partial transfer function" means that the file includes only the updated information.
2. **Source position:** `Vector<Position>` type. Source localization information for the transfer function. When the Filetype is "partial transfer function", this parameter is empty.
3. **Neighbor information for source positions:** `Vector<Neighbor>` type. Neighbor information for the source. When the Filetype is "partial transfer function", this parameter is empty.
4. **Microphone position:** `Vector<Position>` type. Microphone information for the transfer function. When the Filetype is "partial transfer function", this parameter is empty.
5. **Transfer function for localization:** `Map<ID,Matrix<complex<float>>>` type. The transfer function for localization.
6. **Transfer function for separation:** `Map<ID,Matrix<complex<float>>>` type. The transfer function for separation.

Position type

Position type, which expresses the microphone and source position of the transfer function, has the following information below:

1. **ID:** int type. ID of the position.
2. **Coordinate type:** Coordinate type. Coordinate type.
3. **Coordinate:** float type. Coordinate of the position.
4. **Path:** string type. Path of the wave file.
5. **Matrix data:** Matrix<complex<float>> type. Matrix data. This parameter is not used.
6. **Enable the channel set information:** int type. Enable the channel set information. This parameter is not used.
7. **Channel set information:** Vector<int> type. Channel set information. This parameter is not used.

Neighbor type

Neighbor type, which expresses the neighbor information of the source, has the following information below:

1. **ID:** Vector<int> type. IDs which has the neighbors.
2. **Neighbor(ID):** Vector< Vector<int> > type. Neighbor information by ID.
3. **Neighbor(Position):** Vector< Vector< Position > > type. Neighbor information by Position type.
4. **Algorithm:** NeighborAlgorithm type. The search algorithm for the neighbors.

Config type

Config type, which expresses the configuration information, has the following information below:

1. **Comment:** string type. The description of the file. Any string is acceptable.
2. **Synchronous Average:** int type. The number of repetition of a signal used for transfer function measurement (TSP signal).
3. **Path:** string type. The path of the audio file for transfer function measurement (TSP signal).
4. **Offset:** int type. The offset during transfer function calculation.
5. **Length:** int type. The length of a signal for transfer function measurement (TSP signal) in a sample.
6. **Begin index for peak search:** int type. The begin index when searching for the peak of the direct sound during transfer function calculation.
7. **End index for peak search:** int type. The end index when searching for the peak of the direct sound during transfer function calculation.
8. **FFT length:** int type. The length of Fourier transform during transfer function calculation.
9. **Sampling rate:** int type. The sampling rate.
10. **signal Max:** int type. The maximum amplitude of the recorded signal for transfer function measurement.

Problem

Read this to know more about nodes that uses (`Map < ., . >`) for input/output such as `MFCCEXtraction` and `SpeechRecognitionClient` .

Solution

The `Map` data type is a set of key and the data that corresponds to that key. For example, in a 3-speaker simultaneous speech recognition, features are separated for each speaker. Then, each feature is assigned a key based on the speaker's speech index ID. The key and data are then handled as a set. Through this each speaker and speech can be distinguished.

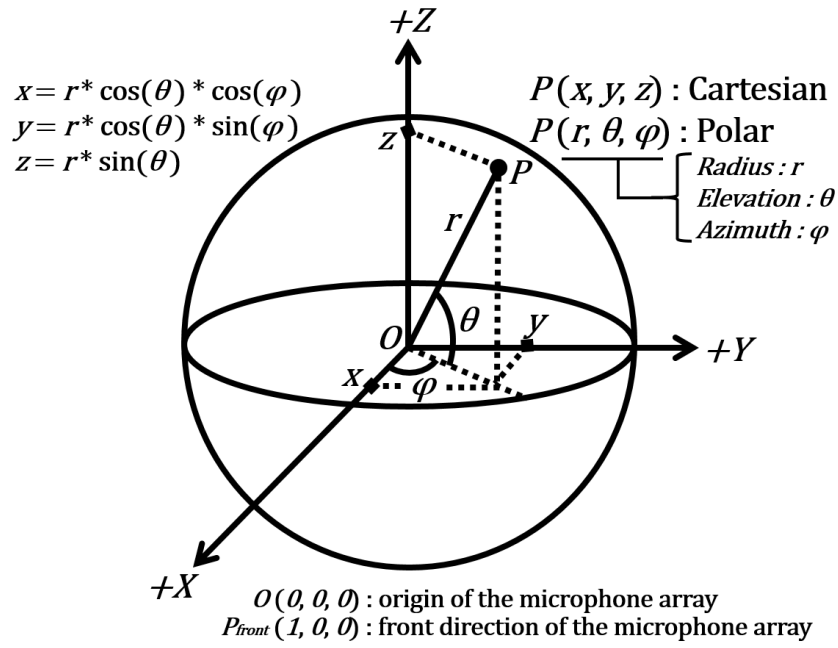


Figure 4.2: HARK standard coordinate system

4.5 HARK Standard Coordinate System

The center of HARK's coordinate system is the origin specified by the user (usually the center of the microphone array). The x positive axis is the front, y positive axis is towards the left, and z positive axis is in the upward direction. The unit is in meters. It is assumed that the front angle is 0[deg] and is rotating in a counterclockwise direction (ex: left direction is 90[deg]). The angle of elevation is defined as the XY plane and formed angle of the direction vector of the coordinates.

Chapter 5

File Formats

This chapter describes the file types, as well as the file formats used in HARK . The earlier versions of HARK used a variety of file formats, which made it difficult to grasp overall, especially in the analysis of complex binary formats of transfer function files. Because of that, from HARK 2.1, file formats has been simplified by implementing the 2 design policies below.

1. Lessen the use of HARK special formats and use standardized format if possible.
2. Utilize libraries that offers file input/output API.

To comply with these policies, standardized format and compounds of standardized format are now being implemented besides Matrix binary, which is the only HARK-specific file format. In addition, a library that supports file input/output functions called "libharkio3" was created for easy file manipulation.

HARK mainly uses the following three formats:

1. **XML:** Files that render the position. The extension is .xml
2. **Matrix binary:** Files that represent a matrix. The extensions is .mat
3. **Zip:** An organization of files mentioned above used for complex file formats such as transfer functions, etc. The extension is .zip

Other file formats used in earlier versions are either integrated to the file formats above or converted into a standardized format.

Table ?? shows the list of files that can be configured in the node's input/output or property settings.

The rest of the chapter will discuss the 3 file formats in detail. Regarding Julius format, see the Julius document for reference. Regarding the difference between the file formats from the original Julius , see the discussion on JuliusMFT . For Raw Audio Format and PCM Wave Format, see the discussion on standard format.

5.1 XML Format

XML is a file format that is used to save information on the position of sound sources, such as microphone positions and sound localization results. As shown in Figure ??, the root element is `hark_xml` , while the child elements are `config`, `positions`, `neighbors`, and `channels` .

5.1.1 hark_xml

What is this node for?

This serves as the root element of all XML files used in HARK .

Attributes

`hark_xml` has a mandatory attribute `version`. The current version is "1.3".

Table 5.1: HARK nodes that uses file I/O

Node name	Where to use	File type	New format	Old format
SaveRawPCM	Output	Raw Audio file	Raw Audio	No change
SaveWavePCM	Output	Wave file	PCM Wave	No change
LocalizeMUSIC	Property	TF(*) for localization	Zip	HGTF binary
SaveSourceLocation	Output	Localization result	XML	Localization result text
LoadSourceLocation	Property	Localization result	XML	Localization result text
GHDSS	Property	TF(*) for separation	Zip	HGTF binary
	Property	Microphone positions	XML	HARK text
	Property	Stationary noise positions	XML	HARK text
	Property	Initial separation matrix	Zip	HGTF binary
	Output	Separation matrix	Zip	HGTF binary
SaveFeatures	Output	Features	Matrix binary	float binary
SaveHTKFeatures	Output	Features	HTK format	No change
DataLogger	Output	Map data	XML	Map text
CMSave	Property	Correlation matrix	Zip	Correlation matrix text
CMLoad	Output	Correlation matrix	Zip	Correlation matrix text
JuliusMFT	Commandline in jconf file in jconf file	Configuration	jconf (text)	No change
		AM(*), phoneme list	julius format	No change
		LM(*), dictionary	julius format	No change
harktool	harktool	Sound source positions list	XML	srcinf text
	harktool	Impulse response	PCM Wave	float binary

TF: Transfer Function, AM: Acoustic Model, LM: Language Model

Child Elements

The child elements config, positions and neighbors are discussed below. Each child element is optional.

5.1.2 config

What is this node for?

It contains the general description of the XML file.

Attributes

No attributes.

Child Elements

config has the following optional child elements. All except "comment" are used for transfer function file.

comment A description of the file. Any string is acceptable.

SynchronousAverage The number of repetition of a signal used for transfer function measurement (TSP signal). A natural number is required.

TSPpath The path of the audio file for transfer function measurement (TSP signal). A string is required.

TSPOffset An offset during transfer function calculation. A natural number is required.

PeakSearch It has the attributes from and to. It is used to specify the range when searching for the peak of the direct sound during transfer function calculation. These attributes are mandatory and must be natural numbers.

nfft The length of Fourier transform during transfer function calculation. A natural number is required.

```

<hark_xml version="1.3">
  <config>
    <comment>Test file</comment>
    <SynchronousAverage>16</SynchronousAverage>
    <TSPpath>/home/tsp.wav</TSPpath>
    <TSPOffset>2</TSPOffset>
    <PeakSearch from="0" to="100"/>
    <nfft>1024</nfft>
    <samplingRate>0</samplingRate>
    <signalMax>0</signalMax>
    <TSPLength>0</TSPLength>
  </config>
  <positions type="tsp" coordinate="cartesian">
    <position x="0.100" y="0.100" z="0.100" id="0" path="/home/tsp1.wav"/>
    <position x="0.150" y="0.100" z="0.100" id="1" path="/home/tsp2.wav"/>
    <position x="0.200" y="0.200" z="0.200" id="2" path="/home/tsp3.wav"/>
  </positions>
  <neighbors algorithm="NearestNeighbor">
    <neighbor id="0" ids="0;1;2;"/>
    <neighbor id="1" ids="1;0;2;"/>
    <neighbor id="2" ids="2;1;0;"/>
  </neighbors>
</hark_xml>

```

Figure 5.1: A sample of XML format

samplingRate The sampling frequency of the recorded signal for transfer function measurement. A natural number is required. The initial sampling rate is 16kHz.

signalMax The maximum amplitude of the recorded signal for transfer function measurement. A natural number is required. If the wave file's sample width is 16bit, the value must be 32767.

TSPLength The length of a signal for transfer function measurement (TSP signal) in a sample. A natural number is required. The initial value is 16384 samples.

5.1.3 positions

What is this node for?

It is an element for describing a set of positions within an XML file.

Attributes

It has 3 attributes.

type Mandatory. It is used to specify the kind of positions. Currently, there are 5 classifications of the type attribute.

- noise: Noise position
- microphone: Microphone positions
- source: Localized sound source positions
- tsp: Positions of TSP recording files
- impulse: Positions of impulse response files

coordinate Mandatory. It describes the position's coordinate system. The value must be `cartesian` for Cartesian coordinate, or `polar` for Polar coordinate

frame Optional. It is used when the position corresponds to a frame number.

Child Elements

The value of each **position** should be greater than or equal to zero. The attributes are mandatory and should have a fixed value.

id : A unique number that signifies a position.

path : The path of the file that corresponds to the position.

The coordinate attribute below depends on the position attribute settings of the root element.

If coordinate = "cartesian" The attributes that represents the coordinate are **x**, **y**, and **z**. The unit is [mm].

If coordinate = "polar" The attributes that represents the coordinate are **azimuth**, **elevation**, and **radius**. The unit is degree for both azimuth and elevation, and mm for radius.

5.1.4 neighbors

What is this node for?

It is an element used to define the the relationship between the neighboring **positions** of sibling elements.

Attributes

It has a mandatory attribute called **algorithm**. It describes the algorithm for calculating the correlation of neighboring positions. Currently, the only algorithm available is the **NearestNeighbor**, which calculates the correlation of neighboring positions nearest to the Euclidean distance.

Child Elements

Each **positions** has a **neighbor** child element that describes the neighboring positions.

Each **neighbor** has two mandatory attributes.

id The id that represents the neighbor position. An integer is required.

ids It includes its own id and the neighboring position's id, divided by a semicolon.

For example, a position with id "1" that has a neighborhood position with id "2" will be shown as below:

```
<neighbor id="1" ids="1;2"/>
```

5.2 Matrix Binary Format

It is used to describe a matrix, such as a direction's transfer function. An overview is shown in Fig. ??.

The first 32 bytes is a string which indicates that the format is a Matrix binary. The next 32 bytes indicates the data type. They are **int32**, **float32**, and **complex**, with the size 4 bytes integer, 4 bytes float, and complex which has 4 bytes float for real numbers and 4 bytes float for imaginary numbers. Next is the dimension of the matrix with a size of 4 bytes integer (Currently the value is fixed to "2" since tensor is not used). Finally, the contents of the rows and columns are saved in the following manner: 1st row 1st column, 1st row 2nd column, 1st row 3rd column, etc. with a size of 4 bytes each.

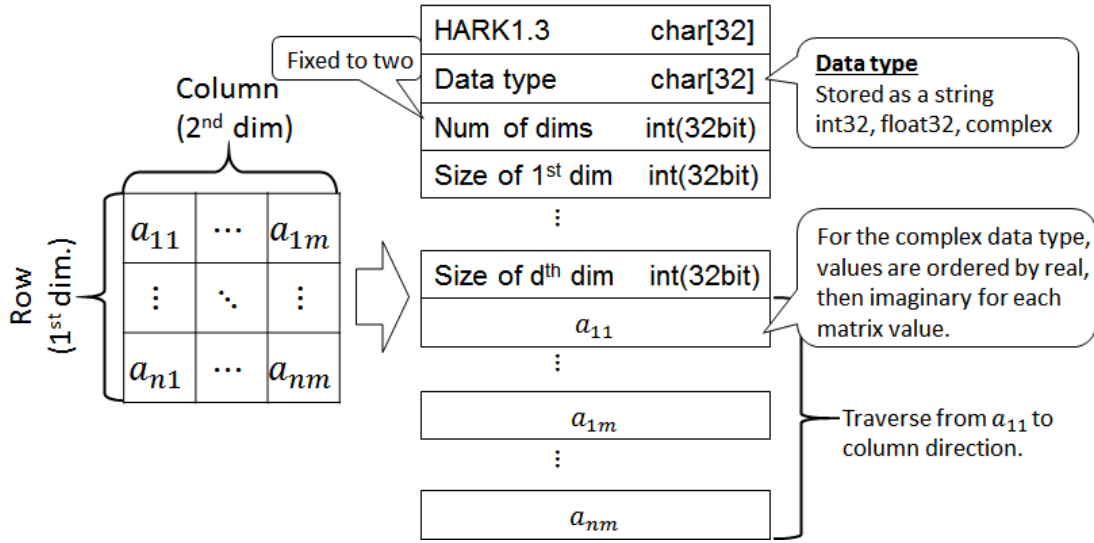


Figure 5.2: Overview of Matrix Binary Format

5.3 Zip Format

Zip is a file format used to describe complex information such as localization of multiple direction transfer function, etc. Text files, XML files and Matrix binary files are simplified, independent and are organized in a directory tree inside the zip file.

Although the files inside the Zip format can be organized in a directory tree, only 1 specific structure of a directory tree is currently supported. This structure is used in transfer function file format, separation matrix of GHDSS node's Export_W, CMLoad, CMSave and such nodes that uses correlation matrix.

5.3.1 Directory tree for transfer functions

The directory tree of a transfer function file is as follows. Note that a name that ends with / is a directory.

```
transferFunction/ --- whatisthis.txt
                  |- microphone.xml
                  |- source.xml
                  |- localization/ --- tf00000.mat
                  |               |- tf00001.mat ...
                  |
                  |- separation/   --- tf00000.mat
                  |               |- tf00001.mat ...
```

The root directory is transferFunction. The whatisthis.txt indicates the file type, which in this case is transfer function. The microphone and source localization information are defined in microphone.xml and source.xml respectively. The transfer function for localization is stored in localization/ directory, while those for separation is stored in separation/ directory. All the files in each directory are in tf%05d.mat format. The filename of the transfer function file is the 5-digit ID number (padded by zero) specified in source.xml.

Note that it doesn't matter if either one or both of the localization/ and separation/ folders are empty. For example, If the localization/ folder is empty, it means that the transfer function for separation used is in the old format. If both localization/ and separation/ folders contain Matrix binary files, it means that the transfer function of both localization and separation was integrated into a single file.

5.3.2 Directory tree for separation matrix of GHDSS

The directory tree of a separation matrix file is as follows. Note that a name that ends with / is a directory.

```
transferFunction/ --- whatisthis.txt
                  |- microphone.xml
                  |- source.xml
                  |- localization/ --- (empty)
                  |- separation/   --- tf000000.mat
                                      |- tf000001.mat ...
```

The root directory is `transferFunction` (though the file is not a transfer function). The `whatisthis.txt` indicates the file type, which in this case is `separation matrix`. The microphone array and the sound source localization result during the separation matrix measurement is defined in `microphone.xml` and `source.xml` respectively.

The separation matrix of GHDSS ($m \times n$, where m = number of microphone and n = complex matrix of the size of frequency bins) is saved in the `separation/` directory. The transfer function files for separation inside the `separation/` directory will have the source ID as its file name. The naming format is `tf%05d.mat`, which is the 5 digit ID number (padded by zero) specified in `source.xml`. The `localization/` will always be empty.

5.3.3 Directory Tree for Correlation Matrix for Localization of CMSave/CMLoad

The directory tree of a correlation matrix file for localization is as follows. Note that a name that ends with / is a directory.

```
transferFunction/ --- whatisthis.txt
                  |- microphone.xml
                  |- source.xml
                  |- localization/ --- tf000000.mat
                                      |- tf000001.mat ...
                  |- separation/   --- (empty)
```

The root directory is `transferFunction` (Although the file is not a transfer function). `whatisthis.txt` represents what the file is. In this case, the content of the file is `correlation matrix`. `microphone.xml` is empty since the microphone is not related to the file. `source.xml` exists only for the compatibility with a transfer function file.

Correlation matrices for localization (A complex square matrix of size n , where n denotes the number of microphones.) are stored in `localization/` directory. Each file name corresponds to each frequency bin. The format of the file name is `tf%05d.mat`, i.e., "tf", a number with 5 digits, and ".mat". The number corresponds to the index of the frequency bin. `separation/` is always empty.

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 ?? season. Details of the contents of the file are described in Section ??.
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 ??. 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 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.

- \mathbf{W} is used for separation matrixes and \mathbf{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, \dots, m_M	Symbol indicating each microphone
n	Index of sound source
N	Number of sound source
s_1, \dots, s_N	Symbol indicating each sound source
i	Index of frequency bin
K	Number of frequency bin
$k_0 \dots k_{K-1}$	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
\mathbf{x}	Vector variable
$[x(1), \dots, x(D)]$	D-dimensional row vector
$[x(1), \dots, x(D)]^T$	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 ??.

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 ?? and ?? show an example usage of the `AudioStreamFromMic` node.

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 ?? 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.

2. RME Hammerfall DSP Multiface series Figures ?? and ?? 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 ??.) 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.

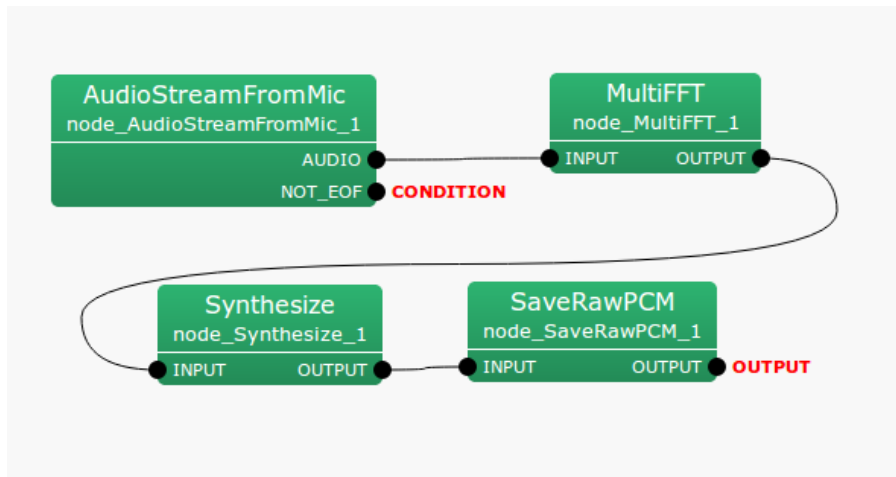


Figure 6.1: Connection example of AudioStreamFromMic 1

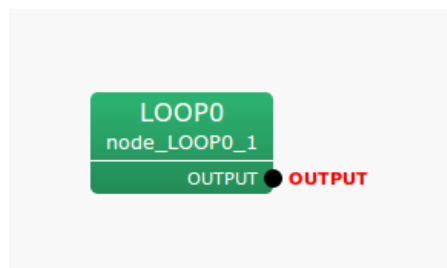


Figure 6.2: Connection example of AudioStreamFromMic 2



Figure 6.3: Wireless RASP

Input-output and property of node

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.



Figure 6.4: Front view of RME Hammerfall DSP Multiface



Figure 6.5: Back view of RME Hammerfall DSP Multiface

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.

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.

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 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	Multiples 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(socket_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 on Windows OS:

On Windows, this node can accept DirectSound devices in addition to wireless RASP, RASP-24, and socket connection. You can designate these devices by entering device name with the DEVICE parameter. Note that you cannot use multi-byte characters for DEVICE parameter.

You have two ways to know the device name. One is to use Device Manager, and the other is to use “Sound Device List” which is provided by HARK. If you want to use Sound Device List, click [Start] → [Programs] → [HARK] → [Sound Device List]. Then, it lists up all the name of sound devices connected to your PC like Figure ?? . You can also use a partial name. For instance, if “Hammerfall DSP” is listed, you can use “Hammerfall” for DEVICE parameter. AudioStreamFromMic uses the top one on the list when more than two candidates are matched.

For three devices, Kinect Xbox, PlayStation Eye, and Microcone, you can use the parameters shown in the next section for DEVICE parameter.

Device corresponding to ASIO on Windows OS:

For ASIO devices, such as Microcone or RME Hammerfall DSP series Multiface, you need to download and install HARK ASIO Plugin from HARK web page.

In this case, you need to use AudioStreamFromASIO instead of AudioStreamFromMic .

- 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 **TAMAGO** on Windows OS

```
CHANNEL_COUNT  8
DEVICETYPE     DS
DEVICE         TAMAGO or tamago
```

- Parameter settings for using **RASP-ZX** on Windows OS

```
CHANNEL_COUNT  8 or 16
DEVICETYPE     DS
DEVICE         rasp
```

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

```
CHANNEL_COUNT  8
DEVICETYPE     ASIO
DEVICE         ASIO Hammerfall DSP
```

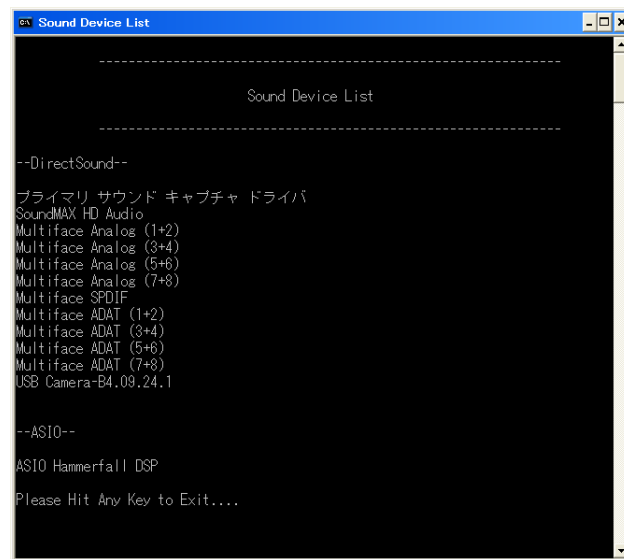


Figure 6.6: Confirmation of the device name

6.1.2 MultiAudioStreamFromMic

Outline of the node

Takes multi-channel speech waveform data from multi microphone arrays. This node is an enhanced version of AudioStreamFromMic to correspond to multiple devices. MultiAudioStreamFromMic corrects the difference in the number of frames between microphone devices when data is received over a long period of time.

Necessary file

No files are required.

Usage

When to use

This node is used to deal with multi-channel speech waveform data from multi microphone arrays as the input for HARK system. Note that it requires that the all microphone arrays are the same model, or that they have the same specification.

Typical connection

Figure ?? shows a connection example of the MultiAudioStreamFromMic .

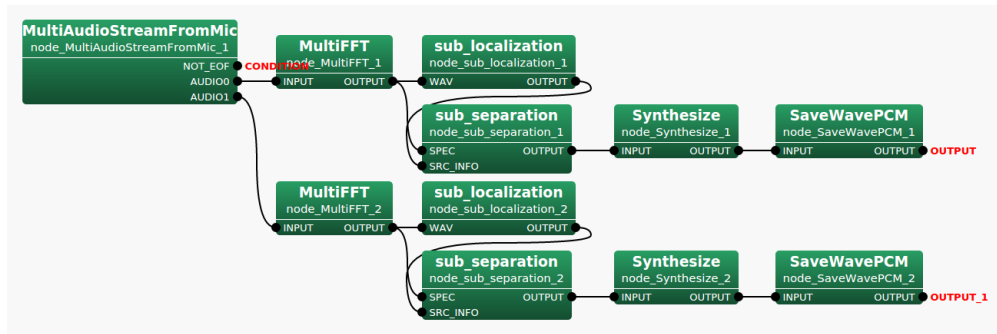


Figure 6.7: Example of the MultiAudioStreamFromMic in LOOP0

Input-output and property of the node

Table 6.3: Parameter list of MultiAudioStreamFromMic

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	/dev/null		A list of identification names required to access the devices.
FRAME_COUNT_SKEW_TOLERANCE	float	5.0	[sec]	The tolerance in frame number in seconds.

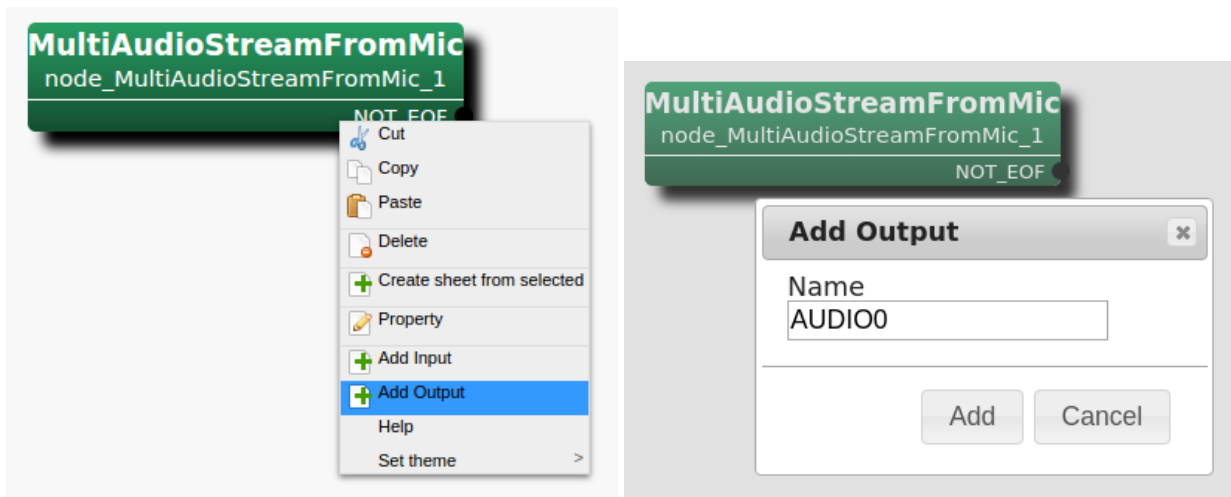
Input

None.
Output

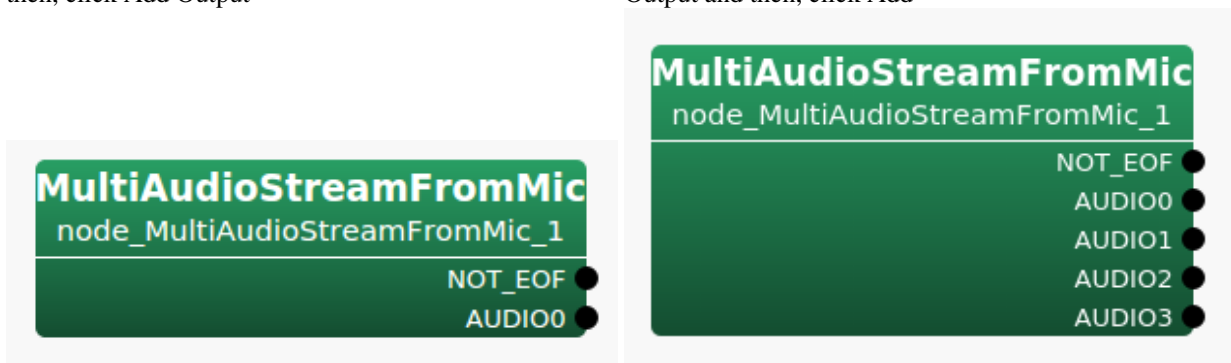
AUDIO00 : `Matrix<float>` type. Indexed, multichannel audio waveform data with rows as channels and columns as samples. Size of the column is equal to the parameter `LENGTH`. This output terminal is hidden by default. The output terminal corresponding to each device needs to be added manually.

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.

Please see the figure ?? to add the output terminals to the node.



Step 1: Right click on MultiAudioStreamFromMic and Step 2: Type AUDIO00 in the input field for Name of Add Output and then, click Add



Step 3: The output terminal, AUDIO00, is added to the node Step 4: Repeat from Step 1 to Step 3 to add the output terminals as needed

Figure 6.8: Steps to add output terminals

Parameter

LENGTH, ADVANCE, CHANNEL_COUNT, SAMPLING_RATE, DEVICETYPE, GAIN : Refer the parameter of the AudioStreamFromMic node. These values are the same for each device.

DEVICE : `string` type. A list of identification names required to access the device. Specify identification names of multiple devices separated by space characters. Each separated identification name of a device is the same as **DEVICE** of `AudioStreamFromMic` .

FRAME_COUNT_SKEW_TOLERANCE : `float` type. The tolerance in frame number in seconds.

Details of the node

This node is an enhanced version of `AudioStreamFromMic` to deal with multiple devices. Each device specified in the list of devices separated by blank space of the **DEVICE** parameter corresponds to each output terminal **AUDIO** arranged in ascending order of the serial number. In the **FRAME_COUNT_SKEW_TOLERANCE** parameter, specify the timing to correct the difference in the number of frames. When the difference between the maximum value and the minimum value in the speech waveform data from multiple devices reaches the value specified by **FRAME_COUNT_SKEW_TOLERANCE**, the data that is the maximum will be deleted by the amount equivalent to the value specified in the **FRAME_COUNT_SKEW_TOLERANCE** parameter.

6.1.3 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 ?? and ?? show an usage example of the `AudioStreamFromWave` node. Figure ?? 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 ?? . Further, connect the output of the `InputStream` node to the iterator subnetwork(`LOAD.WAVE` in Figure ??), which contains networks of various nodes of HARK such as `AudioStreamFromWave`.

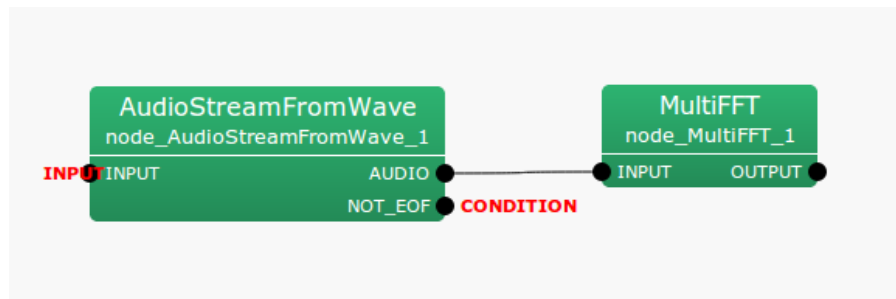


Figure 6.9: Connection example of `AudioStreamFromWave` 1

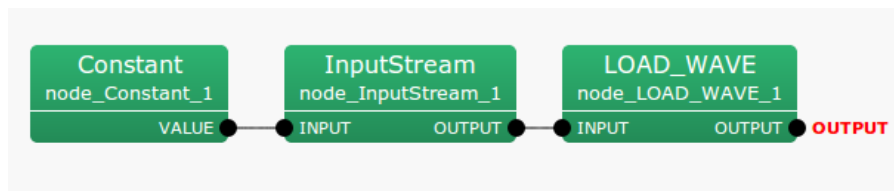


Figure 6.10: Connection example of `AudioStreamFromWave` 2

Input-output and property of node

Input

Table 6.4: 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

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

Output

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 $1/3 \sim 1/2$ of LENGTH would be appropriate for ADVANCE. In the case that sampling frequency is 16000 [Hz], the default values of LENGTH and ADVANCE are 32 and 10 [ms], respectively.

6.1.4 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 `AudioStreamFromMic` node.

Typical connection

Figures ?? and ?? show a usage example of `SaveRawPCM`. Figure ?? 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 ?? 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`.

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 or the format of the file name. See <code>SaveWavePCM</code> for details.
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_BITS	string	as_BITS	[bit]	Quantization bit rate of input speech waveform.

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>`.

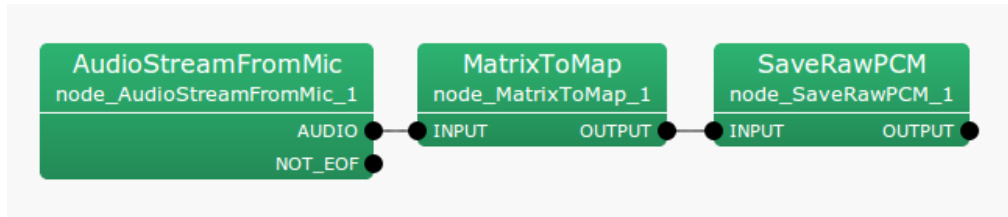


Figure 6.11: Connection example of SaveRawPCM 1

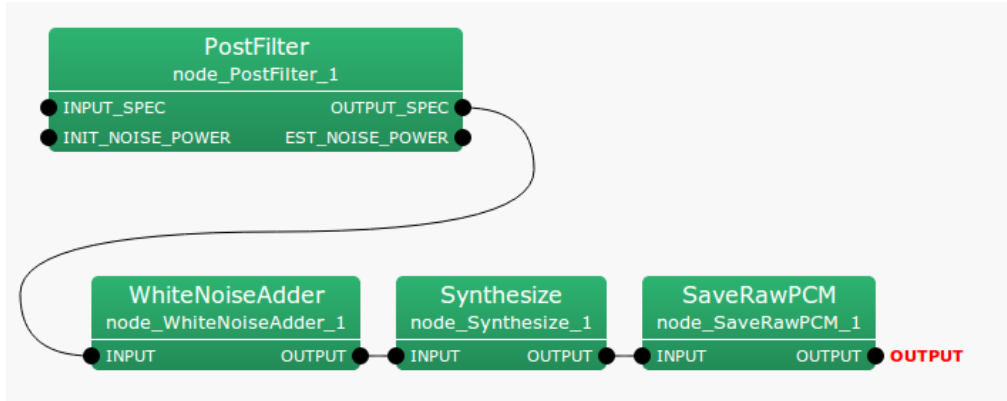


Figure 6.12: Connection example of SaveRawPCM 2

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.

INPUT_BITS : string type. Quantization bit rate of input speech waveform. Select 16 or 24. as_BITS means the same value as BITS.

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.

6.1.5 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. ?? and ??.

Input-output and property of the node

Table 6.6: Parameter list of `SaveRawPCM`

Parameter name	Type	Default value	Unit	Description
BASENAME	string	sep		Prefix or format string of the file name to save.
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_BITS	string	as_BITS	[bit]	Quantization bit rate of input speech waveform.

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.

SOURCES : `Vector<ObjectRef>` type. Sound source localization results (`Vector` of `Source` type objects) is acceptable. This input is optional.

Output

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

Parameter

BASENAME : string type. It is used for formatting the file name. The default value is `sep_`. See the next section for details.

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.

INPUT_BITS : string type. Quantization bit rate of input speech waveform. Select int16 or int24. as_BITS means the same value as BITS.

Details of the node

Format of the files saved: The files are saved as Wave PCM sound data format with 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.

The file name determination: This node provides two ways of file name determination.

1. Prefix [default]

The value of BASENAME parameter is used as prefix. The source ID and an extension .wav is concatenated afterwards. For example, if the BASENAME is sep_ and the input source IDs are 1, 2, 3, then, the file names become sep_0.wav, sep_1.wav, sep_2.wav.

2. Format string [after HARK 2.3.1]

If the BASENAME contains a special pattern {tag:format}, the value is interpreted as a format string. Using this formatting, the users can set more flexible file names.

The number of acceptable values for tag is four (Table ??) srcid and date, meaning the source ID and the time, respectively, can be used without any limitations. However, to use azimuth and elevation for the format string, the SOURCES input of the node must be connected to utilize the sound source localization results.

The value format is optional. This part can be used to specify the digits, such as %3d. The expression is the same as printf.

Table 6.7: Tag List

Tag	Description	Unit
srcid	Source ID	integer
date	Time when the file is saved	String (yyyyMMDD-HH:mm:ss)
azimuth	Azimuth of the localized sound (SOURCES input required)	degree(integer rounded to the nearest decimal point)
elevation	Elevation of the localized sound (SOURCES input required)	degree(integer rounded to the nearest decimal point)

Example format strings

Insert source ID at the middle of the file name

- FORMAT: wav_id_{srcid}_output
- OUTPUT: wav_id_0_output.wav, wav_id_1_output.wav, ...

Insert source ID with zero padding

- FORMAT: wav_id_{srcid:%03d}

- OUTPUT: wav_id_000_output.wav, wav_id_001_output.wav, ...

Insert azimuth to the filename

- FORMAT: wav_az_{azimuth}
- OUTPUT: wav_az_30.wav, wav_az_-10.wav, ...

Note that, in this case, the file is overwritten if two localization results have the same azimuth. To avoid this, adding srcid in the format is recommended to ensure the uniqueness of the file name.

6.1.6 SaveWavePCM2

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. The difference from the `SaveWavePCM` node is that the input terminal `ENABLE` can control whether the file is saved or not at any timing. Also, the time stamp is inserted into the file name, so that the uniqueness of the file name is always ensured. Note that the file is not saved because the input terminal `ENABLE` is recognized as `false` if it is not connected. If you always want to save the file as a replacement for `SaveWavePCM`, set the `Constant` node to `true` and connect.

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 `SaveWavePCM2` node in Fig. ?? and ??. Note that the `ENABLE` terminal must be connected with the `Constant` node set to `true`. If you want to control whether or not files are saved dynamically, you need to be able to control `true` and `false` externally, for example using HARK-Python.

Input-output and property of the node

Table 6.8: Parameter list of `SaveWavePCM2`

Parameter name	Type	Default value	Unit	Description
<code>BASENAME</code>	<code>string</code>	<code>sep_</code>		Prefix or format string of the file name to save.
<code>ADVANCE</code>	<code>int</code>	<code>160</code>	<code>[pt]</code>	Shift length of the analysis frame of the speech waveform to be saved in a file.
<code>SAMPLING_RATE</code>	<code>int</code>	<code>16000</code>	<code>[Hz]</code>	Sampling rate. This parameter is set in the header.
<code>BITS</code>	<code>string</code>	<code>int16</code>	<code>[bit]</code>	Quantization bit rate of speech waveform to be saved in a file. Choose <code>int16</code> or <code>int24</code> .
<code>INPUT_BITS</code>	<code>string</code>	<code>as_BITS</code>	<code>[bit]</code>	Quantization bit rate of input speech waveform.

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.

SOURCES : `Vector<ObjectRef>` type. Sound source localization results (`Vector` of `Source` type objects) are acceptable. This input is optional.

ENABLE : `bool` type. The file is saved only when this input terminal is 1 or `true`. Differences from `SaveWavePCM`.

Output

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

OUTPUT_FILE_NAME : string type. If the input is Matrix<float> type, the name of the currently saved file is output. If the input is Map<int, ObjectRef> type, it is an empty string. Difference from SaveWavePCM .

OUTPUT_FRAME_COUNT : int type. When the input is Matrix<float> type, the number of frames included in the saved file is output. If the input is of type Map<int, ObjectRef> , then the value is undefined (usually 0 but not guaranteed as it is uninitialized). Difference from SaveWavePCM .

Parameter

BASENAME : string type. It is used for formatting the file name. The default value is sep_. See the next section for details.

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.

INPUT_BITS : string type. Quantization bit rate of input speech waveform. Select int16 or int24. as_BITS means the same value as BITS.

Details of the node

Format of the files saved: The files are saved as Wave PCM sound data format with 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.

The file name determination: This node provides two ways of file name determination.

1. Prefix [default]

The value of BASENAME parameter is used as prefix. After that, the sound source ID is concatenated and the time stamp follows in the form .YYYYMMDD.hhmmss.uuuuuu. Finally, the extension .wav is attached. For example, if the BASENAME is sep_ and the input source IDs are 1, 2, 3, then, the file names become sep_0.20210101.120001.987654.wav, sep_1.20210101.120003.123456.wav, sep_2.20210101.120005.246890.wav.

2. Format string [after HARK 2.3.1]

If the BASENAME contains a special pattern {tag:format}, the value is interpreted as a format string. Using this formatting, the users can set more flexible file names.

The number of acceptable values for tag is four (Table ??) the srcid and date, meaning the source ID and the time, respectively, can be used without any limitations. However, to use azimuth and elevation for the format string, the SOURCES input of the node must be connected to utilize the sound source localization results.

The value format is optional. This part can be used to specify the digits, such as 03d. The expression is the same as printf.

Example format strings

Table 6.9: Tag List

Tag	Description	Unit
srcid	Source ID	integer
date	Time when the file is saved	String (yyyyMMdd-HH:mm:ss)
azimuth	Azimuth of the localized sound (SOURCES input required)	degree(integer rounded)
elevation	Elevation of the localized sound (SOURCES input required)	degree(integer rounded)

Insert source ID at the middle of the file name

- FORMAT: wav_id_{srcid}_output
- OUTPUT: wav_id_0_output.20210101.120001.987654.wav, wav_id_1_output.20210101.120003.123456.wav, ...

Insert source ID with zero padding

- FORMAT: wav_id_{srcid:03d}
- OUTPUT: wav_id_000.20210101.120001.987654.wav, wav_id_001.20210101.120003.123456.wav ...

Insert azimuth to the filename

- FORMAT: wav_az_{azimuth}
- OUTPUT: wav_az_30.20210101.120001.987654.wav, wav_az_-10.20210101.120003.123456.wav ...

Note that, in this case, the file is overwritten if two localization results have the same azimuth. To avoid this, adding srcid in the format is recommended to ensure the uniqueness of the file name.

6.1.7 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
- Arbitrary Texts
- Arbitrary Matrices
- Arbitrary Vectors

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 ??, 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”.

Input-output and property of the node

Table 6.10: 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	5530		Port number for outbound network communication
ADVANCE	int	160	[pt]	Shift length of frame
BUFFER_SIZE	int	512		Size of float variables for socket communication
FRAMES_PER_SEND	int	1	[frm]	Frequency of socket communication in frame unit
TIMESTAMP_TYPE	string	GETTIMEOFDAY		Time stamped to the sent data
SAMPLING_RATE	int	16000	[Hz]	Sampling frequency
DEBUG_PRINT	bool	false		ON/OFF for outputting debugging information
SOCKET_ENABLE	bool	true		Flag to determine whether or not to use the socket output

Input

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

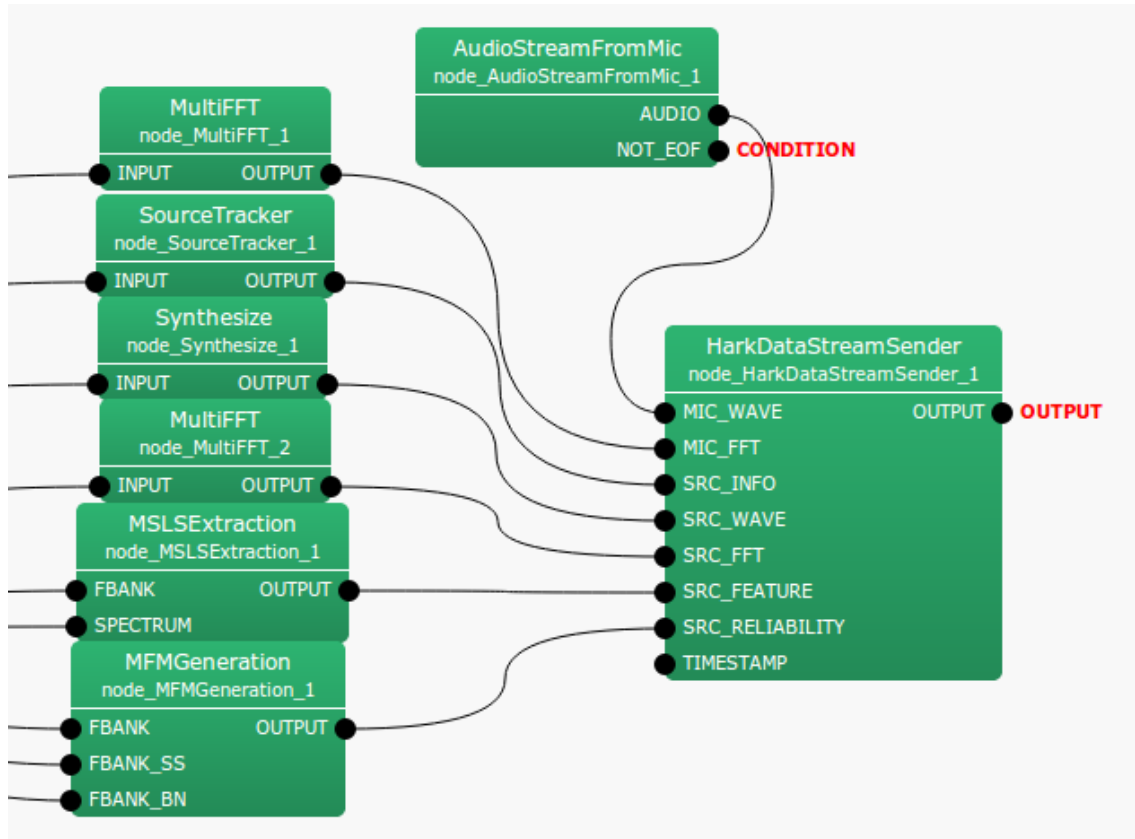


Figure 6.13: Connection example of HarkDataStreamSender

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.

TEXT : ~type. Arbitrary texts.

MATRIX : Matrix<float> or Matrix<complex<float> > type. Arbitrary matrices.

VECTOR : Vector<float> or Vector<complex<float> > type. Arbitrary vectors.

TIMESTAMP : TimeStamp type. Time stamp in the sent data.

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.

TIMESTAMP_TYPE : string type. Setting for time stamped to sent data. If **TIMESTAMP_TYPE**=**GETTIMEOFDAY**, the time taken by **gettimeofday** is stamped. If **TIMESTAMP_TYPE**=**CONSTANT_INCREMENT**, the frame time calculated by **SAMPLING_RATE** is incremented to the stamped current time.

SAMPLING_RATE : int type. Sampling frequency of the input signal. This is valid only when **TIMESTAMP_TYPE**=**CONSTANT_INCREMENT**.

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. **TIMESTAMP_TYPE** is the setting for time stamped to sent data. **SAMPLING_RATE** is the sampling frequency of the input signal. **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 ???. 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) + 2 * sizeof(int64)

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 0000000010 in binary number. Moreover, the transmitted data becomes only a frequency spectrum for each microphone.

Table 6.11: 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 ??.
advance	int	Shift length of a frame
count	int	Frame number of HARK
tv_sec	int64	timestamp of HARK in seconds
tv_usec	int64	timestamp of HARK in micro-seconds

Table 6.12: 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
The eighth column	TEXTS	Arbitrary texts
The ninth column	MATRIX	Arbitrary matrices
The tenth column	VECTOR	Arbitrary vectors

Example 2) In the case that the three input terminals of MIC_WAVE, SRC_INFO and SRC_FEATURE are connected, the type is 0000100101 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 0000000000 in binary.

- **HDH_MicData**

Description: Structural information on the array size for sending two-dimensional arrays

Data size: 3 * sizeof(int)

Table 6.13: 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)

- **HDH_SrcData**

Description: Structural information on the array size for sending one-dimensional arrays

Data size: 2 * sizeof(int)

Table 6.14: 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

Table 6.15: 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, <code>length * sizeof(float)</code> .

(B-2) Transmitted data

Table 6.16: 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	TEXT	MATRIX
(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(HDH_SrcData)					○*				
(l)	float[]	HDH_SrcData.data_bytes					○*				
(m)	float[]	HDH_SrcData.data_bytes					○*				
(n)	HDH_SrcData	sizeof(HDH_SrcData)						○*			
(o)	float[]	HDH_SrcData.data_bytes						○*			
(p)	HDH_SrcData	sizeof(HDH_SrcData)							○*		
(q)	float[]	HDH_SrcData.data_bytes							○*		
(r)	HDH_SrcData	sizeof(HDH_SrcData)								○*	
(s)	char[]	HDH_SrcData.data_bytes								○*	
(t)	HDH_MicData	sizeof(HDH_MicData)									○*
(u)	float[]	HDH_MicData.data_bytes									○*
(v)	HDH_SrcData	sizeof(HDH_SrcData)									
(w)	float[]	HDH_SrcData.data_bytes									

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

Table 6.17: Details of the transmitted data

	Description	Debug
(a)	Transmitted data header. See Table ??.	○
(b)	Structure of acoustic signals (number of microphones, frame length, byte count for transmission). See Table ??.	○
(c)	Acoustic signal (number of microphones \times float type matrix of frame length)	
(d)	Structure of frequency spectra (number of microphones, number of frequency bins, byte count for transmission). See Table ??.	○
(e)	Real part of frequency spectrum (number of microphones \times float type matrix of number of frequency bins)	
(f)	Imaginary part of frequency spectrum (number of microphones \times float type matrix of number of frequency bins)	
(g)	Number of sound sources detected	○
(h)	Source of a source location result. See Table ??.	○
(i)	Structure that indicates that of acoustic signals for each sound source ID (advance length, byte count for transmission). See Table ??.	○
(j)	Acoustic signal for each sound source ID (short int type linear array of advance length)	
(k)	Structure that indicates that of frequency spectra for each sound source ID (number of frequency bins, byte count for transmission). See Table ??.	○
(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 ??.	○
(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 ??.	○
(q)	MFM for each sound source ID (float type linear array of dimension number of features)	
(r)	Text information (number of characters, byte count for transmission). See Table ??.	○
(s)	Text (char type linear array of dimension number of features)	
(t)	Structure of the sent matrix (number of rows and columns and byte count). See Table ??.	○
(u)	Data of the matrix (float type matrix)	
(v)	Structure of the sent vector (size of the vector and byte count). See Table ??.	○
(w)	Data of the vector (float type linear array)	

```

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'.)
        FOR 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

    IF TEXT is connected
        Send (r)
        Send (s)
    ENDIF

    IF MATRIX is connected
        Send (t)
        Send (u)
    ENDIF

    IF VECTOR is connected
        Send (v)
        Send (w)
    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)-(w) in the code correspond to (a)-(w) in Tables ?? and ??.

6.1.8 PlayAudio

Outline of the node

This node plays the waveform data.

Necessary files

No files are required.

Usage

When to use

This node is used when wishing to listen to the separated speech waveform data or the recorded waveform data from microphones.

Typical connection

Figure ?? and Figure ?? show example usages of the PlayAudio node.

Figure ?? shows the example usage of listening to the separated speech waveform data. The separated speech waveform data can be calculated by the GHDSS node and the Synthesize node. When you use the stereo device, the waveform data for L channel should be input to INPUT1, and the waveform data for R channel should be input to INPUT2. When the waveform data is the multichannel data, the waveform data of all channel are mixed in each channel.

Figure ?? shows the example usage of listening to the recorded waveform data from microphones. The multichannel waveform data taken by AudioStreamFromWave should be input to INPUT_MULTI_CHANNEL. The assignment to the output channel is set using the MULTI_CHANNEL_ASSIGN parameter.

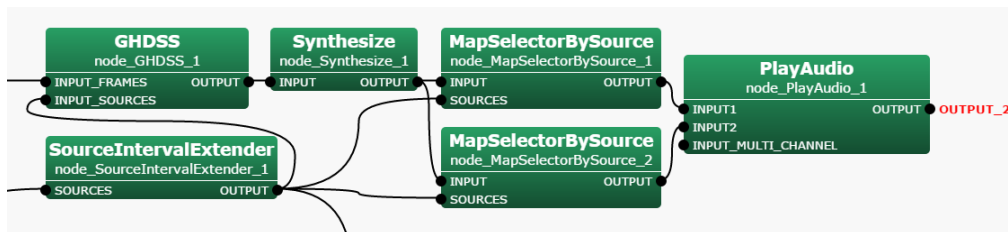


Figure 6.14: Connection example of PlayAudio 1

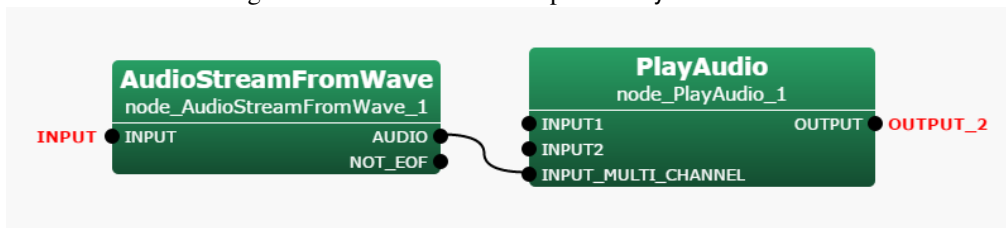


Figure 6.15: Connection example of PlayAudio 2

Input-output and property of the node

Input

Table 6.18: Parameter list of PlayAudio

Parameter name	Type	Default value	Unit	Description
MASTER_VOLUME	float	0	[dB]	Master volume of PlayAudio node.
DEVICE	int			The device index for playing. When this parameter is empty, PlayAudio uses the default device.
CHANNEL_COUNT	int	2		The number of channel for playing. The maximum value of this parameter is 2.
MULTI_CHANNEL_ASSIGN	Vector<int>	See below.		The assignment to the output channel. This parameter is used with INPUT_MULTI_CHANNEL.
LATENCY	int	1000	[msec]	The delay time for the stable playing.
LENGTH	int	512	[pt]	FFT length
ADVANCE	int	160	[pt]	Shift length
SAMPLING_RATE	int	16000	[Hz]	Sampling rate

INPUT1 : Matrix<float>, Map<int, ObjectRef> or Vector<float> type. ObjectRef of Map<int, ObjectRef> is Vector<float>. The waveform data for the channel 1 of the sound device. When the waveform data is the multichannel data, the waveform data of all channel are mixed.

INPUT2 : Matrix<float>, Map<int, ObjectRef> or Vector<float> type. ObjectRef of Map<int, ObjectRef> is Vector<float>. The waveform data for the channel 2 of the sound device. When the waveform data is the multichannel data, the waveform data of all channel are mixed.

INPUT_MULTI_CHANNEL : Matrix<float> or Map<int, ObjectRef> type. The assignment to the output channel is set using the MULTI_CHANNEL_ASSIGN parameter.

Output

OUTPUT : Matrix<float> type. The waveform data for the output channels. This data is not affected by LATENCY parameter.

Parameter

MASTER_VOLUME : float type. Master volume of PlayAudio node.

DEVICE : int type. The device index for playing. When this parameter is empty, PlayAudio uses the default device.

CHANNEL_COUNT : int type. The number of channel for playing. The maximum value of this parameter is 2.

MULTI_CHANNEL_ASSIGN : Vector<int> type. The assignment to the output channel. This parameter is used with INPUT_MULTI_CHANNEL. You should set the index or ID of the input data for the output channel. For example, when you wish to listen to the index 1 and 2 of the Matrix<float>, you should set this parameter <Vector<int> 1 2>. When the element of this parameter is larger than the size of Matrix<float>, PlayAudio plays the waveform data of the available channel, and PlayAudio shows the actual parameters. When this parameter is empty, PlayAudio plays the waveform data of the higher index with the number of the output channel.

LATENCY : int type. The delay time for the stable playing.

LENGTH : int type. FFT length; must be equal to the other node(AudioStreamFromMic, MultiFFT).

ADVANCE : int type. Shift length; must be equal to the other node(AudioStreamFromMic, MultiFFT).

SAMPLING_RATE : int type. Sampling rate; must be equal to the other node(AudioStreamFromMic, MultiFFT).

Details of the node

You should use either INPUT# or INPUT_MUITL.CHANNEL. When both INPUT# and INPUT_MUITL.CHANNEL are used, PlayAudio plays the mixed data.

Only one PlayAudio can be placed in the network file.

To change the assinment to the output channel:

When INPUT1 and INPUT2 is used, you should replace the connection. When INPUT_MUITL.CHANNEL is used, you should change the MULTI.CHANNEL_ASSIGN parameter.

To show the list of the output sound device:

The device list index can be shown by “Output Sound Device List”, which is provided by HARK. If you want to use Output Sound Device List, click [Start] → [Programs] → [HARK] → [Output Sound Device List]. Then, it lists up all the name of sound devices connected to your PC like Figure ??.

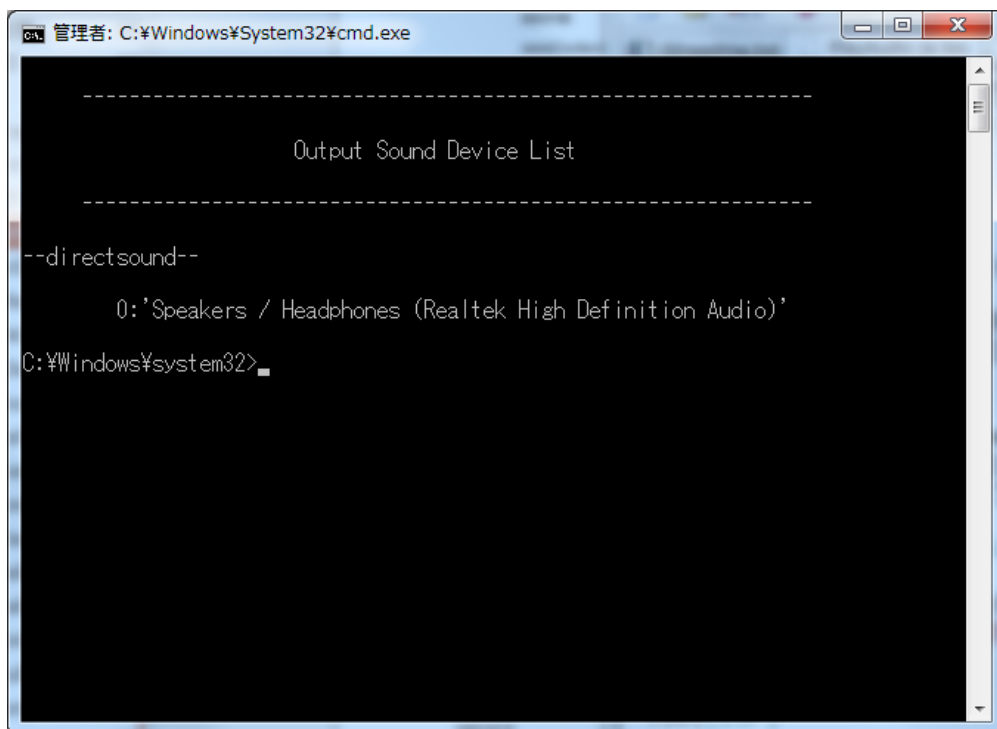


Figure 6.16: Output sound device list

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 ?? shows the usage example for the CMLoad node.

- **Before Version 2.0**

FILENAMER and FILENAMEI are inouts with type string, indicating the files containing the real and imaginary parts respectively of the correlation matrix.

- **After Version 2.1**

Load a zip file. Only FILENAMER is used, and FILENAMEI is ignored.

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.

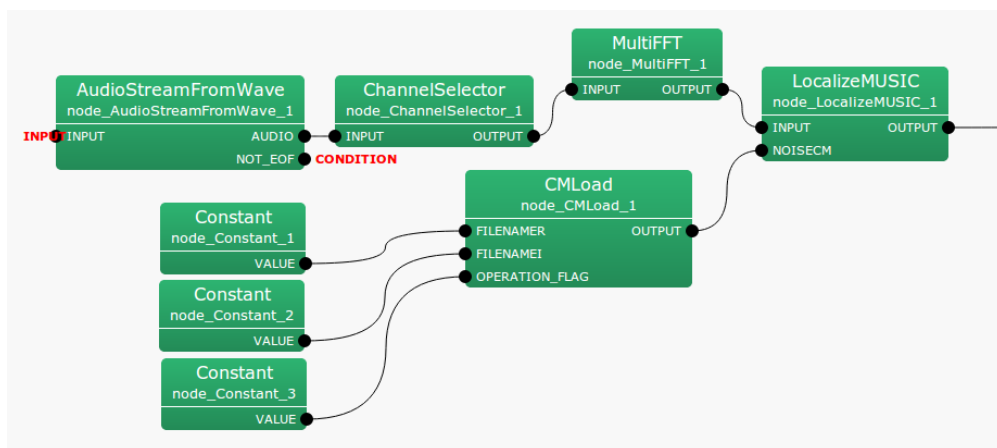


Figure 6.17: Network Example using CMLoad

I/O and property setting of the node

Input

Table 6.19: Parameter list of CMLoad

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

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

OUTPUT : 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. ?? shows a usage example of `CMSave` node.

`INPUTCM` input terminal is connected with a correlation matrix calculated from `CMMakerFromFFT` or `CMMakerFromFFTwithFlag`, etc.

- **Before Version 2.0**

The 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.

- **After Version 2.1**

Saved as a zip file. Only `FILENAMER` is used, and `FILENAMEI` is ignored.

`OPERATION_FLAG` is `int` type or `bool` type input, specifying when the correlation matrix is read. (Figure. ?? 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).

I/O and property setting of the node

Table 6.20: Parameter list of `CMSave`

Parameter	Type	Default	Unit	Description
<code>ENABLE_DEBUG</code>	<code>bool</code>	<code>false</code>		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.

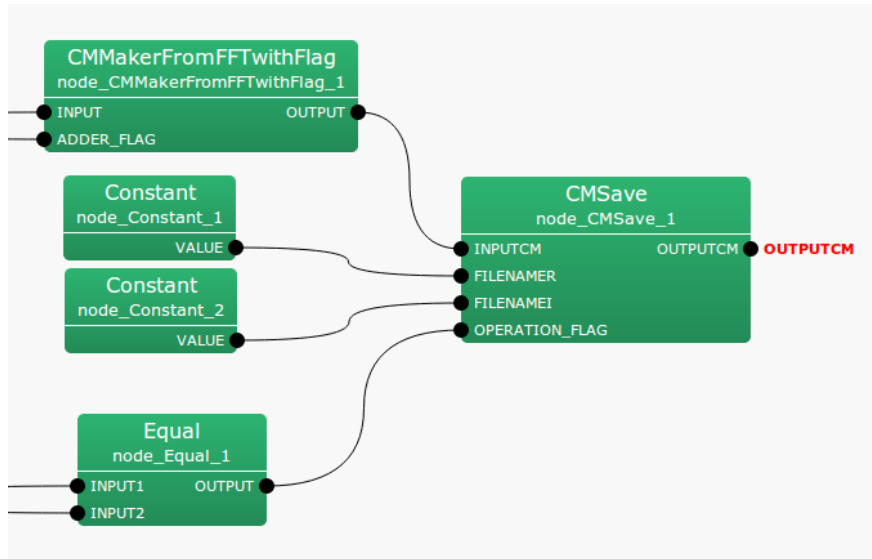


Figure 6.18: Network Example using CMSave

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

- **Before Version 2.0**

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.

- **After Version 2.1**

Save a set of correlation matrices of a M -th order complex square array for each frequency bin as a zip file.

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. ?? 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).

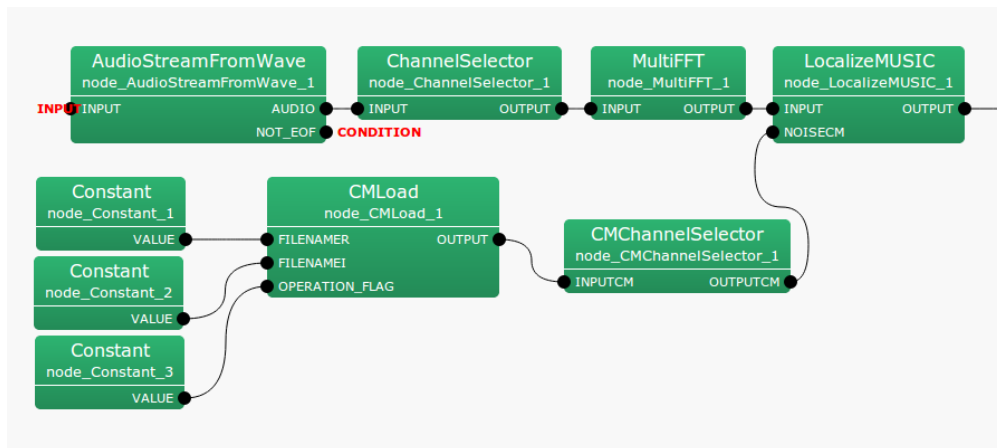


Figure 6.19: Network Example using CMChannelSelector

I/O and property setting of the node

Table 6.21: 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. ?? 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.

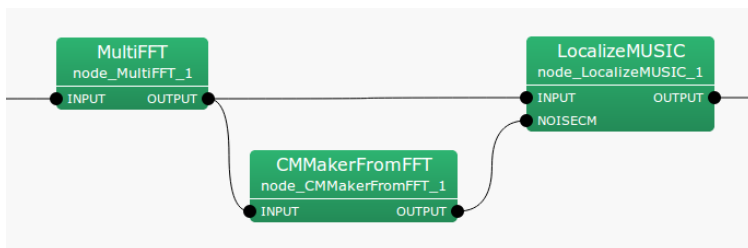


Figure 6.20: Network Example using CMMakerFromFFT

I/O and property setting of the node

Table 6.22: 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
WINDOW_TYPE	string	FUTURE		Frame selection to normalize CM
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).

OPERATION_FLAG : `bool` type. This outputs `true` if the correlation matrix from OUTPUT is updated. Otherwise `false`. This port is invisible by the default. To visualize it, see Fig. ?? in LocalizeMUSIC .

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.

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 MIDDLE, 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.

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.

$$\mathbf{X}(\omega, f) = [X_1(\omega, f), X_2(\omega, f), X_3(\omega, f), \dots, X_M(\omega, f)]^T \quad (6.1)$$

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 $\mathbf{X}(\omega, f)$ can be defined as follows for every frequency and frame.

$$\mathbf{R}(\omega, f) = \mathbf{X}(\omega, f) \mathbf{X}^*(\omega, f) \quad (6.2)$$

Here, $()^*$ denotes the complex conjugate transpose operator. There is no problem if this $\mathbf{R}(\omega, f)$ 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.

$$\mathbf{R}'(\omega, f) = \frac{1}{\text{WINDOW}} \sum_{i=W_i}^{W_f} \mathbf{R}(\omega, f + i) \quad (6.3)$$

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

$\mathbf{R}'(\omega, f)$ 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. ?? 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.

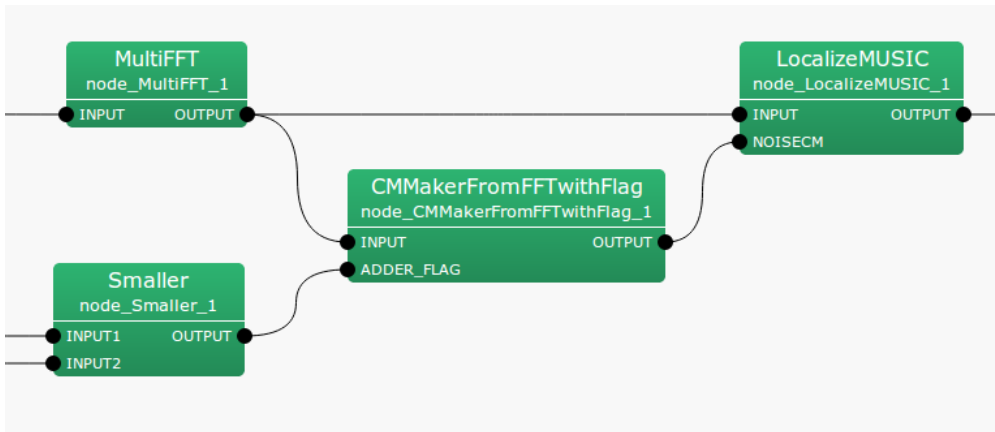


Figure 6.21: Network Example using CMMakerFromFFTwithFlag

I/O and property setting of the node

Input

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

Table 6.23: 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
WINDOW_TYPE	string	FUTURE		Frame selection to normalize CM
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

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).

OPERATION_FLAG : bool type. This outputs true if the correlation matrix from OUTPUT is updated. Otherwise false. This port is invisible by the default. To visualize it, see Fig. ?? in LocalizeMUSIC .

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=WINDOW_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=WINDOW_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=WINDOW_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.

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 MIDDLEW, 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.

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 correlation 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.

$$\mathbf{R}'(\omega, f) = \frac{1}{\text{PERIOD}} \sum_{i=W_i}^{W_f} \mathbf{R}(\omega, f + i) \quad (6.4)$$

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

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.

$$\mathbf{R}'(\omega) = \mathbf{O}$$

$$\text{PERIOD} = 0$$

where $\mathbf{O} \in \mathbb{C}^{(NFFT/2+1) \times M \times M}$ represents the zero matrix.

B) When ADDER_FLAG is 1 (or true)

- Add the correlation matrix.

$$\mathbf{R}'(\omega) = \mathbf{R}'(\omega) + \mathbf{R}(\omega, f + i)$$

$$\text{PERIOD} = \text{PERIOD} + 1$$

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.

$$\mathbf{R}_{out}(\omega, f) = \frac{1}{\text{PERIOD}} \mathbf{R}'(\omega)$$

D) When ADDER_FLAG is 0 (or false)

- Keep the correlation matrix generated in the end.

$$\mathbf{R}_{out}(\omega, f)$$

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

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

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. ?? 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.

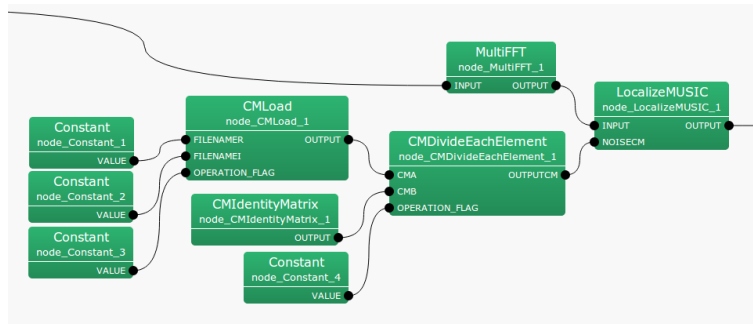


Figure 6.22: Network Example using CMDivideEachElement

I/O and property setting of the node

Table 6.24: 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 j = 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. ?? 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.

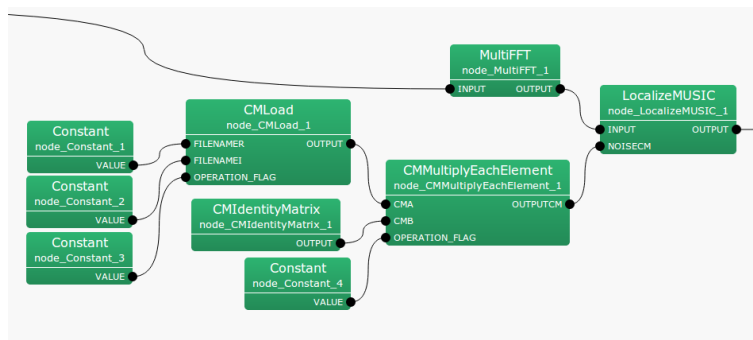


Figure 6.23: Network Example using CMMultiplyEachElement

I/O and property setting of the node

Table 6.25: 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 j = 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 CMConjEachElement

Module Overview

Performs conjugate of a 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 of taking conjugate of each component.

Typical Examples

Figure. ?? shows the usage example of CMConjEachElement node. The 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).

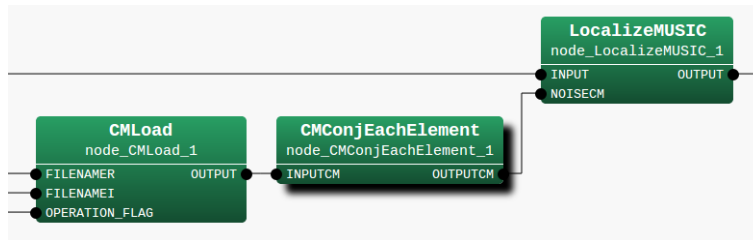


Figure 6.24: Network Example using CMConjEachElement

I/O and property setting of the node

Input

INPUTCM : `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).

Output

OUTPUTCM : `Matrix<complex<float> >` type. The correlation matrix after taking conjugate of INPUTCM is output.

Parameter

No parameters.

Module Description

Performs conjugate of a 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.

```
calculate{
    FOR i = 1 to k
        FOR j = 1 to M
            FOR i = 1 to M
                OUTPUTCM[i][j][k] = conj(INPUTCM[i][j][k])
            ENDFOR
        ENDFOR
    ENDFOR
}
```

6.2.9 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. ?? 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.

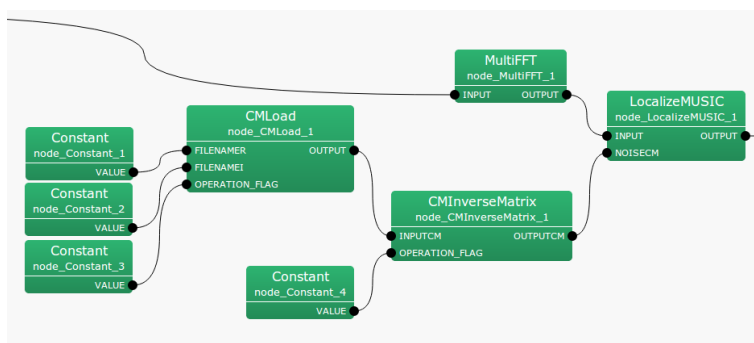


Figure 6.25: Network Example using CMInverseMatrix

I/O and property setting of the node

Table 6.26: 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.

6.2.10 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. ?? shows the usage example of CMMultiplyMatrix node.

CMA input terminal is connected to the correlation matrix calculated from CMMakerFromFFT , 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 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.

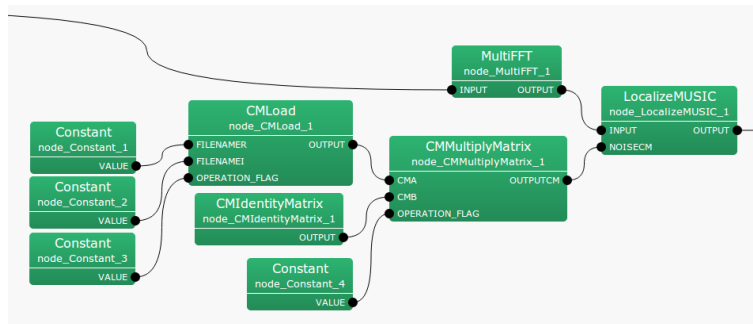


Figure 6.26: Network Example using CMMultiplyMatrix

I/O and property setting of the node

Table 6.27: 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 corre-

sponding 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.

6.2.11 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. ?? 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.

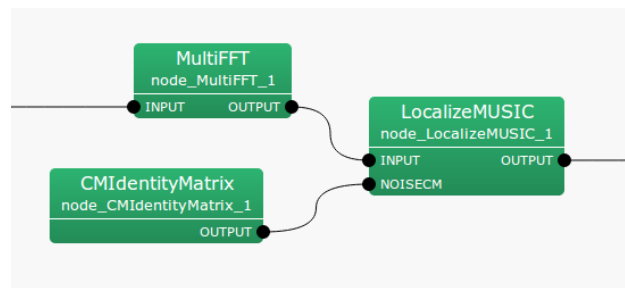


Figure 6.27: Network Example using CMIdentityMatrix

I/O and property setting of the node

Table 6.28: 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

OUTPUT : 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.

6.2.12 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 ?? and ?? show a connection example. This network continuously displays constant localization results.

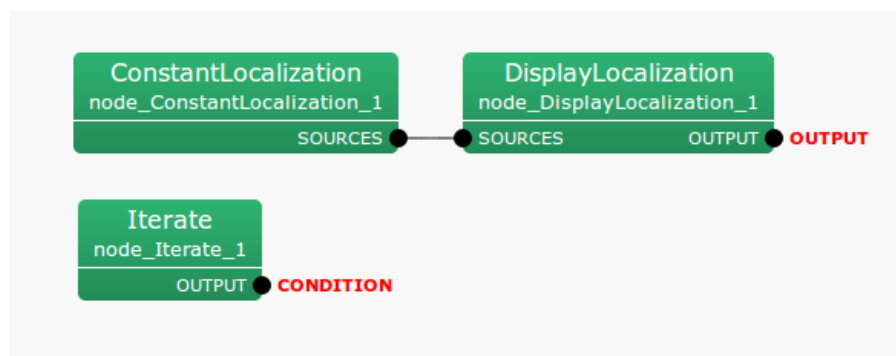


Figure 6.28: Connection example of ConstantLocalization 1

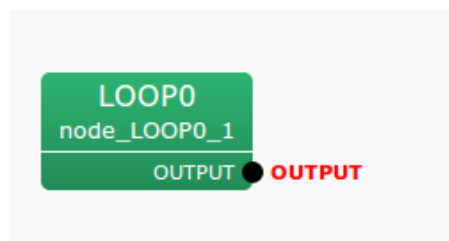


Figure 6.29: Connection example of ConstantLocalization 2

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

Table 6.29: Parameter list of `ConstantLocalization`

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

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

ELEVATIONS : `Vector< float >` type. Elevation angle (up or down) of the direction that the sound source comes 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 > $a_1 \dots a_N$ >`

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

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.

$$x_i = \cos(a_i\pi/180) \cos(e_i\pi/180) \quad (6.5)$$

$$y_i = \sin(a_i\pi/180) \cos(e_i\pi/180) \quad (6.6)$$

$$z_i = \sin(e_i\pi/180) \quad (6.7)$$

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.

6.2.13 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 ?? continuously displays fixed localization results from `ConstantLocalization`.

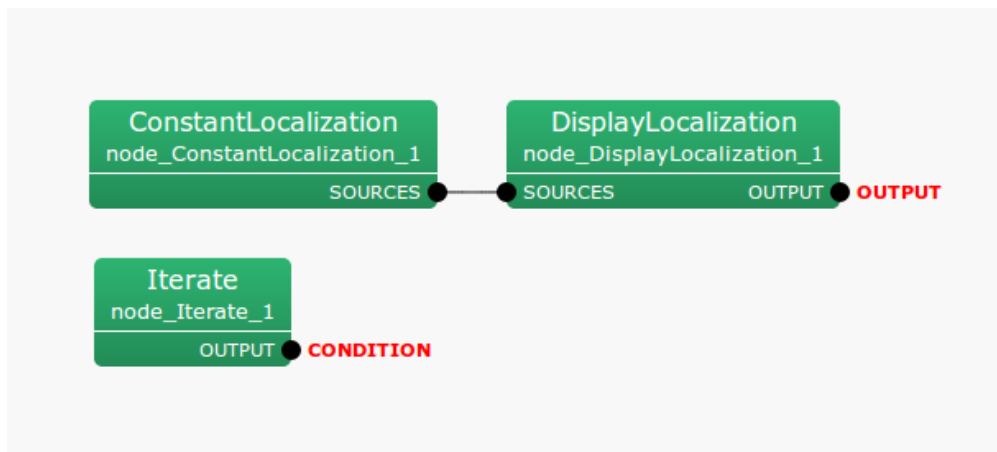


Figure 6.30: Connection example of DisplayLocalization

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

Table 6.30: Parameter list of DisplayLocalization

Parameter name	Type	Default value	Unit	Description
WINDOW_NAME	string	Source Location	Frame	Name of the GUI window that displays
WINDOW_LENGTH	int	1000		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

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].

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.

6.2.14 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

When to use

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 ?? shows a typical connection example.

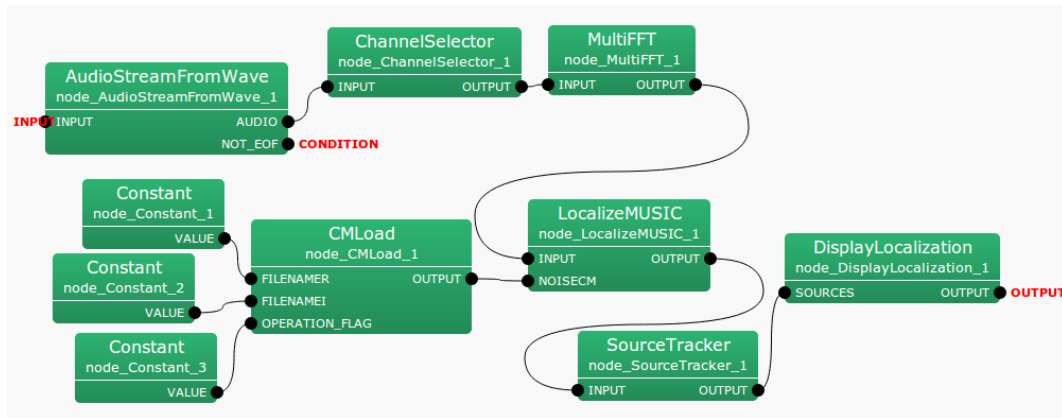


Figure 6.31: Connection example of LocalizeMUSIC

Input-output and property of the node

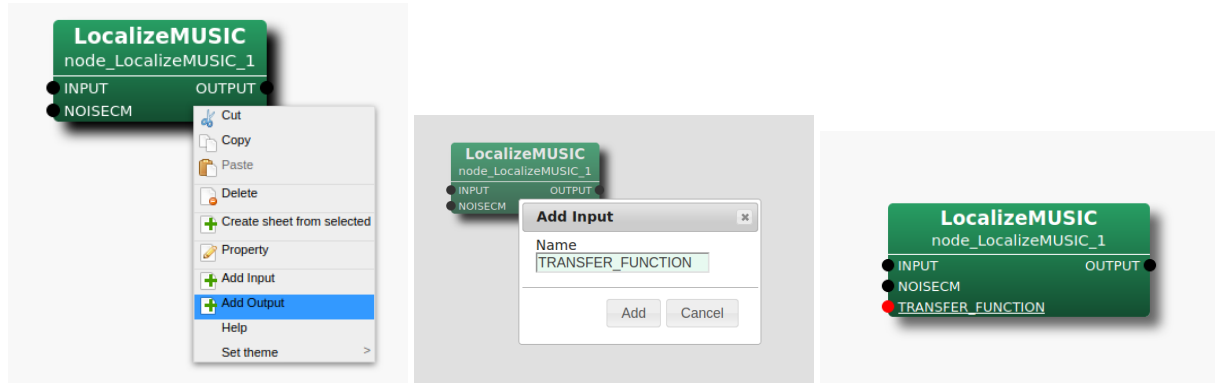
Input

INPUT : $\text{Matrix}\langle \text{complex}\langle \text{float} \rangle \rangle$, Complex frequency representation of input signals with size $M \times (NFFT/2 + 1)$.

NOISECM : $\text{Matrix}\langle \text{complex}\langle \text{float} \rangle \rangle$ 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 $\text{Matrix}\langle \text{complex}\langle \text{float} \rangle \rangle$ 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.

TRANSFER_FUNCTION : `TransferFunction` type. Instead of loading the transfer function file, this node can also receive the transfer function output from an `EstimateTF` node and others through the input terminal of `TransferFunction` type. In that case, the parameter `TF_INPUT.TYPE` is set to `ONLINE`. This input terminal is not displayed by default.

Refer to Figure ?? for the addition method of hidden input.



Step 1: Right-click LocalizeMUSIC and click Add Input. Step 2: TRANSFER_FUNCTION in the input, then, click Add. Step 3: The TRANSFER_FUNCTION input terminal is added to the node.

Figure 6.32: Usage example of hidden inputs : Display of TRANSFER_FUNCTION terminal

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.

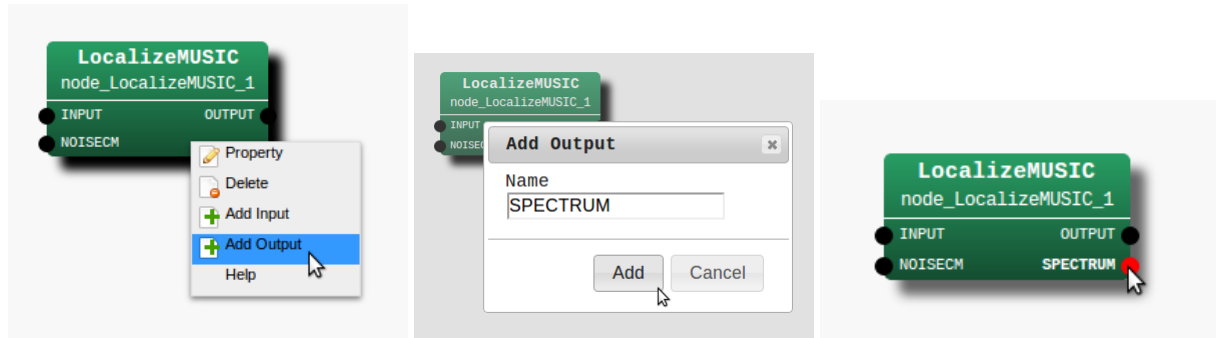
SPECTRUM : `Vector<float>` type. Power of the MUSIC spectrum for every direction. The output is equivalent to $\bar{P}(\theta)$ in Eq. (??). In case of three dimensional sound source localization, θ is a three dimensional vector, and $\bar{P}(\theta)$ becomes three dimensional data. Please refer to node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

Refer to Figure ?? for the addition method of hidden 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



Step 1: Right-click LocalizeMUSIC and click Add Output. Step 2: Enter SPECTRUM in the input, then, click Add. Step 3: The SPECTRUM output terminal is added to the node.

Figure 6.33: Usage example of hidden outputs : Display of SPECTRUM terminal

Table 6.31: 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 (<i>NFFT</i>)
SAMPLING_RATE	int	16000	[Hz]	Sampling rate
TF_INPUT_TYPE	string	FILE		Selection of TF Input
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
ENABLE_INTERPOLATION	bool	false		Enable interpolation of TFs
INTERPOLATION_TYPE	string	FTDLI		Selection of TF interpolation
HEIGHT_RESOLUTION	float	1.0	[deg]	Interval of elevation
AZIMUTH_RESOLUTION	float	1.0	[deg]	Interval of azimuth
RANGE_RESOLUTION	float	1.0	[m]	Interval of radius
PEAK_SEARCH_ALGORITHM	string	LOCAL_MAXIMUM		Peak search algorithm
MAXNUM_OUT_PEAKS	int	-1		Max. num. of output peaks
DEBUG	bool	false		ON/OFF of debug output

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.

TF_INPUT_TYPE : string type. 'FILE' is the default. When 'FILE' is selected, the transfer function file with the name specified by A_MATRIX is used, and when 'ONLINE' is selected, the input of TRANSFER_FUNCTION is used as the transfer function. An error occurs if the TRANSFER_FUNCTION input is connected when 'FILE' is selected or if the input is not connected when 'ONLINE' is selected.

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 MIDDLEW, 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 $0 \leq \omega_{min} \leq \text{SAMPLING_RATE}/2$.

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 $\omega_{min} < \omega_{max} \leq \text{SAMPLING_RATE}/2$.

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.

ENABLE_INTERPOLATION : bool type. False is the default value. In case of true, the spatial resolution of sound source localization can be improved by the interpolation of transfer functions specified by `A_MATRIX`. `INTERPOLATION_TYPE` specifies the method for the interpolation. The new resolution after the interpolation can be specified by `HEIGHT_RESOLUTION`, `AZIMUTH_RESOLUTION`, and `RANGE_RESOLUTION`, respectively.

INTERPOLATION_TYPE : string type. FTDLI is the default value. This specifies the interpolation method for transfer functions.

HEIGHT_RESOLUTION : float type. 1.0[deg] is the default value. This specifies the interval of elevation for the transfer function interpolation.

AZIMUTH_RESOLUTION : float type. 1.0[deg] is the default value. This specifies the interval of azimuth for the transfer function interpolation.

RANGE_RESOLUTION : float type. 1.0[m] is the default value. This specifies the interval of radius for the transfer function interpolation.

PEAK_SEARCH_ALGORITHM : string type. LOCAL_MAXIMUM is the default. This selects the algorithm for searching peaks from the MUSIC spectrum. If LOCAL_MAXIMUM, peaks are defined as the maximum point among all adjacent points (local maximum). If HILL_CLIMBING, peaks are firstly searched only on the horizontal plane (azimuth search) and then searched in the vertical plane of detected azimuth (elevation search).

MAXNUM_OUT_PEAKS : int type. -1 is the default. This parameter defines the maximum number of output peaks of MUSIC spectrum (sound sources). If 0, all the peaks are output. If `MAXNUM_OUT_PEAKS` > 0, `MAXNUM_OUT_PEAKS` peaks are output in order of their power. If -1, `MAXNUM_OUT_PEAKS` = `NUM.SOURCE`.

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. (??) is outputted as is. Next, "MUSIC spectrum:" is outputted after a line feed, and the value of $\bar{P}(\theta)$ of Eq. (??) 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.

$$\mathbf{H}(\theta, \omega) = [h_1(\theta, \omega), \dots, h_M(\theta, \omega)] \quad (6.8)$$

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.

$$\mathbf{X}(\omega, f) = [X_1(\omega, f), X_2(\omega, f), X_3(\omega, f), \dots, X_M(\omega, f)]^T, \quad (6.9)$$

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 $\mathbf{X}(\omega, f)$ can be defined as follows for every frame and for every frequency.

$$\mathbf{R}(\omega, f) = \mathbf{X}(\omega, f) \mathbf{X}^*(\omega, f) \quad (6.10)$$

where $()^*$ represents the conjugate transpose operator. If this $\mathbf{R}(\omega, f)$ 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.

$$\mathbf{R}'(\omega, f) = \frac{1}{\text{WINDOW}} \sum_{i=W_i}^{W_f} \mathbf{R}(\omega, f+i) \quad (6.11)$$

The frames used for the averaging can be changed by WINDOW.TYPE. If WINDOW.TYPE=FUTURE, $W_i = 0$ and $W_f = \text{WINDOW} - 1$. If WINDOW.TYPE=MIDDLE, $W_i = -\text{WINDOW}/2$ and $W_f = \text{WINDOW}/2 + \text{WINDOW}\%2 - 1$. If WINDOW.TYPE=PAST, $W_i = -\text{WINDOW} + 1$ 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 $\mathbf{R}'(\omega, f)$ found in the Eq. (??) 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.

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

where $\mathbf{E}(\omega, f)$ represents the matrix $\mathbf{E}(\omega, f) = [e_1(\omega, f), e_2(\omega, f), \dots, e_M(\omega, f)]$ which consists of singular vectors which perpendicularly intersect each other, and $\mathbf{\Lambda}(\omega)$ represents the diagonals matrix using the eigenvalue corresponding to individual eigenvectors as the diagonal component. In addition, the diagonal component of $\mathbf{\Lambda}(\omega)$, $[\lambda_1(\omega), \lambda_2(\omega), \dots, \lambda_M(\omega)]$ is considered to have been sorted in descending order.

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

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

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

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

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

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

Since the eigenvalue (or singular-value) corresponding to eigen vector space $\mathbf{E}(\omega, f)$ 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 $[e_1(\omega), \dots, e_{N_s}(\omega)]$ corresponds to the sound, are eigen vector $[e_{N_s+1}(\omega), \dots, e_M(\omega)]$ 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.

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

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.

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

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 ???. 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 ??, 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 ??.
- 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 ??.
- 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 ???. 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. (??) 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 N_e , N_d , and N_r 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

```

Search of sound :

Next, the peak is detected from the range in θ_{min} to θ_{max} for $\bar{P}(\theta, f)$ of Eq. (??), and the power of the MUSIC spectrum corresponding to DoA for the top MAXNUM_OUT_PEAKEs 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 MAXNUM_OUT_PEAKEs. The algorithm for searching peaks can be selected by PEAK_SEARCH_ALGORITHM whether it is the local maximum searching or the hill-climbing method. In LocalizeMUSIC, θ_{min} 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.

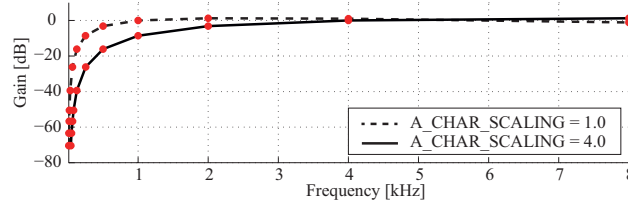


Figure 6.34: Frequency characteristic of characteristic weight when considering as SPECTRUM_WEIGHT_TYPE=A.Characteristic

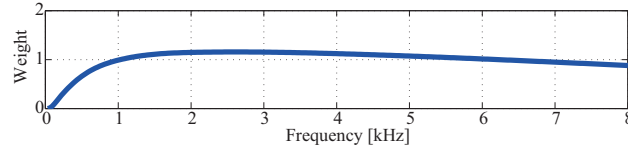


Figure 6.35: $W_{\omega}(\omega, f)$ in case of SPECTRUM_WEIGHT_TYPE=A.Characteristic and A_CHAR_SCALING=1

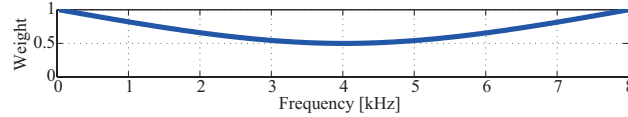


Figure 6.36: $W_{\omega}(\omega, f)$ in case of SPECTRUM_WEIGHT_TYPE=Manual Spline

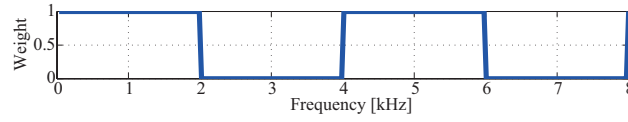


Figure 6.37: $W_{\omega}(\omega, f)$ in case of SPECTRUM_WEIGHT_TYPE=Manual_Square

Discussion :

Finally, we describe the effect that whitening (noise suppression) has on MUSIC spectrum in Eq. (??) 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 ??(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. (??). 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 ??(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 ??(a).

Figure ??(c) is the MUSIC spectrum when generating $\mathbf{K}(\omega, f)$ from the information on Figure ??(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 $\mathbf{K}(\omega, f)$ 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.

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.

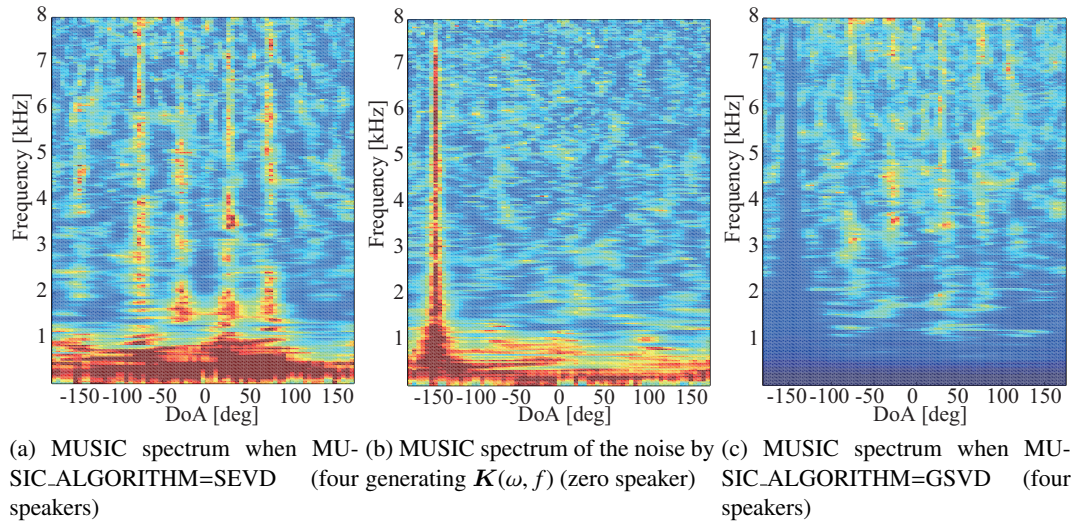


Figure 6.38: Comparison of MUSIC spectrum

- (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.

6.2.15 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 ?? , ?? show a network that reads and displays saved localization results.

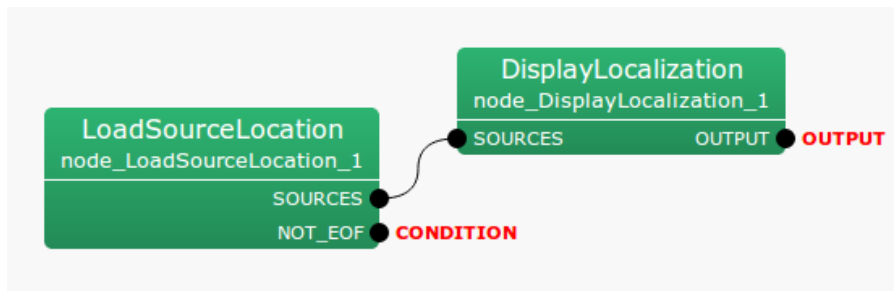


Figure 6.39: Connection example of LoadSourceLocation 1

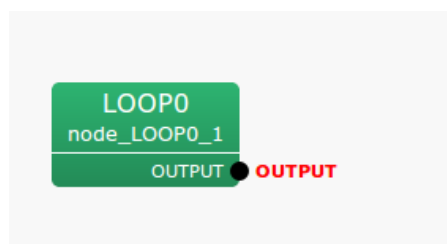


Figure 6.40: Connection example of LoadSourceLocation 2

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 6.32: Parameter list of `LoadSourceLocation`

Parameter name	Type	Default value	Unit	Description
<code>FILENAME</code>	<code>string</code>			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.

6.2.16 NormalizeMUSIC

Outline of the node

This module normalizes the MUSIC spectrum calculated by LocalizeMUSIC into the range [0 1] so as to stabilize the sound source detection by the thresholding in SourceTracker .

Necessary file

None.

Usage

When to use

Sound source localization and detection functions in HARK are achieved by applying a threshold on the MUSIC spectra for each discretized time and direction¹. This module helps you to determine an appropriate threshold in SourceTracker by normalizing the MUSIC spectra calculated by LocalizeMUSIC instead of by directly applying a threshold on the MUSIC spectra. By using this module and setting the threshold in SourceTracker at a value close to 1, e.g. 0.95, the source localization performance becomes stable.

This module internally calculates the normalization parameters and normalizes the MUSIC spectrum values between [0 1]. The estimation of normalization parameters are carried out at a certain interval using the observed MUSIC spectrum frames. In this estimation, the distributions corresponding to the presence or absence of sound sources. The MUSIC spectrum is normalized for each time frame by a particle filter using the estimated normalization parameters.

Typical connection

Figure ?? shows a typical usage of NormalizeMUSIC module. In Fig. ??, the outputs of LocalizeMUSIC (OUTPUT: source locations and corresponding MUSIC spectrum values, SPECTRUM: MUSIC spectrum for each direction) are connected to the corresponding inputs of NormalizeMUSIC module. SOURCES_OUT of NormalizeMUSIC can be connected with SourceTracker in the same way as the output of LocalizeMUSIC .

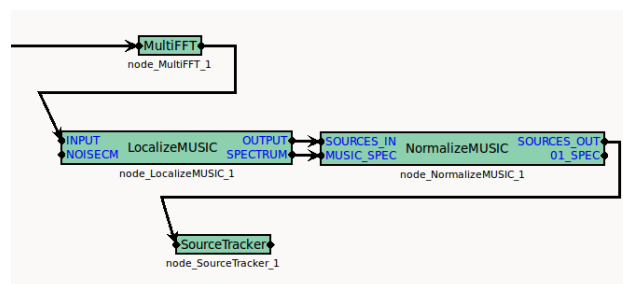


Figure 6.41: Typical usage of NormalizeMUSIC

Input-output and property of the node

Inputs

SOURCES_IN : Vector<ObjectRef> type. Connected with OUTPUT node of LocalizeMUSIC . This contains source information (location and MUSIC spectrum value).

¹For instance, 10 ms time resolution and 5° directional resolution

Table 6.33: Parameter list of NormalizeMUSIC

Parameter name	Type	Default value	Unit	Description
SHOW_ALL_PARAMETERS	bool	false		Show or hide the parameters except INITIAL_THRESHOLD.
INITIAL_THRESHOLD	float	30		An approximate boundary on MUSIC spectra where a sound source exists or not.
ACTIVE_PROP	float	0.05		A threshold whether to update the normalization parameters.
UPDATE_INTERVAL	int	10		The interval of updating the normalization parameters, and the number of frames used for the process.
PRIOR_WEIGHT	float	0.02		Regularization parameter for the calculation of the normalization parameters.
MAX_SOURCE_NUM	int	2		The maximum number of sources that each particle can assume in the particle filter below.
PARTICLE_NUM	int	100		The number of particles used by the particle filter that normalizes the MUSIC spectra.
LOCAL_PEAK_MARGIN	float	0		A margin of MUSIC spectrum between adjacent directions for the detection of local peaks in the MUSIC spectrum.

MUSIC_SPEC : Vector<float> type. Connected with SPECTRUM node of LocalizeMUSIC . This contains the MUSIC spectrum value for each direction.

Outputs

SOURCES_OUT : Vector<ObjectRef> type. This contains the same source information as SOURCES_IN except that the MUSIC spectrum value of each source is replaced with the normalized value in [0 1].

01_SPEC : Vector<float> type. Normalized values of the input node MUSIC_SPEC.

Parameters

SHOW_ALL_PARAMETERS : bool type, set false by default. If set true, all parameters are displayed. In many cases, parameters other than INITIAL_THRESHOLD require no modification from the default values.

INITIAL_THRESHOLD : float type, set 30 by default. This value is used in two ways: (1) used as a prior belief of the boundary on the MUSIC spectrum between the presence and absence of sound source, and (2) used to determine whether to carry out the first estimation of the normalization parameters.

ACTIVE_PROP : float type, set 0.05 by default. This is a threshold to determine whether to update the normalization parameters when UPDATE_INTERVAL frames of MUSIC spectra are accumulated. Let T be the number of frames of MUSIC spectra and D be the number of directions, that is TD MUSIC values in total, and θ denote ACTIVE_PROP. If we have more time-direction points than θTD in the MUSIC spectra with a larger value than INITIAL_THRESHOLD, the normalization parameters are updated.

UPDATE_INTERVAL : int type, set 10 by default. The number of frames used to update the normalization parameters. This value controls the interval of the updates. By default, HARK system is configured as follows. The multichannel audio signal is sampled at 16000 (Hz). In MultiFFT , the short-time Fourier transform is carried out with 160 (pt) shift, that is 0.01 (sec). LocalizeMUSIC module calculates the MUSIC spectrum every 50

frames, i.e., 0.5 (sec). Therefore, if UPDATE_INTERVAL is 10, the normalization parameters are calculated with the MUSIC spectra for 5 (sec).

PRIOR_WEIGHT : float type. This is a regularization parameter to stabilize the estimation of the normalization parameters. Specific explanation is provided in **Technical details**, and recommended configuration is given in **Troubleshooting** below, respectively.

MAX_SOURCE_NUM : int type. The maximum source number that each particle can hypothesize in the particle filter for the MUSIC spectrum normalization. Regardless of the actual number of sound sources, setting this parameter at 1 – 3 produces good results.

PARTICLE_NUM : int type, set 100 by default. The number of particle used in a particle filter to normalize the MUSIC spectrum in each time frame. Empirically, 100 is a sufficiently large number when we have 72 directions (5° resolution on the azimuth plane). If you handle more directions (for example, multiple elevations like 72×30 directions), more particles may be necessary.

LOCAL_PEAK_MARGIN : float type. In the particle filter for the MUSIC spectrum normalization, each particle can hypothesize an existence of sound sources at local peaks of the observed MUSIC spectrum. This value is the margin allowed to take the local peaks. Setting a too large value for this parameter may risk false detections of sound sources.

Details

Technical details: The interested readers can refer to the paper in ?? Reference below. In the paper, the normalization parameter estimation corresponds to the estimation of the posterior distribution of VB-HMM (variational Bayes hidden Markov model), and the MUSIC spectrum normalization corresponds to the online inference using a particle filter.

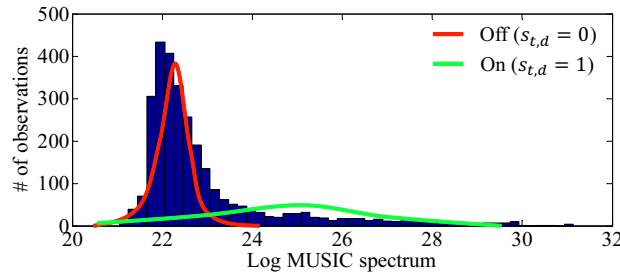


Figure 6.42: Estimation of normalization parameters from observed MUSIC spectra

Roughly speaking, this module fits to Gaussian distributions, sound presence in green in Fig. ?? and sound absence in red, to the observed MUSIC spectra (blue histogram in Fig. ??).

Here we provide how the variables used in the paper correspond to the parameters of this module. The input of the VB-HMM for the normalization parameter estimation is MUSIC spectra with T time frames and D directions. T is specified by UPDATE_INTERVAL and D is determined in LocalizeMUSIC module. VB-HMM uses some hyperparameters $\alpha_0, \beta_0, m_0, a_0$, and b_0 . $\alpha_0 a_0$, and b_0 are set at 1 in the same way as the reference. m_0 is the INITIAL_THRESHOLD, and β_0 is set at $TD\varepsilon$ provided that PRIOR_WEIGHT is ε .

Flowchart: Figure ?? illustrates the flowchart of NormalizeMUSIC module. Blue lines are a flow of source information from SOURCES_IN to SOURCES_OUT. Red solid lines are the observed MUSIC spectrum and red dotted lines are the normalized MUSIC spectrum. Two processes are carried out for each time frame: (1) Normalization of MUSIC spectrum using the latest normalization parameters (middle column). (2) Replacement of the MUSIC spectrum value in the source information with the normalized value (left column). SourceTracker module that follows NormalizeMUSIC module refers to the MUSIC spectrum value in the source information to detect and localize the sound sources.

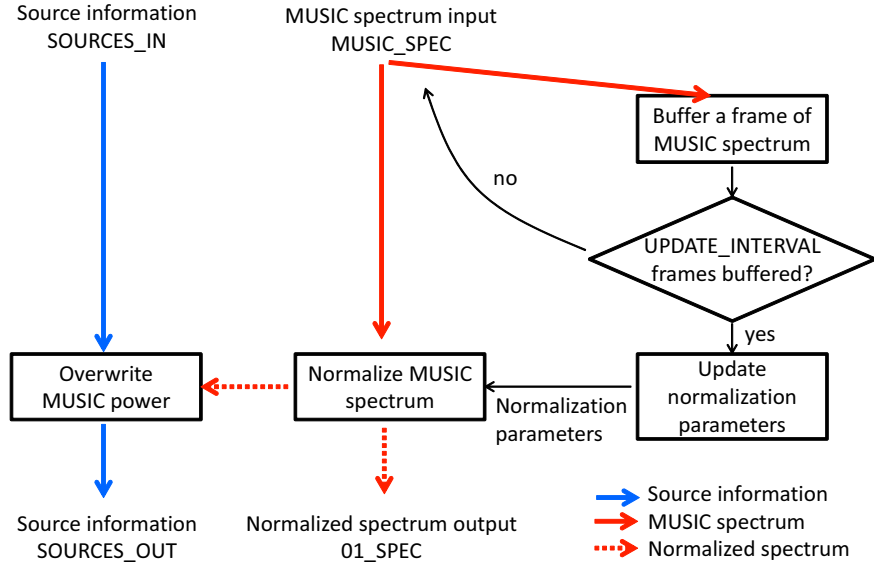


Figure 6.43: Flowchart of NormalizeMUSIC

The right column is the update of normalization parameters. When the following two conditions are satisfied, the normalization parameters are updated.

1. UPDATE_INTERVAL frames of MUSIC spectra are accumulated, and
2. the proportion of time-direction points with sound existence exceeds ACTIVE_PROP.

When 1. is satisfied, whether condition 2. is satisfied is examined using the observed MUSIC spectra $x_{t,d}$ with T frames and D directions. Let θ be ACTIVE_PROP.

First update: In case no normalization parameter estimation has been carried out since the program was started, the first estimation of the normalization parameters are carried out if the number of time-direction points $x_{t,d}$ more than INITIAL_THRESHOLD exceeds θTD .

Updates afterwards: If the summation of the normalized MUSIC spectrum values exceeds θTD , the normalization parameters are updated using the latest observation of MUSIC spectra.

Troubleshooting: We outline how to configure the parameters when something is wrong with the localization and detection of sound sources.

Visualize MUSIC spectrum: Verify that the values of MUSIC spectrum are low in the absence of sound sources and high in the presence. You can obtain the MUSIC spectrum values by setting DEBUG parameter at true in LocalizeMUSIC module. Then, the values are streamed into the standard output. You can visualize these values by using an appropriate tool such as python + matplotlib or matlab. You can find a similar topic in “3.3 Sound source localization fails” of HARK cookbook. If there seems to be something wrong with the calculation of MUSIC spectra, see “8 Sound source localization” of HARK cookbook. The calculation of MUSIC spectra is sometimes stabilized by setting NUM_SOURCE parameter at 1 and LOWER_BOUND_FREQUENCY parameter at 1000 (Hz) of LocalizeMUSIC.

Try first: Set ACTIVE_PROP and PRIOR_WEIGHT at 0. Try a low value for INITIAL_THRESHOLD (for example 20). Set THRESH parameter at 0.95 of SourceTracker module connected with SOURCES_OUT of NormalizeMUSIC module. If nothing is localized, the calculation of MUSIC spectra is likely to be wrong. If too many sounds are detected, adjust INITIAL_THRESHOLD by increasing at 5 intervals (e.g., $20 \rightarrow 25 \rightarrow 30$).

Too many false detections: Possible reasons are (1) MUSIC spectrum value is high in the absence of a sound source and (2) the mean value of the distribution for sound presence in Fig. ?? is too low. In case of (1), we can try the MUSIC algorithm that uses noise correlation matrix (see LocalizeMUSIC for details), or increasing LOWER_BOUND_FREQUENCY of LocalizeMUSIC to 800–1000 (Hz). In case of (2), we can try increasing INITIAL_THRESHOLD (for example, from 30 to 35), or PRIOR_WEIGHT (for example, set around 0.05–0.1).

Nothing is detected: Possible reasons are (1) MUSIC spectrum value is low even in the presence of a sound source and (2) INITIAL_THRESHOLD is too large. In case of (1), we have to adjust the parameters of LocalizeMUSIC . Set NUM_SOURCES at the actual number of sources, or try a larger value like $M - 1$ or $M - 2$, where M is the number of microphones. Specify LOWER_BOUND_FREQUENCY and UPPER_BOUND_FREQUENCY to meet the frequency range of the target sound. In case of (2), decrease INITIAL_THRESHOLD.

Reference

- (1) Takuma Otsuka, Kazuhiro Nakadai, Tetsuya Ogata, Hiroshi G. Okuno: Bayesian Extension of MUSIC for Sound Source Localization and Tracking, *Proceedings of International Conference on Spoken Language Processing (Interspeech 2011)*, pp.3109-3112. ²

²<http://winnie.kuis.kyoto-u.ac.jp/members/okuno/Public/Interspeech2011-Otsuka.pdf>

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

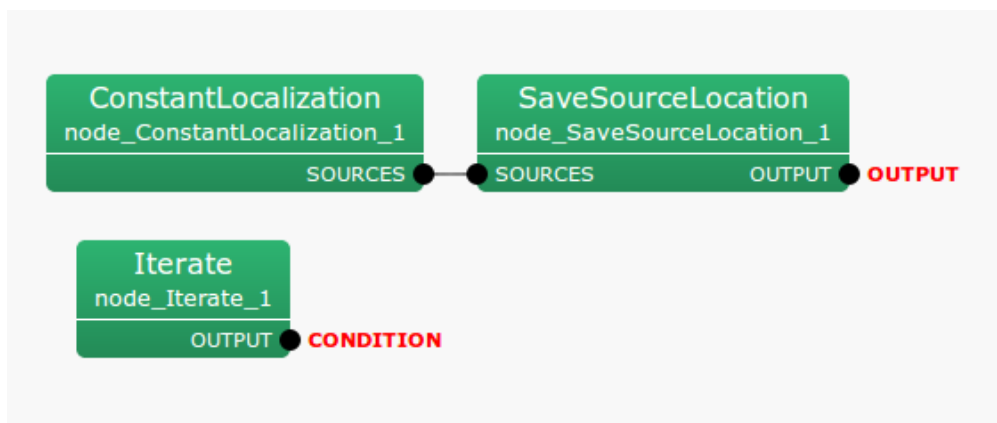


Figure 6.44: Connection example of SaveSourceLocation

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 6.34: 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.

6.2.18 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 ?? 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.

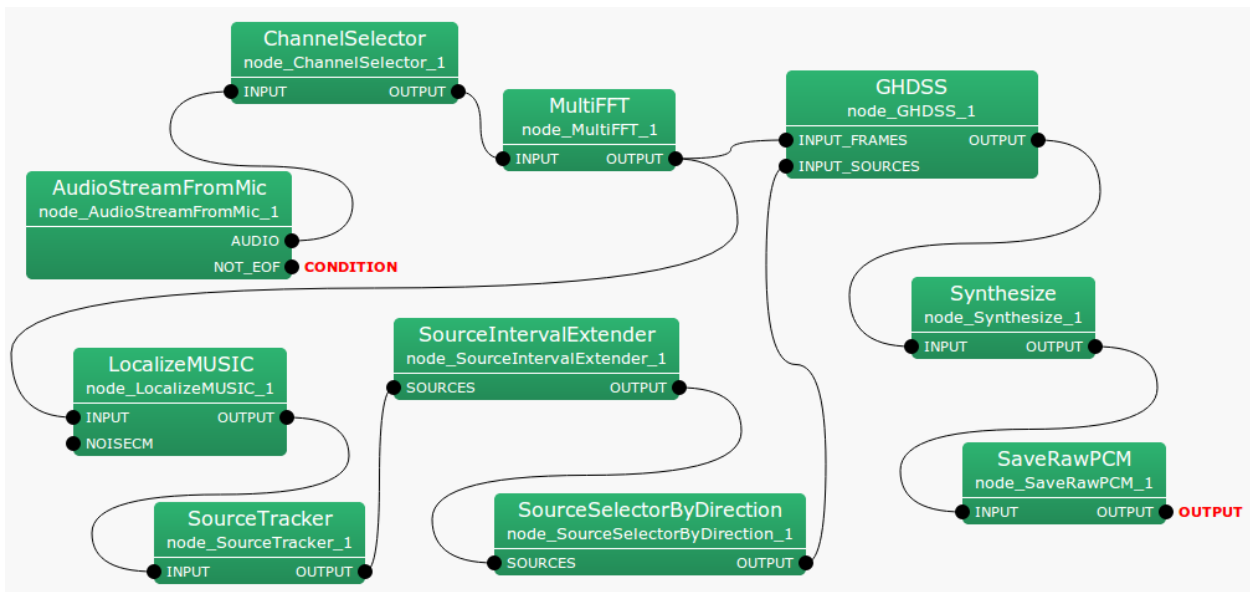


Figure 6.45: Connection example of SourceIntervalExtender

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 6.35: 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 ???. 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 ??)

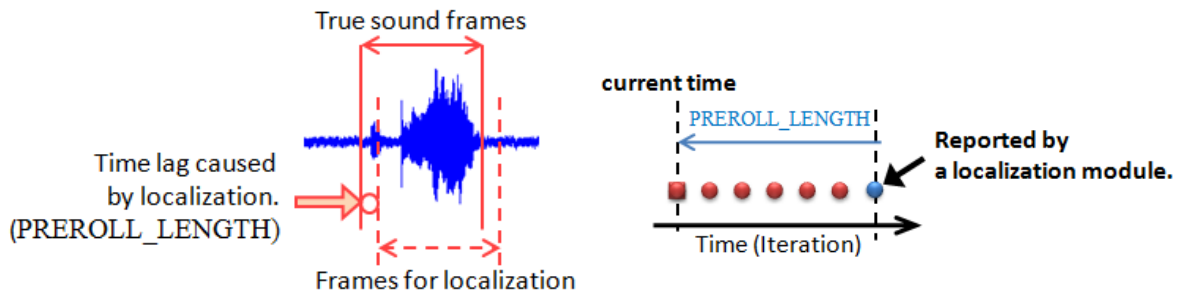


Figure 6.47: Operation of SourceIntervalExtender

Figure 6.46: Necessity to output a source location result sooner than usual

6.2.19 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 ?? 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

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

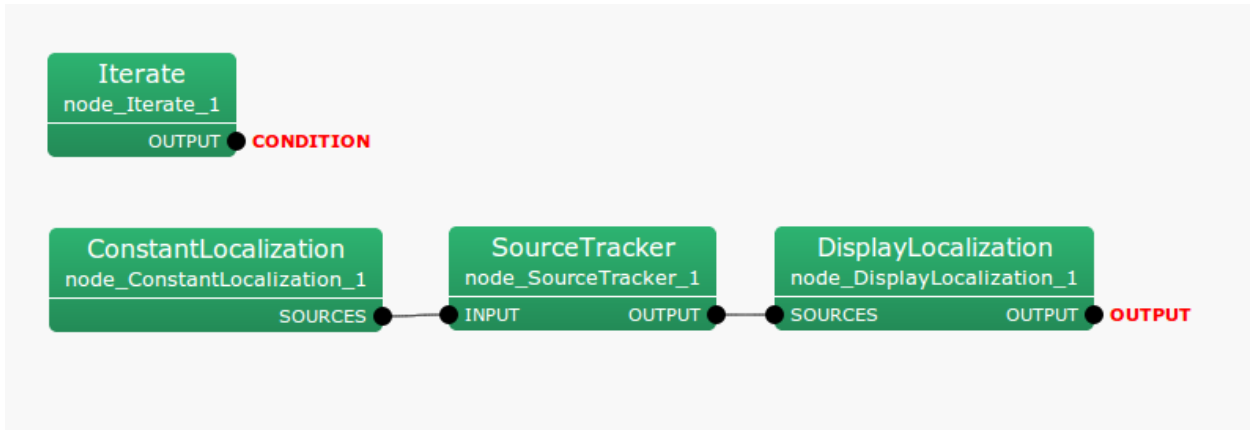


Figure 6.48: Connection example of SourceTracker

Table 6.36: Parameter list of SourceTracker

Parameter name	Type	Default value	Unit	Description
THRESH	float			To be ignored if the MUSIC power of the sound source is smaller than THRESH.
PAUSE_LENGTH	float	800	[frame*10]	Length when assuming that the localized sound continues.
COMPARE_MODE	string	DEG	DEG or TFINDEX	Method for comparing inter-source distance. If the value is DEG, they are calculated using triangular functions. If the value is TFINDEX, they are calculated using index comparison.
MIN_SRC_INTERVAL	float	20	[deg]	Threshold value of angular difference for judging that the sound source is same as another. (Valid if COMPARE_MODE = DEG)
MIN_TFINDEX_INTERVAL	int	3		Threshold value of index difference for judging the sound source is same as another. (Valid if COMPARE_MODE = TFINDEX)
MIN_ID	int	0		
DEBUG	bool	false		

Parameter

THRESH : float type. This parameter judges by the MUSIC power whether the source localization result is noise to be ignored. The result is considered noise if the MUSIC 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.

COMPARE_MODE : ~type. If the value is DEG, comparison is done using triangular functions. If the value is

TFINDEX, comparison is done by index comparison. Since triangular function is heavier, TFINDEX reduces the computation time. TFINDEX is useful only if the index difference and angular difference are equivalent, i.e., transfer functions are recorded in order.

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. This parameter is valid if COMPARE_MODE = DEG.

MIN_TFINDEX_INTERVAL : int type. Almost the same as MIN_SRC_INTERVAL. The difference is that the compared values are not angles but indexes. This parameter is valid if COMPARE_MODE = TFINDEX.

MIN_ID :

DEBUG : bool type. If this value is true, the localization results are given to stderr.

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 MUSIC 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, MUSIC 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 $\mathbf{q}_1 = (x_1, y_1, z_1)$ and $\mathbf{q}_2 = (x_2, y_2, z_2)$, the angle θ is expressed as follows.

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = |\mathbf{q}_1||\mathbf{q}_2| \cos \theta \quad (6.17)$$

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

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

In order to simplify the indication, the angle between the i th sound source and the j th 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 ?? . 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 MUSIC 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.

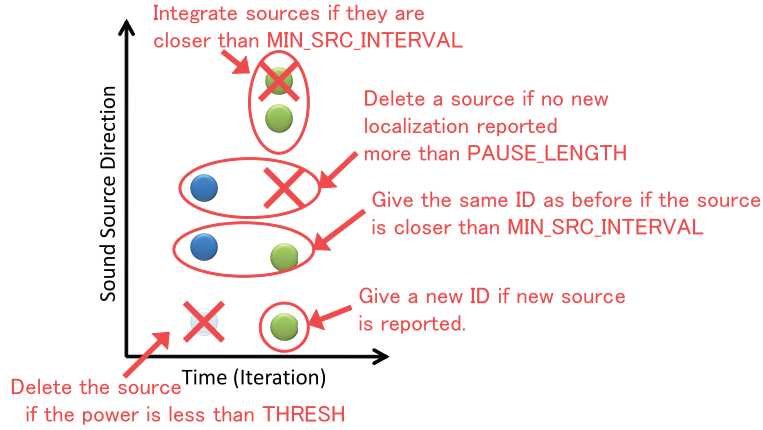


Figure 6.49: Sound source tracking method of SourceTracker

6.2.20 SourceTrackerPF

Outline of the node

A compatible node with SourceTracker . SourceTracker uses an algorithm that gives the same ID to source localization results when sound sources are sufficiently close. However, this node gives ID to source localization results per particle group by performing clustering with particle filter based on the angle of Direction of Arrival. While this nodes enables more robust sound source tracking compared to SourceTracker , its computational cost is larger than that of SourceTracker . Note that the estimation result will be varied due to nonlinear modeling using a random walk model for the particle state transition model. Estimated sound source location is updated every frame.

Necessary file

No files are required.

Usage

When to use

Since the directions of arrival of a sound source obtained by the sound source localization node such as LocalizeMUSIC are discrete values and fluctuate, tracking the sound source localization results is required. SourceTrackerPF gives a sound source ID to each particle group for the sound source whose power is higher than the threshold value by performing clustering using information on the Direction of Arrival with particle filters. Recommended to use this node when sound source tracking does not work with SourceTracker . While this node enables more robust sound source tracking compared to SourceTracker , its computational cost increases larger than that of SourceTracker . Making the number of particle, one of the parameters, smaller reduces the processing time for computation. The parameter should be tuned taking into consideration the trade-off with the accuracy of estimation.

Typical connection

Normally, have the output of sound source localization node such as ConstantLocalization or LocalizeMUSIC connected to the input of this node. The appropriate sound source ID will be given to the localization results so then have it connected to GHDSS , sound source separation node, based on the localization results, or to localization result display node, DisplayLocalization .

Figure ?? shows a connection example.

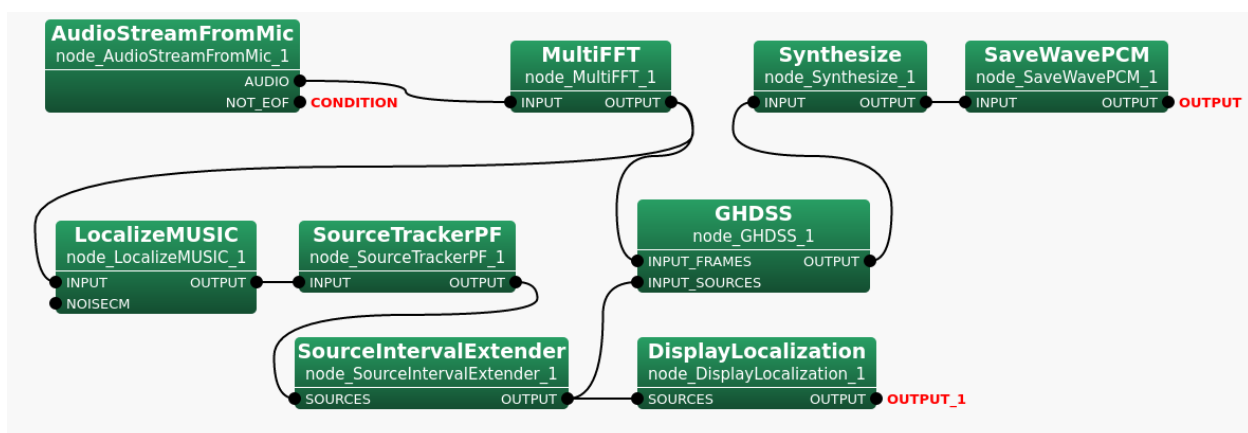


Figure 6.50: Connection example of SourceTrackerPF

Input-output and property of the node

Input

INPUT : Vector<ObjectRef> type. Sound source localization results with no sound source ID.

Output

OUTPUT : Vector<ObjectRef> type. Sound source localization results with sound source ID.

Parameter

Table 6.37: Parameter list of SourceTrackerPF

Parameter name	Type	Default value	Unit	Description
THRESH_SOURCE_POWER	float			The threshold to determine whether or not the localization result should be ignored as noise.
TOTAL_PARTICLE	int	1000		The number of scattered particles.
SOURCE_MAX	int	2		The maximum number of sound sources.
ANOTHER_SOURCE	float	0.00001		The likelihood threshold of a particle group considered as the same sound source.
IGNORE_SOURCE	float	0.00001		The likelihood threshold to generate a new sound source.
REMOVE_SOURCE	int	150	[frame]	The number of frames left of particle groups not associated with the observation value.
OUTPUT_RANGE	float	3.0	[deg]	The angle range to recognize particles as the ones in the group 鐮 ¥ s neighborhood.
LIKELIHOOD_SIGMA	float	1.0	[deg]	The variance of probability distribution assumed to obtain the likelihood of the observation value.
STATE_UPDATE_SIGMA	float	1.0	[deg]	The variance of a Random Walk.
SUM_W	float	0.4		The parameter to avoid divergence.
HISTORY_LOG	bool	false		Enable or disable to show the history log.

THRESH_SOURCE_POWER : float type. The threshold value to determine whether or not the localization result should be ignored as noise. When the MUSIC power is smaller than this value, the result is determined to be noise so as not outputted. Values which are too small will cause a lot of noises in the output whereas values which are too large will make difficult to localize the target sound source. It is vital to tune the parameter to find the value that satisfies this trade-off.

TOTAL_PARTICLE : int type. The total number of particles to compute. While increasing this value causes high computational cost, placing the too small value worsens the accuracy of estimation on the probability distribution.

SOURCE_MAX : int type. The maximum number of sound sources. Increasing this value will not affect the total number of particles. It will split particles and use them.

ANOTHER_SOURCE : float type. The likelihood threshold of a particle group considered as the same sound source. When the maximum likelihood within the particle group falls below this value, it ends associating with the observation value.

IGNORE_SOURCE : float type. The likelihood threshold to generate a new sound source. Prepare the new sound source generation when the likelihood for the all particle groups is under this value.

REMOVE_SOURCE : int type. The number of frames left of particle groups not associated with the observation value.

OUTPUT_RANGE : float type. The angle range from the center of gravity of the particle group to recognize particles as the ones in the group's neighborhood. Particles within this range will be computed as being included in the particle group.

LIKELIHOOD_SIGMA : float type. The parameter for likelihood calculation. The variance of probability distribution assumed to obtain the likelihood of the observation value. Specifying the variance too large will result high likelihood irrespective of the observation value. Specifying the variance too small will lower likelihood and diverge.

STATE_UPDATE_SIGMA : float type. The variance of a Random Walk during the state transition. The random value is based on the normal distribution.

SUM_W : float type. The parameter to avoid divergence. The threshold value as to whether or not to weight each particle. Output the average value of the particles in the group when the sum of the weighting values is smaller than the threshold since the calculation result will diverge when the importance is too small.

HISTORY_LOG : bool type. Setting the value to true shows the history log of the particle group. The default value is false.

Details of the node

First, this node determine whether or not the sound source localization result with no sound source ID that is given as the input is less than the THRESH_SOURCE_POWER parameter value. The results whose MUSIC power is less than the threshold will be considered as noise and discarded.

For the results whose MUSIC power is equal to or greater than the THRESH_SOURCE_POWER parameter value, the node will give sound source ID and estimate the direction of the sound source by the following sound source tracking method based on the particle filter.

A sound source tracking method using particle filters

In particle filters, define both the transition model $p(\mathbf{x}(t) | \mathbf{x}(t-1))$ and the observation model $p(\mathbf{y}(t) | \mathbf{x}(t))$ as stochastic expression where the internal state is $\mathbf{x}(t)$. Note that $\mathbf{y}(t)$ indicates an observation vector. The i th particle is holding the importance $w_i(t)$ that indicates how much both the internal state $\mathbf{x}_i(t)$ and the particles would contribute to sound source tracking results. The importance is generally defined as likelihood.

The processing of this node consists of five steps; Initialization, Creation and Annihilation of a Sound Source, Selection, and Output.

Step 1 - Initialization

In the initialization, distribute all particles uniformly and randomly. In addition, adopt a particle group and define the importance w_i as follows so that multiple sound sources can be handled.

$$\sum_{i \in P_k} w_i = 1 \quad (6.19)$$

$$\sum_{k=1}^S N_k = N \quad (6.20)$$

Here, N_k is the number of particles that P_k , the k th particle group, has, S is the number of sound sources, and N is the total number of particles.

Step 2 - Creation and Annihilation of a Sound Source

This step is for dealing with multiple sound sources. The internal state of the particle group P_k is defined as follows.

$$\hat{x}_k(t) = \sum_{i \in P_k} x_i \cdot w_i(t) \quad (6.21)$$

When the j th observation at the time t is y_j , the angle between y_j and $\hat{x}_k(t)$ is $\angle\theta$, and the threshold for the angle obtained by ANOTHER_SOURCE parameter is $\angle\theta_{th}$, do the following processing.

- Let y_j associate with P_k if $\angle\theta < \angle\theta_{th}$ is true.
- Create a new particle group if no particle group associated with y_j is found.
- Annihilate P_k if the observation associated with the particle group P_k cannot be obtained within the time period specified in the REMOVE_SOURCE parameter.
- In any case, redistribute the particles to meet the equations (??, ??).

Step 3 - Importance sampling

The flow of the importance sampling are as follows.

1. Estimate the state $x_i(t)$ from $x_i(t-1)$ using the transition model, $p(x(t)|x(t-1))$.
2. Update the importance $w_i(t)$ using the equation (??).
3. Normalize $w_i(t)$ according to the equations (??, ??).

The transition model of the azimuth angle $\theta_i(t)$ and the elevation angle $\phi_i(t)$ of the sound source direction, the elements of $x_i(t)$, are defined based on the Random Walk Process as follows.

$$\theta_i(t) = \theta_i(t-1) + r_\theta \quad (6.22)$$

$$\phi_i(t) = \phi_i(t-1) + r_\phi \quad (6.23)$$

r_* is a random number based on normal distribution. Specify the variance in the STATE_UPDATE_SIGMA parameter.

When the angle between $x_i(t)$ and y_j is $\angle\psi$, the likelihood can be defined below.

$$l(t) = \exp\left(-\frac{\angle\psi^2}{2R}\right) \quad (6.24)$$

Update w_i with the following equation at the end.

$$w_i(t) = l(t) \cdot w_i(t-1) \quad (6.25)$$

Step 4 - Selection

Update particles according to the importance w_i .

The number of particles for the i that satisfies $i \in P_k$ is updated by the following equation.

$$N(k_i) = \text{round}(N_k \cdot w_i) \quad (6.26)$$

In the below case, R_k particles are left not updated.

$$R_k = N_k - \sum_{i \in P_k} N(k_i) \quad (6.27)$$

These particles above are also distributed according to the residual weight parameter, $R(w_i)$.

$$R(w_i) = w_i - N(k_i) \Big/ \sum_{j \in P_k} N(k_j) \quad (6.28)$$

Sampling Importance Resampling (SIR) is used.

Step 5 - Output

Estimate the posterior probability $p(\mathbf{x}(t) | \mathbf{x}(t))$ from the density of updated particles.

The internal state of the particle group for the sound source k is estimated by the equation (??).

Repeat Step2 to Step5 until tracking the sound source is completed.

Output the $\hat{\mathbf{x}}_k(t)$ of each particle group as the estimation result of the sound source position.

Assign a sound source ID per particle group so the sound source ID will be taken over within the same particle group.

References

- (1) K. Nakadai, K. Nakajima, M. Murase, H. Okuno, Y. Hasegawa and H. Tsujino: "Tracking of Multiple Sound Source by Integration of Robot-Embedded and In-Room Microphone Arrays", Journal of the Robotics Society of Japan, Vol.25, no.6 (2007).

6.2.21 CSP

Outline of the node

This estimates a sound's direction in the horizontal plane using the CSP method from 2ch waveform data.

Necessary file

No files are required.

Usage

When to use

This node estimates a sound's direction using the CSP method. The orientation result outputted from this node is used for post-processing such as tracking and source separation.

Typical connection

Figure ?? shows a typical connection example.

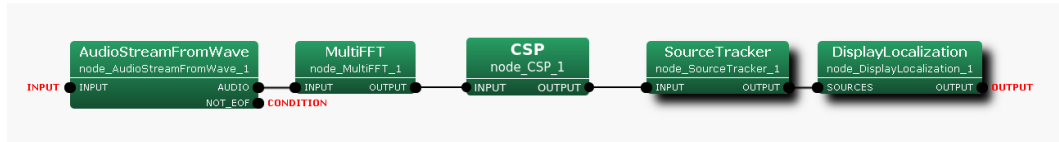


Figure 6.51: Connection example of CSP

Input-output and property of the node

Input

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

Output

OUTPUT : Source position (direction) is expressed as Vector<ObjectRef> type. ObjectRef is a Source and is a structure which consists of CSP value of the source and its direction. The element number of Vector is a sound number (N).

CSPVALUE : Vector<float> type. CSP value for every direction. The output is equivalent to $CS P_{i,j}(k)$ in Eq.(??). This output terminal is not displayed by default.

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

Parameter

DISTANCE_BETWEEN_MICS : float type. 0.3 is default value. The distance between 2 microphones.

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.

SPEED_OF_SOUND : float type. 340 is default value. The speed of sound.



Step 1: Right-click CSP and click Add Output. Step 2: Enter CSPVALUE in the input, then, click Add. Step 3: The CSPVALUE output terminal is added to the node.

Figure 6.52: Usage example of hidden outputs : Display of CSPVALUE terminal

Table 6.38: Parameter list of CSP

Parameter name	Type	Default value	Unit	description
DISTANCE_BETWEEN_MICS	float	0.3	[m]	Distance between microphones
SAMPLING_RATE	int	16000	[Hz]	Sampling rate
SPEED_OF_SOUND	float	340	[m/s]	Speed of sound
LENGTH	int	512	[pt]	FFT points (<i>NFFT</i>)
LOWER_BOUND_FREQUENCY	int	500	[Hz]	Lower bound frequency
UPPER_BOUND_FREQUENCY	int	2800	[Hz]	Upper bound frequency
MANUAL_WEIGHT_SQUARE	Matrix<float>	See below.		Key point of rectangular weight
MIN_DEG	int	0	[deg]	Minimum azimuth
MAX_DEG	int	180	[deg]	Maximum azimuth
WINDOW	int	50	[frame]	Frames to normalize CrossSpectrum
WINDOW_TYPE	string	FUTURE		Frame selection to normalize CrossSpectrum
PERIOD	int	50	[frame]	The cycle to compute SSL
CSP_THRESHOLD	float	0		Threshold value of CSP value
MAXNUM_OUT_PEAKS	int	-1		Max. num. of output peaks
DEBUG	bool	false		ON/OFF of debug output

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.

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 $0 \leq \omega_{min} \leq \text{SAMPLING_RATE}/2$.

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 $\omega_{min} < \omega_{max} \leq \text{SAMPLING_RATE}/2$.

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 Cross 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 Cross spectrum from 2000 [Hz] to 4000 [Hz] and 6000 [Hz] to 8000 [Hz] can be suppressed.

MIN_DEG : int type. 0 is the default value. It is the minimum angle for peak search.

MAX_DEG : int type. 180 is the default value. It is the maximum angle for peak search.

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 MIDDLEW, 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.

CSP_THRESHOLD : float type. 0 is default value. This node pick up the local-peak from CSP value which is larger than this value.

MAXNUM.OUT.PEAKS : int type. -1 is the default. This parameter defines the maximum number of output peaks of CSP value (sound sources). If -1 or 0, all the peaks are output. If $MAXNUM.OUT.PEAKS > 0$, $MAXNUM.OUT.PEAKS$ peaks are output in order of their value.

DEBUG : bool type. ON/OFF of the debug output and the format of the debug output is CSP value.

Details of the node

CSP method estimates the sound's direction from CSP value and Time Difference of Arrival (TDOA), which are calculated from 2ch signals ($s_i(n)$, $s_j(n)$) recording with 2 microphones (M_i , M_j). CSP value and TDOA are expressed as follows.

$$CSP_{i,j}(k) = DFT^{-1} \left[\frac{DFT[s_i(n)]DFT[s_j(n)]^*}{|DFT[s_i(n)]||DFT[s_j(n)]|} \right] \quad (6.29)$$

$$\tau = \operatorname{argmax}_k(CSP_{i,j}(k)) \quad (6.30)$$

τ is the time (samples) difference of the sound, and CSP value has a local peak at the time. The sound's direction is expressed as follows with the time difference τ , the speed of sound c , the distance between 2 microphones and the sampling rate F_s .

$$\theta = \cos^{-1} \left(\frac{c\tau/F_s}{d} \right) \quad (6.31)$$

References

- (1) Shun Tsunasawa, Shinji Ohyama, "Multi-speaker Localization and Tracking Based on TDOA Derived from Multi-frame CSP Coefficient" *Transactions of the Society of Instrument and Control Engineers*, Vol.53, No.12, 644/653 (2017).

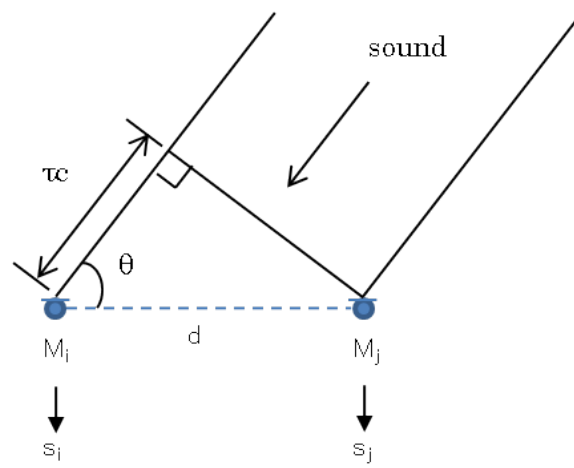


Figure 6.53: CSP method

6.2.22 LocalizeBFDS

Outline of the node

This estimates a sound's direction using delay-and-sum (DS) beamforming (BF) method from multichannel waveform data.

Necessary file

The transfer function file or the position files is required. The transfer function file consists of a steering vector. The position files are the microphone position file and source position file which are created by harktool. When the position files are used, this node generates the transfer function based on the positional relationship between the microphone and sound.

Usage

When to use

This node estimates a sound's direction using delay-and-sum beamforming (DS) method. This node is made for the test of EstimateTF node, so the output doesn't have the enough information for post-processing such as tracking and source separation.

Typical connection

Figure ?? shows a typical connection example.

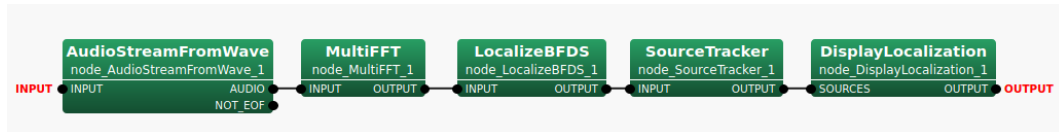


Figure 6.54: Connection example of LocalizeBFDS

Input-output and property of the node

Input

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

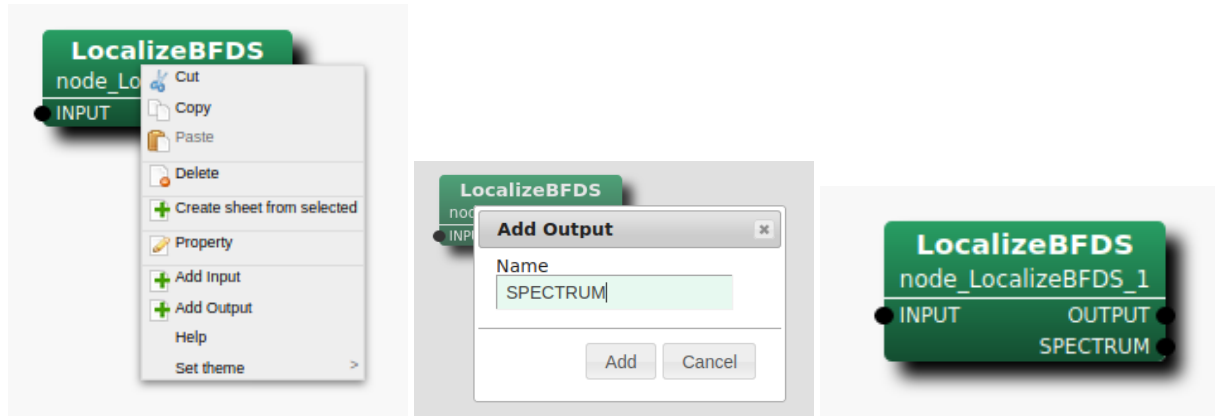
Output

OUTPUT : Source position (direction) is expressed as `Vector<ObjectRef>` type. `ObjectRef` is a `Source` and is a structure which consists of the BF power of the source and its direction. The element number of `Vector` is a sound number (N), but $N = 1$.

SPECTRUM : `Vector<float>` type. BF spectrum for every direction. This output terminal is not displayed by default.

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

Parameter



Step 1: Right-click LocalizeBFDS and click Add Output. Step 2: Enter SPECTRUM in the input, then, click Add. Step 3: The SPECTRUM output terminal is added to the node.

Figure 6.55: Usage example of hidden outputs : Display of SPECTRUM terminal

Table 6.39: Parameter list of LocalizeBFDS

Parameter name	Type	Default value	Unit	description
TF_CHANNEL_SELECTION	Vector<int>	See below.		Channel number used
SAMPLING_RATE	int	16000	[Hz]	Sampling rate
LENGTH	int	512	[pt]	FFT points (<i>NFFT</i>)
USE_TF_FILE	bool	false		Using flag for Transfer function file
TF_FILENAME	string			Transfer function file name
MIC_POSITIONS	string			Microphone position file name
SRC_POSITIONS	string			Source position file name
SPEED_OF_SOUND	float	340	[m/s]	Speed of sound
DEBUG	bool	false		ON/OFF of debug output

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.

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.

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.

USE_TF_FILE : bool type. In the case of true, this node uses the transfer function from the transefer function file. In the case of false, this node uses the transfer function based on the positional relationship between the microphone and sound.

TF_FILENAME : string D There is no default value. The file name of the transfer function file is designated. This parameter is shown when USE_TF_FILE = *true*.

MIC_POSITIONS : string . There is no default value. The file name of the microphone position file is designated. This parameter is shown when USE_TF_FILE = *false*.

SRC_POSITIONS : string . There is no default value. The file name of the source position file is designated. This parameter is shown when `USE_TF_FILE = false`.

SPEED_OF_SOUND : float type. 340 is default value. The speed of sound.

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. Then, BF spectrum of the frame is shown.

Details of the node

Localization using delay-and-sum beamforming :

In the localization using delay-and-sum beamforming, the spatial spectrum is estimated by scanning the beam for the every direction. The beam is formed by following steps.

1. add the delay for the particular direction to each input signal.
2. sum the delayed signals.

In this node, the spatial spectrum $P(\theta)$ can be expressed as follows with the steering vector w^H and the input signal x .

$$P(\theta) = w^H(\theta)x \quad (6.32)$$

$w(\theta)$ for the direction θ can be expressed as follows with the transfer function $a(\theta)$.

$$w(\theta) = \frac{a(\theta)}{|a(\theta)|} \quad (6.33)$$

Transfer function based on the positional relationship between the microphone and sound :

In this node, the transfer function based on the positional relationship between the microphone and sound is calculated assuming plane wave propagation.

References

- (1) Futoshi Asano, "Array signal processing for acoustics" Acoustical Society of Japan

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 ?? . 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.

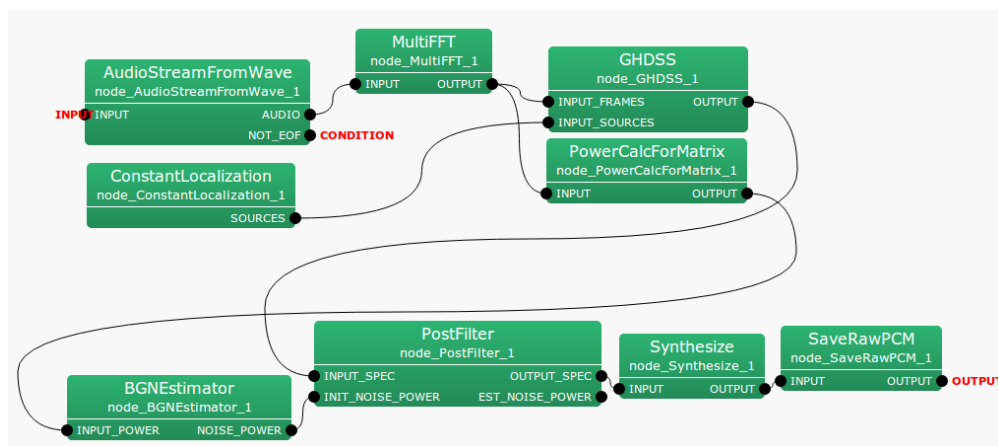


Figure 6.56: Connection example of BGNEstimator

Input-output and property of the node

Input

INPUT_POWER : Matrix<float> type. Multichannel power spectrum

Output

Table 6.40: 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

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 ???. The

Table 6.41: Variable

Variable name	Corresponding parameter or description
$S(f, k_i) = [S_1(f, k_i), \dots, S_M(f, k_i)]^T$	Time frame f , input power spectrum of the frequency bin k_i
$\lambda(f, k_i) = [\lambda_1(f, k_i), \dots, \lambda_M(f, k_i)]^T$	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

derivation flow is as shown in Figure ???.

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

$$S_m^{smo,t}(f, k_i) = \alpha_s \lambda_m(f - 1, k_i) + (1 - \alpha_s) S_m(f, k_i) \quad (6.34)$$

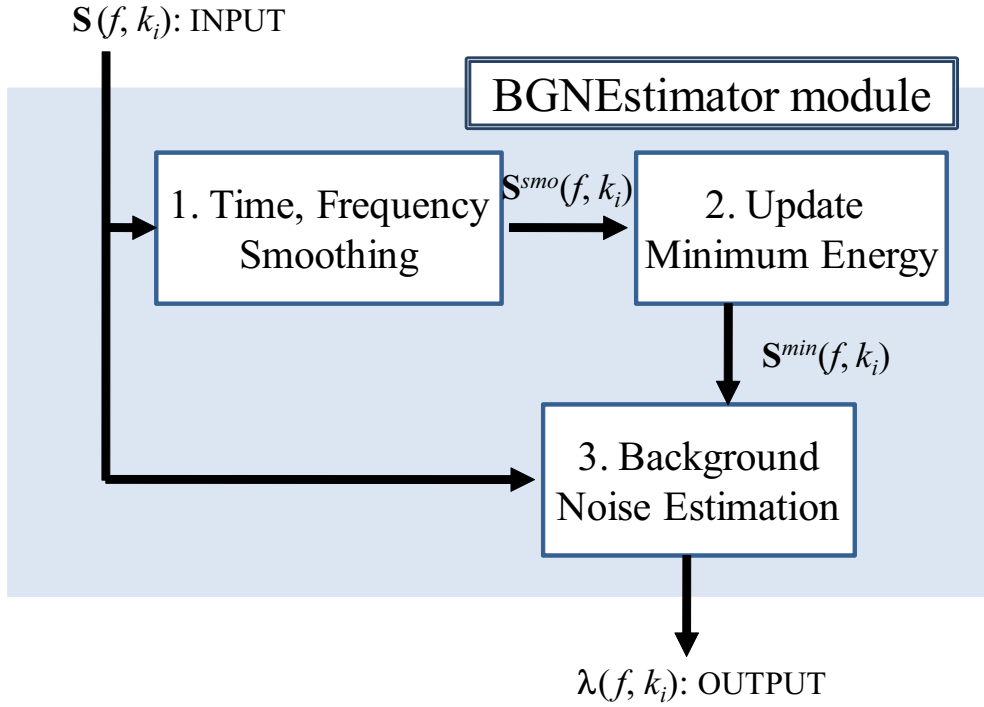


Figure 6.57: Flow of stationary noise estimation

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

$$S_m^{smo}(f, k_i) = 0.25S_m^{smo}(f, k_{i-1}) + 0.5S_m^{smo}(f, k_i) + 0.25S_m^{smo}(f, k_{i+1}) \quad (6.35)$$

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

$$S_m^{tmp}(f, k_i) = \begin{cases} S_m^{smo}(f, k_i), & \text{if } f = nL \\ \min\{S_m^{tmp}(f-1, k_i), S_m^{smo}(f, k_i)\}, & \text{if } f \neq nL \end{cases} \quad (6.36)$$

$$S_m^{min}(f, k_i) = \begin{cases} \min\{S_m^{tmp}(f-1, k_i), S_m^{smo}(f, k_i)\}, & \text{if } f = nL \\ \min\{S_m^{min}(f-1, k_i), S_m^{smo}(f, k_i)\}, & \text{if } f \neq nL \end{cases} \quad (6.37)$$

Here, n is an arbitrary integer.

3 罫 ¥ 愁 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.

$$S_m^{smo}(f, k_i) < \delta S_m^{min}(f, k_i) \text{ or} \quad (6.38)$$

$$f < N \text{ or} \quad (6.39)$$

$$S_m^{smo}(f, k_i) < \lambda_m(f-1, k_i) \quad (6.40)$$

2. Calculation of smoothing coefficient

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

$$\alpha_d = \begin{cases} \frac{1}{f+1}, & \text{if } (\frac{1}{f+1} \geq \alpha_d^{min}) \\ \alpha_d^{min}, & \text{if } (\frac{1}{f+1} < \alpha_d^{min}) \\ 0 & \text{(When the target sound is contained)} \end{cases} \quad (6.41)$$

Steady noise is obtained by the following equations.

$$\lambda_m(f, k_i) = (1 - \alpha_d)\lambda_m(f - 1, k_i) + \alpha_d\theta S_m(f, k_i) \quad (6.42)$$

6.3.2 BeamForming

Outline of the node

BeamForming node performs sound source separation based on the following methods:

- DS : Delay-and-Sum beamforming
- WDS : Weighted Delay-and-Sum beamforming
- NULL : NULL beamforming
- ILSE : Indefinite term and Least Square Estimator based beamforming
- LCMV Linearly Constrained Minimum Variance beamforming
- GJ : Linearly Constrained Minimum Variance beamforming, Griffiths-Jim type
- GICA : Geometrically constrained Independent Component Analysis
- GHDSS : Geometrically constrained Higher-order Decorrelation-based Source Separation

Node inputs are:

- Multi-channel complex spectrum of the input acoustic signal,
- Direction of localized sound sources,
- Direction or localized noise sources.

Note outputs are a set of complex spectrum of each separated sound.

Necessary files

Table 6.42: Necessary files for BeamForming

Corresponding parameter name	Description
TF_CONJ_FILENAME	Transfer function of microphone array

Usage

When to use

This node is used to perform sound source separation on the sound source direction originated using 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 ?? shows connection examples of the BeamForming . The node has three inputs as follows:

1. INPUT_FRAMES takes a multi-channel complex spectrum containing the mixture of sounds produced by for example MultiFFT ,
2. INPUT_SOURCES takes the results of sound source localization produced by for example LocalizeMUSIC or ConstantLocalization ,
3. INPUT_NOISE_SOURCES takes the results of sound source localization for noise sources produced by for example LocalizeMUSIC or ConstantLocalization .

The output is the separated signals.

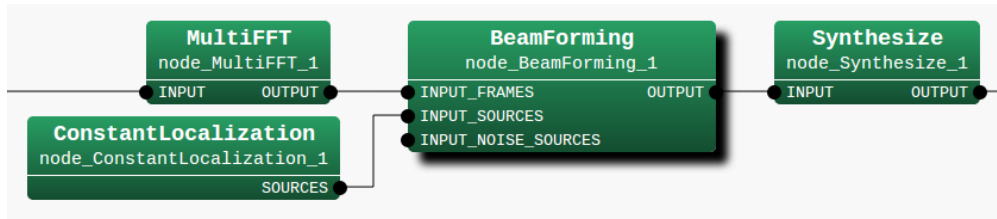


Figure 6.58: Example of Connections of BeamForming

Input-output and property of the node

Input

INPUT_FRAMES : `Matrix<complex<float> >` type. Multi-channel complex spectra. Corresponding to the complex spectrum of input waveform from each microphone, the rows correspond to the channels and the columns correspond to the frequency bins.

INPUT_SOURCES : `Vector<ObjectRef>` type. A Vector array of the Source type object in which sound source localization results are stored. Typically, takes the output of SourceIntervalExtender connected to SourceTracker .

INPUT_NOISE_SOURCES : `Vector<ObjectRef>` type. A Vector array of the Source type object where noise source localization results are stored. The input is optional. It enables to avoid lowering performance of beam-forming due to noise.

Output

OUTPUT : `Map<int, ObjectRef>` type. A pair containing the sound source ID and the complex spectrum of the separated sound (`Vector<complex<float> >` type). Output as many as the number of sound sources .

Parameter

LENGTH : `int` type. Analysis frame length [samples], which must be equal to the values at a preceding node (e.g. AudioStreamFromMic or the MultiFFT). The default is 512.

ADVANCE : `int` type. Shift length of a frame [samples], which must be equal to the values at a preceding node (e.g. AudioStreamFromMic or the MultiFFT). The default is 160.

SAMPLING_RATE : `int` type. Sampling frequency of the input waveform [Hz]. The default is 16000.

LOWER_BOUND_FREQUENCY : `int` type. This parameter is the minimum frequency used when separation 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 : `int` type. This parameter is the maximum frequency used when separation 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 ?? 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. If specified, a matrix in INITW_FILENAME is used as an initial separation matrix. Otherwise, the pre-measured separation matrix obtained from a database in TF_CONJ_FILENAME or the separation matrix estimated from a geometric relation is used as an initial separation matrix.

GICA_SS.METHOD : string type. Select a stepsize of ICA (Independent Component Analysis) calculation method based for a blind source separation. If FIX, GICA_SS.MYU designed fix value is used. If LC.MYU, GICA.LC.MYU associated with the step size based on geometric constraint is used. If ADAPTIVE, the stepsize is adaptively determined.

GICA_SS.MYU : float type. Designate the stepsize to be used when updating a separation matrix based on blind source separation. The default value is 0.001. If GICA_SS.METHOD = FIX, GICA_SS.MYU is the designated value for the stepsize. If GICA_SS.METHOD = LC.MYU, this parameter is ignored. If GICA_SS.METHOD = ADAPTIVE, the adaptive stepsize is multiplied by GICA_SS.MYU, resulting the final stepsize. By setting this value and GICA.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.

GHDSS_SS.METHOD : string type. Select a stepsize calculation method based for HDSS (Higher-order Decorrelation-based Source Separation). If FIX, GHDSS_SS.MYU designed fix value is used. If LC.MYU, GHDSS.LC.MYU associated with the step size based on geometric constraint is used. If ADAPTIVE, the stepsize is adaptively determined.

GHDSS_SS.MYU : float type. Designate the stepsize to be used when updating a separation matrix based on HDSS (Higher-order Decorrelation-based Source Separation). The default value is 0.001. If GHDSS_SS.METHOD = FIX, GHDSS_SS.MYU is the designated value for the stepsize. If GHDSS_SS.METHOD = LC.MYU, this parameter is ignored. If GHDSS_SS.METHOD = ADAPTIVE, the adaptive stepsize is multiplied by GHDSS_SS.MYU, resulting the final stepsize. By setting this value and GHDSS.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.

LCMV.LC.METHOD : string type. Select a stepsize calculation method for separation based on linearly constraints. This parameter affects the stepsize for the source separation based on linearly constraints (LC). If FIX, LCMV.LC.MYU designed fix value is used. If ADAPTIVE, the stepsize is adaptively determined.

LCMV.LC.MYU : float type. Designate the stepsize to be used when updating a separation matrix based on linearly constraints. The default value is 0.001. If LCMV.LC.METHOD = FIX, LCMV.LC.MYU is the designated value for the stepsize. If LCMV.LC.METHOD = ADAPTIVE, the adaptive stepsize is multiplied by LCMV.LC.MYU, resulting the final stepsize.

GJ.LC.METHOD : string type. Select a stepsize calculation method for separation based on linearly constraints. If FIX, GJ.LC.MYU designed fix value is used. If ADAPTIVE, the stepsize is adaptively determined.

GJ.LC.MYU : float type. Designate the stepsize to be used when updating a separation matrix based on linearly constraints. The default value is 0.001. If GJ.LC.METHOD = FIX, GJ.LC.MYU is the designated value for the stepsize. If GJ.LC.METHOD = ADAPTIVE, the adaptive stepsize is multiplied by GJ.LC.MYU, resulting the final stepsize.

GICA.LC.METHOD : string type. Select a stepsize calculation method for separation based on geometric constraints (GC). If FIX, GICA.LC.MYU designed fix value is used. If ADAPTIVE, the stepsize is adaptively determined.

GICA.LC.MYU : float type. Designate the stepsize to be used when updating a separation matrix based on geometric constraints. The default value is 0.001. If GICA.LC.METHOD = FIX, GICA.LC.MYU is the designated value for the stepsize. If GICA.LC.METHOD = ADAPTIVE, the adaptive stepsize is multiplied by LCMV.LC.MYU, resulting the final stepsize. By setting this value and GICA_SS.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.

GHDSS.LC.METHOD : string type. Select a stepsize calculation method for separation based on geometric constraints. This parameter affects the stepsize for the source separation based on geometric constraints (GC). If FIX, GHDSS.LC.MYU designed fix value is used. If ADAPTIVE, the stepsize is adaptively determined.

GHDSS_LC_MYU : float type. Designate the stepsize to be used when updating a separation matrix based on geometric constraints. The default value is 0.001. If GHDSS_LC_METHOD = FIX, GHDSS_LC_MYU is the designated value for the stepsize. If GHDSS_LC_METHOD = ADAPTIVE, the adaptive stepsize is multiplied by GHDSS_LC_MYU, resulting the final stepsize. By setting this value and GHDSS_SS_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.

GHDSS_LC_CONST : string type. Select geometric constraint method. If DIAG, use only diagonal components (direct sound components) for geometric constraints. If FULL, use off-diagonal components in addition to diagonal components (direct sound components) for geometric constraints. Highly precise separation is possible even with DIAG, because the blind spot is automatically formed by higher order decorrelation.

GICA_SS_SCAL : float type. Designate the scale factor of a hyperbolic tangent function (tanh) in calculation of the higher-order correlation matrix. The default value is 1.0. 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.

GHDSS_SS_SCAL : float type. Designate the scale factor of a hyperbolic tangent function (tanh) in calculation of the higher-order correlation matrix. The default value is 1.0. 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.

GHDSS_NOISE_FLOOR : float type. Specify the amplitude threshold (upper limit) that regards the input signal as noise. The default value is 0. When the amplitude of the input signal is less than this value, it is regarded as a noise section and the separation matrix is not updated. Specify a positive real number, if the noise is large and the separation matrix does not converge stably.

GHDSS_UPDATE : string type. Decide how to update separation matrix. If STEP, update based on geometric constraints is performed after updating based on higher order decorrelation. If TOTAL, perform updates based on higher order decorrelation and update based on geometric constraints at the same time.

UPDATE_METHOD_W : string type. Recalculation of separation matrix is required when sound source position information changes. At this time, designate a method to consider sound source position information changed. The separating matrix is internally stored together with the coordinates of the sound source ID and the sound source direction for a certain period of time, and once the sound is stopped, if a sound judged to be a sound source from the same direction is detected, the separating matrix saved again The separation process is performed. At this time, a criterion as to whether or not to update the separation matrix is set. If ID, it is judged by the sound source ID whether it is the same direction sound source or not. If POS, it is judged by comparing the coordinates of the sound source direction. If ID.POS, at first sound source IDs are compared, if they are not determined to be identical, further, judgment is made by comparing the coordinates of the sound source direction.

UPDATE_ACCEPT_DISTANCE : float type. Distance to be regarded as the same sound source for movement of the sound source [mm]. If it is within the set distance range, calculation is performed using the updated separation matrix. The default value is 300.0.

EXPORT_W : bool type. The user determines if the results of the separation matrix updated will be output. When true, select EXPORT_W_FILENAME.

EXPORT_W_FILENAME : string type. Designate the name of the file into which a separation matrix will be output. For its format, see Section ??.

BF_METHOD : string type. Designate the sound source separation method. Currently, this node supports the following separation methods:

- DS : Delay-and-Sum beamforming [1]
- WDS : Weighted Delay-and-Sum beamforming [1]
- NULL : NULL beamforming [1]

- ILSE : Iterative Least Squares with Enumeration [2]
- LCMV : Linearly Constrained Minimum Variance beamforming [3]
- GJ : Griffiths-Jim beamforming [4]
- GICA : Geometrically constrained Independent Component Analysis [5]
- GHDSS : Geometrically constrained Higher-order Decorrelation-based Source Separation [5]

ENABLE_DEBUG : bool type. Setting the value to true outputs the separation status to the standard output. The default value is false.

Table 6.43: Parameter list of BF_METHOD = DS,WDS,NULL,ILSE

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.
ENABLE_DEBUG	bool	false		Enabling debug output.

Table 6.44: Parameter list of BF_METHOD = LCMV

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.
LCMV_LC_METHOD	string	ADAPTIVE		A stepsize calculation method based on geometric constraints.If FIX, LCMV_LC_MYU designed fix value is used.If ADAPTIVE, the stepsize is adaptively determined.
LCMV_LC_MYU	float	0.001		The stepsize when updating a separation matrix based on geometric constraints.
UPDATE_METHOD_W	string	ID		How to regard sound source position information as changed.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	Distance to be regarded as the same sound source with respect to movement of the sound source.
EXPORT_W	bool	false		Designate whether separation matrixes are to be written to files.
EXPORT_W_FILENAME	string			The name of the file to which the separation matrix is written.
ENABLE_DEBUG	bool	false		Enabling debug output.

Table 6.45: Parameter list of BF_METHOD = GJ

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.
GJ_LC_METHOD	string	ADAPTIVE		A stepsize calculation method based on geometric constraints.If FIX, GJ_LC_MYU designed fix value is used.If ADAPTIVE, the stepsize is adaptively determined.
GJ_LC_MYU	float	0.001		The stepsize when updating a separation matrix based on geometric constraints.
UPDATE_METHOD_W	string	ID		How to regard sound source position information as changed.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	Distance to be regarded as the same sound source with respect to movement of the sound source.
EXPORT_W	bool	false		Designate whether separation matrixes are to be written to files.
EXPORT_W_FILENAME	string			The name of the file to which the separation matrix is written.
ENABLE_DEBUG	bool	false		Enabling debug output.

Details of the node

Technical details: Basically, the technical detail of each separation method can be found in the references below.

Bried explanation of sound source separation:

Table ?? shows the notation of variables used in sound source separation problems. Since the source separation is performed frame-by-frame in the frequency domain, all the variable is computed in a complex field. Also, the separation is performed for all K frequency bins ($1 \leq k \leq K$). Here, we omit k from the notation. Let N , M , and f denote the number of sound sources and the number of microphones, and the frame index, respectively.

We use the following linear model for the signal processing:

$$X(f) = HS(f) + N(f). \quad (6.43)$$

The purpose of the separation is to estimate $W(f)$ based on the following equation:

$$Y(f) = W(f)X(f) \quad (6.44)$$

so that $Y(f)$ is getting closer to $S(f)$. After separation, the estimated $W(f)$ can be saved by setting EXPORT_W=true and put a certain name in EXPORT_W_FILENAME.

TF_CONJ_FILENAME specifies the transfer function matrix H which is pre-measured or pre-calculated. Hereinafter, we denote this pre-measured transfer function as \hat{H} to distinguish from H .

Separation by BF_METHOD = DS,WDS,NULL,ILSE: $W(f)$ is directly determined using \hat{H} and corresponding the directions of target recieved at the INPUT_SOURCES terminal and the directions of noise recieved at the INPUT_NOISE_SOURCES terminal.

Separation by BF_METHOD = LCMV,GJ: The cont function $J_L(W(f))$ for updating the separation matrix is defined by the directions of target recieved at the INPUT_SOURCES terminal and the directions of noise recieved

Table 6.46: Parameter list of BF_METHOD = GICA

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.
GICA_SS_METHOD	string	ADAPTIVE		A stepsize calculation method based on blind source separation.If FIX, GICA_SS_MYU designed fix value is used.If LC_MYU, GICA_LC_MYU associated with the step size based on geometric constraint is used.If ADAPTIVE, the stepsize is adaptively determined.
GICA_SS_MYU	float	0.001		The stepsize when updating a separation matrix based on blind source separation.
GICA_LC_METHOD	string	ADAPTIVE		A stepsize calculation method based on geometric constraints.If FIX, GICA_LC_MYU designed fix value is used.If ADAPTIVE, the stepsize is adaptively determined.
GICA_LC_MYU	float	0.001		The stepsize when updating a separation matrix based on geometric constraints.
GICA_SS_SCAL	float	1.0		The scale factor in a higher-order correlation matrix computation.
UPDATE_METHOD_W	string	ID		How to regard sound source position information as changed.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	Distance to be regarded as the same sound source with respect to movement of the sound source.
EXPORT_W	bool	false		Designate whether separation matrixes are to be written to files.
EXPORT_W_FILENAME	string			The name of the file to which the separation matrix is written.
ENABLE_DEBUG	bool	false		Enabling debug output.

at the INPUT_NOISE_SOURCES terminal. The equation for updating the separation matrix is described simply as follows:

$$\mathbf{W}(f+1) = \mathbf{W}(f) + \mu \nabla_{\mathbf{W}} \mathbf{J}_{\mathbf{L}}(\mathbf{W})(f), \quad (6.45)$$

where $\nabla_{\mathbf{W}} \mathbf{J}_{\mathbf{L}}(\mathbf{W}) = \frac{\partial \mathbf{J}_{\mathbf{L}}(\mathbf{W})}{\partial \mathbf{W}}$. LC_MYU specifies the value of μ . If LC_METHOD = ADAPTIVE, this node computes the adaptive stepsize based on the following equation.

$$\mu = \left. \frac{\mathbf{J}_{\mathbf{L}}(\mathbf{W})}{|\nabla_{\mathbf{W}} \mathbf{J}_{\mathbf{L}}(\mathbf{W})|^2} \right|_{\mathbf{W}=\mathbf{W}(f)} \quad (6.46)$$

Separation by BF_METHOD = GICA: The cost function $J_{\mathbf{G}}(\mathbf{W}(f))$ for updating the separation matrix is defined by the directions of target received at the INPUT_SOURCES terminal and the directions of noise received at the INPUT_NOISE_SOURCES terminal.

$$J_{\mathbf{G}}(\mathbf{W}(f)) = J_{\mathbf{SS}}(\mathbf{W}(f)) + J_{\mathbf{LC}}(\mathbf{W}(f)), \quad (6.47)$$

where $J_{\mathbf{SS}}(\mathbf{W}(f))$ is the cost function for the blind source separation, $J_{\mathbf{LC}}(\mathbf{W}(f))$ is the cost function for the source separation based on geometric constraints. The equation for updating the separation matrix is described simply as

Table 6.47: Parameter list of BF_METHOD = 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.
SPEED_OF_SOUND	float	343.0	[m/s]	Sound speed.
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.
GHDSS_SS_METHOD	string	ADAPTIVE		A stepsize calculation method based on higher-order decorrelation.If FIX, GHDSS_SS_MYU designed fix value is used.If LC_MYU, GHDSS_LC_MYU associated with the step size based on geometric constraint is used.If ADAPTIVE, the stepsize is adaptively determined.
GHDSS_SS_MYU	float	0.001		The stepsize when updating a separation matrix based on higher-order decorrelation.
GHDSS_LC_METHOD	string	ADAPTIVE		A stepsize calculation method based on geometric constraints.If FIX, GHDSS_LC_MYU designed fix value is used.If ADAPTIVE, the stepsize is adaptively determined.
GHDSS_LC_MYU	float	0.001		The stepsize when updating a separation matrix based on geometric constraints.
GHDSS_LC_CONST	string	FULL		Determine geometric constraints.If DIAG, use only diagonal components.If FULL, use whole part.
GHDSS_SS_SCAL	float	1.0		The scale factor in a higher-order correlation matrix computation.
GHDSS_NOISE_FLOOR	float	0.0		The threshold value of the amplitude for judging the input signal as noise (upper limit).
GHDSS_UPDATE	string	STEP		The method to update separation matrixes.If STEP, update based on geometric constraints is performed after updating based on higher order decorrelation.If TOTAL, perform updates at the same time.
UPDATE_METHOD_W	string	ID		How to regard sound source position information as changed.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	Distance to be regarded as the same sound source with respect to movement of the sound source.
EXPORT_W	bool	false		Designate whether separation matrixes are to be written to files.
EXPORT_W_FILENAME	string			The name of the file to which the separation matrix is written.
ENABLE_DEBUG	bool	false		Enabling debug output.

follows:

$$\mathbf{W}(f+1) = \mathbf{W}(f) + \mu_{SS} \nabla_{\mathbf{W}} J_{SS}(\mathbf{W})(f) + \mu_{LC} \nabla_{\mathbf{W}} J_{LC}(\mathbf{W})(f), \quad (6.48)$$

where $\nabla_{\mathbf{W}}$ means the partial derivative in respect of \mathbf{W} same as Eq. (??). The μ_{SS} and μ_{LC} in the equation can be specified by SS_MYU and LC_MYU, respectively. If SS_METHOD = ADAPTIVE, μ_{SS} is adaptively determined by

$$\mu_{SS} = \frac{J_{SS}(\mathbf{W})}{|\nabla_{\mathbf{W}} J_{SS}(\mathbf{W})|^2} \bigg|_{\mathbf{W}=\mathbf{W}(f)}. \quad (6.49)$$

Table 6.48: Notation of variables

Variables	Description
$\mathbf{S}(f) = [S_1(f), \dots, S_N(f)]^T$	Complex spectrum of target sound sources at the f -th frame
$\mathbf{X}(f) = [X_1(f), \dots, X_M(f)]^T$	Complex spectrum of a microphone observation at the f -th frame, which corresponds to INPUT_FRAMES.
$\mathbf{N}(f) = [N_1(f), \dots, N_M(f)]^T$	Complex spectrum of added noise
$\mathbf{H} = [\mathbf{H}_1, \dots, \mathbf{H}_N] \in \mathbb{C}^{M \times N}$	Transfer function matrix from the n -th sound source ($1 \leq n \leq N$) to the m -th microphone ($1 \leq m \leq M$)
$\mathbf{W}(f) = [\mathbf{W}_1, \dots, \mathbf{W}_M] \in \mathbb{C}^{N \times M}$	Separation matrix at the f -th frame
$\mathbf{Y}(f) = [Y_1(f), \dots, Y_N(f)]^T$	Complex spectrum of separated signals

If LC_METHOD = ADAPTIVE, μ_{LC} is adaptively determined by

$$\mu_{LC} = \left. \frac{J_{LC}(\mathbf{W})}{|\nabla_{\mathbf{W}} J_{LC}(\mathbf{W})|^2} \right|_{\mathbf{W}=\mathbf{W}(f)}. \quad (6.50)$$

Trouble shooting: Basically, follow the trouble shooting of the GHDSS node.

References

- [1] H. Krim and M. Viberg, 'Two decades of array signal processing research: the parametric approach', in IEEE Signal Processing Magazine, vol. 13, no. 4, pp. 67–94, 1996. D. H. Johnson and D. E. Dudgeon, Array Signal Processing: Concepts and Techniques, Prentice-Hall, 1993.
- [2] S. Talwar, et al.: 'Blind separation of synchronous co-channel digital signals using an antenna array. I. Algorithms', IEEE Transactions on Signal Processing, vol. 44, no. 5, pp. 1184 - 1197.
- [3] O. L. FrostIII, 'An Algorithm for Lineary Constrained Adaptive array processing', Proc. of the IEEE, Vol. 60, No.8, 1972
- [4] L. Griffiths and C. Jim, 'An alternative approach to linearly constrained adaptive beamforming', IEEE trans. on ant. and propag. Vol. AP-30, No.1, 1982
- [5] H. Nakajima, et al.: 'Blind Source Separation With Parameter-Free Adaptive Step-Size Method for Robot Audition', IEEE Trans. ASL Vol.18, No.6, pp.1476-1485, 2010.

6.3.3 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 ??). 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 ?? 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 SpectralGainFilter .

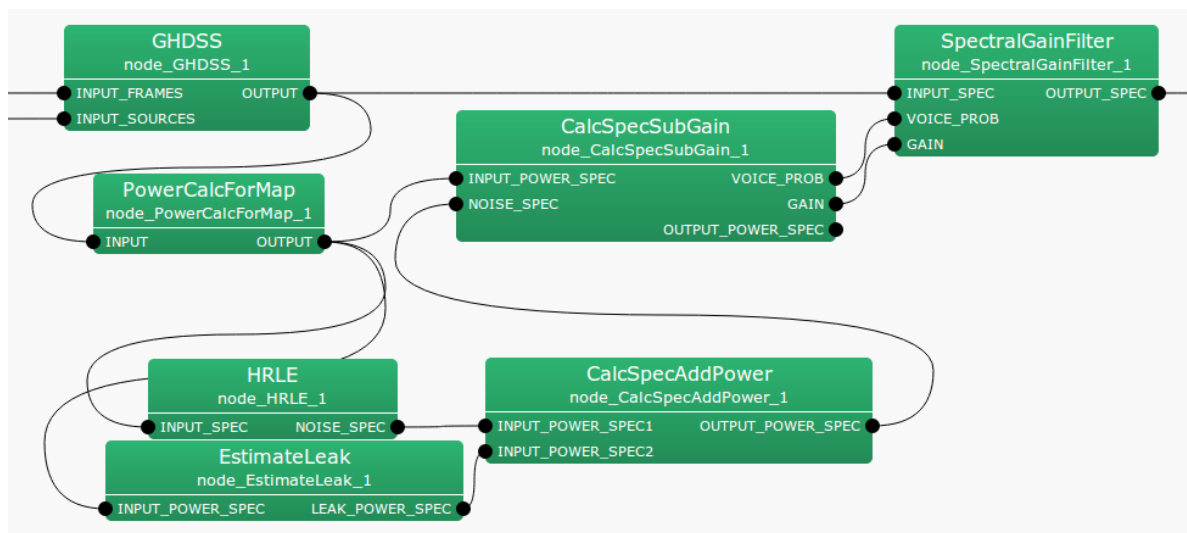


Figure 6.59: Connection example of CalcSpecSubGain

Input-output and property of the node

Table 6.49: 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 ??.) 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.

$$Y_n(k_i) = X_n(k_i) - N_n(k_i) \quad (6.51)$$

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

$$G_n(k_i) = \begin{cases} \text{ALPHA} \frac{Y_n(k_i)}{X_n(k_i)}, & \text{if } Y_n(k_i) > \text{BETA}, \\ \text{BETA}, & \text{if otherwise.} \end{cases} \quad (6.52)$$

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.

6.3.4 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 ?? 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 .

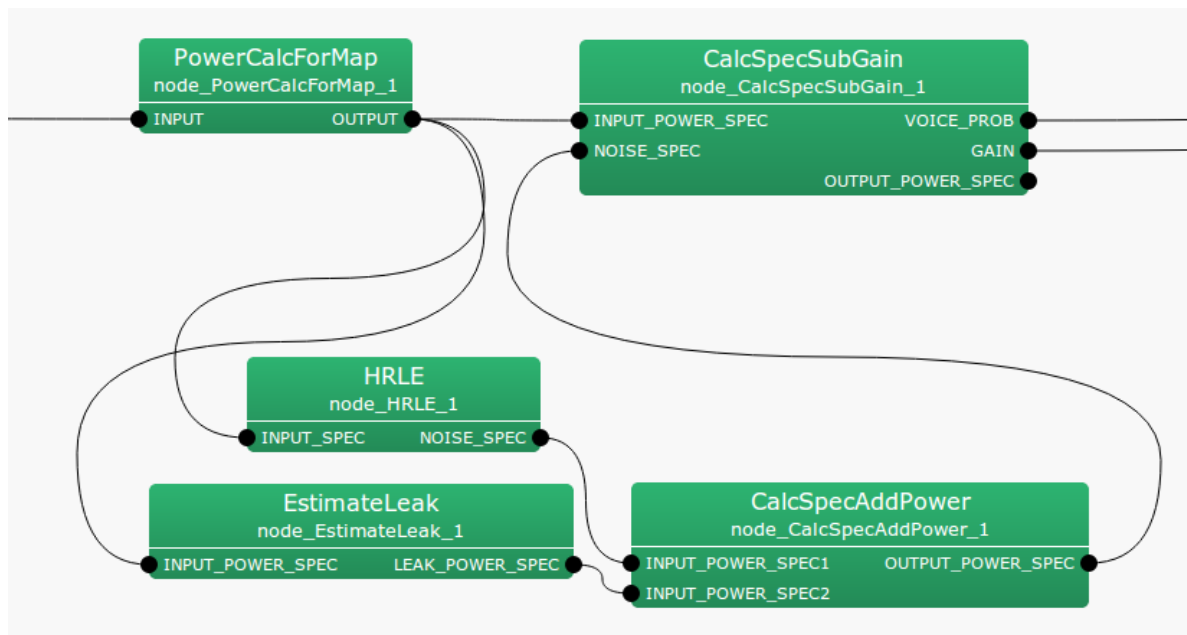


Figure 6.60: Connection example of CalcSpecAddPower

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 parameters.

Details of the node

This node outputs the sum of two spectra.

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

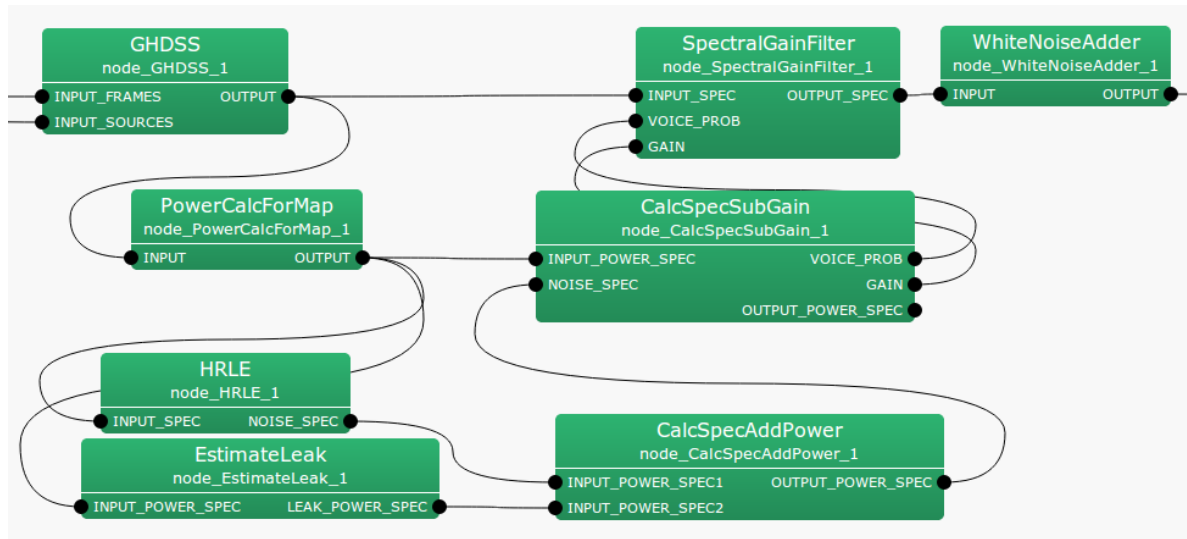


Figure 6.61: Connection example of EstimateLeak

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 6.50: 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 ??.

6.3.6 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. Directivity 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. Node outputs are a set of complex spectrum of each separated sound. **Changes from HARK 2.0 to HARK 2.1**

1. Using Zip Format

In HARK 2.1, GHDSS uses Zip Format for TF_CONJ_FILENAME, INITW_FILENAME and EXPORT_W_FILENAME 録

2. Changing the parameter for calculating error at updating W

In HARK 2.1, GHDSS uses error calculated by the distance from the former frame localization. The unit is [mm].

Necessary files

Table 6.51: 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 ?? 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.??).

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.??).

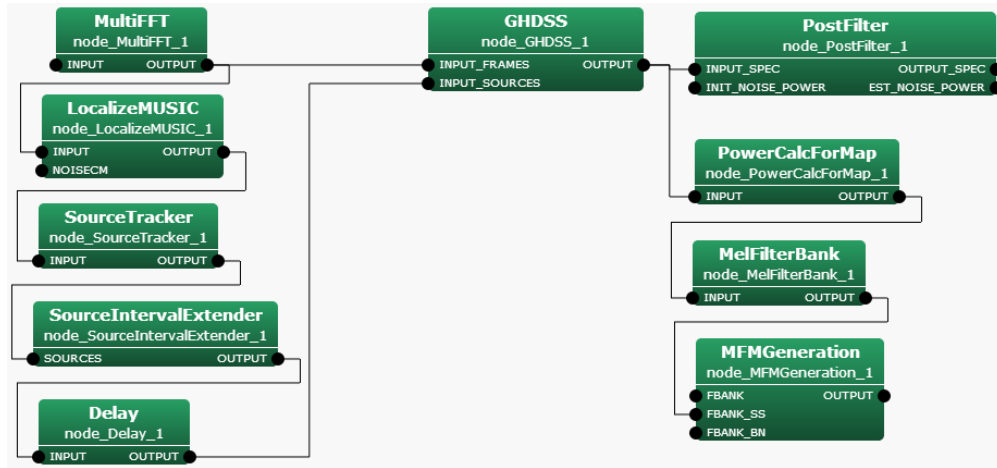


Figure 6.62: Example of Connections of GHDSS

Input-output and property of the node

Table 6.52: 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.
TF_INPUT_TYPE	string	FILE		Selection of TF Input
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	FULL		Determine geometric constraints. Select DIAG or FULL. If DIAG, the geometric constraints contain only direct sound part. If FULL, the geometric constraints use whole part.
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_ACCEPT_DISTANCE	float	300.0	[mm]	The threshold value of distance 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 EXPORT_W_FILENAME	string			The following is valid when true for EXPORT_W. 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.

TRANSFER_FUNCTION : `TransferFunction` type. Instead of loading the transfer function file, this node can also receive the transfer function output from an EstimateTF node and others through the input terminal of TransferFunction type. In that case, the parameter TF_INPUT.TYPE is set to ONLINE. This input terminal is not displayed by default.

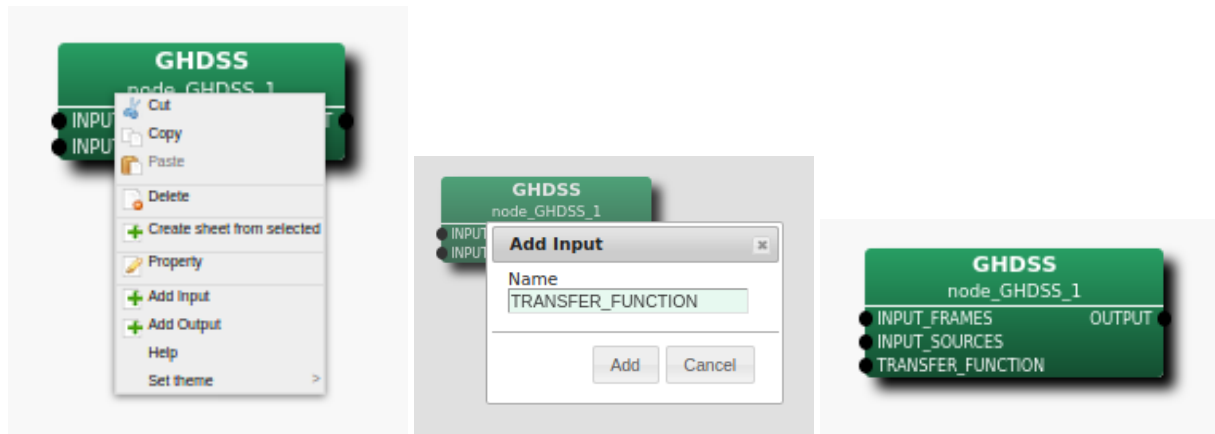
Refer to Figure ?? for the addition method of hidden input.

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).



Step 1: Right-click GHDSS and Step 2: TRANSFER_FUNCTION in Step 3: The TRANSFER_FUNCTION input terminal is added to the node.

Figure 6.63: Usage example of hidden inputs : Display of TRANSFER_FUNCTION terminal

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.

TF_INPUT_TYPE : string type. 'FILE' is the default. When 'FILE' is selected, the transfer function file with the name specified by TF_CONJ_FILENAME is used, and when 'ONLINE' is selected, the input of TRANSFER_FUNCTION is used as the transfer function. An error occurs if the TRANSFER_FUNCTION input is connected when 'FILE' is selected or if the input is not connected when 'ONLINE' is selected.

LOWER_BOUND_FREQUENCY : int type. 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 : int type. 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 ?? 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 ??.

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. Set it as DIAG to use only the diagonal parts (direct sound parts) in the geometric constraints. Select FULL if you want to use whole geometric constraints. Since a blind spot is formed automatically by the higher-order decorrelation, a highly precise separation is achieved in DIAG. The default is FULL.

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_METHOD_W : 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 position 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 positions of the sound source direction.

UPDATE_ACCEPT_DISTANCE : float type. The default value is 300.0. The unit is [mm]. The user sets an allowable error of distance 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 ??.

Details of the node

Formulation of sound source separation: Table ?? shows symbols used for the formulation of the sound source separation problem. The meaning of the indices is in Table ?. 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, \dots, k_{K-1} of K frequencies.

Table 6.53: Definitions of the parameters

Parameter	Description
$S(k_i) = [S_1(k_i), \dots, S_N(k_i)]^T$	The sound source complex spectrum corresponding to the frequency bin k_i .
$X(k_i) = [X_1(k_i), \dots, X_M(k_i)]^T$	The vector of a microphone observation complex spectrum, which corresponds to INPUT_FRAMES.
$N(k_i) = [N_1(k_i), \dots, N_M(k_i)]^T$	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 ($N \times M$).
$Y(k_i) = [Y_1(k_i), \dots, Y_N(k_i)]^T$	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 (??).

$$X(k_i) = H(k_i)S(k_i) + N(k_i). \quad (6.53)$$

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 (??).

$$H_{Dm,n}(k_i) = \exp(-j2\pi l_i r_{m,n}), \quad (6.54)$$

$$l_i = \frac{2\pi\omega_i}{c}, \quad (6.55)$$

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.

$$Y(k_i) = W(k_i)X(k_i) \quad (6.56)$$

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 (??))

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 , $E[\cdot]$ and $\phi(\cdot)$ 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.

$$\phi(Y) = [\phi(Y_1), \phi(Y_2), \dots, \phi(Y_N)]^T \quad (6.57)$$

$$\phi(Y_k) = \tanh(\sigma|Y_k|) \exp(j\angle(Y_k)) \quad (6.58)$$

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 ($W(k_i)H_D(k_i) = I$) 鑑

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.

$$J(W) = \alpha J_1(W) + \beta J_2(W), \quad (6.59)$$

$$J_1(W) = \sum_{i \neq j} |R_{i,j}^{\phi(y)y}|^2, \quad (6.60)$$

$$J_2(W) = \|WH_D - I\|^2, \quad (6.61)$$

Here, α and β are weighting factors. Moreover, the norm of a matrix is defined as below. $\|M\|^2 = \text{tr}(MM^H) = \sum_{i,j} |m_{i,j}|^2$ An update equation of the separation matrix to minimize Equation (??) is obtained by the gradient method that uses the complex gradient calculation $\frac{\partial}{\partial W^*}$.

$$W(k_i, f+1) = W(k_i, f) - \mu \frac{\partial J}{\partial W^*}(W(k_i, f)) \quad (6.62)$$

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 (??), 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 (??), which uses only one frame, is used.

$$W(k_i, f+1) = W(k_i, f) - \left[\mu_{SS} \frac{\partial J_1}{\partial W^*}(W(k_i, f)) + \mu_{LC} \frac{\partial J_2}{\partial W^*}(W(k_i, f)) \right], \quad (6.63)$$

$$\frac{\partial J_1}{\partial W^*}(W) = (\phi(Y)Y^H - \text{diag}[\phi(Y)Y^H]) \tilde{\phi}(WX)X^H, \quad (6.64)$$

$$\frac{\partial J_2}{\partial W^*}(W) = 2(WH_D - I)H_D^H, \quad (6.65)$$

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

$$\tilde{\phi}(Y) = [\phi(\tilde{Y}_1), \phi(\tilde{Y}_2), \dots, \phi(\tilde{Y}_N)]^T \quad (6.66)$$

$$\tilde{\phi}(Y_k) = \phi(Y_k) + Y_k \frac{\partial \phi(Y_k)}{\partial Y_k} \quad (6.67)$$

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

$$\mu_{SS} = \frac{J_1(W)}{2\|\frac{\partial J_1}{\partial W}(W)\|^2} \quad (6.68)$$

$$\mu_{LC} = \frac{J_2(W)}{2\|\frac{\partial J_2}{\partial W}(W)\|^2} \quad (6.69)$$

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

$$W(k_i) = H_D^H(k_i)/M, \quad (6.70)$$

Here, M indicates the number of microphones.

Processing flow The main processing for time frame f in the GHDSS node is shown in Figure ???. The detailed

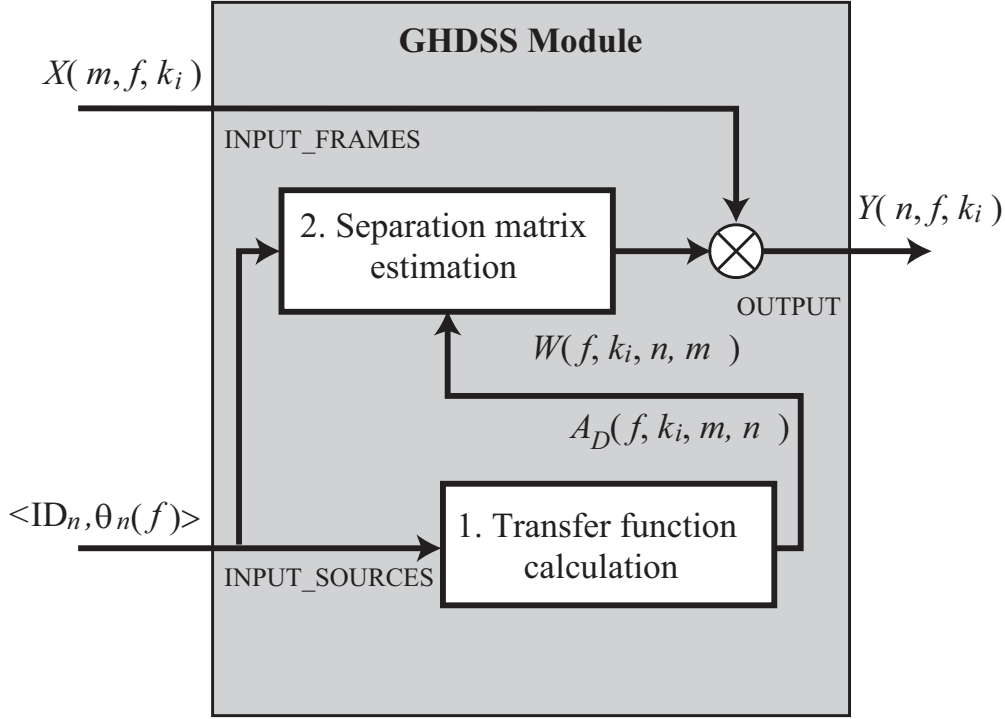


Figure 6.64: Flowchart of GHDSS

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 (??)
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 current 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

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_DISTANCE: Succeed
- Error is more than UPDATE_ACCEPT_DISTANCE: 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 ??.

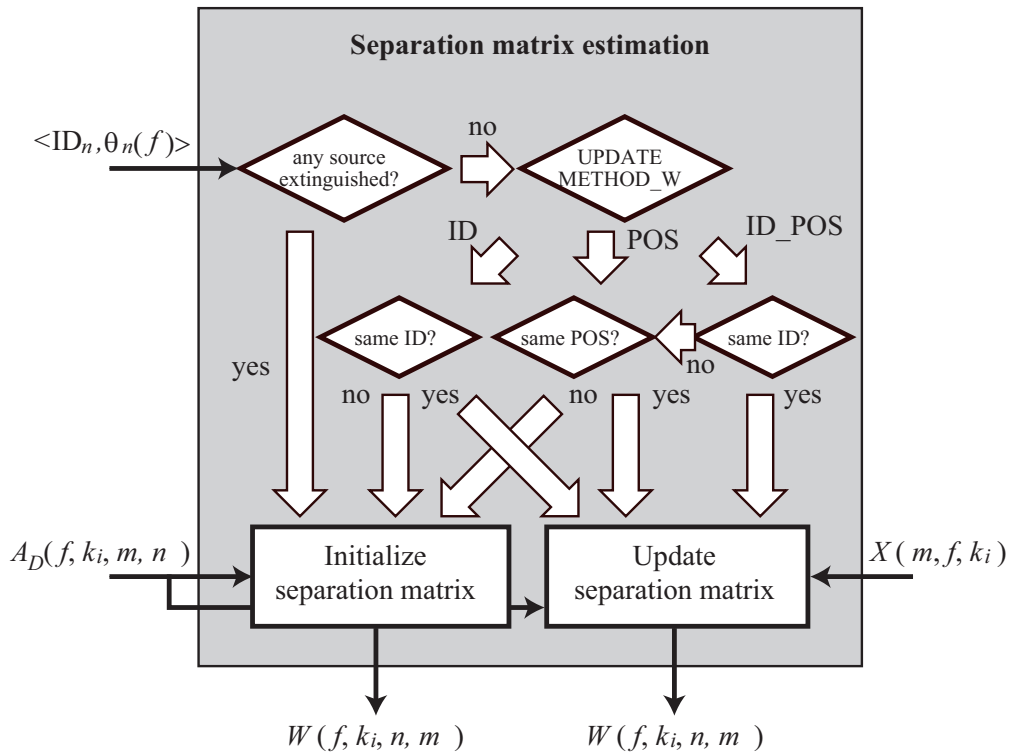


Figure 6.65: Flowchart of separation matrix estimation

Here, the previous frame is updated based on Equation (??) or an initial value of the separation matrix is derived by the transfer function based on Equation (??).

- 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.

— 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_DISTANCE: Update W
 - Error is more than UPDATE_ACCEPT_DISTANCE: 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_DISTANCE: Update W
 - Error more than UPDATE_ACCEPT_DISTANCE: 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_DISTANCE: Overwrite and save W
 - Error more than UPDATE_ACCEPT_DISTANCE: Save additionally W

6.3.7 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 ??, 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 ?? shows a connection example when EstimateLeak is used together.

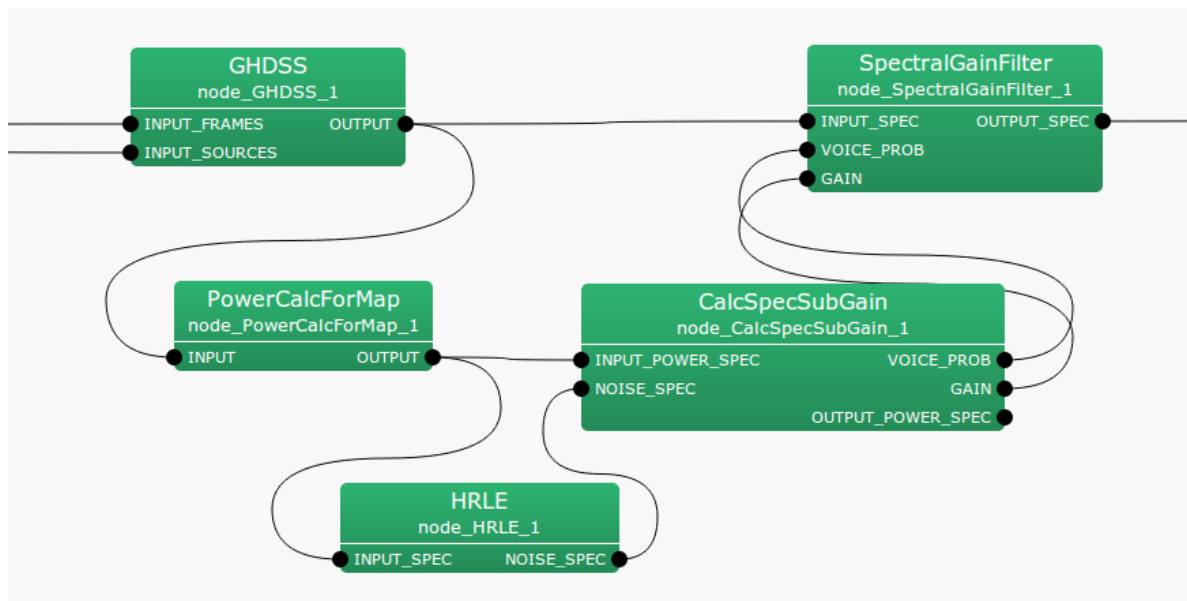


Figure 6.66: Connection example of HRLE 1

Input-output and property of the node

Input

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

Output

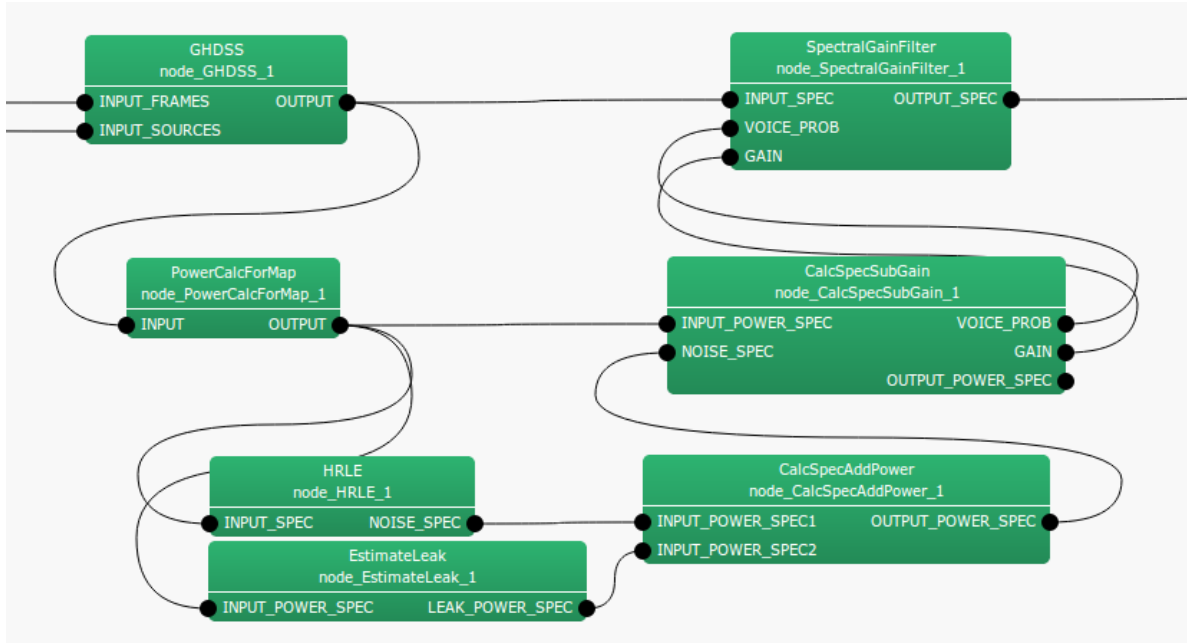


Figure 6.67: Connection example of HRLE 2

Table 6.54: Parameter list of HRLE

Parameter name	Type	Default value	Unit	Description
LX	float	0.85		Normalization accumulation frequency (Lx value).
TIME_CONSTANT_METHOD	string	LEGACY		Time constant value definition.
TIME_CONSTANT	float		[pt]	Time constant.
DECAY_FACTOR	int		[ms]	Time constant.
ADVANCE	int	160	[pt]	Shift length of a frame.
SAMPLING_RATE	int	16000	[Hz]	Sampling frequency.
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.

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

Parameters

LX : float type. 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. The default value is 0.85.

TIME_CONST_METHOD : string type. Time constant value definition. "LEGACY" uses time constant value in time sample unit,"MILLISECOND" uses time constant value in milliseconds. The default value is LEGACY.

TIME_CONSTANT : float type. A time constant (more than zero) is designated in time sample unit.

DECAY_FACTOR : int type. A time constant (more than zero) is designated in milliseconds.

ADVANCE : int type. Shift length of a frame [samples], which must be equal to the values at a preceding node (e.g. AudioStreamFromMic or the MultiFFT). The default value is 160.

SAMPLING_RATE : int type. Sampling frequency of the input waveform [Hz]. The default value is 16000.

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

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

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

DEBUG : bool type. Designate the debugging mode. The default value is *false*. When it is set to *true*, the values of the cumulative histogram are output every 100 frames in the comma-separated text file format to the standard output. 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:

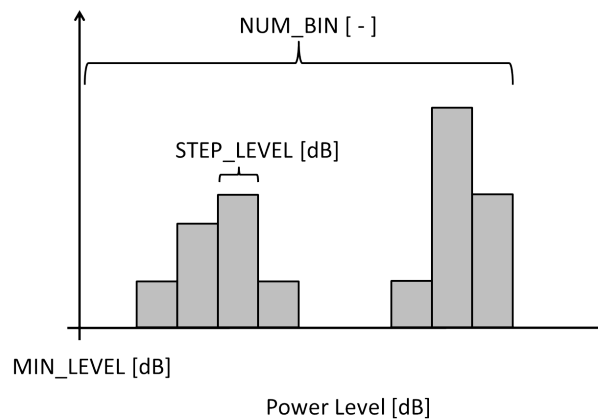


Figure 6.68: Parameters (NUM_BIN, MIN_LEVEL, STEP_LEVEL)

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

Details of the node

Figure ?? shows a processing flow of HRLE. HRLE obtains a level histogram from the input power and estimates the L_x level from the cumulative distribution. The L_x level, as shown in Figure ??, is the level that normalization accumulation frequency in an accumulation frequency distribution becomes x . x is a parameter. For example, when $x = 0$, the minimum value is estimated, when $x = 1$, maximum value is estimated and when $x = 0.5$, a median is estimated in its processing. The details of the processing in HRLE are expressed by the following seven equations

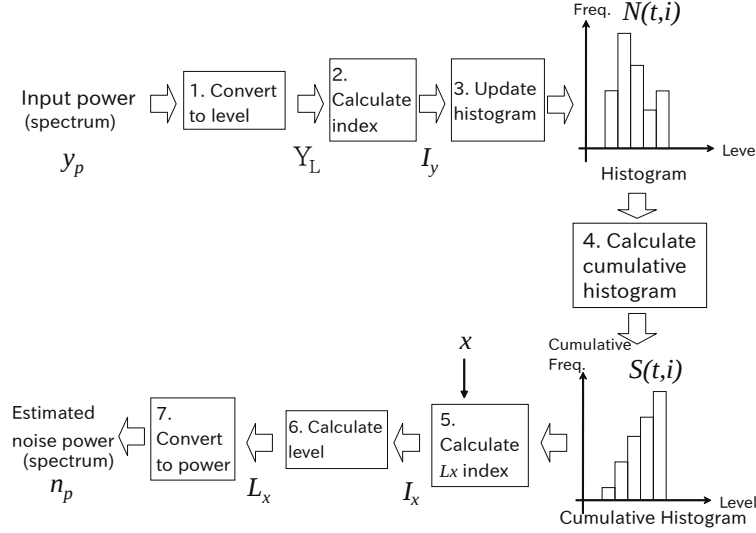


Figure 6.69: Processing flow of HRLE

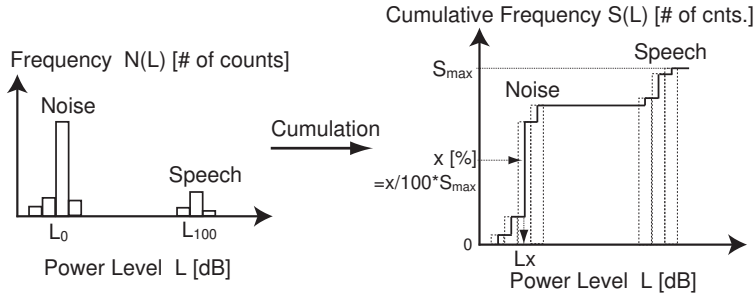


Figure 6.70: Estimation of L_x value

(corresponding to Figure ??). 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.

$$Y_L(t) = 10 \log_{10} y_p(t), \quad (6.71)$$

$$I_y(t) = \lfloor (Y_L(t) - L_{min}) / L_{step} \rfloor, \quad (6.72)$$

$$N(t, l) = \alpha N(t-1, l) + (1 - \alpha) \delta(l - I_y(t)), \quad (6.73)$$

$$S(t, l) = \sum_{k=0}^l N(t, k), \quad (6.74)$$

$$I_x(t) = \underset{l}{\operatorname{argmin}} \left[S(t, l) \frac{x}{100} - S(t, l) \right], \quad (6.75)$$

$$L_x(t) = L_{min} + L_{step} \cdot I_x(t), \quad (6.76)$$

$$n_p(t) = 10^{L_x(t)/10} \quad (6.77)$$

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).

6.3.8 ML

Outline of the node

Perform sound source separation using the method of maximum likelihood (Maximum Likelihood estimation). In this algorithm, obtain a separation matrix on condition that likelihood function is maximized assuming that the input signal is a sum of a single target sound source and Gaussian noise. Transfer function information from the sound source to the microphone, the period information of the sound source (detection result of the speech period), and a correlation matrix of the known noise are required.

Node inputs are:

- Multi-channel complex spectrum of mixed sound,
- Direction of localized sound sources,
- A correlation matrix of known noise.

Note outputs are a set of complex spectrum of each separated sound.

Necessary file

Table 6.55: Necessary files for ML

Corresponding parameter name	Description
TF_CONJ_FILENAME	Transfer function of a microphone array

Usage

When to use

This node is used to perform sound source separation on the sound source direction originated using a microphone array. The sound source direction can be either a value estimated by sound source localization or a constant value.

Typical connection

Figure ?? shows a connection example of the ML . The node has three inputs as follows:

1. INPUT_FRAMES takes a multi-channel complex spectrum containing the mixture of sounds produced by for example MultiFFT ,
2. INPUT_SOURCES takes the results of sound source localization produced by for example LocalizeMUSIC or ConstantLocalization ,
3. INPUT_NOISE_CM takes a correlation matrix of known noise produced by for example CMLoad .

The output is the separated signals.

Input-output and property of the node

Input

INPUT_FRAMES : Matrix<complex<float> > type. Multi-channel complex spectra. Corresponding to the complex spectrum of input waveform from each microphone, the rows correspond to the channels and the columns correspond to the frequency bins.

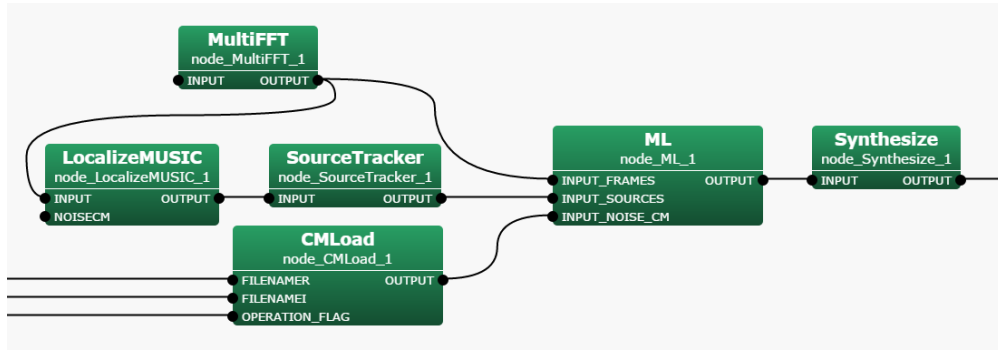


Figure 6.71: Network Example using ML

INPUT_SOURCES : `Vector<ObjectRef>` type. A Vector array of the Source type object in which sound source localization results are stored. Typically, takes the output of `SourceIntervalExtender` connected to `SourceTracker`.

INPUT_NOISE_CM : `Matrix<complex<float>>` type. A correlation matrix for each frequency bin. The rows represent the frequency bin ($NFFT/2 + 1$ rows) and the columns represent the M -th order complex square correlation array ($M * M$ columns).

Output

OUTPUT : `Map<int, ObjectRef>` type. A pair containing the sound source ID and the complex spectrum of the separated sound (`Vector<complex<float>>` type). Output as many as the number of sound sources.

Parameter

LENGTH : `int` type. Analysis frame length [samples], which must be equal to the values at a preceding node (e.g. `AudioStreamFromMic` or the `MultiFFT`). The default is 512.

ADVANCE : `int` type. Shift length of a frame [samples], which must be equal to the values at a preceding node (e.g. `AudioStreamFromMic` or the `MultiFFT`). The default is 160.

SAMPLING_RATE : `int` type. Sampling frequency of the input waveform [Hz]. The default is 16000.

LOWER_BOUND_FREQUENCY : `int` type. This parameter is the minimum frequency used when separation 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 : `int` type. This parameter is the maximum frequency used when separation 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 ?? for the detail of the file format.

REG_FACTOR : `float` type. The coefficient. See the equation (??). The default value is 0.0001.

ENABLE_DEBUG : `bool` type. The default value is `false`. Setting the value to `true` outputs the separation status to the standard output.

Table 6.56: Parameter list of ML

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 frequency value used for separation processing.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	The maximum frequency value used for separation processing.
TF_CONJ_FILENAME	string			File name of transfer function database of your microphone array.
REG_FACTOR	float	0.0001		The coefficient 鐽 ¥ 愁 See the equation (??)
ENABLE_DEBUG	bool	false		Enable or disable to output the separation status to standard output.

Details of the node

Technical details: Please refer to the following reference for the details.

Brief explanation of sound source separation: Table ?? shows the notation of variables used in sound source separation problems. Since the source separation is performed frame-by-frame in the frequency domain, all the variable is computed in a complex field. Also, the separation is performed for all K frequency bins ($1 \leq k \leq K$). Here, we omit k from the notation. Let N , M , and f denote the number of sound sources and the number of microphones, and the frame index, respectively.

Table 6.57: Notation of variables

Variables	Description
$\mathbf{S}(f) = [S_1(f), \dots, S_N(f)]^T$	Complex spectrum of target sound sources at the f -th frame.
$\mathbf{X}(f) = [X_1(f), \dots, X_M(f)]^T$	Complex spectrum of a microphone observation at the f -th frame, which corresponds to INPUT_FRAMES.
$\mathbf{N}(f) = [N_1(f), \dots, N_M(f)]^T$	Complex spectrum of added noise.
$\mathbf{H} = [\mathbf{H}_1, \dots, \mathbf{H}_N] \in \mathbb{C}^{M \times N}$	Transfer function matrix from the n -th sound source ($1 \leq n \leq N$) to the m -th microphone ($1 \leq m \leq M$).
$\mathbf{K}(f) \in \mathbb{C}^{M \times M}$	Correlation matrix of known noise.
$\mathbf{W}(f) = [\mathbf{W}_1, \dots, \mathbf{W}_M] \in \mathbb{C}^{N \times M}$	Separation matrix at the f -th frame.
$\mathbf{Y}(f) = [Y_1(f), \dots, Y_N(f)]^T$	Complex spectrum of separated signals.

Use the following linear model for the signal processing:

$$\mathbf{X}(f) = \mathbf{H}\mathbf{S}(f) + \mathbf{N}(f) \quad (6.78)$$

The purpose of the separation is to estimate $\mathbf{W}(f)$ based on the following equation:

$$\mathbf{Y}(f) = \mathbf{W}(f)\mathbf{X}(f) \quad (6.79)$$

so that $\mathbf{Y}(f)$ is getting close to $\mathbf{S}(f)$.

The separation matrix \mathbf{W}_{ML} based on the maximum likelihood method is expressed by the following equation.

$$\mathbf{W}_{\text{ML}}(f) = \frac{\tilde{\mathbf{K}}^{-1}(f)\mathbf{H}}{\mathbf{H}^H \tilde{\mathbf{K}}^{-1}(f)\mathbf{H}} \quad (6.80)$$

$\tilde{\mathbf{K}}(f)$ can be expressed as below.

$$\tilde{\mathbf{K}}(f) = \mathbf{K}(f) + \|\mathbf{K}(f)\|_{\text{F}} \alpha \mathbf{I} \quad (6.81)$$

$\|K(f)\|_F$ is the Frobenius norm of the known noise correlation matrix $K(f)$ 罫々修 α is the REG_FACTOR 罫々修 and I is an identity matrix.

Trouble shooting: Basically, follow the GHDSS node troubleshooting.

Reference

- [1] F. Asano: 'Array signal processingfor acoustics —Localization, tracking and separation of sound sources—, The Acoustical Society of Japan, 2011.

6.3.9 MSNR

Outline of the node

Perform sound source separation using the method of maximum SNR (Maximum Signal-to-Noise Ratio). In this algorithm, perform sound source separation by updating the separation matrix so that the ratio of the gain in the target sound source direction and the gain in the known noise direction is maximized. Transfer function information from the sound source to the microphones in advance is not required; however, the section information on the sound source (detection result of the utterance section) is necessary.

Node inputs are:

- Multi-channel complex spectrum of mixed sound,
- Direction of localized sound sources.

Note outputs are a set of complex spectrum of each separated sound.

Necessary files

No files are required.

Usage

When to use

This node is used to perform sound source separation on the sound source direction originated using a microphone array. The sound source direction can be either a value estimated by sound source localization or a constant value. Since this node uses the ratio of the gain in the target sound source and the gain in the known noise, it requires the speech period information of the known noise. This node treats the time period during which no sound source direction data is input as the period with noise, the Noise Period.

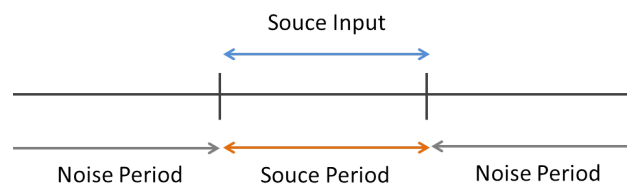


Figure 6.72: Noise Period vs Source Period

Typical connection

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

1. INPUT_FRAMES takes a multi-channel complex spectrum containing the mixture of sounds produced by for example MultiFFT ,
2. INPUT_SOURCES takes the results of sound source localization produced by for example LocalizeMUSIC or ConstantLocalization ,

The output is the separated signals.

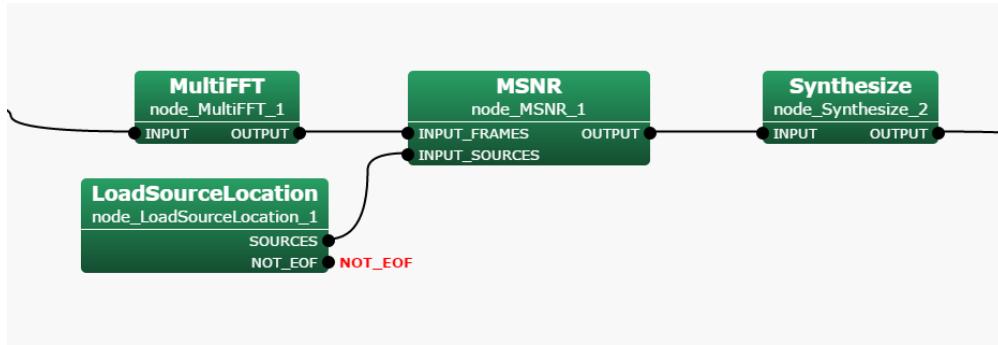


Figure 6.73: Example of connection of the MSNR

Input-output and property of the node

Input

INPUT_FRAMES : `Matrix<complex<float> >` type. Multi-channel complex spectra. Corresponding to the complex spectrum of input waveform from each microphone, the rows correspond to the channels and the columns correspond to the frequency bins.

INPUT_SOURCES : `Vector<ObjectRef>` type. A Vector array of the Source type object in which sound source localization results are stored. Typically, takes the output of `SourceIntervalExtender` connected to `SourceTracker`.

Output

OUTPUT : `Map<int, ObjectRef>` type. A pair containing the sound source ID and the complex spectrum of the separated sound (`Vector<complex<float> >` type). Output as many as the number of sound sources.

Parameter

LENGTH : `int` type. Analysis frame length [samples], which must be equal to the values at a preceding node (e.g. `AudioStreamFromMic` or the `MultiFFT`). The default is 512.

ADVANCE : `int` type. Shift length of a frame [samples], which must be equal to the values at a preceding node (e.g. `AudioStreamFromMic` or the `MultiFFT`). The default is 160.

SAMPLING_RATE : `int` type. Sampling frequency of the input waveform [Hz]. The default is 16000.

LOWER_BOUND_FREQUENCY : `int` type. This parameter is the minimum frequency used when separation 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 : `int` type. This parameter is the maximum frequency used when separation 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.

DECOMPOSITION_ALGORITHM : `string` type. The decomposition algorithm to perform sound source separation. GEVD represents generalized eigenvalue decomposition. GSVD represents generalized singular value decomposition. GEVD has better noise suppression performance than GSVD whereas GEVD costs longer calculation time than GSVD. Select the appropriate algorithm according to the purpose and the computer environment.

ALPHA : `float` type. The stepsize for updating correlation matrices. The default value is 0.99.

ENABLE_DEBUG : `bool` type. The default value is `false`. Setting the value to `true` outputs the separation status to the standard output.

Table 6.58: Parameter list of MSNR

Parameter list	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 frequency value used for separation processing.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	The maximum frequency value used for separation processing.
DECOMPOSITION_ALGORITHM	string	GEVD		The decomposition algorithm.
ALPHA	float	0.99		The stepsize for updating correlation matrices.
ENABLE_DEBUG	bool	false		Enable or disable to output the separation status to standard output.

Details of the node

Technical details: Please refer to the following reference for the details.

Brief explanation of sound source separation: Table ?? shows the notation of variables used in sound source separation problems. Since the source separation is performed frame-by-frame in the frequency domain, all the variable is computed in a complex field. Also, the separation is performed for all K frequency bins ($1 \leq k \leq K$). Here, we omit k from the notation. Let N , M , and f denote the number of sound sources and the number of microphones, and the frame index, respectively.

Table 6.59: Notation of variables

Variables	Description
$\mathbf{S}(f) = [S_1(f), \dots, S_N(f)]^T$	Complex spectrum of target sound sources at the f -th frame.
$\mathbf{X}(f) = [X_1(f), \dots, X_M(f)]^T$	Complex spectrum of a microphone observation at the f -th frame, which corresponds to INPUT_FRAMES.
$\mathbf{N}(f) = [N_1(f), \dots, N_M(f)]^T$	Complex spectrum of added noise.
$\mathbf{H} = [\mathbf{H}_1, \dots, \mathbf{H}_N] \in \mathbb{C}^{M \times N}$	Transfer function matrix from the n -th sound source ($1 \leq n \leq N$) to the m -th microphone ($1 \leq m \leq M$)
$\mathbf{K}(f) \in \mathbb{C}^{M \times M}$	Correlation matrix of known noise.
$\mathbf{W}(f) = [\mathbf{W}_1, \dots, \mathbf{W}_M] \in \mathbb{C}^{N \times M}$	Separation matrix at the f -th frame.
$\mathbf{Y}(f) = [Y_1(f), \dots, Y_N(f)]^T$	Complex spectrum of separated signals.

Use the following linear model for the signal processing:

$$\mathbf{X}(f) = \mathbf{H}\mathbf{S}(f) + \mathbf{N}(f) \quad (6.82)$$

The purpose of the separation is to estimate $\mathbf{W}(f)$ based on the following equation:

$$\mathbf{Y}(f) = \mathbf{W}(f)\mathbf{X}(f) \quad (6.83)$$

so that $\mathbf{Y}(f)$ is getting close to $\mathbf{S}(f)$.

Assuming that the correlation matrix of the target sound signal is $\mathbf{R}_{ss}(f)$ and the correlation matrix of the noise signal is $\mathbf{R}_{nn}(f)$, the evaluation function $J_{\text{MSNR}}(\mathbf{W}(f))$ for updating the separation matrix is expressed as follows.

$$J_{\text{MSNR}}(\mathbf{W}(f)) = \frac{\mathbf{W}(f)\mathbf{R}_{ss}(f)\mathbf{W}(f)^H}{\mathbf{W}(f)\mathbf{R}_{nn}(f)\mathbf{W}(f)^H} \quad (6.84)$$

In the MSNR, obtain $\mathbf{W}(f)$ that maximizes $J_{\text{MSNR}}(\mathbf{W}(f))$ using generalized eigenvalue decomposition or generalized singular value decomposition.

Here, the correlation matrix $\mathbf{R}_{ss}(f)$ of the target sound signal is updated as follows using the correlation matrix $\mathbf{R}_{xx}(f)$ obtained from the signal in the period during which the INPUT_SOURCES input terminal receives target sound source direction data.

$$\mathbf{R}_{ss}(f + 1) = \alpha \mathbf{R}_{ss}(f) + (1 - \alpha) \mathbf{R}_{xx}(f) \quad (6.85)$$

On the other hand, the correlation matrix $\mathbf{R}_{nn}(f)$ of the noise signal is updated as follows using the correlation matrix $\mathbf{R}_{xx}(f)$ obtained from the signal in the period (the Noise Period) during which the INPUT_SOURCES input terminal receives no data.

$$\mathbf{R}_{nn}(f + 1) = \alpha \mathbf{R}_{nn}(f) + (1 - \alpha) \mathbf{R}_{xx}(f) \quad (6.86)$$

The α in the equation (??) and the equation (??) can be specified in the ALPHA property.

$\mathbf{W}(f)$ is updated by $\mathbf{R}_{ss}(f)$ and $\mathbf{R}_{nn}(f)$ and so separated.

Trouble shooting: Basically, same as GHDSS node troubleshooting.

Reference

- [1] P. W. Howells, 'Intermediate Frequency Sidelobe Canceller', U.S. Patent No.3202990, 1965.

6.3.10 MVDR

Outline of the node

Perform sound source separation using the method of Minimum Variance Distortionless Response, MVDR. In this algorithm, obtain a separation matrix that minimizes the output power under the linear constraint condition that does not distort the target sound source. Transfer function information from the sound source to the microphone, the period information of the sound source (detection result of the speech period), and a correlation matrix of the known noise are required.

Node inputs are:

- Multi-channel complex spectrum of mixed sound,
- Direction of localized sound sources,
- A correlation matrix of known noise.

Note outputs are a set of complex spectrum of each separated sound.

Necessary file

Table 6.60: Necessary files for MVDR

Corresponding parameter name	Description
TF_CONJ_FILENAME	Transfer function of a microphone array.

Usage

When to use

This node is used to perform sound source separation on the sound source direction originated using a microphone array. The sound source direction can be either a value estimated by sound source localization or a constant value.

Typical connection

Figure ?? shows a connection example of the MVDR . The node has three inputs as follows:

1. INPUT_FRAMES takes a multi-channel complex spectrum containing the mixture of sounds produced by for example MultiFFT ,
2. INPUT_SOURCES takes the results of sound source localization produced by for example LocalizeMUSIC or ConstantLocalization ,
3. INPUT_NOISE_CM takes a correlation matrix of known noise produced by for example CMLoad .

The output is the separated signals.

Input-output and property of the node

Input

INPUT_FRAMES : Matrix<complex<float> > type. Multi-channel complex spectra. Corresponding to the complex spectrum of input waveform from each microphone, the rows correspond to the channels and the columns correspond to the frequency bins.

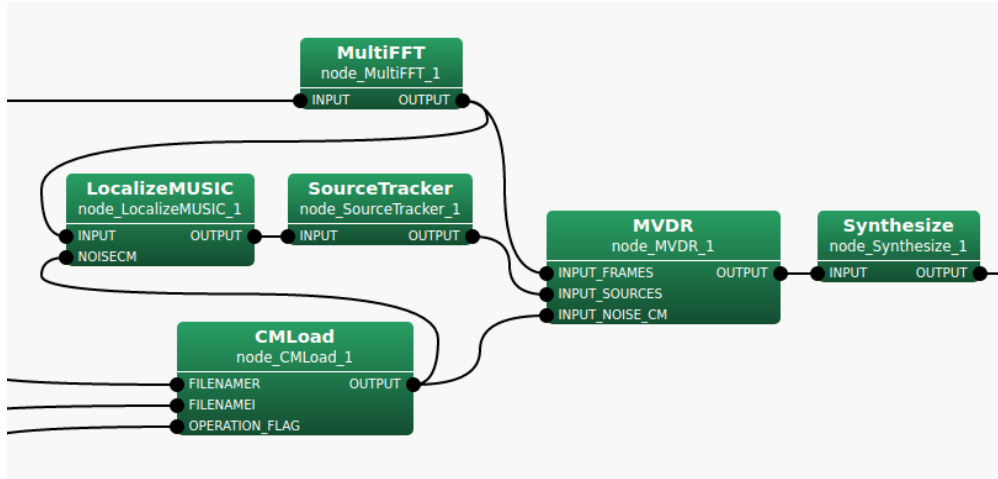


Figure 6.74: Network Example using MVDR

INPUT_SOURCES : `Vector<ObjectRef>` type. A Vector array of the Source type object in which sound source localization results are stored. Typically, takes the output of `SourceIntervalExtender` connected to `SourceTracker`.

INPUT_NOISE_CM : `Matrix<complex<float> >` type. A correlation matrix for each frequency bin. The rows represent the frequency bin ($NFFT/2 + 1$ rows) and the columns represent the M -th order complex square correlation array ($M * M$ columns). This input is required. If there is no need for correlation matrix input, connect `CMIdentityMatrix` to generate the correlation matrix.

Output

OUTPUT : `Map<int, ObjectRef>` type. A pair containing the sound source ID and the complex spectrum of the separated sound (`Vector<complex<float> >` type). Output as many as the number of sound sources.

Parameter

LENGTH : `int` type. Analysis frame length [samples], which must be equal to the values at a preceding node (e.g. `AudioStreamFromMic` or the `MultiFFT`). The default is 512.

ADVANCE : `int` type. Shift length of a frame [samples], which must be equal to the values at a preceding node (e.g. `AudioStreamFromMic` or the `MultiFFT`). The default is 160.

SAMPLING_RATE : `int` type. Sampling frequency of the input waveform [Hz]. The default is 16000.

LOWER_BOUND_FREQUENCY : `int` type. This parameter is the minimum frequency used when separation 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 : `int` type. This parameter is the maximum frequency used when separation 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 ?? for the detail of the file format.

REG_FACTOR : `float` type. The coefficient. See the equation (?). The default value is 0.001.

ENABLE_DEBUG : `bool` type. The default value is `false`. Setting the value to `true` outputs the separation status to the standard output.

Table 6.61: Parameter list of MVDR

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 frequency value used for separation processing.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	The maximum frequency value used for separation processing.
TF_CONJ_FILENAME	string			File name of transfer function database of your microphone array.
REG_FACTOR	float	0.001		The coefficient. See the equation(?).
ENABLE_DEBUG	bool	false		Enable or disable to output the separation status to standard output.

Details of the node

Technical details: Please refer to the following reference for the details.

Brief explanation of sound source separation: Table ?? shows the notation of variables used in sound source separation problems. Since the source separation is performed frame-by-frame in the frequency domain, all the variable is computed in a complex field. Also, the separation is performed for all K frequency bins ($1 \leq k \leq K$). Here, we omit k from the notation. Let N , M , and f denote the number of sound sources and the number of microphones, and the frame index, respectively.

Table 6.62: Notation of variables

Variables	Description
$\mathbf{S}(f) = [S_1(f), \dots, S_N(f)]^T$	Complex spectrum of target sound sources at the f -th frame.
$\mathbf{X}(f) = [X_1(f), \dots, X_M(f)]^T$	Complex spectrum of a microphone observation at the f -th frame, which corresponds to INPUT_FRAMES.
$\mathbf{N}(f) = [N_1(f), \dots, N_M(f)]^T$	Complex spectrum of added noise.
$\mathbf{H} = [\mathbf{H}_1, \dots, \mathbf{H}_N] \in \mathbb{C}^{M \times N}$	Transfer function matrix from the n -th sound source ($1 \leq n \leq N$) to the m -th microphone ($1 \leq m \leq M$)
$\mathbf{K}(f) \in \mathbb{C}^{M \times M}$	Correlation matrix of known noise.
$\mathbf{W}(f) = [\mathbf{W}_1, \dots, \mathbf{W}_M] \in \mathbb{C}^{N \times M}$	Separation matrix at the f -th frame.
$\mathbf{Y}(f) = [Y_1(f), \dots, Y_N(f)]^T$	Complex spectrum of separated signals.

Use the following linear model for the signal processing:

$$\mathbf{X}(f) = \mathbf{H}\mathbf{S}(f) + \mathbf{N}(f) \quad (6.87)$$

The purpose of the separation is to estimate $\mathbf{W}(f)$ based on the following equation:

$$\mathbf{Y}(f) = \mathbf{W}(f)\mathbf{X}(f) \quad (6.88)$$

so that $\mathbf{Y}(f)$ is getting close to $\mathbf{S}(f)$.

The separation matrix \mathbf{W}_{MVDR} based on the MVDR method is expressed by the following equation.

$$\mathbf{W}_{\text{MVDR}}(f) = \frac{\tilde{\mathbf{K}}^{-1}(f)\mathbf{H}}{\mathbf{H}^H \tilde{\mathbf{K}}^{-1}(f)\mathbf{H}} \quad (6.89)$$

$\tilde{\mathbf{K}}(f)$ can be expressed as below.

$$\tilde{\mathbf{K}}(f) = \mathbf{K}(f) + \alpha \mathbf{I} \quad (6.90)$$

Here, α is the REG_FACTOR parameter, \mathbf{I} is a diagonal matrix to perform normalization of the correlation matrix.

Trouble shooting: Basically, follow the GHDSS node trouble shooting.

Reference

- [1] F. Asano: 'Array signal processing for acoustics —Localization, tracking and separation of sound sources—, The Acoustical Society of Japan, 2011.

6.3.11 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.

Usage

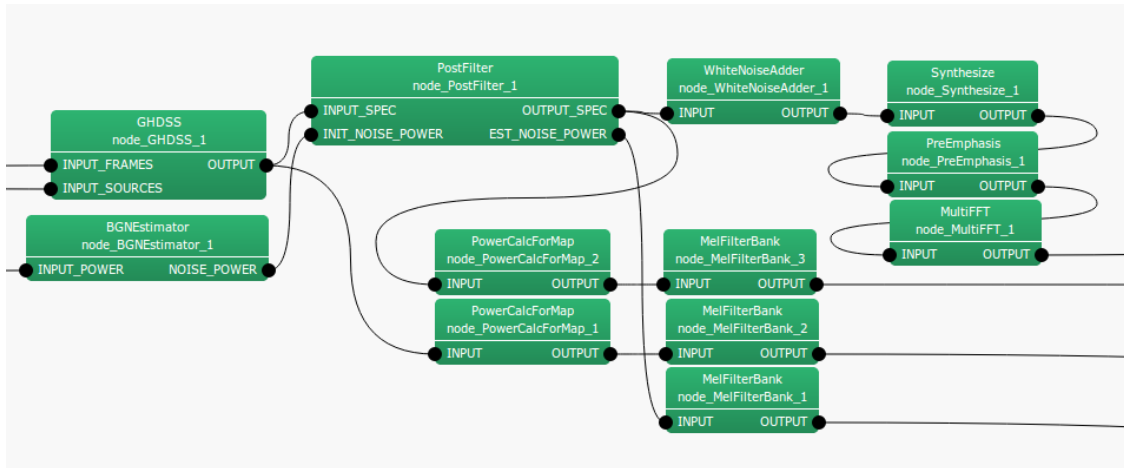
When to use

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 ?? 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 ?? 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 6.63: 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 6.64: 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

The subscripts used in the equations are based on the definitions in Table ?? . Moreover, the time frame index f is abbreviated in the following equations unless especially needed. Figure ?? 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 ?? shows the processing flow of noise estimation . The three kinds of noise that the 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.

$$\lambda(f, k_i) = \lambda^{sta}(f, k_i) + \lambda^{leak}(f, k_i) + \lambda^{rev}(f - 1, k_i) \quad (6.91)$$

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.

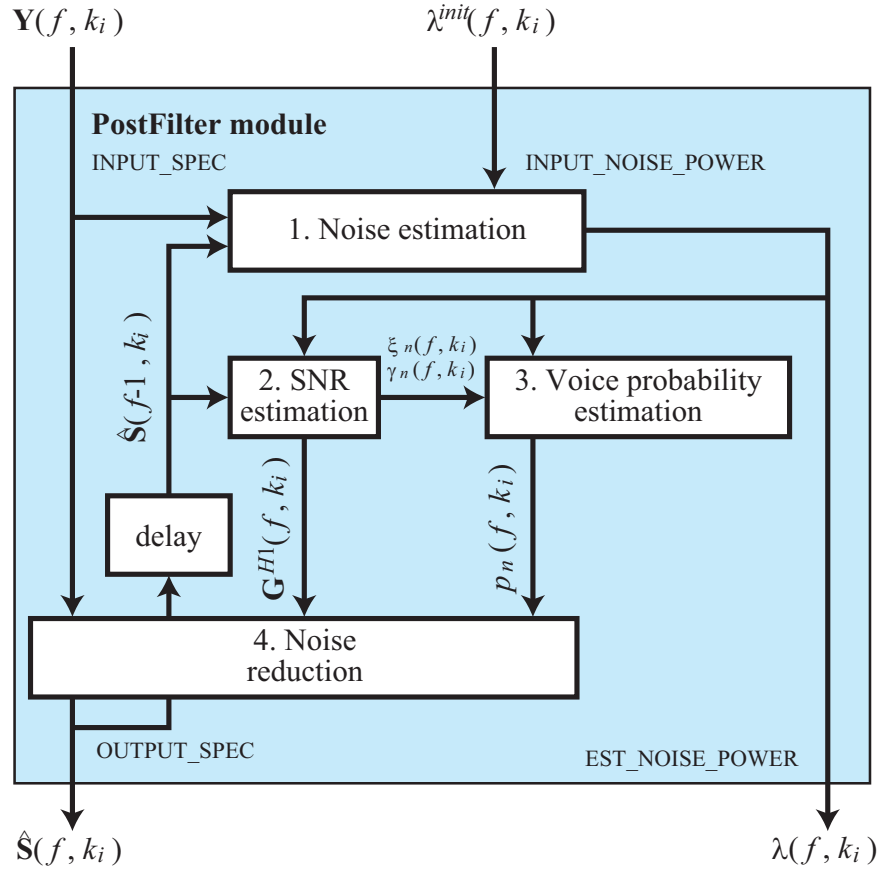


Figure 6.75: Flowchart of PostFilter

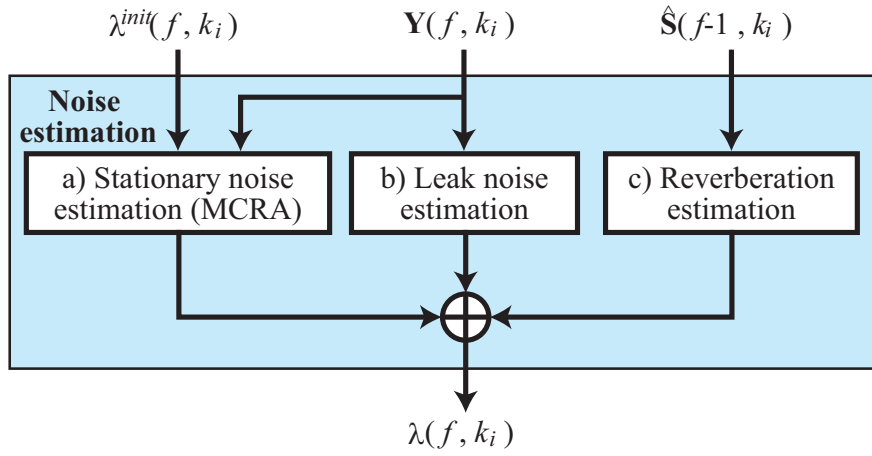


Figure 6.76: Procedure of noise estimation

Table 6.65: Definition of variable

Parameter	Description, Corresponding parameter
$Y(k_i) = [Y_1(k_i), \dots, Y_N(k_i)]^T$	Complex spectrum of separated sound corresponding to the frequency bin k_i
$\lambda^{init}(k_i) = [\lambda_1^{init}(k_i), \dots, \lambda_N^{init}(k_i)]^T$	Initial value power spectrum used for the stationary noise estimation
$\lambda^{sta}(k_i) = [\lambda_1^{sta}(k_i), \dots, \lambda_N^{sta}(k_i)]^T$	Estimated stationary noise power spectrum.
α_s	Smoothing coefficient of the input power spectrum. Parameter SPEC_SMOOTH_FACTOR. The default value is 0.5
$S^{tmp}(k_i) = [S_1^{tmp}(k_i), \dots, S_N^{tmp}(k_i)]$	Temporary parameter for minimum power calculation.
$S^{min}(k_i) = [S_1^{min}(k_i), \dots, S_N^{min}(k_i)]$	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

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

$$S_n(f, k_i) = \alpha_s S_n(f-1, k_i) + (1 - \alpha_s) |Y_n(k_i)|^2 \quad (6.92)$$

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

$$S_n^{min}(f, k_i) = \begin{cases} \min\{S_n^{min}(f-1, k_i), S_n(f, k_i)\} & \text{if } f \neq nL \\ \min\{S_n^{tmp}(f-1, k_i), S_n(f, k_i)\} & \text{if } f = nL \end{cases}, \quad (6.93)$$

$$S_n^{tmp}(f, k_i) = \begin{cases} \min\{S_n^{tmp}(f-1, k_i), S_n(f, k_i)\} & \text{if } f \neq nL \\ S_n(f, k_i) & \text{if } f = nL \end{cases}, \quad (6.94)$$

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.

$$S_n^r(k_i) = \frac{S_n(k_i)}{S_n^{min}(k_i)}, \quad (6.95)$$

$$I_n(k_i) = \begin{cases} 1 & \text{if } S_n^r(k_i) > \delta \\ 0 & \text{if } S_n^r(k_i) \leq \delta \end{cases} \quad (6.96)$$

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.

$$\alpha_{d,n}^C(k_i) = (\alpha_d - 1)I_n(k_i) + 1. \quad (6.97)$$

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

$$S_n^{leak}(k_i) = \sum_{p=1}^N |Y_p(k_i)|^2 - |Y_n(k_i)|^2, \quad (6.98)$$

$$S_n^0(k_i) = |Y_n(k_i)|^2 - qS_n^{leak}(k_i), \quad (6.99)$$

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

$$S_n^0(k_i) = S_{floor} \quad (6.100)$$

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 .

$$\lambda_n^{sta}(f, k_i) = \begin{cases} \alpha_{d,n}^C(k_i)\lambda_n^{sta}(f-1, k_i) + (1 - \alpha_{d,n}^C(k_i))rS_n^0(f, k_i) & \text{nochange in source position} \\ \alpha_{d,n}^C(k_i)\lambda_n^{init}(f, k_i) + (1 - \alpha_{d,n}^C(k_i))rS_n^0(f, k_i) & \text{if Change in source position} \end{cases} \quad (6.101)$$

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

Table 6.66: 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 \times OVER.CANCEL_FACTOR
$S_n(f, k_i)$	Smoothing power spectrum obtained by Equation (??)

Some parameters are calculated as follows.

$$\beta = -\frac{\alpha^{leak}}{1 - (\alpha^{leak})^2 + \alpha^{leak}(1 - \alpha^{leak})(N-2)} \quad (6.102)$$

$$\alpha = 1 - (N-1)\alpha^{leak}\beta \quad (6.103)$$

With this parameter, mix the smoothed spectrum $S(k_i)$, 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 (??).

$$Z_n(k_i) = \alpha S_n(k_i) + \beta S_n^{leak}(k_i), \quad (6.104)$$

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.

$$\lambda_n^{leak} = \alpha^{leak} \left(\sum_{n' \neq n} Z_{n'}(k_i) \right) \quad (6.105)$$

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

$$\lambda_n^{leak} = \alpha^{leak} \left(\sum_{n' \neq n} Z_{n'}(k_i) \right) \quad (6.106)$$

Table 6.67: Definition of variable

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

$$\lambda_n^{rev}(f, k_i) = \gamma \left(\lambda_n^{rev}(f-1, k_i) + \Delta |\hat{S}_n(f-1, k_i)|^2 \right) \quad (6.107)$$

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

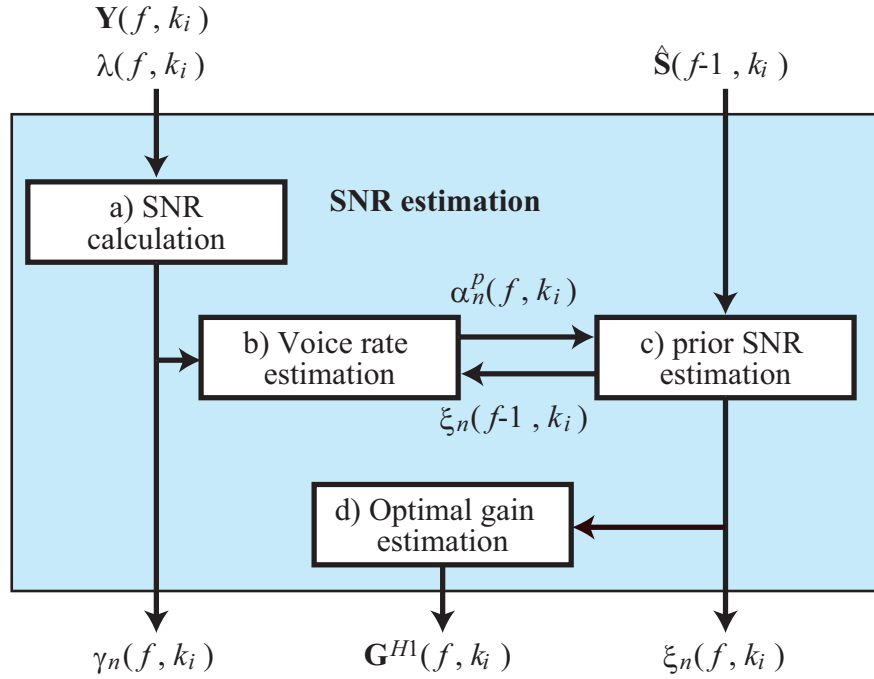


Figure 6.77: Procedure of SNR estimation

- a) Calculation of SNR
- b) Preliminary SNR estimation before noise mixture
- c) Estimation of a speech content rate
- d) Estimation of an optimal gain

Table 6.68: 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

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

2-a) Calculation of SNR The variables used in 2-a) are based on Table ?. 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.

$$\gamma_n(k_i) = \frac{|Y_n(k_i)|^2}{\lambda_n(k_i)} \quad (6.108)$$

$$\gamma_n^C(k_i) = \begin{cases} \gamma_n(k_i) & \text{if } \gamma_n(k_i) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (6.109)$$

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 ???. The speech content rate

Table 6.69: Definition of variable

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

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

$$\alpha_n^p(f, k_i) = \alpha_{mag}^p \left(\frac{\xi_n(f-1, k_i)}{\xi_n(f-1, k_i) + 1} \right)^2 + \alpha_{min}^p \quad (6.110)$$

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

Table 6.70: 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.

$$\xi_n(k_i) = (1 - \alpha_n^p(k_i)) \xi_{imp} + \alpha_n^p(k_i) \gamma_n^C(k_i) \quad (6.111)$$

$$\xi_{imp} = a \frac{|\hat{S}_n(f-1, k_i)|^2}{\lambda_n(f-1, k_i)} + (1 - a) \xi_n(f-1, k_i) \quad (6.112)$$

Here, ξ_{imp} 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 ???. Prior to calculating an optimal

Table 6.71: 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 $\gamma_n(k_i)$.

$$v_n(k_i) = \frac{\xi_n(k_i)}{1 + \xi_n(k_i)} \gamma_n(k_i) \quad (6.113)$$

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

$$G_n^{H1}(k_i) = \frac{\xi_n(k_i)}{1 + \xi_n(k_i)} \exp \left\{ \frac{1}{2} \int_{v_n(k_i)}^{\infty} \frac{e^{-t}}{t} dt \right\} \quad (6.114)$$

Here,

$$\begin{aligned} G_n^{H1}(k_i) &= 1 & \text{if } v_n(k_i) < \theta^{min} \\ G_n^{H1}(k_i) &= 1 & \text{if } G_n^{H1}(k_i) > 1. \end{aligned} \quad (6.115)$$

3) Estimation of probability of speech presence Figure ?? shows the flow of estimation of probability of speech

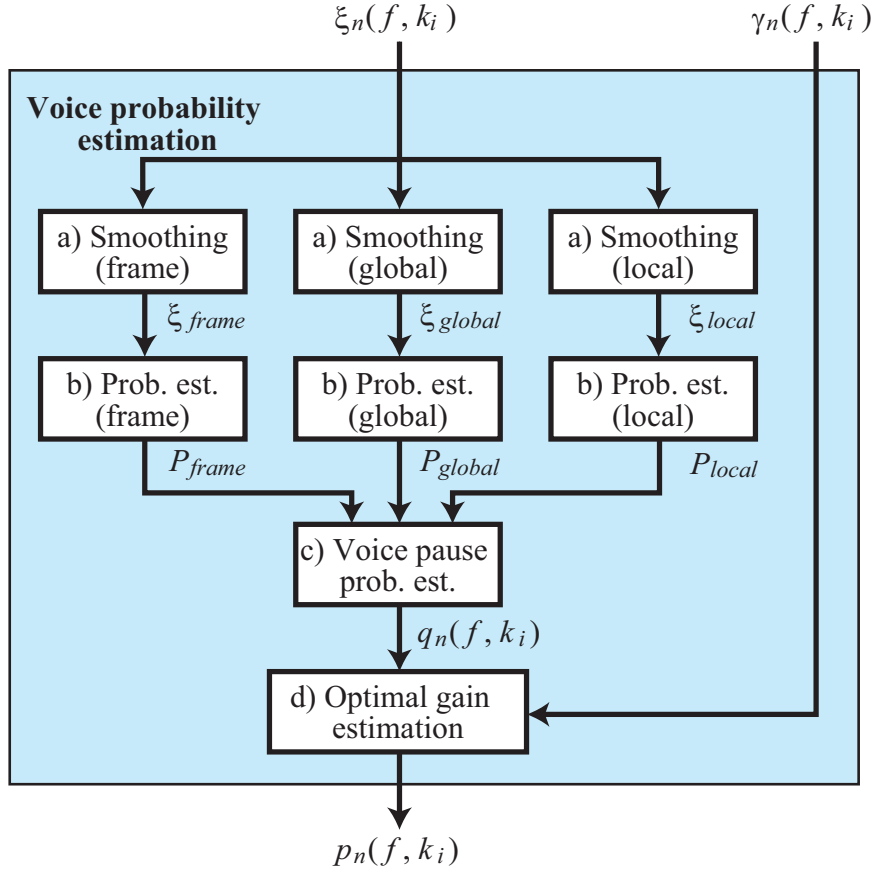


Figure 6.78: Procedure for estimation of probability of speech presence:

presence. Estimation of the probability of speech presence consists of:

- Smoothing of the preliminary SNR for each of the 3 types of bands
- Estimation with the temporal probability of speech presence based on the smoothed SNR in each band
- Speech quiescent probability is estimated based on three provisional probability.
- Estimation of the final probability of speech presence.

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

$$\zeta_n(f, k_i) = b\zeta_n(f - 1, k_i) + (1 - b)\xi_n(f, k_i) \quad (6.116)$$

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

Table 6.72: 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 by averaging is performed in the frequency bin range $F_{st} \sim F_{en}$.

$$\zeta_n^f(k_i) = \frac{1}{F_{en} - F_{st} + 1} \sum_{k_j=F_{st}}^{F_{en}} \zeta_n(k_j) \quad (6.117)$$

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

$$\zeta_n^g(k_i) = \sum_{j=-(G-1)/2}^{(G-1)/2} w_{han}(j + (G-1)/2) \zeta_n(k_{i+j}), \quad (6.118)$$

$$w_{han}(j) = \frac{1}{C} \left(0.5 - 0.5 \cos \left(\frac{2\pi j}{G} \right) \right), \quad (6.119)$$

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.

$$\zeta_n^l(k_i) = 0.25\zeta_n(k_i - 1) + 0.5\zeta_n(k_i) + 0.25\zeta_n(k_i + 1) \quad (6.120)$$

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

Table 6.73: 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_SNR, 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.

$$\zeta_n^{peak}(f, k_i) = \begin{cases} \zeta_n^f(f, k_i), & \text{if } \zeta_n^f(f, k_i) > Z_{thres} \zeta_n^f(f-1, k_i) \\ \zeta_n^{peak}(f-1, k_i), & \text{if otherwise.} \end{cases} \quad (6.121)$$

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,

$$\zeta_n^{peak}(k_i) = \begin{cases} Z_{min}^{peak}, & \text{if } \zeta_n^{peak}(k_i) < Z_{min}^{peak} \\ Z_{max}^{peak}, & \text{if } \zeta_n^{peak}(k_i) > Z_{max}^{peak} \end{cases} \quad (6.122)$$

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

$$P_n^f(k_i) = \begin{cases} 0, & \text{if } \zeta_n^f(k_i) < \zeta_n^{peak}(k_i) Z_{min}^f \\ 1, & \text{if } \zeta_n^f(k_i) > \zeta_n^{peak}(k_i) Z_{max}^f \\ \frac{\log(\zeta_n^f(k_i)/\zeta_n^{peak}(k_i) Z_{min}^f)}{\log(Z_{max}^f/Z_{min}^f)}, & \text{otherwise} \end{cases} \quad (6.123)$$

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

$$P_n^g(k_i) = \begin{cases} 0, & \text{if } \zeta_n^g(k_i) < Z_{min}^g \\ 1, & \text{if } \zeta_n^g(k_i) > Z_{max}^g \\ \frac{\log(\zeta_n^g(k_i)/Z_{min}^g)}{\log(Z_{max}^g/Z_{min}^g)}, & \text{otherwise} \end{cases} \quad (6.124)$$

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

$$P_n^l(k_i) = \begin{cases} 0, & \text{if } \zeta_n^l(k_i) < Z_{min}^l \\ 1, & \text{if } \zeta_n^l(k_i) > Z_{max}^l \\ \frac{\log(\zeta_n^l(k_i)/Z_{min}^l)}{\log(Z_{max}^l/Z_{min}^l)}, & \text{otherwise} \end{cases} \quad (6.125)$$

3-c) Estimation of the probability of speech pause The variables used in 3-c) are shown in Table ?? . As shown

Table 6.74: 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 $P_n^{f,g,l}(k_i)$.

$$q_n(k_i) = 1 - \left(1 - a^l + a^l P_n^l(k_i)\right) \left(1 - a^g + a^g P_n^g(k_i)\right) \left(1 - a^f + a^f P_n^f(k_i)\right), \quad (6.126)$$

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 (??).

$$p_n(k_i) = \left\{ 1 + \frac{q_n(k_i)}{1 - q_n(k_i)} (1 + \zeta_n(k_i)) \exp(-v_n(k_i)) \right\}^{-1} \quad (6.127)$$

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 $Y_n(k_i)$.

$$\hat{S}_n(k_i) = Y_n(k_i)G_n^{H1}(k_i)p_n(k_i) \quad (6.128)$$

6.3.12 SemiBlindICA

Overview

This module removes a known signal (e.g., the utterance from a spoken dialogue system) from the multichannel observed mixture. The algorithm presented in Reference ⁽¹⁾ is implemented in this module.

Necessary files

N/A.

Usage

When to use

Here is an example in the context of a spoken dialogue system. When a spoken dialogue system dispenses with a close-talk microphone, the microphone may capture the mixture of the utterance of the user of the system and the utterance from the system itself because the microphone is located in a distance from the mouth of the user. In this situation, the speech recognition quality is degraded because the input signal captured by the system contains both the target voice of the user and interfering system utterance.

Generally speaking, when the multichannel observed signal recorded by a microphone array contains a known signal component, we can remove the known signal component from the observed mixture signal. In the example above, the system utterance corresponds to the known signal component. Here, a “known” signal means that we know the waveform of the signal when it is played. For example, when we have a wav file to play with a loudspeaker, the signal is known. Usually, the waveform of an observed signal by a microphone is usually different from the waveform that was played from a loudspeaker. This is because the spatial propagation of the sound changes the waveform and adds some reverberation. Besides, there is some time difference of arrival due to the distance between the sound source (e.g., a loudspeaker) and receivers (e.g., microphones). Since SemiBlindICA module explicitly models this propagation of the known sounds, the original waveform being played is sufficient information to remove the known signal component.

Typical connection

Figures ?? and ?? show the typical usage of SemiBlindICA node. In Figure ??, the multichannel observed mixture including the known and unknown signal components is connected to INPUT. The known component is connected to REFERENCE terminal. These signals are converted into the time-frequency domain with MultiFFT node. From OUTPUT terminal, the unknown signal component is produced where the REFERENCE is removed from INPUT. This OUTPUT can be connected to LocalizeMUSIC node for further analysis of the unknown component.

Figure ?? illustrates an example that uses a stereo wav file. In the wav file, the first (left) channel contains the known signal component while the second (right) channel contains the mixture of known and unknown components. ChannelSelector node is used to extract each channel from a multichannel wav file. In Figure ??, the separated unknown signal component is finally saved as another wav file using SaveWavePCM node.

Input-output and property of the node

Input

INPUT : Matrix<complex<float> > type. Multichannel observation that contains both known and unknown signal components. This is complex-valued spectra obtained by MultiFFT node.

REFERENCE : Matrix<complex<float> > type. Complex-valued spectra of the known signal component.

Output

OUTPUT : Matrix<complex<float> > type. The output signal is generated by suppressing the known signal REFERENCE from the observed mixture INPUT. This is in the same multichannel format as INPUT terminal.

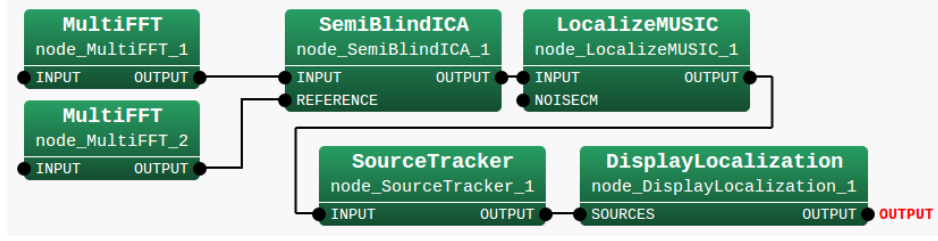


Figure 6.79: Typical connection of SemiBlindICA node

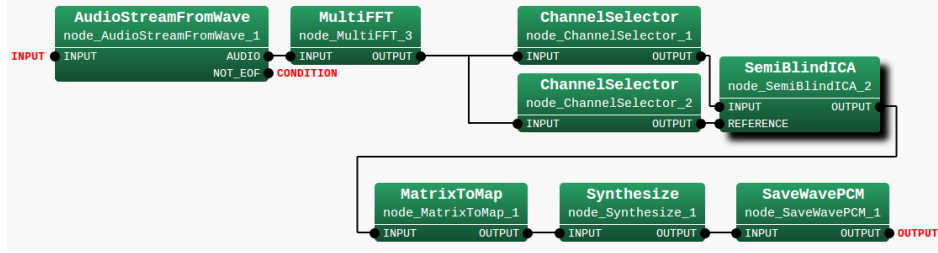


Figure 6.80: Extraction of unknown signal component with SemiBlindICA node by using the left and right channels

Parameter

CHANNEL Number of channels of the input mixture INPUT.

LENGTH Windows length of the short-time Fourier transform. HARK uses 512 [pt] by default.

INTERVAL This parameter adjusts the length of the filter to remove the known signal component depending on the step size of the short-time Fourier transform. This *multirate repeating* is introduced to improve the convergence of the filter learning⁽²⁾. This value is denoted by K in the math expressions below.

TAP_LOWERFREQ The length of the filter at 0 [Hz]. The length of the filter accounts for the time difference between the REFERENCE signal and the known component in INPUT, and the reverberation in the observed signal. When the environment contains a long reverberation, this value is set larger accordingly. This value is denoted by M_L .

TAP_UPPERFREQ The length of the filter at the Nyquist frequency. The filter length at each frequency bin is determined by the linear interpolation of TAP_LOWERFREQ and TAP_UPPERFREQ. This value is denoted by M_H .

DECAY The decaying parameter for the learning rate of each element of the filter to remove the known signal component. In a reverberant environment, such as an indoor situation, the filter element corresponding to the past time frame exponentially decays. Since the filter element values follow an exponential curve, the learning of the filter values becomes efficient when the learning rate for each filter value is also set exponentially decaying values. When this value is set 1, the learning rate becomes identical for all filter values. We empirically set this value at 0.6–0.8. This value is denoted by λ .

MU_FILTER The filter values are obtained though the stochastic gradient method. This is the learning rate for the learning procedure. Generally speaking, a large learning rate is able to drastically update the parameters while there is a risk of the fluctuation of the parameters around the (local) optimum value. On the other hand, a small learning rate can avoid the risk of the fluctuation, while the number of parameter updates may increase before convergence. This is denoted by μ_w .

MU_REFERENCE Learning rate for the normalization parameter of the known signal component. The normalization of the known signal is carried out to accelerate the learning. This value is denoted by μ_α .

MU_UNKNOWNSIGNAL Learning rate for the normalization parameter of the unknown signal component. Similarly to MU_REFERENCE, this normalization is introduced for an efficient learning. This is denoted by μ_β .

Table 6.75: Parameters of SemiBlindICA

Parameter	Type	Default value	Unit	description
CHANNEL	int	1		Number of channels of INPUT terminal
LENGTH	int	512	[pt]	Length of FFT window
INTERVAL	int	1		This parameter adjusts the length of the filter to remove the known signal depending on the step size of short-time Fourier transform. If the overlap of the windows (i.e., the step size is small), take a larger value.
TAP_LOWERFREQ	int	8	[frame]	The length of the filter at 0 Hz frequency bin
TAP_UPPERFREQ	int	4	[frame]	The length of the filter at the Nyquist frequency
DECAY	float	0.8		Decaying parameter for the learning rate of each element of the filter to remove the known signal
MU_FILTER	float	0.01		Learning rate of the filter to remove the known signal component. Specify a positive value.
MU_REFERENCE	float	0.01		Learning rate of the normalization parameter of the known signal. Specify a positive value.
MU_UNKNOWN SIGNAL	float	0.01		Learning rate of the normalization parameter of the unknown signal component. Specify a positive value.
IS_ZERO	float	0.0001		A threshold to detect an active signal in INPUT. This threshold is applied to the power at each time frequency point in the time-frequency domain.
FILE_FILTER_IN	string	-null		File name (path) that contains the filter value used to initialize the processing. If “-null” (the default value) is specified, no file is used for the initialization.
FILE_FILTER_OUT	string	-null		File name (path) to write out the filter values. If “-null” (the default value) is specified, no file is written.
OUTPUT_FREQ	int	150	[frame]	The interval to save the filter values in terms of time frames.

IS_ZERO To save the computational resource, the filter update procedure is omitted if the INPUT contains no signal. This is the threshold to detect the existence of a signal. When the power of the INPUT is below this value, this time frame is ignored. Note that this threshold is applied to the power of the signal in the time-frequency domain, instead of the waveform in the time domain.

FILE_FILTER_IN File name (path) that contains the initial filter values. If “-null” is specified, no file is used for the initialization.

FILE_FILTER_OUT File name (path) to write out the filter values. If “-null” is specified, no file is written out.

OUTPUT_FREQ The interval in terms of time frame to save the filter values.

Detail of the node

SemiBlindICA node suppresses the known signal component from the multichannel observation that contains both the known and unknown signals using independent component analysis (ICA). The ICA algorithm is derived based on

the mixing process in the time-frequency domain and the statistical independence between the known and unknown signals.

Mixing model and the separation process: SemiBlindICA node uses the following mixing process in a reverberant environment. This model is a linear mixing process in the time-frequency domain. Let ω be the frequency bin index, f be the time frame index, and $X(\omega, f)$ be the observed signal at frequency ω and time f . The observation is modeled as

$$X(\omega, f) = N(\omega, f) + \sum_{m=0}^M H(\omega, m) S(\omega, f - m),$$

where $N(\omega, f)$, $S(\omega, f)$, and $H(\omega, m)$ denote the unknown signal, the known signal, and the propagation coefficient with a time lag of m frames.

The separation process is derived as follows using ICA.

$$\begin{aligned} \begin{pmatrix} \hat{N}(\omega, f) \\ \mathbf{S}(\omega, f) \end{pmatrix} &= \begin{pmatrix} a(\omega) & -\mathbf{w}^T(\omega) \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} X(\omega, f) \\ \mathbf{S}(\omega, f) \end{pmatrix}, \\ \mathbf{S}(\omega, f) &= [S(\omega, f), S(\omega, f - K), \dots, S(\omega, f - M(\omega)K)]^T, \\ \mathbf{w}(\omega) &= [w_0(\omega), w_1(\omega), \dots, w_{M(\omega)}(\omega)]^T, \\ M(\omega) &= \text{floor}(\omega/\omega_{\text{nyq}}(M_U - M_L)) + M_L. \end{aligned} \quad (6.129)$$

Here, ω_{nyq} denotes the maximum value of the frequency bin index (corresponding to the Nyquist frequency). The separation filter $\mathbf{w}(\omega)^T$ is a $M(\omega) + 1$ -dimensional filter, where \mathbf{w}^T is the transpose of vector \mathbf{w} . K is the factor for multirate repeating⁽²⁾ introduced for an efficient convergence of the filter.

Estimation of the separation filter: The separation filter is estimated through ICA processing: the filter is obtained by minimizing the Kullback-Leibler divergence (KLD) between the product of the probability density function of \hat{N} and \mathbf{S} , and the joint distribution of these variables. The update procedure is derived by using the nonholonomic constraint⁽³⁾ and natural gradient method. Let \hat{N}_n be the normalized unknown signal. The separation filter to suppress the known signal component is incrementally updated as follows.

$$\begin{aligned} \mathbf{w}(\omega, f + 1) &= \mathbf{w}(\omega, f) + \mu_w \Phi_{\hat{N}_n(\omega)}(\hat{N}_n(\omega, f)) \tilde{\mathbf{S}}_n(\omega, f), \\ a(\omega) &= 1, \end{aligned}$$

where \hat{x} denotes the complex conjugate of x . Here, the higher-order correlation is defined as $\Phi_x(x) = \tanh(|x|)e^{j\theta(x)}$. μ_w is defined as follows.

$$\mu_w = \text{diag}(\mu_w, \mu_w \lambda^{-1}, \dots, \mu_w \lambda^{-M(\omega)}). \quad (6.130)$$

This element-wise learning rate has an exponential decay so as to accelerate the learning of the separation filter. From Eq. (??), the unknown signal component \hat{N} is obtained as follows.

$$\hat{N}(\omega, f) = X(\omega, f) - \mathbf{w}(\omega, f)^T \mathbf{S}_n(\omega, f). \quad (6.131)$$

To satisfy the nonholonomic constraint, \hat{N} should be normalized. This is because the constraint requires $E[1 - \Phi_x(x\alpha_x)\bar{x}\bar{\alpha}_x] = 1$. In the general framework of minimization of KLD based on natural gradient method, variable x has a normalization factor v_x that is incrementally updated as follows:

$$v_x(f + 1) = v_x(f) + \mu_x[1 - \Phi_x(x(f)v_x(f))\bar{x}(f)\bar{v}_x(f)]v_x(f)$$

This is applied to the normalization of the estimated unknown signal component \hat{N} using the factor α as

$$\hat{N}_n(f) = \alpha(f)\hat{N}(f), \quad (6.132)$$

$$\alpha(f + 1) = \alpha(f) + \mu_\alpha[1 - \Phi_{\hat{N}_n}(\hat{N}_n(f))\bar{\hat{N}}_n(f)]\alpha(f). \quad (6.133)$$

For an efficient convergence of the separation filter, SemiBlindICA node normalizes the observation signal using normalization factor β . Similarly to the case of \hat{N} , these quantities are updated as

$$S_n(f) = \beta(f)S(f), \quad (6.134)$$

$$\beta(f+1) = \beta(f) + \mu_\beta[1 - \Phi_{S_n}(S_n(f))\bar{S}_n(f)]\beta(f), \quad (6.135)$$

$$S_n(f) = [S_n(f), S_n(f-K), \dots, S_n(f-MK)].$$

Flow: The main algorithm of SemiBlindICA node consists of Eqs. (??–??) for each frequency bin ω and time frame f . Algorithm ?? summarizes the core procedures at a certain frequency bin and a channel.

Algorithm 1 The core algorithm of SemiBlindICA node

At frequency ω and frame f , the following procedures are carried out.

Calculate $\hat{N}(\omega, f)$ by Eq. (??).

Normalize $\hat{N}(\omega, f)$ and $S(\omega, f)$ by Eqs. (??, ??)

Update the separation filter $w(\omega, f)$ using Eq. (??)

Update the normalization factors $\alpha(\omega, f)$ and $\beta(\omega, f)$ using Eq. (??, ??)

Output $\hat{N}(\omega, f)$

Algorithm ?? presents the overall procedures applied to all the frequency bins and channels. Every time a new time frame is observed, Algorithm ?? is applied to each channel and frequency bin.

Algorithm 2 Overall procedures

The following procedures are carried out with a new observation of a time frame.

$f \leftarrow f + 1$

for ch in $0, \dots, C$ **do**

for ω in $0, \dots, \omega_{nqt}$ **do**

 Run Algorithm ?? for ω, ch .

end for

end for

Reference

- (1) R. Takeda et al., “Barge-in-able Robot Audition Based on ICA and Missing Feature Theory,” in Proc. of IROS, pp. 1718–1723, 2008.
- (2) H. Kiya et al., “Improvement of convergence speed for subband adaptive digital filter using the multirate repeating method,” Electronics and Communications in Japan, Part III, Vol. 78, no. 10, pp. 37–45, 1995.
- (3) C. Choi et al., “Natural gradient learning with nonholonomic constraint for blind deconvolution of multiple channels,” in Proc. of Int’l Workshop on ICA and BBS, pp. 371–376

6.3.13 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 ?? 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.

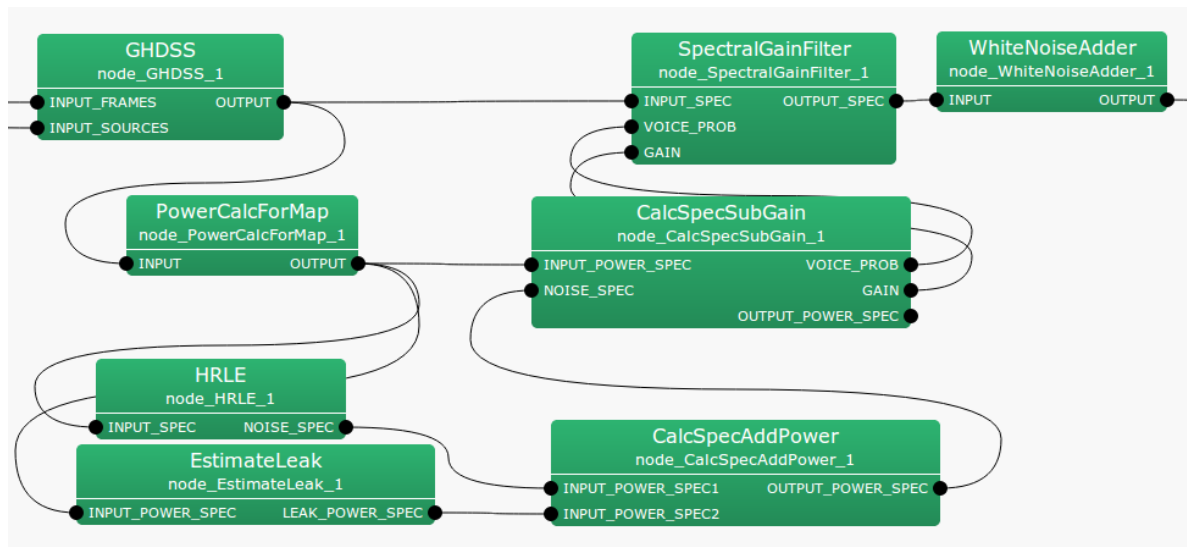


Figure 6.81: Connection example of SpectralGainFilter

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.

$$Y_n(k_i) = X_n(k_i)G_n(k_i)p_n(k_i) \quad (6.136)$$

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

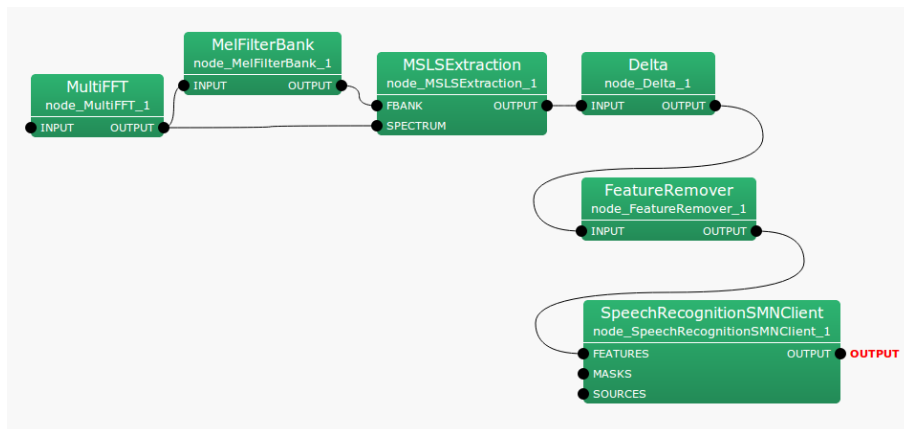


Figure 6.82: Typical connection example for Delta

Input-output and property of the node

Table 6.76: Parameter list of Delta

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	int	13		Dimension number of static feature

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.

$$x(f) = [x(f, 0), x(f, 1), \dots, x(f, P-1)]^T \quad (6.137)$$

Here, P is FBANK_COUNT.

$$y(f) = [x(f, 0), x(f, 1), \dots, x(f, 2P-1)]^T \quad (6.138)$$

Each output vector element is expressed as,

$$y(f, p) = \begin{cases} x(f, p), & \text{if } p = 0, \dots, P-1, \\ w \sum_{\tau=-2}^2 \tau \cdot x(f+\tau, p), & \text{if } p = P, \dots, 2P-1, \end{cases} \quad (6.139)$$

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

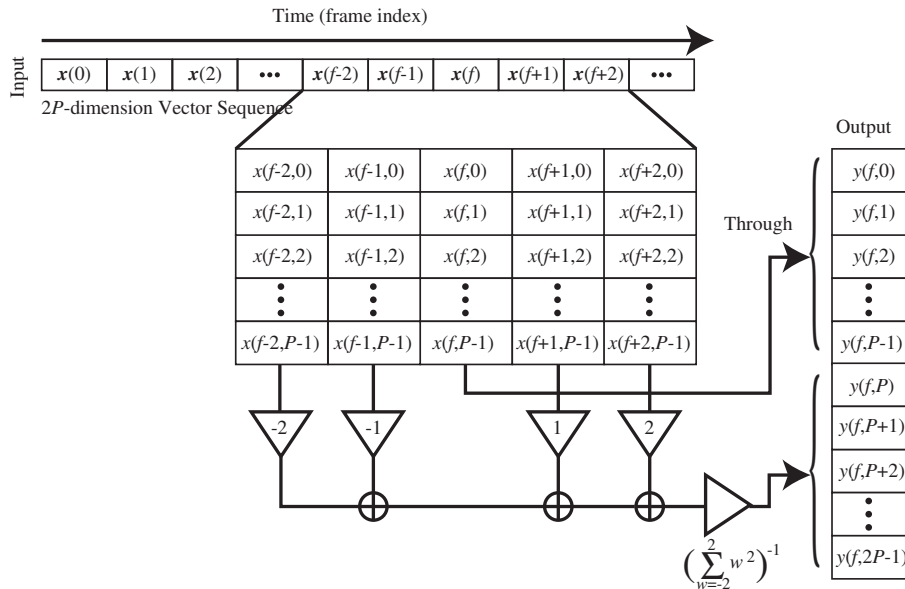


Figure 6.83: Input-output flow of Delta .

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.

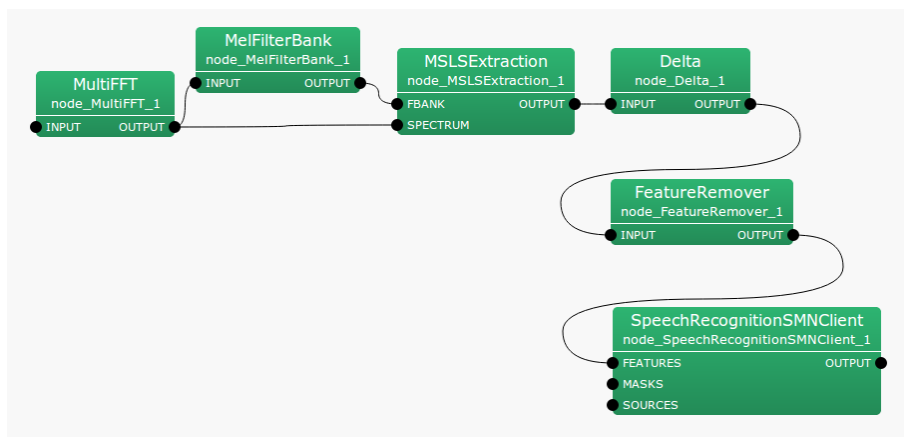


Figure 6.84: Typical connection example of FeatureRemover

Input-output and property of the node

Table 6.77: 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

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 MSLSEExtraction .

Typical connection

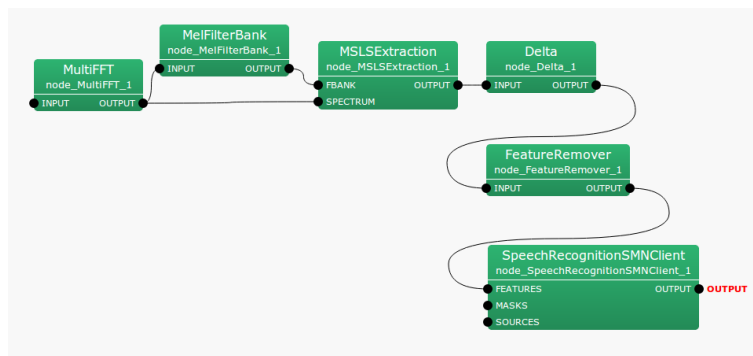


Figure 6.85: Connection example of MelFilterBank

Input-output and property of the node

Table 6.78: 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
CUTOFF	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.

CUTOFF : 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 CUTOFF.

MAX_FREQUENCY : int type. Upper limit frequency of the filter bank. Its range is positive integers and less than CUTOFF.

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_RATECUTOFF/LENGTH. Transformation of the linear scale and mel scale is expressed as follows.

$$m = 1127.01048 \log(1.0 + \frac{\lambda}{700.0}) \quad (6.140)$$

However, expression on the linear scale is λ (Hz) and that on the mel scale is m . Figure ?? shows an example of the transformation by 8000 Hz. The red points indicate the center frequency of each bank when SAMPLING_RATE is 16000Hz, CUTOFF 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 ?? 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 ?? 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 ?? 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.

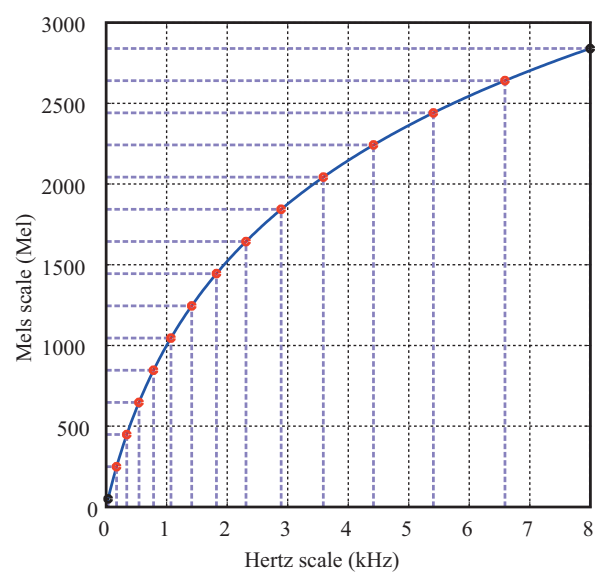


Figure 6.86: Correspondence between linear scale and a mel scale

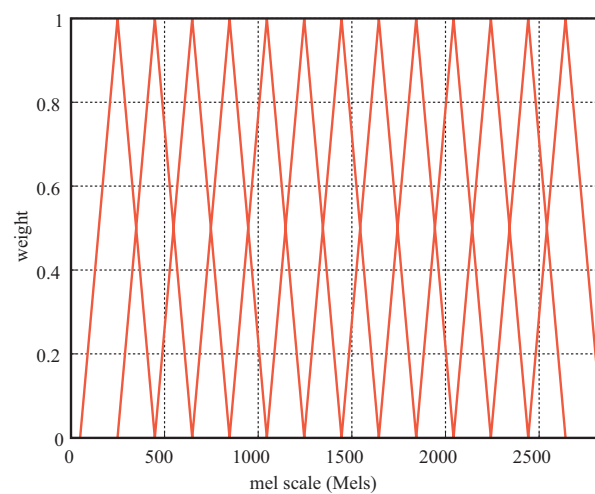


Figure 6.87: Window function on mel scale

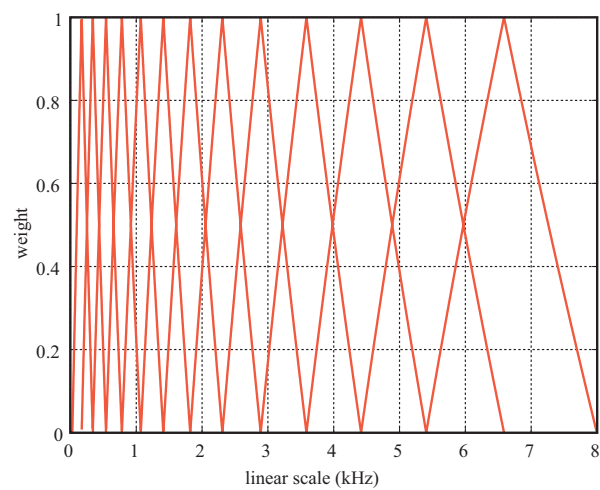


Figure 6.88: Window function on linear scale

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

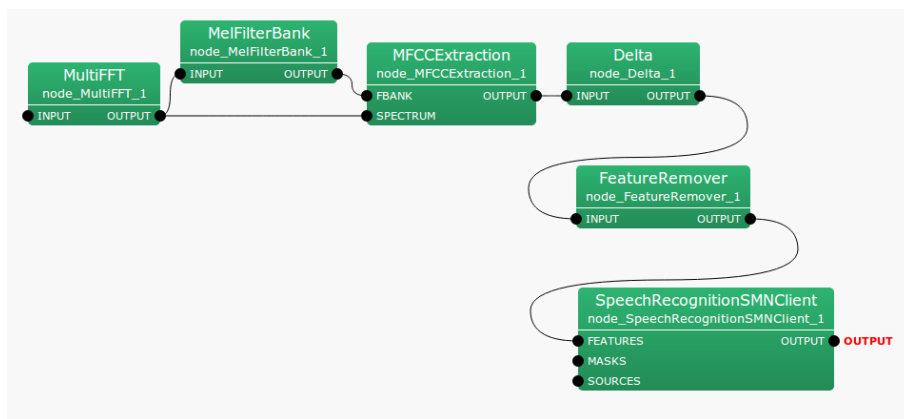


Figure 6.89: Typical connection example of MFCCExtraction

Input-output and property of the node

Table 6.79: 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.

$$x(f) = [x(f, 0), x(f, 1), \dots, x(f, P - 1)]^T \quad (6.141)$$

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 P th 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;

$$y(f) = [y(f, 0), y(f, 1), \dots, y(f, P - 1), E]^T \quad (6.142)$$

$$y(f, p) = L(p) \cdot \sqrt{\frac{2}{P}} \cdot \sum_{q=0}^{P-1} \left\{ \log(x(q)) \cos\left(\frac{\pi(p+1)(q+0.5)}{P}\right) \right\} \quad (6.143)$$

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

$$L(p) = 1.0 + \frac{Q}{2} \sin\left(\frac{\pi(p+1)}{Q}\right), \quad (6.144)$$

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

$$s = [s(0), \dots, s(K - 1)]^T, \quad (6.145)$$

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

$$E = \log\left(\frac{1}{K} \sum_{k=0}^{K-1} s(k)\right) \quad (6.146)$$

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

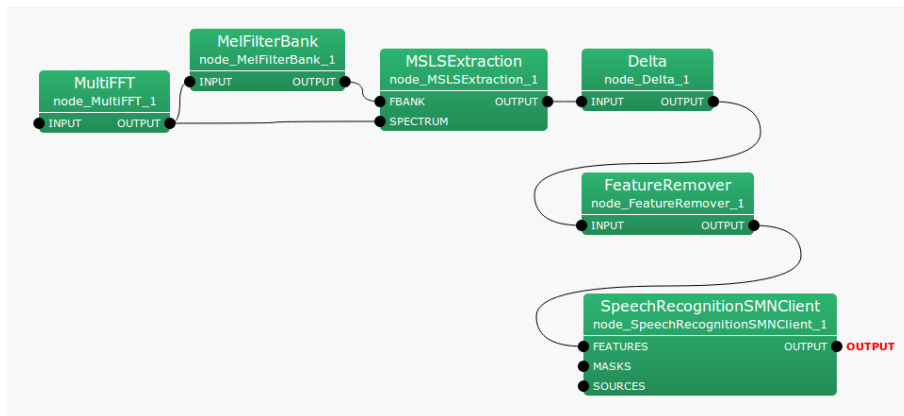


Figure 6.90: Typical connection example of MSLSExtraction

Input-output and property of the node

Table 6.80: 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 is optimized to the value 13.
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 ?? 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 default 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:

$$x = [x(0), x(1), \dots, x(P-1)]^T \quad (6.147)$$

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;

$$y = [y(0), y(1), \dots, y(P-1), E]^T \quad (6.148)$$

$$y(p) = \frac{1}{P} \sum_{q=0}^{P-1} \left\{ L(q) \cdot \sum_{r=0}^{P-1} \left\{ \log(x(r)) \cos\left(\frac{\pi q(r+0.5)}{P}\right) \right\} \cos\left(\frac{\pi q(p+0.5)}{P}\right) \right\} \quad (6.149)$$

Here, the liftering coefficient is expressed as;

$$L(p) = \begin{cases} 1.0, & (p = 0, \dots, P-1), \\ 0.0, & (p = P, \dots, 2P-1), \end{cases} \quad (6.150)$$

Here, $Q = 22$.

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

$$x = [x(0), x(1), \dots, x(P-1)]^T \quad (6.151)$$

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 P th 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:

$$y = [y(0), y(1), \dots, y(P-1), E]^T \quad (6.152)$$

$$y(p) = \begin{cases} (\log(x(p)) - \mu) - 0.9(\log(x(p-1)) - \mu), & \text{if } p = 1, \dots, P-1 \\ \log(x(p)), & \text{if } p = 0, \end{cases} \quad (6.153)$$

$$\mu = \frac{1}{P} \sum_{q=0}^{P-1} \log(x(q)), \quad (6.154)$$

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:

$$s = [s(0), s(1), \dots, s(N-1)]^T \quad (6.155)$$

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:

$$p = \log\left(\frac{1}{N} \sum_{n=0}^{N-1} s(n)\right) \quad (6.156)$$

2P-dimension vector sequence

Static feature (P-dimensional vector)	{	$y(f-2,0)$	$y(f-1,0)$	\dots	$y(f-2,0)$	$y(f-1,0)$	$y(f,0)$	$y(f+1,0)$	$y(f+2,0)$
		$y(f-2,1)$	$y(f-1,1)$	\dots	$y(f-2,1)$	$y(f-1,1)$	$y(f,1)$	$y(f+1,1)$	$y(f+2,1)$
		$y(f-2,2)$	$y(f-1,2)$	\dots	$y(f-2,2)$	$y(f-1,2)$	$y(f,2)$	$y(f+1,2)$	$y(f+2,2)$
		\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
		$y(f-2,P-1)$	$y(f-1,P-1)$	\dots	$y(f-2,P-1)$	$y(f-1,P-1)$	$y(f,P-1)$	$y(f+1,P-1)$	$y(f+2,P-1)$
Dynamic feature (P-dimensional vector)	{	$y(f-2,P)$	$y(f-1,P)$	\dots	$y(f-2,P)$	$y(f-1,P)$	$y(f,P)$	$y(f+1,P)$	$y(f+2,P)$
		$y(f-2,P+1)$	$y(f-1,P+1)$	\dots	$y(f-2,P+1)$	$y(f-1,P+1)$	$y(f,P+1)$	$y(f+1,P+1)$	$y(f+2,P+1)$
		$y(f-2,P+2)$	$y(f-1,P+2)$	\dots	$y(f-2,P+2)$	$y(f-1,P+2)$	$y(f,P+2)$	$y(f+1,P+2)$	$y(f+2,P+2)$
		\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
		$y(f-2,2P-1)$	$y(f-1,2P-1)$	\dots	$y(f-2,2P-1)$	$y(f-1,2P-1)$	$y(f,2P-1)$	$y(f+1,2P-1)$	$y(f+2,2P-1)$

Time (frame index) \rightarrow

*Shadowed elements are filled with ZERO.

Figure 6.91: 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

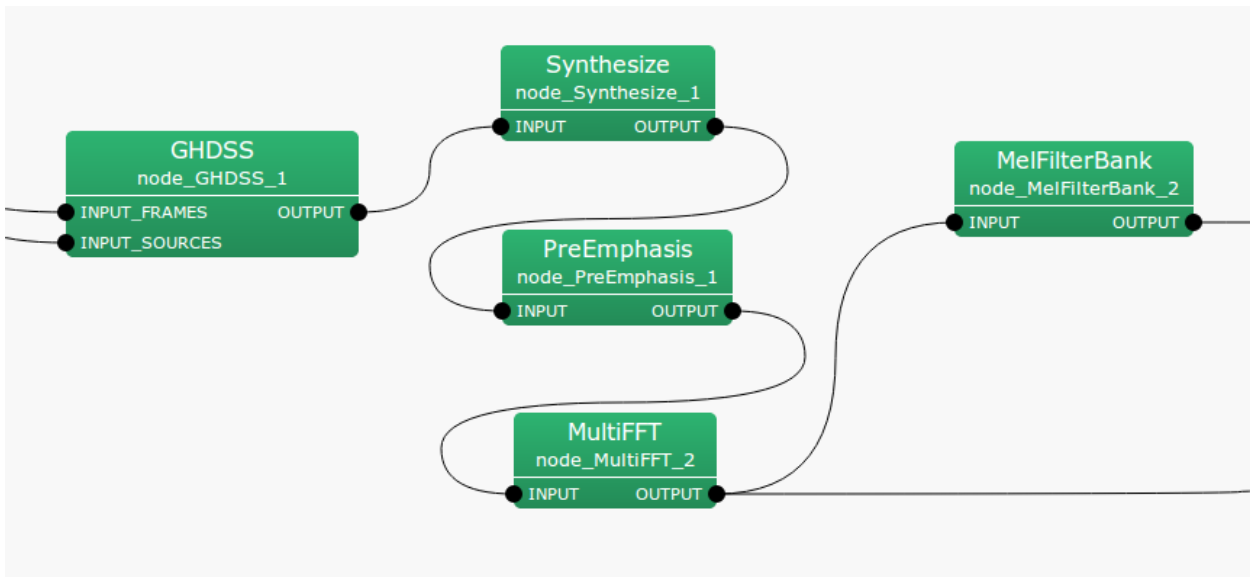


Figure 6.92: Connection example of PreEmphasis

Input-output and property of the node

Table 6.81: 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.

$$p[t] = \begin{cases} s[t] - c_p \cdot s[t-1] & t > 0 \\ (1 - c_p) \cdot s[0] & t = 0 \end{cases} \quad (6.157)$$

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 $fs/2 - 100\text{Hz}$) 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

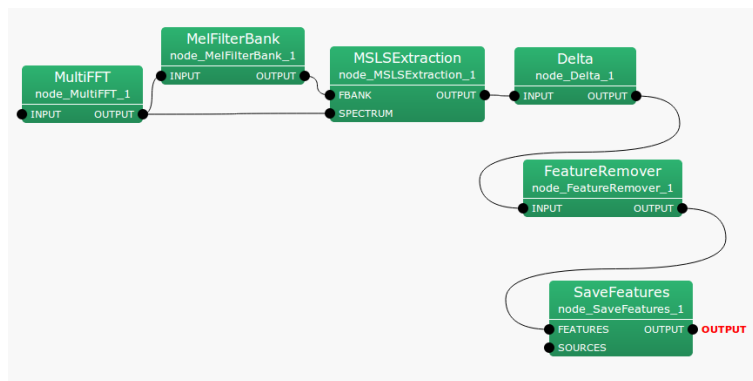


Figure 6.93: Connection example of SaveFeatures

Input-output and property of the node

Table 6.82: 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 754 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

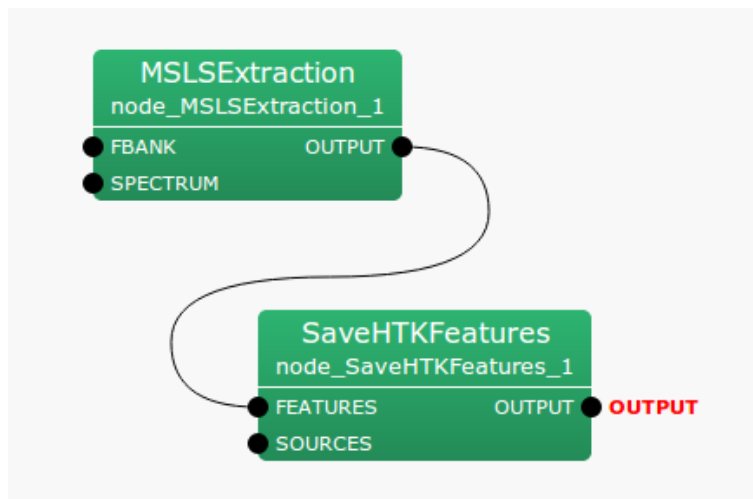


Figure 6.94: Connection example of `SaveHTKFeatures`

Input-output and property of the node

Table 6.83: 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 754 big 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

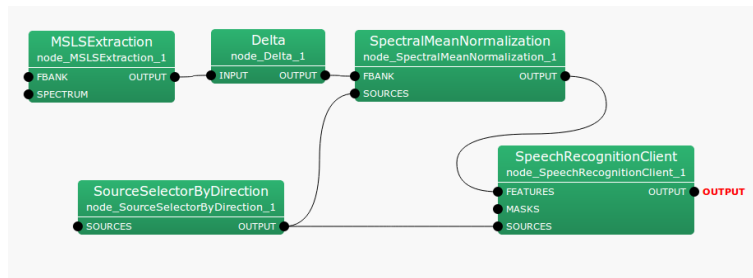


Figure 6.95: Connection example of SpectralMeanNormalization

Input-output and property of the node

Table 6.84: 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.

6.4.10 SpectralMeanNormalizationIncremental

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.

In the SpectralMeanNormalization node Real-time average removal is achieved by using the mean of the previous utterances in the same sound source direction. In the SMNIncremental node There is a difference that real-time average removal is realized by calculating and reflecting the mean of the utterances at regular intervals.

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

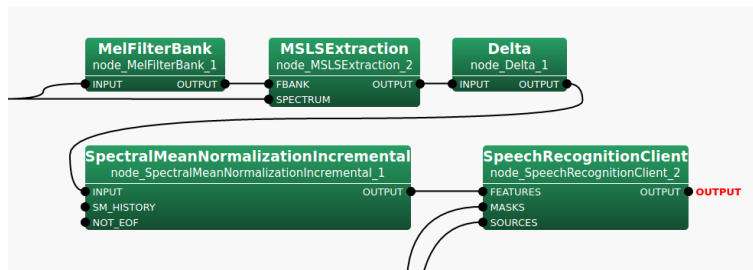


Figure 6.96: Connection example of SpectralMeanNormalizationIncremental

Input-output and property of the node

Input

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

SM_HISTORY : Map<int, ObjectRef> type(ObjectRef is Vector<ObjectRef> and ObjectRef in Vector<ObjectRef> is Vector<float>). This input is optional. The average spectral history for each source.

Table 6.85: Parameter list of SpectralMeanNormalizationIncremental

Parameter name	Type	Default value	Unit	Description
FBANK_COUNT	int	13	[frames]	Dimension number of input feature parameters
PERIOD	int	20		Period for calculating the average
SM_ALGORITHM	string	INCREMENTAL	[frames]	Algorithm for calculating the initial mean value
SM_FILENAME	string			CSV file name of initial average
SM_HISTORY_FILENAME	string	0	[frames]	CSV file name for initial average history
IGNORE_FRAMES	int			Number of frames to exclude from the calculation of the average spectrum
BASENAME	string	"smn"		Base name of the SMN file
OUTPUT_FNAME	string	"out.txt"		Dimension number of input feature parameters
SM_EXPORT_FILENAME	string			Name of CSV file to output the average
SM_EXPORT_ALGORITHM	string	LAST_SRC		Algorithm for calculating the average to be output

NOT_EOF : bool type. This input is optional. This input is only used when specifying the OUTPUT_FNAME or SM_EXPORT_FILENAME parameters.

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. Specify the number of banks.

PERIOD : int type. Its range is 0 or a positive integer. Specify the period for calculating the average spectrum. In other words, if 1 is specified, the average spectrum is calculated every frame. However, if 0 is specified, it means that the average spectrum is calculated indefinitely, that is, SM_ALGORITHM is used for all frames.

SM_ALGORITHM : string type. Default value is INCREMENTAL. The possible values are [INCREMENTAL, PREV_SM, ZERO, FILE]. Determine the initial algorithm to calculate the average spectrum to reach the *PERIOD + IGNORE_FRAME* frame. In the case of INCREMENTAL, the average spectrum is calculated using all frames from the first frame of the sound source. In the case of PREV_SM, the average spectrum calculated by the immediately preceding sound source is used. In the case of ZERO, the average spectrum is assumed to be 0 and processed. In the case of FILE, the average spectrum is read from the file specified by the SM_FILENAME parameter.

SM_FILENAME : string type. Specifies the CSV file name that gives the initial average spectrum when FILE is specified in SM_ALGORITHM.

SM_HISTORY_FILENAME : string type. Specify the CSV file name to save the initial average spectrum history until the *PERIOD + IGNORE_FRAME* frame is reached.

IGNORE_FRAMES : int type. Its range is 0 or a positive integer. Specifies the number of frames to ignore for calculating the average spectrum.

BASENAME : string type. Default value is "smn.". The base name of the file that stores the SMN file. The file name will be *BASENAME + id + ".csv"*.

OUTPUT_FNAME : string type. Default value is "out.txt".

SM_EXPORT_FILENAME : string type. Default value is an empty string. Specify the CSV file name of the average spectrum to be exported by SM_EXPORT_ALGORITHM. The export is enabled only if a file name is specified, and is disabled if an empty string is specified.

SM_EXPORT_ALGORITHM : string type. Default value is LAST_SRC, the possible values are [LAST_SRC, SRC_AVERAGE, FRAME_AVERAGE]. Determine the algorithm for calculating the average spectrum to be exported. In the case of LAST_SRC, the average spectrum of the last sound source is stored. In the case of SRC_AVERAGE, the average spectrum of all sound sources is stored. In the case of FRAME_AVERAGE, the average spectrum of all the frames of all the sources is stored as an average.

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

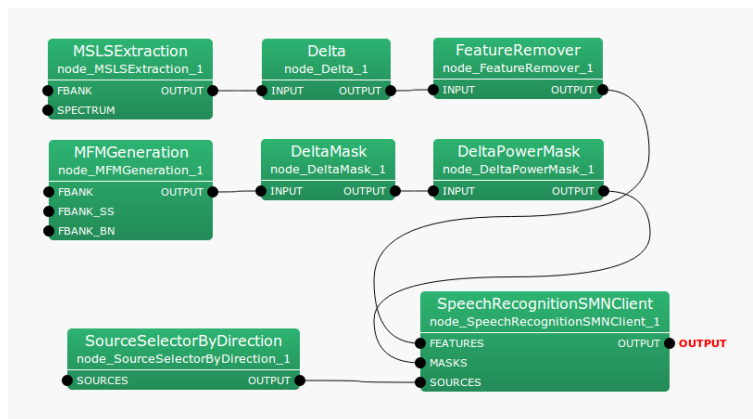


Figure 6.97: Typical connection example of DeltaMask

Input-output and property of the node

Table 6.86: 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 vector of 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;

$$m(f) = [m(f, 0), m(f, 1), \dots, m(f, 2P - 1)]^T \quad (6.158)$$

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.

$$y'(f) = [m'(f, 0), m'(f, 1), \dots, m'(f, 2P - 1)]^T \quad (6.159)$$

$$m'(f, p) = \begin{cases} m(f, p), & \text{if } p = 0, \dots, P - 1, \\ \prod_{\tau=-2}^2 m(f + \tau, p), & \text{if } p = P, \dots, 2P - 1, \end{cases} \quad (6.160)$$

Figure ?? shows an input-output flow of DeltaMask .

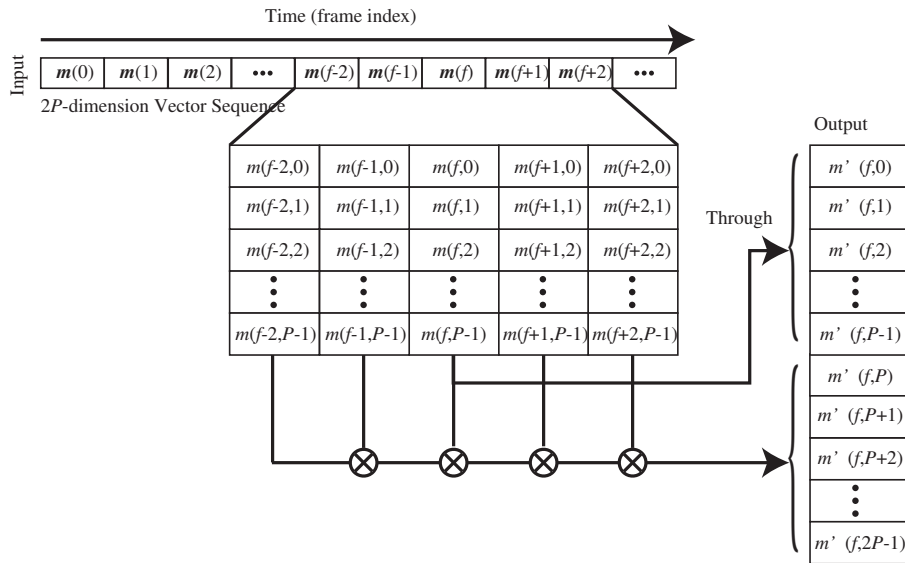


Figure 6.98: Input-output flow of DeltaMask .

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

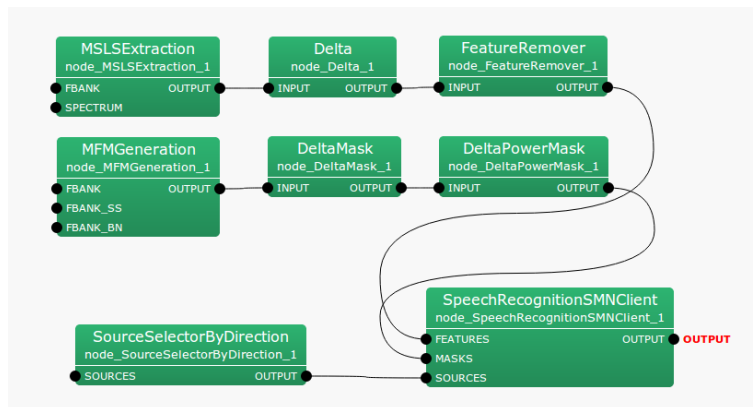


Figure 6.99: Typical connection example of DeltaPowerMask

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

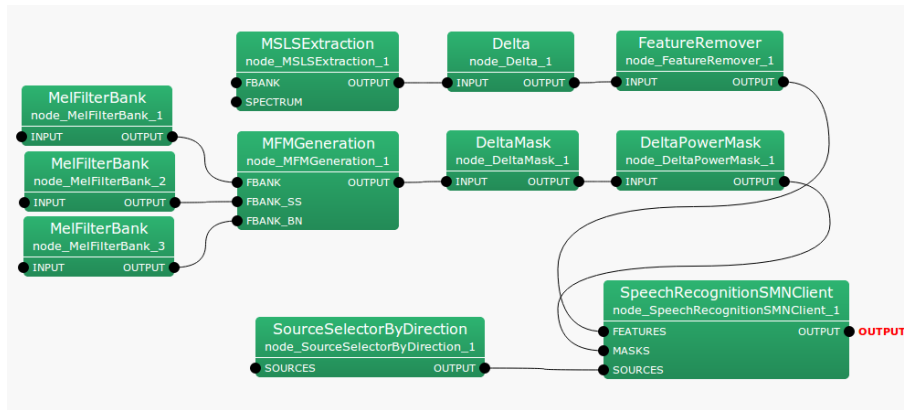


Figure 6.100: Connection example of MFMGeneration

Input-output and property of the node

Table 6.87: 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_SS : 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:

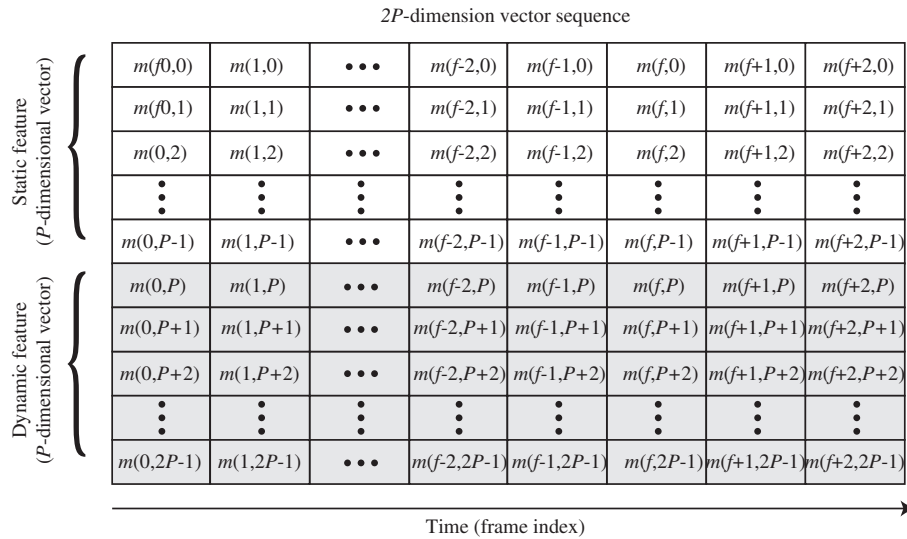
$$m(f) = [m(f, 0), m(f, 1), \dots, m(f, P-1)]^T \quad (6.161)$$

$$m(f, p) = \begin{cases} 1.0, & r(p) > \text{THRESHOLD} \\ 0.0, & r(p) \leq \text{THRESHOLD} \end{cases} \quad (6.162)$$

$$r(p) = \min(1.0, (f(p) + 1.4 * b(p)) / (f(p) + 1.0)), \quad (6.163)$$

$$(6.164)$$

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 ?? shows a schematic view of an output vector sequence.



*Shadowed elements are filled with ZERO.

Figure 6.101: 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

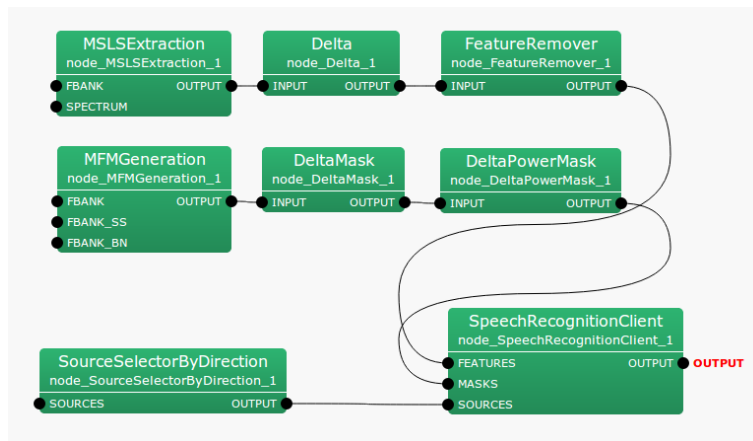


Figure 6.102: Connection example of SpeechRecognitionClient

Input-output and property of the node

Table 6.88: 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, this node uses a method for removing the average of the entire utterance section. Therefore, even when used online, sending is not performed until the utterance is over, and for that reason the processing is not real-time. In order to realize real-time processing, it is necessary to estimate or approximate the mean values of the utterance concerned using some values without obtaining the features values of the entire utterance section. 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

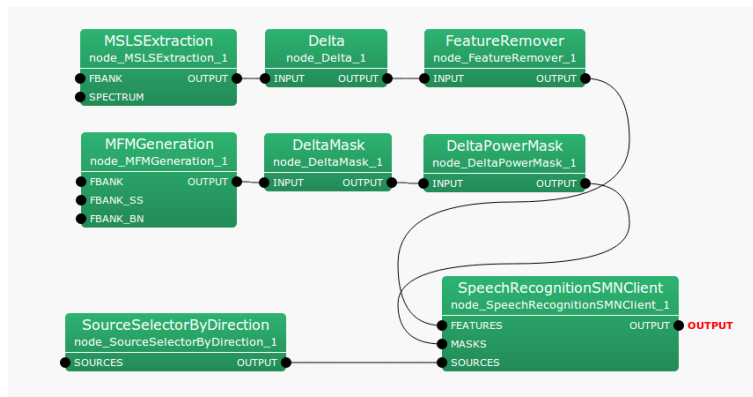


Figure 6.103: Connection example of SpeechRecognitionSMNClient

Input-output and property of the node

Table 6.89: 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

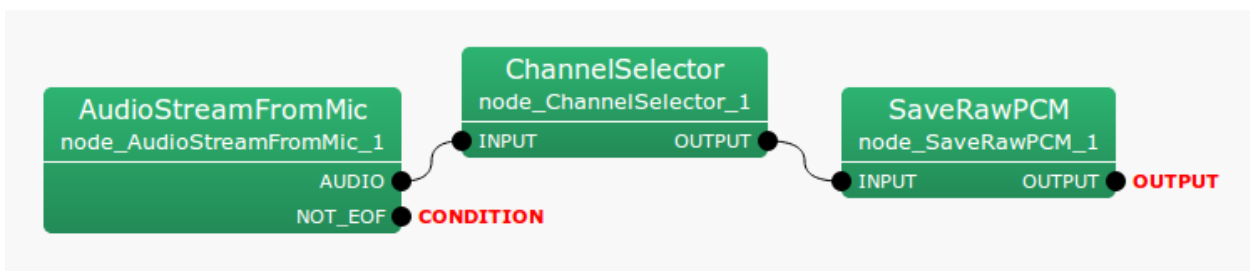


Figure 6.104: Example of a typical connection of ChannelSelector

Figure ?? 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 particular 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 ?? shows the network example that combines the two sound source localization results from `LocalizeMUSIC` and `ConstantLocalization`.

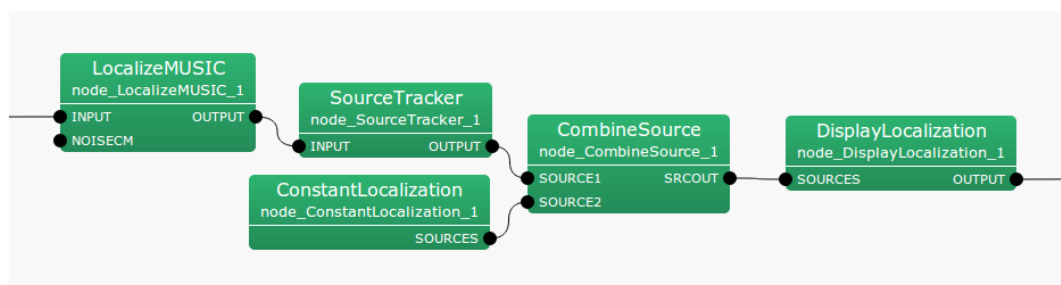


Figure 6.105: Example of a connection of `CombineSource`

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 ??.

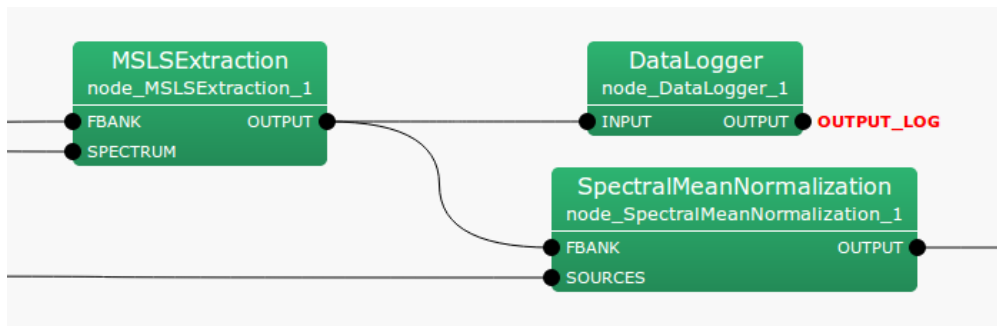


Figure 6.106: Connection example of DataLogger

Input-output and property of the node

Table 6.90: 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 HarkParamsDynReconf

Outline of the node

To reconfigure the parameters of LocalizeMUSIC , SourceTracker , and HRLE dynamically during execution of a network, this node subscribes parameters of the three modules through socket communication and passes the parameter to the three modules.

Necessary file

None.

Usage

When to use

This node is used when wishing to reconfigure parameters of LocalizeMUSIC , SourceTracker , and HRLE dynamically while executing a network.

Typical connection

Figure ?? shows the usage example of HarkParamsDynReconf node.

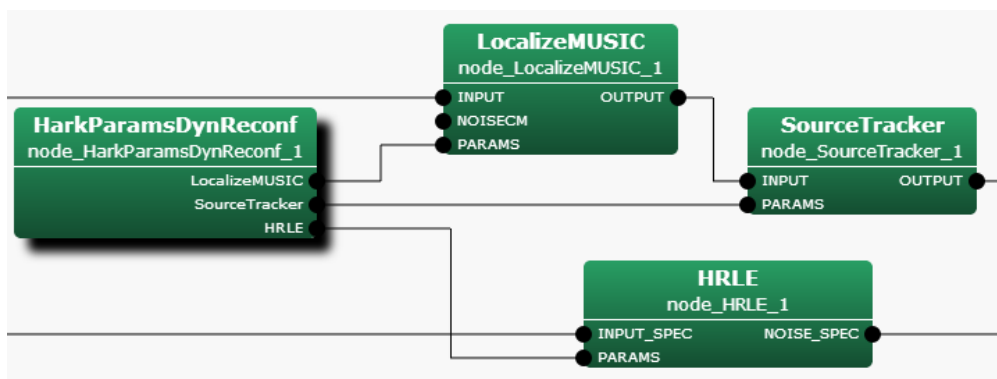


Figure 6.107: Connection example of HarkParamsDynReconf

LocalizeMUSIC , SourceTracker , and HRLE in Figure ?? have the PARAMS input terminals that are not displayed by the default. Figure ?? shows how to add the invisible input terminal PARAMS.

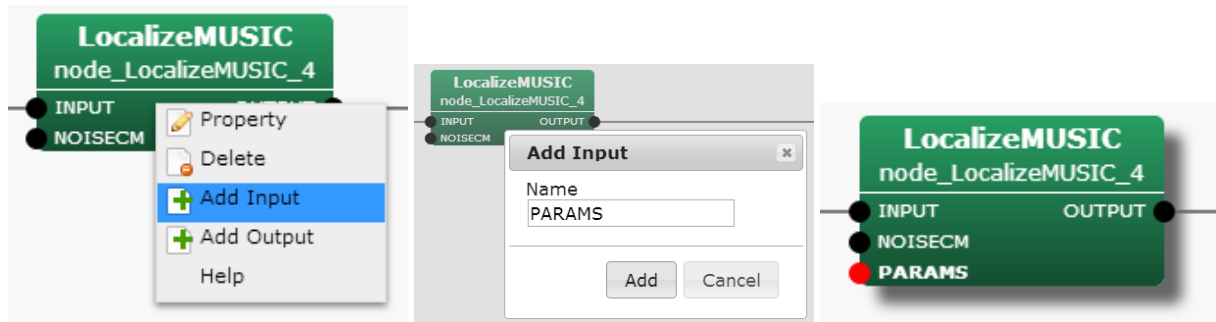
Input-output and property of the node

Input

No inputs.

Output

LocalizeMUSIC : Vector<ObjectRef> type. This outputs the parameters of LocalizeMUSIC . This should be connected to the PARAMS input terminal of LocalizeMUSIC .



Step 1: Right-click the node and “Add Step 2: Input “PARAMS” in the Step 3: PARAMS input terminal is
Input” “Name” form. Click “Add”. added to the node

Figure 6.108: Adding an invisible input terminal PARAMS

SourceTracker : `Vector<ObjectRef>` type. This outputs the parameters of SourceTracker . This should be connected to the PARAMS input terminal of SourceTracker .

HRLE : `Vector<ObjectRef>` type. This outputs the parameters of HRLE . This should be connected to the PARAMS input terminal of HRLE .

Parameter

Table 6.91: Parameter list of HarkParamsDynReconf

Parameter name	Type	Default value	Unit	Description
PORT	int	9999		Port number for the socket connection
ENABLE_DEBUG	bool	false		Enable debug output

PORT : int type. This parameter defines the port number for socket communication.

ENABLE_DEBUG : bool type. If true, the debug message is printed to the standard output.

Details of the node

This node works as a server of a non-blocking socket communication and subscribes parameters of LocalizeMUSIC , SourceTracker , and HRLE from a client program. After receiving the new parameters, this node sends the parameter to the three nodes.

The subscribed data should be a float type vector with the size 12, denoted as buff[12] hereinafter. If the current frame receives new parameters, the parameters are updated. If not, the parameters keep the previous ones.

The received data, buff[12], is decoded as follows and sent to the next nodes.

- **NUM_SOURCE** (LocalizeMUSIC) : (int)buff[0]
- **MIN_DEG** (LocalizeMUSIC) : (int)buff[1]
- **MAX_DEG** (LocalizeMUSIC) : (int)buff[2]
- **LOWER_BOUND_FREQUENCY** (LocalizeMUSIC) : (int)buff[3]
- **UPPER_BOUND_FREQUENCY** (LocalizeMUSIC) : (int)buff[4]
- **THRESH** (SourceTracker) : (float)buff[5]

- **PAUSE_LENGTH** (SourceTracker) : (float)buff[6]
- **MIN_SRC_INTERVAL** (SourceTracker) : (float)buff[7]
- **MIN_TFINDEX_INTERVAL** (SourceTracker) : (float)buff[8]
- **COMPARE_MODE** (SourceTracker) : (Source::CompareMode)buff[9]
- **LX** (HRLE) : (float)buff[10]
- **TIME_CONSTANT** (HRLE) : (int)buff[11]

This node supports the reconnection of client programs.

Here is an example of a client program written in python.

```
#!/usr/bin/python
import socket
import struct

HOST = 'localhost'      # The remote host
PORT = 9999             # The same port as used by the server

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
buff = [2.0, -180.0, 180.0, 500.0, 2800.0, 30.0, 800.0, 20.0, 6.0, 0.0, 0.85, 16000.0]
msg = struct.pack("f"*len(buff), *buff)
sock.send(msg)

sock.close()
```

6.7.5 LoadMapFrames

Outline of the node

Loads data from files created by SaveMapFrames into containers in either the Map <int , Vector<float> > type or the Map <int , Vector<complex<float> > > type.

Necessary files

Data files in either text format (TEXT) or binary format (RAW) which are called "Internal files" and a text file which is called "Main file" containing lists of those data files per frame. All files were created by SaveMapFrames .

Usage

When to use

This nodes is used to use data of frames saved in files in advance by SaveMapFrames to process further such as converting the data into a wav file later.

Typical connection

Figure ?? below shows a connection example using LoadMapFrames in a network. The files that LoadMapFrames loads are to be specified by value in the FILENAME parameter. In this network, Synthesize takes data in the Map <int , Vector<complex<float> > > type, which are output of LoadMapFrames , to convert them into waveforms of time domain. Then, SaveWavePCM saves the data outputted by Synthesize as a sound file. LoadSourceLocation is an optional in this network. The network gives an example that SaveWavePCM optionally takes sound source localization results in the Vector<ObjectRef> type as well. Note that the ObjectRef must refer to data in the Source type. LoadSourceLocation loads data from a file generated by SaveSourceLocation while LoadMapFrames loads data from files generated by SaveMapFrames .

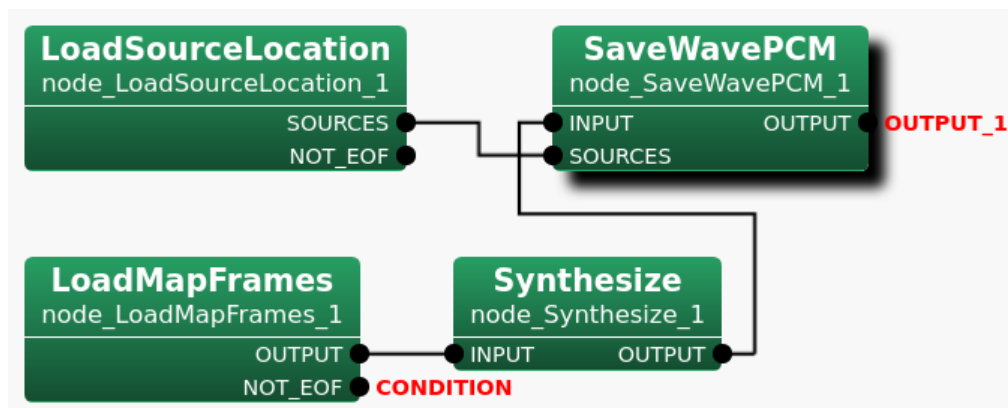


Figure 6.109: Connection example of LoadMapFrames

Input-output and property of the node

Input

No inputs.

Output

OUTPUT : Map<int, Vector<float>> or Map<int, Vector<complex<float>>> of the Map<int, ObjectRef> type.
Data of frames where the data are mapped to keys of the Map.

NOT_EOF : bool type. A flag to indicate whether or not the end of file has been reached when a program is reading the "Main file" which contains lists of "Internal files" for each frame. It is set to false when the program reaches the end of file of the "Main file". Then, the program will be terminated. Otherwise, it is being set to true.

Parameter

Table 6.92: Parameter list of LoadMapFrames

Parameter name	Type	Default value	Unit	Description
FILENAME	string			The name of the text file containing lists of data files for each frame.
INPUT_TYPE	string	TEXT_float		The file format and the data type of the input data files. Select TEXT_float, RAW_float, TEXT_complex_float, or RAW_complex_float.
COL_SIZE	int	512		The column size of the Vector<ObjectRef> of the Map<int, Vector<ObjectRef>> type in which data in the data file are to be restored.

FILENAME : string type. The name of the "Main file" containing lists of data files for each frame, created by SaveMapFrames . For more details on files generated by SaveMapFrames , see the node reference.

INPUT_TYPE : string type. The file format and the data type of input data files. Select appropriate one from the drop-down menu. Four options are available as follows.

- TEXT_float
- RAW_float
- TEXT_complex_float
- RAW_complex_float

The default value is set to TEXT_float. If the wrong value is selected, the Network will not run successfully. Figure ?? gives sample file names for both "Main file" and "Internal files". Each file name of "Internal Files" indicates data type, "Vector_float" for this example. The file extension indicates the file format. Each file has its extension, either "txt" for text (TEXT) or "raw" for binary (RAW).

COL_SIZE : int type. The Vector size of data in the Map<int, Vector<ObjectRef>> type saved in the input data files which can be found in the name of the file to load as shown in Figure ?? . If the wrong value is specified, the Network will not run successfully. The default value is 512.

For more information, see SaveMapFrames specifically in Details of the Node.

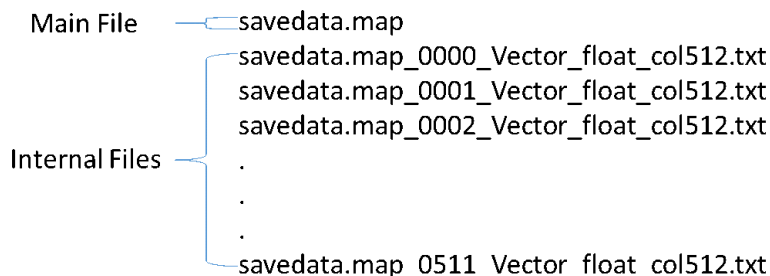


Figure 6.110: SaveMapFrames Generated Files

See the node reference of SaveMapFrames for more details on files generated by SaveMapFrames .

Details of the node

LoadMapFrames reads files generated by SaveMapFrames to restore the data in containers in either the Map <int , Vector<float> > type or the Map <int , Vector<complex<float> > > type. LoadMapFrames loads two types of files. One is data files which are called "Internal file" containing data of frames separated for each key. The other one is a text file which is called "Main file" contains lists of "Internal files" following by each frame number. LoadMapFrames obtains information from both the frame number and the filenames in "Main file" in order to restore data saved in "Internal file" properly into containers in the corresponding data type. Table ?? below shows the all available INPUT_TYPE parameter values and the corresponding output data types.

Table 6.93: LoadMapFrames Output Type

INPUT_TYPE	Output Type
Text Float	Map <int , Vector<float> >
Raw Float	Map <int , Vector<float> >
Text Complex Float	Map <int , Vector<complex<float> > >
Raw Complex Float	Map <int , Vector<complex<float> > >

Please note that LoadMapFrames will output an empty Map when it reads no filenames following by a frame number. This means that no data associated with the frame. When detecting that all data files have been read by reaching the end of file of the "Main file", LoadMapFrames sets NOT_EOF to false so that the iteration process for loading data from files into containers can be terminated.

6.7.6 LoadMatrixFrames

Outline of the node

Loads a data file created by SaveMatrixFrames into the Matrix<float> type or the Matrix<complex<float>> type per frame.

Necessary files

A data file in either text format (TEXT) or binary format (RAW) created by SaveMatrixFrames .

Usage

When to use

This node is used to restore data of frames saved in a file in advance by SaveMatrixFrames to either the Matrix<float> type or the Matrix<complex<float>> type so that the data in the file can be processed.

Typical connection

Figure ?? below shows a connection example using LoadMatrixFrames in a network. This sample network is for producing a wav file in the end from data of frames in the Matrix type saved in a data file. Since Synthesize accepts data only in the Map <int , Vector<complex<float>>> type, MatrixToMap is needed between LoadMatrixFrames and Synthesize for data type conversion.

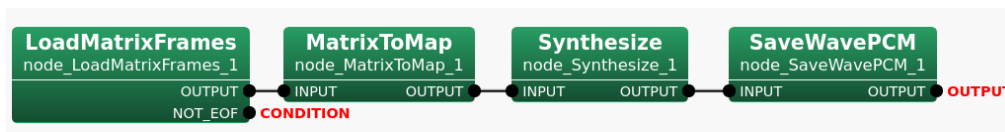


Figure 6.111: Connection Example of LoadMatrixFrames

Input-output and property of the node

Input

No inputs.

Output

OUTPUT : Matrix<float> or Matrix<complex<float>> of the Matrix<ObjectRef> types. Data of frames.

NOT_EOF : bool type. A flag to indicate whether or not the end of file has been reached when a program is reading the data file. It is set to false when the program reaches the end of the data file. Then, the program will be terminated. Otherwise, it is being set to true.

Parameter

FILENAME : string type. The name of the data file to load.

INPUT_TYPE : string type. The file format and the data type of the data file to load. Select appropriate one from the drop-down menu. Four options are available as follows.

- TEXT_float

- RAW_float
- TEXT_complex_float
- RAW_complex_float

The default value is set to TEXT_float. If the wrong value is selected, the Network will not run successfully.

ROW_SIZE : int type. The row size of the Matrix<ObjectRef> type in which data in the data file are to be restored. If the wrong value is specified, the network will not run successfully. Default value is 8.

COL_SIZE : int type. The column size of the Matrix<ObjectRef> type in which data in the data file are to be restored. If the wrong value is specified, the network will not run successfully. Default value is 512.

Table 6.94: Parameter list of LoadMatrixFrames

Parameter name	Type	Default value	Unit	Description
FILENAME	string			The name of the data file to load.
INPUT_TYPE	string	TEXT_float		The file format and the data type of the data file. Select TEXT_float, RAW_float, TEXT_complex_float, or RAW_complex_float.
ROW_SIZE	int	8		The row size of the Matrix<ObjectRef> type in which data in the data file are to be restored.
COL_SIZE	int	512		The column size of the Matrix<ObjectRef> type in which data in the data file are to be restored.

To make the parameter value settings for LoadMatrixFrames easier, it is strongly recommended to use a special pattern, {tag:format}, aka a format string, when saving data in a file by SaveMatrixFrames . The file name will then contain the data type, the row size, and the column size of the Matrix<ObjectRef> type in which the data in the data file are to be restored.

The example format strings and the outputs are given below in the Table ?? below for each parameter except FILENAME. For more details on {tag:format}, refer to SaveMatrixFrames node reference specifically, Tag list of SaveMatrixFrames .

Table 6.95: Example format strings and the outputs for the Matrix<complex<float> > where the row size is 8 and the column size is 512

Format String (FILENAME parameter value)	Output (Formatted Filename)
samplefile.dat	samplefile.dat
samplefile_{datatype}.txt	samplefile_complex_float.txt
samplefile_row{rowsize}.raw	samplefile_row8.raw
samplefile_col{colsize}.dat	samplefile_col512.dat
samplefile_dim{dim}.dat	samplefile_dim2.dat
samplefile_{datatype}_row{rowsize}_col{colsize}_dim{dim}.dat	samplefile_complex_float_row8_col512_dim2.dat

Putting appropriate file extension will also be helpful to save the time to find out the file format. If not, the TEXT type is a text file and the RAW type is a binary file. In Ubuntu platform, RAW files cannot be accessed by text editors whereas they can be opened in Windows. In any case, RAW files are not "human-readable" unlike TEXT files.

Details of the node

LoadMatrixFrames reads a file generated by SaveMatrixFrames and then restore the data in the file to either the Matrix<float> type or the Matrix<complex<float> > type per frame. The iteration process for loading data into the Matrix<ObjectRef> type per frame will be conducted by following the parameter values specified in the INPUT_TYPE, the ROW_SIZE, and the COL_SIZE. Table ?? below shows the all available INPUT_TYPE parameter values and the corresponding output data types in LoadMatrixFrames .

Table 6.96: LoadMatrixFrames Output Type

INPUT.TYPE	Output Type
Text Float	Matrix<float>
Raw Float	Matrix<float>
Text Complex Float	Matrix<complex<float> >
Raw Complex Float	Matrix<complex<float> >

When reaching the end of file of the data file, LoadMatrixFrames sets NOT_EOF to false so that the iteration process for loading data from the file into Matrix<ObjectRef> can be terminated.

6.7.7 LoadVectorFrames

Outline of the node

Loads a data file created by SaveVectorFrames into the Vector<float> type or the Vector<complex<float>> type per frame.

Necessary files

A data file in either text format (TEXT) or binary format (RAW) created by created by SaveVectorFrames .

Usage

When to use

This node is used to restore data of frames saved in a file in advance by SaveMatrixFrames to either the Vector<float> type or the Vector<complex<float>> type so that the data in the file can be processed.

Typical connection

Figure ?? below shows a connection example using LoadVectorFrames in a network. This sample network is for display the location of the input sound source(s) in the end. Instead of the hidden output terminal of LocalizeMUSIC , for the MUSIC_SPEC input terminal of NormalizeMUSIC , LoadVectorFrames outputs power of the MUSIC spectrum for every direction in the Vector<float> type. NormalizeMUSIC then can use both source positons determined by LocalizeMUSIC and MUSIC spectrum provided by LoadVectorFrames to stabilize the sound source detection by SourceTracker .

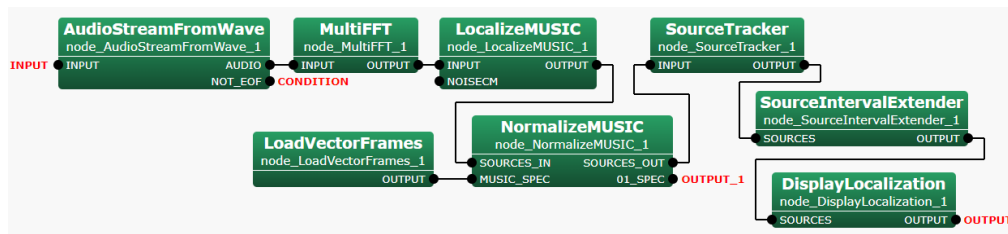


Figure 6.112: Connection Example of LoadVectorFrames

Input-output and property of the node

Input

No inputs.

Output

OUTPUT : Vector<float> or Vector<complex<float>> of the Vector<ObjectRef> type. Data of frames.

NOT_EOF : bool type. A flag to indicate whether or not the end of file has been reached when a program is reading the data file. It is set to false when the program reaches the end of the data file. Then, the program will be terminated. Otherwise, it is being set to true.

Parameter

Table 6.97: Parameter list of LoadVectorFrames

Parameter name	Type	Default value	Unit	Description
FILENAME	string			The name of the data file to load.
INPUT_TYPE	string	TEXT_float		The file format and the data type of the data file. Select TEXT_float, RAW_float, TEXT_complex_float, or RAW_complex_float.
COL_SIZE	int	512		The column size of the Vector<ObjectRef> type in which data in the data file are to be restored.

FILENAME : string type. The name of the data file to load.

INPUT_TYPE : string type. The file format and the data type of the data file to load. Select appropriate one from the drop-down menu. Four options are available as follows.

- TEXT_float
- RAW_float
- TEXT_complex_float
- RAW_complex_float

The default value is set to TEXT_float. If the wrong value is selected, the Network will not run successfully.

COL_SIZE : int type. The column size of the Vector<ObjectRef> type in which data in the data file are to be restored. If the wrong value is specified, the network will not run successfully. Default value is 512.

To make the parameter value settings for LoadVectorFrames easier, it is strongly recommended to use a special pattern, {tag:format}, aka a format string, when saving data in a file by SaveVectorFrames . The file name will then contain the data type and the column size of the Vector<ObjectRef> type in which the data in the data file are to be restored.

The example format strings and the outputs are given below in the Table ?? below for each parameter except FILENAME. For more details on {tag:format}, refer to SaveVectorFrames node reference specifically, Tag list of SaveVectorFrames .

Table 6.98: Example format strings and the outputs for the Vector<complex<float> > where the column size is 20

Format String (FILENAME parameter value)	Output (Formatted Filename)
samplefile.dat	samplefile.dat
samplefile_{datatype}.txt	samplefile_complex_float.txt
samplefile_col{colsize}.raw	samplefile_col20.raw
samplefile_dim{dim}.dat	samplefile_dim1.dat
samplefile_{datatype}_col{colsize}_dim{dim}.dat	samplefile_complex_float_col20_dim1.dat

Putting appropriate file extension will also be helpful to save the time to find out the file format. If not, the TEXT type is a text file and the RAW type is a binary file. In Ubuntu platform, RAW files cannot be accessed by text editors whereas they can be opened in Windows. In any case, RAW files are not "human-readable" unlike TEXT files.

Details of the node

LoadVectorFrames reads a file generated by SaveVectorFrames and then restore the data in the file to either the Vector<float> type or the Vector<complex<float> > type per frame. The iteration process for loading data into the Vector<ObjectRef> type per frame will be conducted by following the parameter values specified in the INPUT_TYPE and the COL_SIZE. Table ?? below shows the all available INPUT_TYPE parameter values and the corresponding output data types in LoadVectorFrames .

When reaching the end of file of the data file, LoadVectorFrames sets NOT_EOF to false so that the iteration process for loading data from the file into Vector<ObjectRef> can be terminated.

Table 6.99: LoadVectorFrames Output Type

INPUT_TYPE	Output Type
Text Float	Vector<float>
Raw Float	Vector<float>
Text Complex Float	Vector<complex<float> >
Raw Complex Float	Vector<complex<float> >

6.7.8 MapIDOffset

Outline of the node

Shift the keys of the given `Map<int, ObjectRef>` by a specified integer value.

Necessary file

No files are required.
subsubsectionUsage

When to use

This node is used to shift the keys of the given `Map<int, ObjectRef>` by specified integer value. The data type of the `ObjectRef` of the `Map<int, ObjectRef>` for the input is `Vector<float>`, `Vector<complex<float> >`, `Matrix<float>`, or `Matrix<complex<float> >`.

Input-output and property of the node

Input

INPUT : `Map<int, Vector<float>>`, `Map<int, Vector<complex<float> >>`, `Map<int, Matrix<float>>` or `Map<int, Matrix<complex<float> >>` of the `Map<int, ObjectRef>` Type.

Output

OUTPUT : `Map<int, Vector<float>>`, `Map<int, Vector<complex<float> >>`, `Map<int, Matrix<float>>` or `Map<int, Matrix<complex<float> >>` of the `Map<int, ObjectRef>` Type.

Parameter

Table 6.100: Parameter list of MapIDOffset

Parameter name	Type	Default value	Unit	Description
ID_OFFSET	int	0		The value by which to shift the keys of the given <code>Map<int, ObjectRef></code>
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

ID_OFFSET : int type. The value by which to shift the keys of the given `Map<int, ObjectRef>`. The value can be negative. The default value is 0.

DEBUG : bool type. Setting the value to true outputs the separation status to the standard output. The default value is false.

6.7.9 MapMatrixValueOverwrite

Outline of the node

Overwrite a specified submatrix of a `Matrix<ObjectRef>` with a specified value where the `Matrix<ObjectRef>` is an `ObjectRef` of a `Map<int, ObjectRef>`.

Necessary file

No files are required.

Usage

When to use

This node is used to overwrite a specified submatrix of a `Matrix<ObjectRef>` with a specified value where the `Matrix<ObjectRef>` is an `ObjectRef` of a `Map<int, ObjectRef>`. The data type of the specified value will be converted to the suitable one depending on the `ObjectRef` of the `Map<int, Matrix<ObjectRef>>` given as the input.

Input-output and property of the node

Input

INPUT : `Map<int, Matrix<ObjectRef>>` type; `Map<int, Matrix<int>>`, `Map<int, Matrix<float>>`, or `Map<int, Matrix<complex<float>>>`.

Output

OUTPUT : `Map<int, Matrix<ObjectRef>>` type; `Map<int, Matrix<int>>`, `Map<int, Matrix<float>>`, or `Map<int, Matrix<complex<float>>>`.

Parameter

OVERWRITTEN_ROW_MIN : `int` type. The row index of the `Matrix<ObjectRef>` of the `Map<int, Matrix<ObjectRef>>` given as the input to specify the first row of the submatrix on which overwrite. The default value is 0.

OVERWRITTEN_ROW_MAX : `int` type. The row index of the `Matrix<ObjectRef>` of the `Map<int, Matrix<ObjectRef>>` given as the input to specify the last row of the submatrix on which overwrite. The default value is 0.

OVERWRITTEN_COL_MIN : `int` type. The column index of the `Matrix<ObjectRef>` of the `Map<int, Matrix<ObjectRef>>` given as the input to specify the first column of the submatrix on which overwrite. The default value is 0.

OVERWRITTEN_COL_MAX : `int` type. The column index of the `Matrix<ObjectRef>` of the `Map<int, Matrix<ObjectRef>>` given as the input to specify the last column of the submatrix on which overwrite. The default value is 0.

OVERWRITE_VALUE_REAL : `float` type. The value with which overwrite a specified submatrix. When the **INPUT** is `Map<int, Matrix<int>>`, the type will be converted to the `int` type. When the **INPUT** is `Map<int, Matrix<complex<float>>>`, the value will be for the real part. The default value is 0.

OVERWRITE_VALUE_IMAG : `float` type. The value for the imaginary part with which overwrite a specified submatrix when the `ObjectRef` of the `Map<int, Matrix<ObjectRef>>` given as the input is the `complex` type. The default value is 0.

DEBUG : `bool` type. Setting the value to `true` outputs the overwrite status to the standard output. The default value is `false`.

Table 6.101: Parameter list of MapMatrixValueOverwrite

Parameter name	Type	Default value	Unit	Description
OVERWRITTEN_ROW_MIN	int	0		The row index of the Matrix given as the input to specify the first row of the submatrix on which overwrite.
OVERWRITTEN_ROW_MAX	int	0		The row index of the Matrix given as the input to specify the last row of the submatrix on which overwrite.
OVERWRITTEN_COL_MIN	int	0		The column index of the Matrix given as the input to specify the first column of the submatrix on which overwrite.
OVERWRITTEN_COL_MAX	int	0		The column index of the Matrix given as the input to specify the last column of the submatrix on which overwrite.
OVERWRITE_VALUE_REAL	float	0		The value with which overwrite a specified submatrix. The data type will be converted to the suitable one depending on the ObjectRef of the Map<int, Matrix<ObjectRef>> given as the input.
OVERWRITE_VALUE_IMAG	float	0		The value for the imaginary part with which overwrite a specified submatrix when the ObjectRef of the Map<int, Matrix<ObjectRef>> given as the input is the complex type.
DEBUG	bool	false		Enable or disable to output the overwriting status to standard output.

Details of the node

<example>

PARAMETER:

OVERWRITTEN_ROW_MIN:0,
OVERWRITTEN_ROW_MAX:0,
OVERWRITTEN_COL_MIN:1,
OVERWRITTEN_COL_MAX:2,
OVERWRITE_VALUE_REAL:9

INPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \right\}$$

OUTPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 9 & 9 \\ 4 & 5 & 6 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 2 & 9 & 9 \\ 5 & 6 & 7 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 3 & 9 & 9 \\ 6 & 7 & 8 \end{bmatrix} \right\}$$

6.7.10 MapOperator

Outline of the node

Perform mathematical operations on the `ObjectRef` that have the same key of the two given `Map<int, ObjectRef>`.

Necessary file

No files are required.

Usage

When to use

This node is used to perform mathematical operations on the `ObjectRef` that have the same key of the two given `Map<int, ObjectRef>`. The data type of the `ObjectRef` should be `Float`, `Complex`, `Vector<float>`, `Vector<complex<float>>`, `Matrix<float>`, or `Matrix<complex<float>>`. The operation will be addition, subtraction, multiplication, division, equal, maximum, minimum, or concatenate. Note that the data type and the size of the `ObjectRef` of the two `Map<int, ObjectRef>` for the input must be the same in order to perform the calculation.

Input-output and property of the node

Input

INPUT1 : `Map<int, Float>`, `Map<int, Complex>`, `Map<int, Vector<float>>`, `Map<int, Vector<complex<float>>>`, `Map<int, Matrix<float>>`, `Map<int or Matrix<complex<float>>>` of `Map<int, ObjectRef>` Type.

INPUT2 : `Map<int, Float>`, `Map<int, Complex>`, `Map<int, Vector<float>>`, `Map<int, Vector<complex<float>>>`, `Map<int, Matrix<float>>` or `Map<int, Matrix<complex<float>>>` of `Map<int, ObjectRef>` Type.

Output

OUTPUT : `Map<int, Float>`, `Map<int, Complex>`, `Map<int, Vector<float>>`, `Map<int, Vector<complex<float>>>`, `Map<int, Matrix<float>>`, `Map<int or Map<int, bool> Matrix<complex<float>>>` of `Map<int, ObjectRef>` Type.

Parameter

Table 6.102: Parameter list of MapOperator

Parameter name	Type	Default value	Unit	Description
OPERATION_TYPE	string	Add		Operation type to perform on the <code>ObjectRef</code> of the two given <code>Map<int, ObjectRef></code> . Select one from Add, Sub, Mul, Div, Equal, Max, Min, or Concat. Perform addition, subtraction, multiplication, division, equality test, maximum, minimum, and concatenation, respectively.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

OPERATION_TYPE : string type. The operation type to perform on the `ObjectRef` of the two given `Map<int, ObjectRef>`. Select one from Add, Sub, Mul, Div, Equal, Max, Min, or Concat. Add performs Addition, Sub performs subtraction, Mul performs multiplication, Div performs division, Equal checks equality, Max takes the greater value, Min takes the smaller value, and Concat does concatenation on or between the two `ObjectRef`. The default value is Add.

DEBUG : bool type. Setting the value to **true** outputs the conversion status to the standard output. The default value is **false**.

Details of the node

Table 6.103: operation and input-output type

Input type	Operation	Output type
Map< int , Float >	Add, Sub, Mul, Div, Max or Min	Map< int , Float >
Map< int , Complex >	Add, Sub, Mul, Div, Max or Min	Map< int , Complex >
Map< int , Vector<float> >	Add, Sub, Mul, Div, Max or Min	Map< int , Vector<float> >
Map< int , Vector<complex<float> > >	Add, Sub, Mul, Div, Max or Min	Map< int , Vector<complex<float> > >
Map< int , Matrix<float> >	Add, Sub, Mul, Div, Max or Min	Map< int , Matrix<float> >
Map< int , Matrix<complex<float> > >	Add, Sub, Mul, Div, Max or Min	Map< int , Matrix<complex<float> > >
Map< int , Float > , Map< int , Complex > , Map< int , Vector<float> > , Map< int , Vector<complex<float> > > , Map< int , Matrix<float> > , Map< int , Matrix<complex<float> > >	Equal	Map< int , bool >
Map< int , Float > , Map< int , Complex > , Map< int , Vector<float> > , Map< int , Vector<complex<float> > > , Map< int , Matrix<float> > , Map< int , Matrix<complex<float> > >	Concat	same as input type, but size is different

6.7.11 MapSelectorBySource

Outline of the node

A filtering node which filters the multiple sound source separation results given as the input to output results that satisfy the conditions specified by parameters based on the information (ID, power, direction) in the Source. Regarding ID or power of a sound source, output the sound source whose ID or power is the minimum, the maximum, or within a certain range specified. Regarding the direction of the sound source, output the sound source whose direction is in the nearest to the specified angle, or angle within a certain range specified.

Necessary file

No files are required.

Usage

When to use

This node is used to obtain the sound source separation results when the information on ID, power, or direction of the sound source is available. It is useful to output some portion of data in the Map type. For example, when replaying the output result of sound source separation nodes such as GHDSS in PlayAudio. In this case, connect Synthesize and MapSelectorBySource to GHDSS node so that sound sources in different directions with Lch and Rch can be played separately.

Typical connection

Figure ?? shows a connection example. As shown in the figure, this node is connected later than the sound source separation module such as GHDSS.

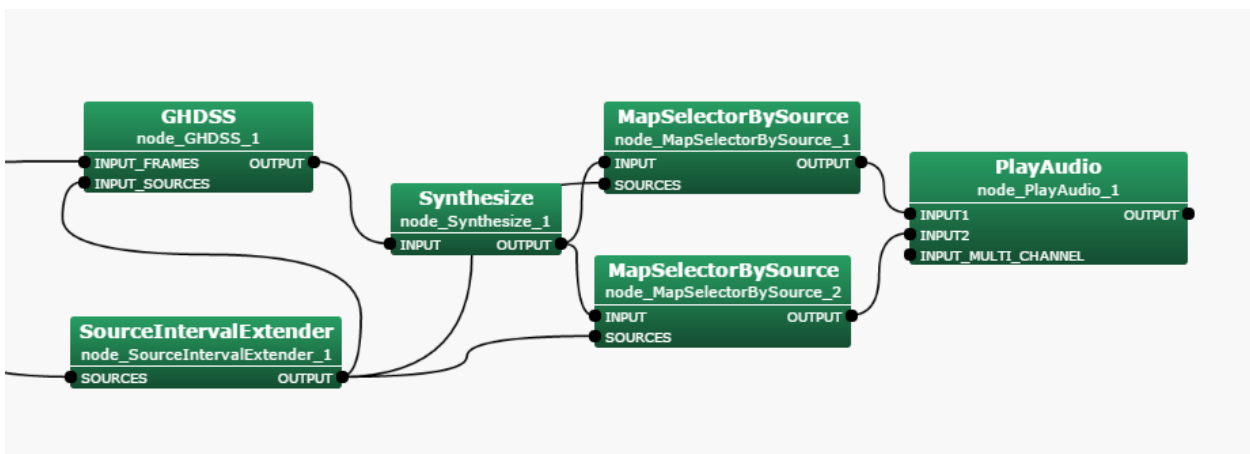


Figure 6.113: Connection example of MapSelectorBySource in an Iterator subnetwork

Input-output and property of the node

Input

INPUT : Map<int, ObjectRef> type. The key of the Map, int, corresponds to the sound source ID since this node is connected to after a sound source separation node in general. The ObjectRef is either Vector<float> (Power Spectral) or Vector<complex<float> > (Complex Spectral) which are indicating sound source separation.

SOURCES : Vector<ObjectRef> type. The directions of sound sources to which ID are assigned. The ObjectRef refers to data in the Source type indicating the sound source information with an ID.

Output

OUTPUT : Map<int, ObjectRef> type. The data that satisfy conditions specified by parameters.

Parameter

SELECTION_TYPE : string type. The filter condition for outputting data. Select ALL, ID, POWER, DIRECTION(*), DIRECTION_AZIMUTH, or DIRECTION_ELEVATION. Selecting ALL outputs all results unconditionally. When selecting ID, POWER, or DIRECTION(*)/DIRECTION_AZIMUTH/DIRECTION_ELEVATION, outputs results that satisfy the conditions specified by the subsequent parameters for the sound source ID, the sound source power, or the sound source direction.

ID_SELECTION_TYPE : string type. The type of the sound source ID available when ID is selected in the SELECTION_TYPE parameter. Select LATEST, OLDEST, or BETWEEN. Selecting LATEST or OLDEST outputs the results of the sound source with the latest ID or the oldest ID. Selecting BETWEEN outputs the results of which ID are in between the values specified in ID_RANGE_MIN and ID_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of the multiple sound sources.

ID_RANGE_MIN : int type. The lower limit of the range for the sound source ID available when BETWEEN is selected in the ID_SELECTION_TYPE parameter.

ID_RANGE_MAX : int type. The upper limit of the range for the sound source ID available when BETWEEN is selected in the ID_SELECTION_TYPE parameter.

POWER_SELECTION_TYPE : string type. The type of the sound source power available when POWER is selected in the SELECTION_TYPE parameter. Select HIGHEST, LOWEST, or BETWEEN. Selecting HIGHEST or LOWEST outputs the results of the sound source with the highest power or the lowest power. Selecting BETWEEN outputs the results whose power is in between the values specified in POWER_RANGE_MIN and POWER_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources.

POWER_RANGE_MIN : float type. The lower limit of the range for the sound source power available when BETWEEN is selected in the POWER_SELECTION_TYPE parameter.

POWER_RANGE_MAX : float type. The upper limit of the range for the sound source power available when BETWEEN is selected in the POWER_SELECTION_TYPE parameter.

DIRECTION_SELECTION_TYPE : string type. The type of the direction of a sound source available when DIRECTION is selected in the SELECTION_TYPE parameter. Select NEAREST or BETWEEN. Selecting NEAREST outputs the result whose direction is located in the nearest to the angle specified in the DIRECTION parameter. Selecting BETWEEN outputs the results whose direction is in between the specific values specified in DIRECTION_RANGE_MIN and DIRECTION_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources. (*)

DIRECTION : float type. The direction angle of a sound source available when NEAREST is selected in the DIRECTION_SELECTION_TYPE parameter. (*)

Table 6.104: Parameter list of MapSelectorBySource

Parameter list	Type	Default value	Unit	Description
SELECTION_TYPE	string	ID		The filter condition for outputting data. Select ALL, ID, POWER, DIRECTION(*), DIRECTION_AZIMUTH, or DIRECTION_ELEVATION.
ID_SELECTION_TYPE	string	LATEST		The type of the sound source ID. Select LATEST, OLDEST or BETWEEN.
ID_RANGE_MIN	int	0		The lower limit of the range for the sound source ID.
ID_RANGE_MAX	int	0		The upper limit of the range for the sound source ID.
POWER_SELECTION_TYPE	string	HIGHEST		The type of the sound source power. Select HIGHEST, LOWEST or BETWEEN.
POWER_RANGE_MIN	float	0		The lower limit of the range for the sound source power.
POWER_RANGE_MAX	float	40.0		The upper limit of the range for the sound source power.
DIRECTION_SELECTION_TYPE(*)	string	BETWEEN		The type of the direction of a sound source. Select NEAREST or BETWEEN.
DIRECTION(*)	float	0	[deg]	The direction angle of a sound source.
DIRECTION_RANGE_MIN(*)	float	0	[deg]	The lower limit of the range for the direction angle of a sound source.
DIRECTION_RANGE_MAX(*)	float	360.0	[deg]	The upper limit of the range for the direction angle of a sound source.
AZIMUTH_SELECTION_TYPE	string	BETWEEN		The type of the azimuth angle of a sound source. Select NEAREST or BETWEEN.
AZIMUTH	float	0	[deg]	The azimuth angle of a sound source.
AZIMUTH_RANGE_MIN	float	0	[deg]	The lower limit of the range for the azimuth angle of a sound source.
AZIMUTH_RANGE_MAX	float	360.0	[deg]	The upper limit of the range for the azimuth angle of a sound source.
ELEVATION_SELECTION_TYPE	string	BETWEEN		The type of the elevation angle of a sound source. Select NEAREST or BETWEEN.
ELEVATION	float	0	[deg]	The elevation angle of a sound source.
ELEVATION_RANGE_MIN	float	0	[deg]	The lower limit of the range for the elevation angle of a sound source.
ELEVATION_RANGE_MAX	float	360.0	[deg]	The upper limit of the range for the elevation angle of a sound source.
DEBUG_PRINT	bool	false		Prints the debug message of this module if set to true.

DIRECTION_RANGE_MIN : float type. The lower limit of the range for the direction angle of a sound source available when BETWEEN is selected in the DIRECTION_SELECTION_TYPE parameter. (*)

DIRECTION_RANGE_MAX : float type. The upper limit of the range for the direction angle of a sound source available when BETWEEN is selected in the DIRECTION_SELECTION_TYPE parameter. (*)

AZIMUTH_SELECTION_TYPE : string type. The type of the azimuth of a sound source available when DIRECTION_AZIMUTH is selected in the SELECTION_TYPE parameter. Select NEAREST or BETWEEN. Selecting NEAREST outputs the result whose azimuth is located the nearest to the azimuth specified in the AZIMUTH parameter. Selecting BETWEEN outputs the results whose azimuth is in between the specific values specified in AZIMUTH_RANGE_MIN and AZIMUTH_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources.

AZIMUTH : float type. The azimuth angle available when NEAREST is selected in the AZIMUTH_SELECTION_TYPE parameter.

AZIMUTH_RANGE_MIN : float type. The lower limit of the range for the azimuth angle of a sound source available when BETWEEN is selected in the AZIMUTH_SELECTION_TYPE parameter.

AZIMUTH_RANGE_MAX : float type. The upper limit of the range for the azimuth angle of a sound source available when BETWEEN is selected in the AZIMUTH_SELECTION_TYPE parameter.

ELEVATION_SELECTION_TYPE : string type. The type of the elevation of a sound source available when DIRECTION_ELEVATION is selected in the SELECTION_TYPE parameter. Select NEAREST or BETWEEN. Selecting NEAREST outputs the result whose elevation is located in the nearest to the elevation specified in the ELEVATION parameter. Selecting BETWEEN outputs the results whose elevation is in between the values specified in ELEVATION_RANGE_MIN and ELEVATION_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources.

ELEVATION : float type. The elevation angle of a sound source available when NEAREST is selected in the ELEVATION_SELECTION_TYPE.

ELEVATION_RANGE_MIN : float type. The lower limit of the range for the elevation angle of a sound source available when BETWEEN is selected in the ELEVATION_SELECTION_TYPE.

ELEVATION_RANGE_MAX : float type. The upper limit of the range for the elevation angle of a sound source available when BETWEEN is selected in the ELEVATION_SELECTION_TYPE.

DEBUG_PRINT : bool type. Prints the debug message of this module if set to true..

(*)Parameters for compatibility

DIRECTION and DIRECTION_AZIMUTH, choices in the SELECTION_TYPE parameter, are compatible. The DIRECTION parameter and the AZIMUTH parameter are compatible. The DIRECTION_RANGE_MIN parameter and the AZIMUTH_RANGE_MIN parameter are compatible so are the DIRECTION_RANGE_MAX parameter and the AZIMUTH_RANGE_MAX parameter.

6.7.12 MapToMap

Outline of the node

Convert the data type of the ObjectRef of the Map<int, ObjectRef> given as the input; between the Map<int, Vector<float>> type and the Map<int, Vector<complex<float>>> type or between the Map<int, Matrix<float>> type and the Map<int, Matrix<complex<float>>> type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the data type of the ObjectRef of the Map<int, ObjectRef> given as the input; the Map<int, Vector<float>> type to the Map<int, Vector<complex<float>>> type, the Map<int, Vector<complex<float>>> type to the Map<int, Vector<float>> type, the Map<int, Matrix<float>> type to the Map<int, Matrix<complex<float>>> type, or the Map<int, Matrix<complex<float>>> type to the Map<int, Matrix<float>> type.

Input-output and property of the node

Input

INPUT : Map<int, Vector<float>>, Map<int, Vector<complex<float>>>, Map<int, Matrix<float>>, or Map<int, Matrix<complex<float>>> of the Map<int, ObjectRef> type.

Output

OUTPUT : Map<int, Vector<float>>, Map<int, Vector<complex<float>>>, Map<int, Matrix<float>>, or Map<int, Matrix<complex<float>>> of the Map<int, ObjectRef> type.

Parameter

METHOD_COMPLEX_TO_FLOAT : string type. The method to convert the Vector<complex<float>> type to the Vector<float> type or to convert the Matrix<complex<float>> type to the Matrix<float> type. Select magnitude, real, or imaginary. Output the absolute value, the real part, and the imaginary part, which are of the complex number, respectively. The default value is magnitude.

METHOD_FLOAT_TO_COMPLEX : string type. The method to convert the Vector<float> type to the Vector<complex<float>> type or to convert the Matrix<float> type to the Matrix<complex<float>> type. Select zero or hilbert. For the imaginary part, output 0 by selecting zero or output the absolute value of the real part after converted to complex by selecting hilbert. The default value is zero.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Details of the node

Table 6.105: Parameter list of MapToMap

Parameter name	type	Default value	Unit	Description
METHOD_COMPLEX_TO_FLOAT	string	magnitude		The method to convert the Vector<complex<float> > type to the Vector<float> type or to convert the Matrix<complex<float> > type to the Matrix<float> type. Select magnitude, real, or imaginary. Output the absolute value, the real part, and the imaginary part, which are of the complex number, respectively.
METHOD_FLOAT_TO_COMPLEX	string	zero		The method to convert the Vector<float> type to the Vector<complex<float> > type or to convert the Matrix<float> type to the Matrix<complex<float> > type. Select zero or hilbert. For the imaginary part, output 0 or the absolute value of the real part after converted to complex .
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

Table 6.106: Conversion table of MapToMap

INPUT	OUTPUT	Parameter to use
Map< int , Vector<float> >	Map< int , Vector<complex<float> > >	METHOD_FLOAT_TO_COMPLEX
Map< int , Matrix<float> >	Map< int , Matrix<complex<float> > >	
Map< int , Vector<complex<float> > >	Map< int , Vector<float> >	METHOD_COMPLEX_TO_FLOAT
Map< int , Matrix<complex<float> > >	Map< int , Matrix<float> >	

6.7.13 MapToMatrix

Outline of the node

Convert the `Map<int, ObjectRef>` type to the `Matrix<ObjectRef>` type when the `ObjectRef` of the given `Map<int, ObjectRef>` is either the `Matrix<float>` type or the `Matrix<complex<float>>` type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the `Map<int, ObjectRef>` type to `Matrix<ObjectRef>` type when the `ObjectRef` of the given `Map<int, ObjectRef>` is either the `Matrix<float>` type or the `Matrix<complex<float>>` type. Taking a `Map<int, Matrix<float>>` as the input will output a `Matrix<float>`. Taking a `Map<int, Matrix<complex<float>>>` as the input will output a `Matrix<complex<float>>`.

Input-output and property of the node

Input

INPUT : `Map<int, Matrix<float>>` or `Map<int, Matrix<complex<float>>>` of the `Map<int, ObjectRef>` type.

Output

OUTPUT : any type. Note that the supported data types are the `Matrix<float>` type and the `Matrix<complex<float>>` type.

Parameter

Table 6.107: Parameter list of MapToMatrix

Parameter name	Type	Default value	Unit	Description
METHOD	string	min_key		The method to convert the <code>Map<int, ObjectRef></code> type to the <code>Matrix<ObjectRef></code> type. Select min_key, max_key, average, or summation.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

METHOD : string type. The method to convert the `Map<int, ObjectRef>` type to the `Matrix<ObjectRef>` type. Select min_key, max_key, average, or summation. Selecting min_key or max_key outputs a `Matrix<float>` or a `Matrix<complex<float>>` whose key is the minimum or the maximum among the input `Map<int, ObjectRef>`. Selecting average or summation outputs a `Matrix<float>` or a `Matrix<complex<float>>` whose values are the summation of or the average of the values of the input `Map<int, ObjectRef>`. The default value is min_key.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Table 6.108: Conversion table of MapToMatrix

INPUT	METHOD	OUTPUT	
Map< int , Matrix<float> >	min_key	Matrix<float>	(1)
	max_key		(2)
	average		(3)
	summation		(4)
Map< int , Matrix<complex<float> > >	min_key	Matrix<complex<float> >	
	max_key		
	average		
	summation		

Details of the node

<example>

INPUT: Three input values, each value consists of a key and a 2x2 matrix.

$$\left\{ 0, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} \right\}$$

OUTPUT(1): a 2x2 matrix for key 0.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

OUTPUT(2): a 2x2 matrix for key 1.

$$\begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix}$$

OUTPUT(3): a 2x2 matrix, the average of matrices whose key is from 0 to 2.

$$\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

OUTPUT(4): a 2x2 matrix, the summation of matrices whose key is from 0 to 2.

$$\begin{bmatrix} 15 & 18 \\ 21 & 24 \end{bmatrix}$$

6.7.14 MapToVector

Outline of the node

Convert the `Map<int, ObjectRef>` type to the `Vector<ObjectRef>` type when the `ObjectRef` of the given `Map<int, ObjectRef>` is either the `Vector<float>` type or the `Vector<complex<float> >` type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the `Map<int, ObjectRef>` type to the `Vector<ObjectRef>` type when the `ObjectRef` of the given `Map<int, ObjectRef>` is either the `Vector<float>` type or the `Vector<complex<float> >` type. Taking a `Map< int , Vector<float> >` as the input will output a `Vector<float>` . Taking a `Map< int , Vector<complex<float> > >` as the input will output a `Vector<complex<float> >` .

Input-output and property of the node

Input

INPUT : `Map< int , Vector<float> >` or `Map< int , Vector<complex<float> > >` of the `Map<int, ObjectRef>` type.

Output

OUTPUT : any type. Note that the supported data types are the `Vector<float>` type and the `Vector<complex<float> >` type.

Parameter

Table 6.109: Parameter list of MapToVector

Parameter name	Type	Default value	Unit	Description
METHOD	string	min_key		The method to convert the <code>Map<int, ObjectRef></code> type to the <code>Vector<ObjectRef></code> type. Select min_key, max_key, average, or summation.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

METHOD : string type. The method to convert the `Map<int, ObjectRef>` type to the `Vector<ObjectRef>` type. Select min_key, max_key, average, or summation. Selecting min_key or max_key outputs a `Vector<float>` or a `Vector<complex<float> >` whose key is the minimum or the maximum among the input `Map<int, ObjectRef>` . Selecting average or summation outputs a `Vector<float>` or a `Vector<complex<float> >` whose values are the summation of or the average of the values of the input `Map<int, ObjectRef>` . The default value is min_key.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Table 6.110: Conversion table of MapToVector

INPUT	METHOD	OUTPUT	
Map< int , Vector<float> >	min_key	Vector<float>	(1)
	max_key		(2)
	average		(3)
	summation		(4)
Map< int , Vector<complex<float> > >	min_key	Vector<complex<float> >	
	max_key		
	average		
	summation		

Details of the node

<example>

INPUT: Three input values, each value consists of a key and a 3 element vector.

$$\{ 0, < 1 \ 2 \ 3 > \}, \quad \{ 1, < 4 \ 5 \ 6 > \}, \quad \{ 2, < 7 \ 8 \ 9 > \}$$

OUTPUT(1): a 3 element vector for key 0.

$$< 1 \ 2 \ 3 >$$

OUTPUT(2): a 3 element vector for key 2.

$$< 7 \ 8 \ 9 >$$

OUTPUT(3): a 3 element vector, the average of vectors whose key is from 0 to 2.

$$< 4 \ 5 \ 6 >$$

OUTPUT(4): a 3 element vector, the summation of vectors whose key is from 0 to 2.

$$< 12 \ 15 \ 18 >$$

6.7.15 MapVectorValueOverwrite

Outline of the node

Overwrite a specified subvector of a `Vector<ObjectRef>` with a specified value where the `Vector<ObjectRef>` is an `ObjectRef` of a `Map<int, ObjectRef>`.

Necessary file

No files are required.

Usage

When to use

This node is used to overwrite a specified subvector of a `Vector<ObjectRef>` with a specified value where the `Vector<ObjectRef>` is an `ObjectRef` of a `Map<int, ObjectRef>`. The data type of the specified value will be converted to the suitable one depending on the `ObjectRef` of the `Map<int, Vector<ObjectRef>>` given as the input.

Input-output and property of the node

Input

INPUT : `Map<int, Vector<ObjectRef>>` type; `Map<int, Vector<int>>`, `Map<int, Vector<float>>`, or `Map<int, Vector<complex<float>>>`.

Output

OUTPUT : `Map<int, Vector<ObjectRef>>` type; `Map<int, Vector<int>>`, `Map<int, Vector<float>>`, or `Map<int, Vector<complex<float>>>`.

Parameter

OVERWRITTEN_MIN : `int` type. The element index of the `Vector<ObjectRef>` of the `Map<int, Vector<ObjectRef>>` given as the input to specify the first element of the subvector on which overwrite. The default value is 0.

OVERWRITTEN_MAX : `int` type. The element index of the `Vector<ObjectRef>` of the `Map<int, Vector<ObjectRef>>` given as the input to specify the last element of the subvector on which overwrite. The default value is 0.

OVERWRITE_VALUE_REAL : `float` type. The value with which overwrite a specified subvector. When the **INPUT** is `Map<int, Vector<int>>`, the type will be converted to the `int` type. When the **INPUT** is `Map<int, Vector<complex<float>>>`, the value will be for the real part. The default value is 0.

OVERWRITE_VALUE_IMAG : `float` type. The value for the imaginary part with which overwrite a specified subvector when the `ObjectRef` of the `Map<int, Vector<ObjectRef>>` given as the input is the complex type. The default value is 0.

DEBUG : `bool` type. Setting the value to `true` outputs the overwrite status to the standard output. The default value is `false`.

Table 6.111: Parameter list of MapVectorValueOverwrite

Parameter name	Type	Default value	Unit	Description
OVERWRITTEN_MIN	int	0		The element index of the Vector given as the input to specify the first element of the subvector on which overwrite.
OVERWRITTEN_MAX	int	0		The element index of the Vector given as the input to specify the last element of the subvector on which overwrite.
OVERWRITE_VALUE_REAL	float	0		The value with which overwrite a specified subvector. The data type will be converted to the suitable one depending on the ObjectRef of the Map<int, Vector<ObjectRef>> given as the input.
OVERWRITE_VALUE_IMAG	float	0		The value for the imaginary part with which overwrite a specified subvector when the ObjectRef of the Map<int, Vector<ObjectRef>> given as the input is the complex type.
DEBUG	bool	false		Enable or disable to output the overwriting status to standard output.

Details of the node

<example>

PARAMETER:

OVERWRITTEN_MIN:1,
OVERWRITTEN_MAX:2,
OVERWRITE_VALUE_REAL:9

INPUT:

{0, <1 2 3 4>}, {1, <3 4 5 6>}, {2, <5 6 7 8>}

OUTPUT:

{0, <1 9 9 4>}, {1, <3 9 9 6>}, {2, <5 9 9 8>}

6.7.16 MatrixToMap

Outline of the node

Convert the `Matrix<ObjectRef>` type to the `Map<int, ObjectRef>` type when the `ObjectRef` of the `Matrix<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type.

Necessary files

No files are required.

Usage

When to use

This node is used to convert the `Matrix<ObjectRef>` type to the `Map<int, ObjectRef>` type when the `ObjectRef` of the `Matrix<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type. Taking a `Matrix<float>` as the input will output `Map<int, Matrix<float>>`. Taking a `Matrix<complex<float>>` as the input will output a `Map<int, Matrix<complex<float>>>`. This node is for connecting to the nodes whose data type for the input is the `Map<int, ObjectRef>` type, for instance, `PreEmphasis`, `MelFilterBank`, or `SaveRawPCM`.

Typical connection

Figure ?? and ?? show connection examples for the `MatrixToMap`. The sample network in Figure ?? takes speech waveform data from microphones using `AudioStreamFromMic`, selects the necessary channels using `ChannelSelector`, and converts the data in the `Matrix<float>` type to the one in the `Map<int, ObjectRef>` type using `MatrixToMap`. By connecting the OUTPUT terminal of `MatrixToMap` to the INPUT terminal of `SaveRawPCM`, save the output data in a file.

The sample network in Figure ?? shows how to use `MatrixToMap` when spectra of waveform data are desired in `Map<int, ObjectRef>` data type. As shown in Figure ??, `MultiFFT` can be connected to `MatrixToMap`. The order of arrangement of nodes does not matter, `MatrixToMap` can be placed before or after `MultiFFT`.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are the `Matrix<float>` type and the `Matrix<complex<float>>` type.

Output

OUTPUT : `Map<int, Matrix<float>>`, `Map<int, Matrix<complex<float>>>`, `Map<int, Vector<float>>`, or `Map<int, Vector<complex<float>>>` of the `Map<int, ObjectRef>` type.

Parameter

Table 6.112: Parameter list of `MatrixToMap`

Parameter name	Type	Default value	Unit	Description
OUTPUT_TYPE	string	map_of_row_vectors		The data type of the output data. Select one from <code>map_of_matrix</code> , <code>map_of_row_vectors</code> , or <code>map_of_column_vectors</code> .
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

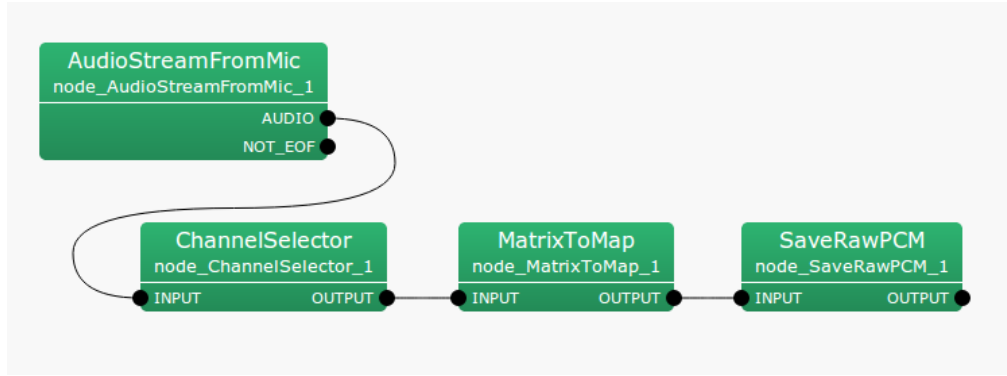


Figure 6.114: Connection example of MatrixToMap to SaveRawPCM

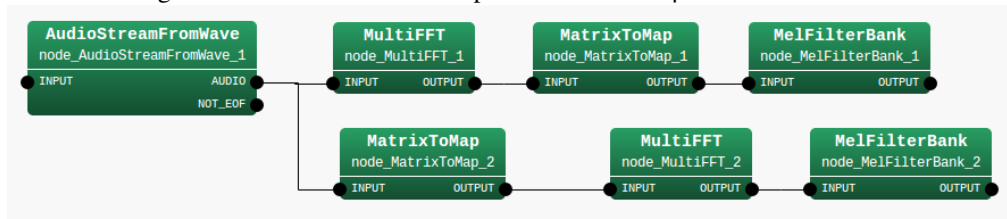


Figure 6.115: Connection example of MatrixToMap to MultiFFT

OUTPUT_TYPE : string type. The data type of the output data. Select one from map_of_matrix, map_of_row_vectors, or map_of_column_vectors. Selecting map_of_matrix outputs a Map whose key is 0 and whose ObjectRef is the Matrix given as the input. Selecting map_of_row_vectors outputs Map as many as the number of rows of the Matrix given as the input. The each Maps key is the row index of the Matrix and the each ObjectRef is a Vector consisting of the row components of the Matrix . Selecting map_of_column_vectors outputs Map as many as the number of columns of the Matrix . The each Map's key is the column index of the Matrix and the each ObjectRef is a Vector consisting of the column components of the Matrix . The default value is map_of_row_vectors.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Details of the node

Table 6.113: Conversion table of MatrixToMap

INPUT		OUT_TYPE	OUTPUT		
type	size		type	size	
Matrix<float>	NxM	map_of_matrix	Map<int , Matrix<float> >	1x{NxM}	(1)
		map_of_row_vectors	Map<int , Vector<float> >	Nx{M}	(2)
		map_of_column_vectors		Mx{N}	(3)
Matrix<complex<float> >		map_of_matrix	Map<int , Matrix<complex<float> > >	1x{NxM}	
		map_of_row_vectors	Map<int , Vector<complex<float> > >	Nx{M}	
		map_of_column_vectors		Mx{N}	

<example>

INPUT:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

OUTPUT(1):

$$\left\{ 0, \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \right\}$$

OUTPUT(2):

$$\{ 0, < 1 \ 2 > \}, \quad \{ 1, < 3 \ 4 > \}, \quad \{ 2, < 5 \ 6 > \}$$

OUTPUT(3):

$$\{ 0, < 1 \ 3 \ 5 > \}, \quad \{ 1, < 2 \ 4 \ 6 > \}$$

6.7.17 MatrixToMatrix

Outline of the node

Convert the data type of the ObjectRef of the Matrix<ObjectRef> given as the input, between the Matrix<float> type and the Matrix<complex<float> > type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the data type of the ObjectRef of the Matrix<ObjectRef> given as the input; the Matrix<float> type to the Matrix<complex<float> > type, the Matrix<complex<float> > type to the Matrix<float> type.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are the Matrix<float> type and the Matrix<complex<float> > type.

Output

OUTPUT : any type. Note that the supported data types are the Matrix<float> type and the Matrix<complex<float> > type.

Parameter

Table 6.114: Parameter list of MatrixToMatrix

Parameter name	Type	Default value	Unit	Description
METHOD_COMPLEX_TO_FLOAT	string	magnitude		The method to convert Matrix<complex<float> > to Matrix<float> . Select magnitude, real, or imaginary. Output the absolute value, the real part, and the imaginary part, which are of the complex number, respectively.
METHOD_FLOAT_TO_COMPLEX	string	zero		The method to convert Matrix<float> to Matrix<complex<float> > . Select zero or hilbert. For the imaginary part, output 0 or the absolute value of the real part after converted to complex .
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

METHOD_COMPLEX_TO_FLOAT : string type. The method to convert the Matrix<complex<float> > type to the Matrix<float> type. Select magnitude, real, or imaginary. Output the absolute value, the real part, and the imaginary part, which are of the complex number, respectively. The default value is magnitude.

METHOD_FLOAT_TO_COMPLEX : string type. The method to convert the `Matrix<float>` type to the `Matrix<complex<float> >` type. Select zero or hilbert. For the imaginary part, output 0 by selecting zero or output the absolute value of the real part after converted to complex by selecting hilbert. The default value is zero.

DEBUG : bool type. Setting the value to `true` outputs the conversion status to the standard output. The default value is `false`.

Details of the node

Table 6.115: Conversion table of `MatrixToMatrix`

INPUT	OUTPUT	Parameter to use
<code>Matrix<float></code>	<code>Matrix<complex<float> ></code>	<code>METHOD_FLOAT_TO_COMPLEX</code>
<code>Matrix<complex<float> ></code>	<code>Matrix<float></code>	<code>METHOD_COMPLEX_TO_FLOAT</code>

6.7.18 MatrixToVector

Outline of the node

Convert the `Matrix<ObjectRef>` type to the `Vector<ObjectRef>` type when the `ObjectRef` of the `Matrix<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the `Matrix<ObjectRef>` type to the `Vector<ObjectRef>` type when the `ObjectRef` of the `Matrix<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type. Taking a `Matrix<float>` as the input will output a `Vector<float>`. Taking a `Matrix<complex<float>>` as the input will output a `Vector<complex<float>>`.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are the `Matrix<float>` type and the `Matrix<complex<float>>` type.

Output

OUTPUT : any type. Note that the supported data types are the `Vector<float>` type and the `Vector<complex<float>>` type.

Parameter

METHOD : string type. The method to convert the `Matrix<ObjectRef>` type to the `Vector<ObjectRef>` type; `Matrix<float>` to `Vector<float>`, or `Matrix<complex<float>>` to `Vector<complex<float>>`. Select reshape or accumulate. Selecting reshape reshapes the `Matrix<ObjectRef>` given as the input into a `Vector` by row-major order or by column-major order. Selecting accumulate calculates the elements of the `Matrix<ObjectRef>` by rows or by columns and then replaces the elements with the accumulated value. The default value is reshape.

RESHAPE_ORDER : string type. The order to reshape the input `Matrix`. When the parameter value of **METHOD** is set to reshape, specify the desired value. Select row, or column. Selecting row reshapes the matrix by row-major order. Selecting column reshapes the matrix by column-major order. The default value is row.

ACCUMULATE_METHOD : string type. The method to accumulate the elements of the `Matrix` when the parameter value of **METHOD** is set to accumulate. Select row_sum, col_sum, row_avg, or col_avg. Calculate the sum of elements in each row, the sum of elements in each column, the average of the elements in each row, and the average of the elements in each column, respectively. The default value is row_sum.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Table 6.116: Parameter list of MatrixToVector

Parameter name	Type	Default value	Unit	Description
METHOD	string	reshape		The method to convert Matrix to Vector . Select reshape or accumulate. Convert by reshaping the matrix with the original elements unchanged or by replacing the original elements of the matrix with the accumulated value.
RESHAPE_ORDER	string	row		The order for storing multidimensional arrays in linear storage. Select row or column. Reshape the matrix given into a vector by row-major order or by column-major order.
ACCUMULATE_METHOD	string	row_sum		The method to accumulate the elements of the matrix. Select one from row_sum, col_sum, row_avg, or col_avg. Calculate the sum of the elements in each row, the sum of the elements in each column, the average the elements in each row, and the average of the elements in each column, respectively
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

Table 6.117: Conversion table of MatrixToVector

INPUT		METHOD	RESHAPE_ORDER	ACCUMULATE_METHOD	OUTPUT	
type	size				type	size
Matrix<float>	NxM	reshape	row	-	Vector<float>	(NxM) (1)
			column			(2)
		accumulate	-	row_sum		N (3)
				row_avg		(4)
				col_sum		M (5)
Matrix<complex<float> >	NxM	reshape	row	-	Vector<complex<float> >	(NxM) (6)
			column			
		accumulate	-	row_sum		N
				row_avg		
				col_sum		M
				col_avg		

Details of the node

<example>

INPUT:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

OUTPUT(1):

< 1 2 3 4 5 6 >

OUTPUT(2):

< 1 3 5 2 4 6 >

OUTPUT(3):

< 3 7 11 > \leftarrow < 1+2 3+4 5+6 >

OUTPUT(4):

$$\langle 1.5 \ 3.5 \ 5.5 \rangle \longleftarrow \langle (1+2)/2 \ (3+4)/2 \ (5+6)/2 \rangle$$

OUTPUT(5):

$$\langle 9 \ 12 \rangle \longleftarrow \langle 1+3+5 \ 2+4+6 \rangle$$

OUTPUT(6):

$$\langle 3 \ 4 \rangle \longleftarrow \langle (1+3+5)/3 \ (2+4+6)/3 \rangle$$

6.7.19 MatrixValueOverwrite

Outline of the node

Overwrite a specified submatrix of a `Matrix<ObjectRef>` with a specified value.

Necessary file

No files are required.

Usage

When to use

This node is used to overwrite a specified submatrix of a `Matrix<ObjectRef>` with a specified value. The data type of the specified value will be converted to the suitable one depending on the `ObjectRef` of the `Matrix<ObjectRef>` given as the input.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are `Matrix<int>`, `Matrix<float>`, and `Matrix<complex<float>>`.

Output

OUTPUT : any type. Note that the supported data types are `Matrix<int>`, `Matrix<float>`, and `Matrix<complex<float>>`.

Parameter

OVERWRITTEN_ROW_MIN : int type. The row index of the `Matrix<ObjectRef>` given as the input to specify the first row of the submatrix on which overwrite. The default value is 0. The default value is 0.

OVERWRITTEN_ROW_MAX : int type. The row index of the `Matrix<ObjectRef>` given as the input to specify the last row of the submatrix on which overwrite. The default value is 0.

OVERWRITTEN_COL_MIN : int type. The column index of the `Matrix<ObjectRef>` given as the input to specify the first column of the submatrix on which overwrite. The default value is 0.

OVERWRITTEN_COL_MAX : int type. The column index of the `Matrix<ObjectRef>` given as the input to specify the last column of the submatrix on which overwrite. The default value is 0.

OVERWRITE_VALUE_REAL : float type. The value with which overwrite a specified submatrix. When the **INPUT** is `Matrix<int>`, the type will be converted to the int type. When the **INPUT** is `Matrix<complex<float>>`, the value will be for the real part. The default value is 0.

OVERWRITE_VALUE_IMAG : float type. The value for the imaginary part with which overwrite a specified submatrix when the `ObjectRef` of the `Matrix<ObjectRef>` given as the input is the complex type. The default value is 0.

DEBUG : bool type. Setting the value to `true` outputs the overwrite status to the standard output. The default value is `false`.

Table 6.118: Parameter list of MatrixValueOverwrite

Parameter name	Type	Default value	Unit	Description
OVERWRITTEN_ROW_MIN	int	0		The row index of the Matrix given as the input to specify the first row of the submatrix on which overwrite.
OVERWRITTEN_ROW_MAX	int	0		The row index of the Matrix given as the input to specify the last row of the submatrix on which overwrite.
OVERWRITTEN_COL_MIN	int	0		The column index of the Matrix given as the input to specify the first column of the submatrix on which overwrite.
OVERWRITTEN_COL_MAX	int	0		The column index of the Matrix given as the input to specify the last column of the submatrix on which overwrite.
OVERWRITE_VALUE_REAL	float	0		The value with which overwrite a specified submatrix. The data type will be converted to the suitable one depending on the ObjectRef of the Matrix<ObjectRef> given as the input.
OVERWRITE_VALUE_IMAG	float	0		The value for the imaginary part with which overwrite a specified submatrix when the ObjectRef of the Matrix<ObjectRef> given as the input is the complex type.
DEBUG	bool	false		Enable or disable to output the overwriting status to standard output.

Details of the node

<example>

PARAMETER:

OVERWRITTEN_ROW_MIN:0,
OVERWRITTEN_ROW_MAX:0,
OVERWRITTEN_COL_MIN:1,
OVERWRITTEN_COL_MAX:2,
OVERWRITE_VALUE_REAL:9

INPUT:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}, \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

OUTPUT:

$$\begin{bmatrix} 1 & 9 & 9 \\ 4 & 5 & 6 \end{bmatrix}, \begin{bmatrix} 2 & 9 & 9 \\ 5 & 6 & 7 \end{bmatrix}, \begin{bmatrix} 3 & 9 & 9 \\ 6 & 7 & 8 \end{bmatrix}$$

6.7.20 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 $\text{SAMPLING_RATE_IN} / \text{SAMPLING_RATE_OUT}$ 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 $\text{SAMPLING_RATE_IN} / \text{SAMPLING_RATE_OUT}$ 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 $\text{SAMPLING_RATE_IN} / \text{SAMPLING_RATE_OUT}$ times as great as the `ADVANCE` value of the node connected afterward (e.g. `GHDSS`). For example, if $\text{SAMPLING_RATE_IN} / \text{SAMPLING_RATE_OUT}$ 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 ?? shows the relationships between symbols and filter properties. Note the correspondence when reading them.

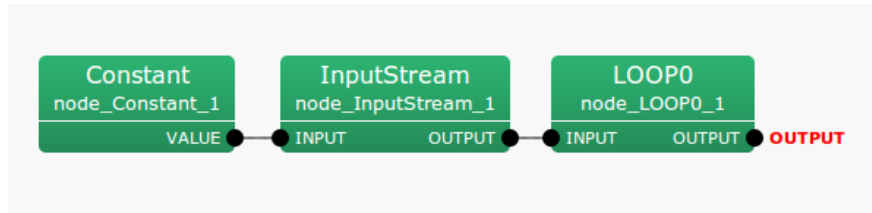


Figure 6.116: Example of a connection of MultiDownSampler : Main network

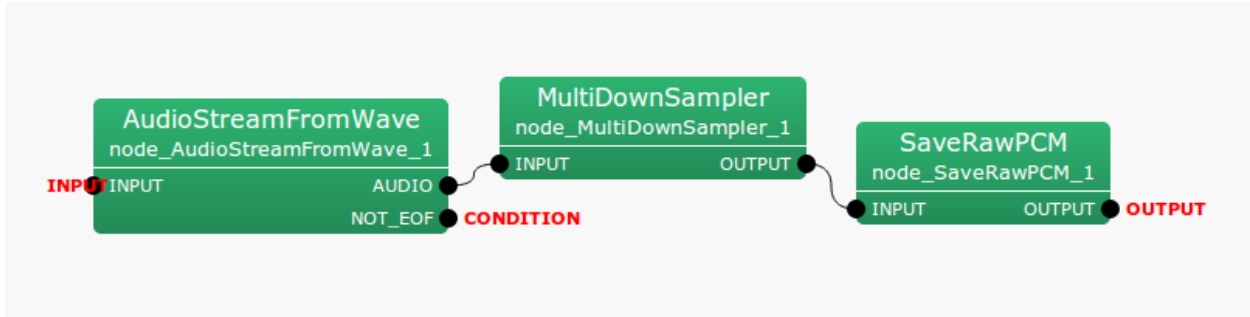


Figure 6.117: Example of a connection of MultiDownSampler : Iteration (LOOP0) Network

Table 6.119: 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 filter 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 W_p , W_s and A_s are set at their default values, W_p and W_s will be around the cutoff frequency W_s . 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.

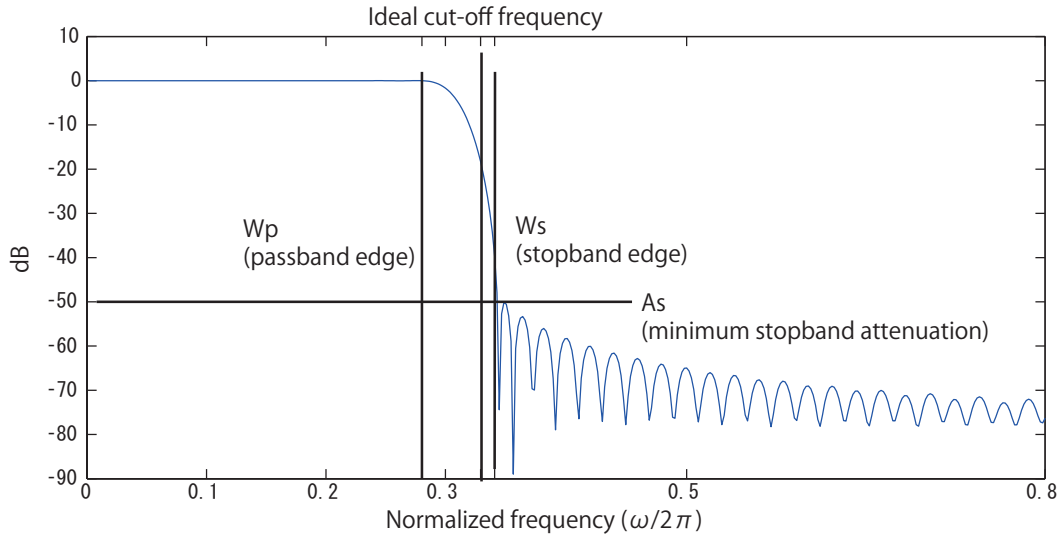


Figure 6.118: Filter properties of default settings.

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 $\text{SAMPLING_RATE_OUT}/\text{SAMPLING_RATE_IN}$ 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

$$s_{\text{out}}(t) = \sum_{i=0}^N h(n)s_{\text{in}}(t - n). \quad (6.165)$$

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

$$H_i(e^{j\omega}) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \text{otherwise} \end{cases} \quad (6.166)$$

This impulse response is expressed as

$$h_i(n) = \frac{\omega_c}{\pi} \left(\frac{\sin(\omega n)}{\omega n} \right), \quad -\infty \leq n \leq \infty \quad (6.167)$$

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.

$$h(n) = \begin{cases} h_i(n), & |n| \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \quad (6.168)$$

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 A_s 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.

$$h(n) = h_i(n)v(n) \quad (6.169)$$

Here, the low-pass filter is designed with the Kaiser window. The Kaiser window is defined by the equation

$$v(n) = \begin{cases} \frac{I_0(\beta \sqrt{1-(nN/2)^2})}{I_0(\beta)}, & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \quad (6.170)$$

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

$$I_0(x) = 1 + \sum_{k=1}^{\infty} \left(\frac{(0.5x)^k}{k!} \right) \quad (6.171)$$

The parameter β is determined by the attenuation obtained by the low-pass filter. Here, it is determined by the index,

$$\beta = \begin{cases} 0.1102(As - 8.7) & As > 50, \\ 0.5842(As - 21)^{0.4} + 0.07886(As - 21) & 21 < As < 50, \\ 0 & As < 21 \end{cases} \quad (6.172)$$

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,

$$N \approx \frac{As - 7.95}{14.36\Delta f} \quad (6.173)$$

Moreover, the cutoff frequency ω_c is set to $0.5(Wp + Ws)$. **Downsampling:** Downsampling is realized by thinning

the sample points of $\text{SAMPLING_RATE_IN} / \text{SAMPLING_RATE_OUT}$ 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.

6.7.21 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 ?? shows an example, in which inputs of `Matrix<float>` and `Map<int, ObjectRef>` types are provided to the MultiFFT node. The path in Figure ?? 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.

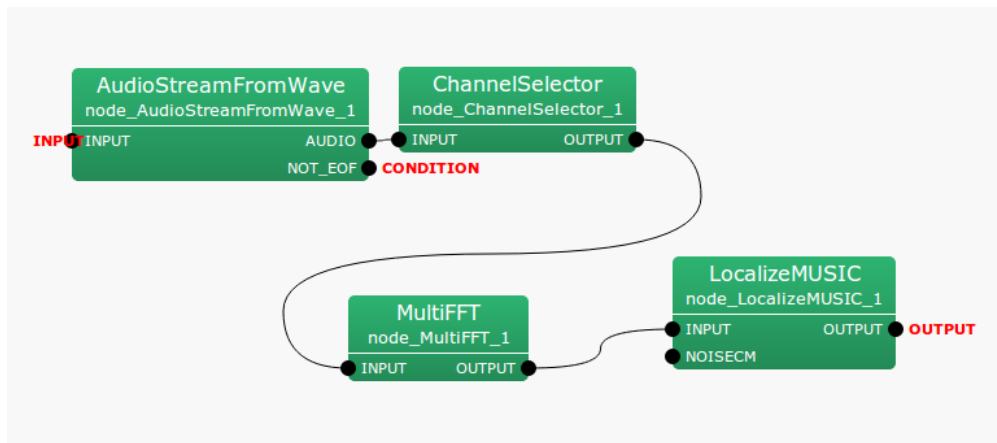


Figure 6.119: Example of a connection of MultiFFT

Input-output and property of the node

Table 6.120: 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, RECTANGLE and HANNING which indicate complex window, hamming window, rectangle window and hanning 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 RECTANGLE` and `HANNING`, which indicate complex, hamming, rectangular and hanning 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 ?? and ?? show the shape and frequency responses of the complex window function.

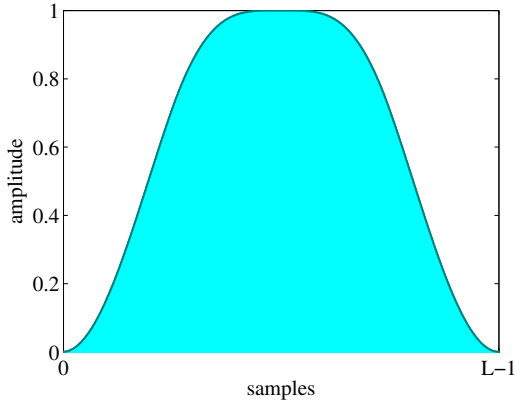


Figure 6.120: Shape of complex window function

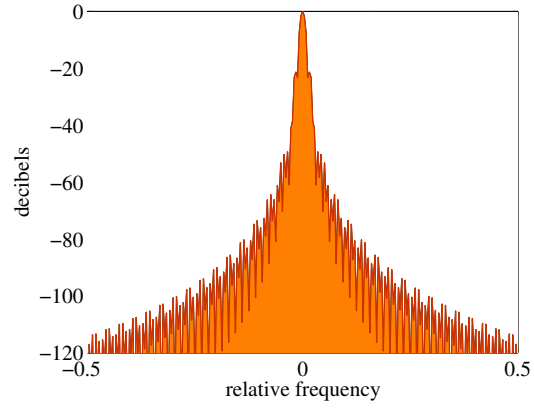


Figure 6.121: Frequency response of complex window function

The horizontal axis in Figure ?? 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:

$$w(k) = \begin{cases} 0.54 - 0.46 \cos \frac{2\pi k}{L-1}, & \text{if } 0 \leq k < L, \\ 0, & \text{if } L \leq k < NFFT \end{cases}$$

Here, π indicates a circular constant. Figures ?? and ?? show the shape and frequency responses of the hamming

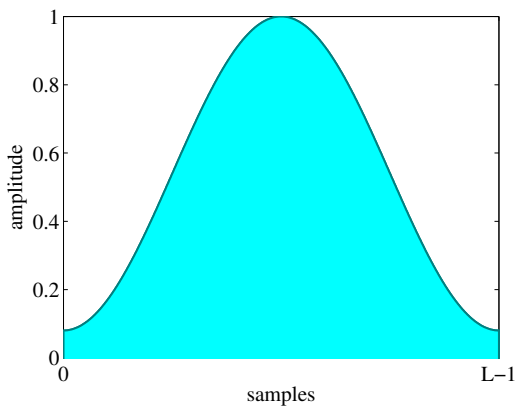


Figure 6.122: Shape of hamming window function

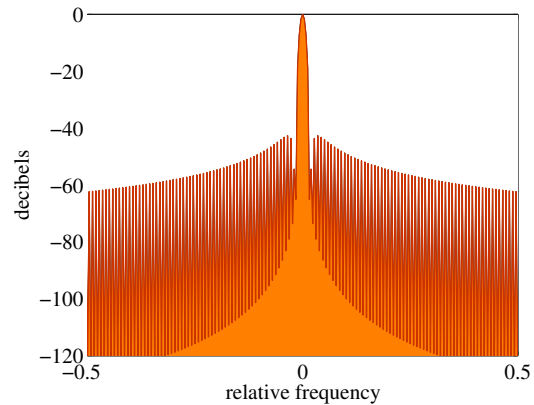


Figure 6.123: Frequency response of hamming window function

window function, respectively.

RECTANGLE, Rectangle window:

$$w(k) = \begin{cases} 1, & \text{if } 0 \leq k < L \\ 0, & \text{if } L \leq k < NFFT \end{cases}$$

Figures ?? and ?? show the shape and frequency responses of the rectangular window function, respectively.

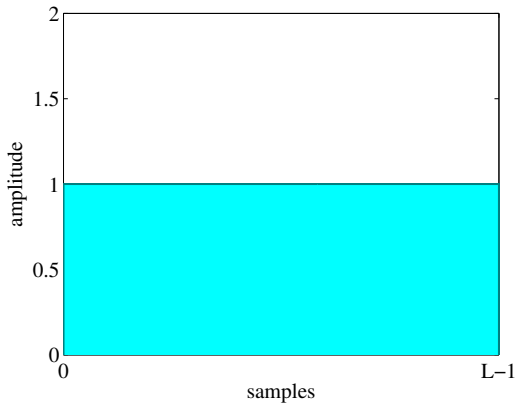


Figure 6.124: Shape of rectangle window function

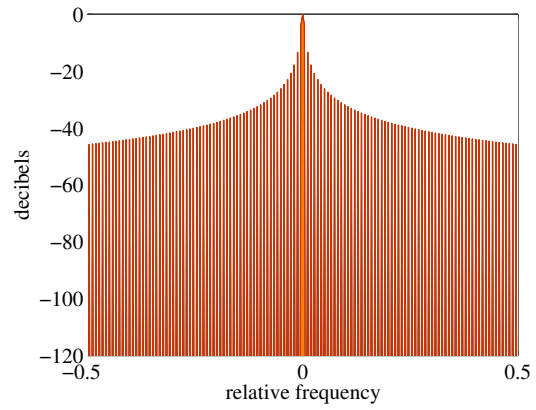


Figure 6.125: Frequency response of rectangle window function

HANNING, hanning window:

$$w(k) = \begin{cases} 0.5 - 0.5 \cos \frac{2\pi k}{L-1}, & \text{if } 0 \leq k < L, \\ 0, & \text{if } L \leq k < NFFT \end{cases}$$

6.7.22 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.

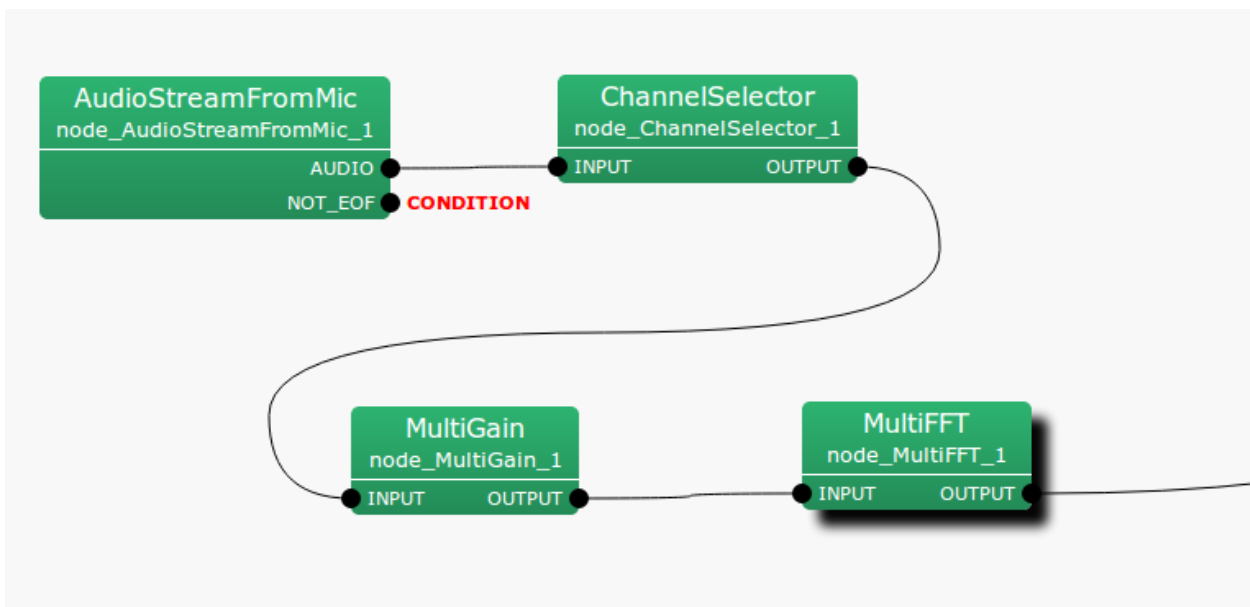


Figure 6.126: Example of a connection of MultiGain

Input-output and properties of the node

Table 6.121: 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.

$$20 \log x = -40 \quad (6.174)$$

$$x = 0.01 \quad (6.175)$$

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 ?? 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.

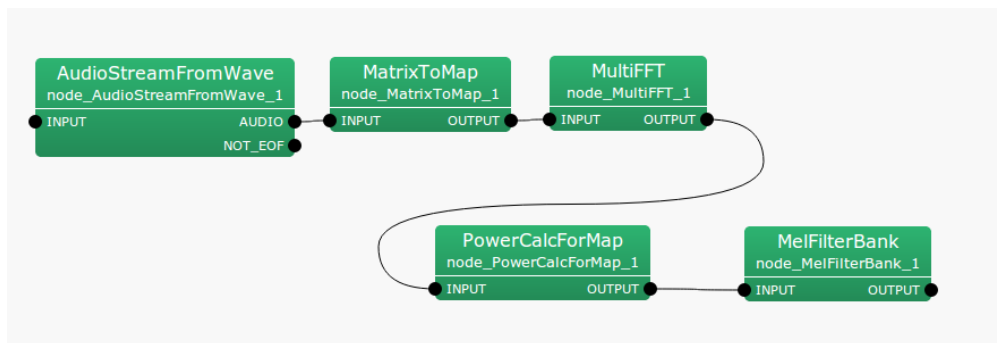


Figure 6.127: Example of a connection of `PowerCalcForMap`

Input-output and properties of the node

Table 6.122: 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:

$$\begin{aligned} N_{i,j} &= M_{i,j}M_{i,j}^* \text{ (if POWER_TYPE=POW),} \\ N_{i,j} &= \text{abs}(M_{i,j}) \text{ (if POWER_TYPE=MAG),} \end{aligned}$$

Here, $M_{i,j}^*$ indicates the complex conjugate of $M_{i,j}$.

6.7.23 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 ?? 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.

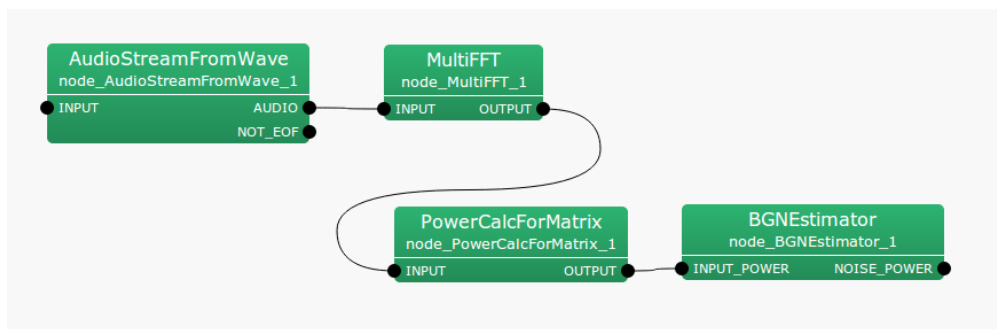


Figure 6.128: Example of a connection of `PowerCalcForMatrix`

Input-output and properties of the node

Table 6.123: 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.

$$\begin{aligned} N_{i,j} &= M_{i,j}M_{i,j}^* \text{ (if POWER_TYPE=POW),} \\ N_{i,j} &= \text{abs}(M_{i,j}) \text{ (if POWER_TYPE=MAG),} \end{aligned}$$

Here, $M_{i,j}^*$ indicates the complex conjugate of $M_{i,j}$.

6.7.24 ResizeMapMatrixValues

Outline of the node

Change the size of a **Matrix** , an **ObjectRef** of a **Map<int, ObjectRef>** type.

Necessary file

No files are required.

Usage

When to use

This node is used to change the size of a **Matrix** , an **ObjectRef** of a **Map<int, ObjectRef>** . When reducing the size, truncate elements from the end of the **Matrix** as many as needed. When expanding the size, add 0s as elements to the end of the **Matrix** as many as needed.

Input-output and property of the node

Input

INPUT : **Map<int, Matrix<int> >**, **Map<int, Matrix<float> >**, or **Map<int, Matrix<complex<float> >>** of **Map<int, ObjectRef>** type.

Output

INPUT : **Map<int, Matrix<int> >**, **Map<int, Matrix<float> >**, or **Map<int, Matrix<complex<float> >>** of **Map<int, ObjectRef>** type.

Parameter

Table 6.124: Parameter list of **ResizeMapMatrixValues**

Parameter name	Type	Default value	Unit	Description
RESIZE_TYPE	string	RELATIVE		The way of using the SIZE_ROW parameter value and the SIZE_COLUMN parameter value. Select RELATIVE or ABSOLUTE. Indicate the relative value or the absolute value, respectively.
SIZE_ROW	int	0		The number of rows to add to the row size or the row size with which replace the row size. The operation performed will depend on the RESIZE_TYPE parameter value.
SIZE_COLUMN	int	0		The number of columns to add to the column size or the column size with which replace the column size. The operation will depend on the RESIZE_TYPE parameter value.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

RESIZE_TYPE : string type. The way of using the SIZE_ROW parameter value and the SIZE.COLUMN parameter value. Select RELATIVE or ABSOLUTE. RELATIVE resizes a **Matrix** by adding the SIZE_ROW parameter value and the SIZE.COLUMN parameter value to the **Matrix** size. ABSOLUTE resizes a **Matrix** by replacing the **Matrix** size with the SIZE_ROW parameter value and the SIZE.COLUMN parameter value. The default value is RELATIVE.

SIZE_ROW : int type. The number of rows to add to the row size or the row size with which replace the row size. The operation performed will depend on the RESIZE_TYPE parameter value.(*) The default value is 0.

SIZE_COLUMN : int type. The number of columns to add to the column size or the column size with which replace the column size. The operation will depend on the RESIZE_TYPE parameter value.(*)

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

(*)

When RELATIVE is selected for the RESIZE_TYPE parameter value, the Matrix size after change will be (A+SIZE_ROW, B+SIZE_COLUMN) for the Matrix whose size is (A, B). When the Matrix size after change is smaller than the size before change, the elements will be truncated from the end of the Matrix as many as needed. When the Matrix size after change is larger than the size before change, 0s will be added as many as needed from the end of the Matrix. When the Matrix size after change goes negative, an error will be generated. Output an empty Matrix when the Matrix size is 0.

Details of the node

<example>

PARAMETER:

RESIZE_TYPE:RELATIVE,
SIZE_ROW:1,
SIZE_COLUMN:2

INPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} \right\}$$

OUTPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 & 0 & 0 \\ 7 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 & 0 & 0 \\ 11 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right\}$$

PARAMETER:

RESIZE_TYPE:RELATIVE,
SIZE_ROW:-1,
SIZE_COLUMN:-1

INPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} \right\}$$

OUTPUT:

$$\left\{ 0, \begin{bmatrix} 1 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 \end{bmatrix} \right\}$$

PARAMETER:

RESIZE_TYPE:ABSOLUTE,
SIZE_ROW:1,
SIZE_COLUMN:5

INPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} \right\}$$

OUTPUT:

$$\left\{ 0, \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 & 0 & 0 & 0 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 & 0 & 0 & 0 \end{bmatrix} \right\}$$

6.7.25 ResizeMapVectorValues

Outline of the node

Change the size of a `Vector`, an `ObjectRef` of a `Map<int, ObjectRef>` type.

Necessary file

No files are required.

Usage

When to use

This node is used to change the size of a `Vector`, an `ObjectRef` of a `Map<int, ObjectRef>` type. When reducing the size, truncate elements from the end of the `Vector` as many as needed. When expanding the size, add 0s as elements to the end of the `Vector` as many as needed.

Input-output and property of the node

Input

INPUT : `Map<int, Vector<int>>`, `Map<int, Vector<float>>`, or `Map<intType, Vector<complex<float>>>` of `Map<int, ObjectRef>` type.

Output

OUTPUT : `Map<int, Vector<int>>`, `Map<int, Vector<float>>`, or `Map<intType, Vector<complex<float>>>` of `Map<int, ObjectRef>` type.

Parameter

Table 6.125: Parameter list of `ResizeMapVectorValues`

Parameter name	Type	Default value	Unit	Description
RESIZE_TYPE	string	RELATIVE		The way of using the SIZE parameter value. Select RELATIVE or ABSOLUTE. Indicate the relative value and the absolute value, respectively.
SIZE	int	0		The number of elements to add to the Vector or the size with which replace the Vector size. The operation will depend on the RESIZE_TYPE parameter value.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

RESIZE_TYPE : string type. The way of using the SIZE parameter value. Select RELATIVE or ABSOLUTE. RELATIVE resizes a Vector by adding the SIZE parameter value to the Vector size. ABSOLUTE resizes a Vector by replacing the Vector size with the SIZE parameter value. The default value is RELATIVE.

SIZE : int type. The number of elements to add to the Vector or the size with which replace the Vector size. The operation will depend on the RESIZE_TYPE parameter value. When RELATIVE is selected for the RESIZE_TYPE parameter value, the Vector size after change will be (A+SIZE) for the Vector whose size is (A). When the Vector size after change is smaller than the size before change, the elements will be truncated from the end of the Vector as many as needed. When the Vector size after change is larger than the size before

change, 0s will be added as many as needed. When the Vector size after change goes negative, an error will be generated. Outputs an empty Vector when the Vector size is 0.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Details of the node

<example>

PARAMETER:

RESIZE_TYPE:RELATIVE,
SIZE:2

INPUT:

{ 0, <1 2 3> }, { 1, <4 5 6> }, { 2, <7 8 9> }

OUTPUT:

{ 0, <1 2 3 0 0> }, { 1, <4 5 6 0 0> }, { 2, <7 8 9 0 0> }

PARAMETER:

RESIZE_TYPE:RELATIVE,
SIZE:-1

INPUT:

{ 0, <1 2 3> }, { 1, <4 5 6> }, { 2, <7 8 9> }

OUTPUT:

{ 0, <1 2> }, { 1, <4 5> }, { 2, <7 8> }

PARAMETER:

RESIZE_TYPE:ABSOLUTE,
SIZE:4

INPUT:

{ 0, <1 2 3> }, { 1, <4 5 6> }, { 2, <7 8 9> }

OUTPUT:

{ 0, <1 2 3 0> }, { 1, <4 5 6 0> }, { 2, <7 8 9 0> }

PARAMETER:

RESIZE_TYPE:ABSOLUTE,
SIZE:2

INPUT:

{ 0, <1 2 3> }, { 1, <4 5 6> }, { 2, <7 8 9> }

OUTPUT:

{ 0, <1 2> }, { 1, <4 5> }, { 2, <7 8> }

6.7.26 SaveMapFrames

Outline of the node

Saves data of frames stored in the `Map<int, ObjectRef>` type of containers in files separately for each key of the Map. Provides a file containing the data file list for each frame in addition to these data files. Note that the supported `ObjectRef` are the `Vector<float>` type and the `Vector<complex<float>>` type.

Necessary file

No files are required.

Usage

When to use

This node is used to save data of frames separately in a file for each key of the `Map<int, ObjectRef>` type in which the data of frames originally were stored. Each file generated for each key will consist of data mapped to the same key. In addition, a file containing the lists of these files per frame will be generated. In the example network given in the "Typical connection" below, `SaveMapFrames` takes data in the `Map<int, Vector<complex<float>>>` type outputted by `GHDSS` as the input. The data are the pairs of a sound source ID of a separated sound and the complex spectrum of the separated sound. Here, keys are representing sound source IDs. Each file generated for each key will consist of data mapped to the same sound source ID.

Typical connection

Figure ?? shows a connection example using `SaveMapFrames` in a network. In this network, sound source separation is performed using `GHDSS` algorithm, and the output will be saved in files by `SaveMapFrames`. For more details on sound separation based on `GHDSS` algorithm, refer to the node reference of `GHDSS`.

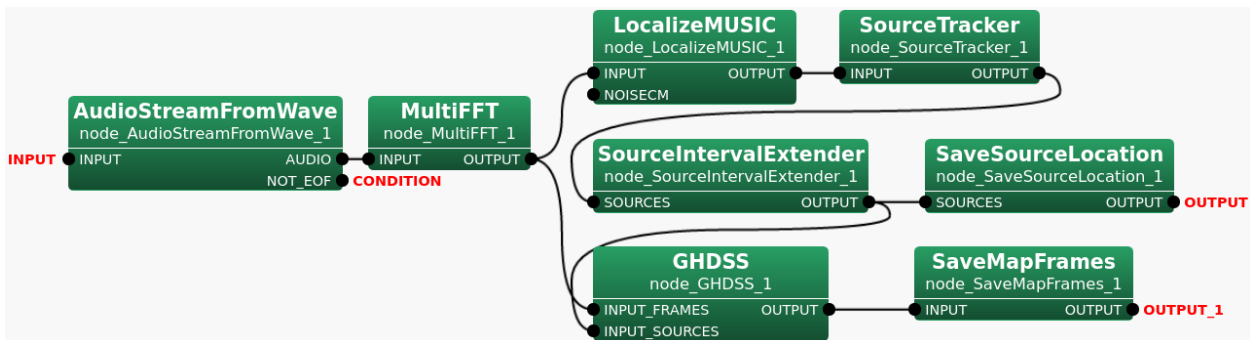


Figure 6.129: Connection example of `SaveMapFrames`

Input-output and property of node

Input

INPUT : Data of frames in the `Map<int, ObjectRef>` type where the data are mapped to keys of the Map. Note that the supported types are the `Map<int, Vector<float>>` type and the `Map<int, Vector<complex<float>>>` type.

Output

OUTPUT : Two types of files. The one is a data file, called "Internal file", and the other one is a list of the "Internal file", called "Main file". The "Internal file" is to be generated as many as the number of keys for the input Map<int, ObjectRef> so the total number of output files will be the number of keys plus 1.

Parameter

Table 6.126: Parameter list of SaveMapFrames

Parameter name	Type	Default value	Unit	Description
FILENAME	string			The file name of the text file to output which contains lists of data files per frame. This file name will be a base name of the file name for the data files to output as well.
OUTPUTTYPE	string	TEXT		The file format options for the data files to output. Select TEXT for text files or RAW for binary files.
TEXTFORMAT	string	FIXED		The file format options when the OUTPUTTYPE is set to TEXT. Select FIXED or SCIENTIFIC.

FILENAME : string type. The file name of the "Main file" in which lists of the "Internal files" for each frame are written.

OUTPUTTYPE : string type. The file format options for the "Internal file". Select TEXT for text files or RAW for binary files. The default value is TEXT. Note that the "Main file" is a text file regardless of the values specified in this parameter.

TEXTFORMAT : string type. The file format options when the OUTPUTTYPE is set to TEXT. Select FIXED or SCIENTIFIC.

Details of the node

SaveMapFrames receives data of frames stored in either the Map<int, Vector<float>> type or the Map<int, Vector<complex<float>>> type of containers for each frame as the input. Then it saves the data in separate files for each key. The output will be two types of files as follows.

Two types of output files:

1. Main file A text file in which the "Internal files (see below)" lists for all frames are written. Each list is separated per frame.
2. Internal files Data files in which data of frames in the Map<int, ObjectRef> type were saved separately for each key. These files will be created as many as the number of the keys. Each file consists of frame data mapped to the same key. The file format is to be specified in the OUTPUTTYPE parameter. When the OUTPUTTYPE parameter is set to RAW, data of frames will be written in IEEE 754 32-bit single-precision floating-point number format by little- endian order. The default value of this parameter, TEXT, will output human-readable data unlike RAW.

For more information on the Map type, refer to Map .

Filename of the output files:

1. Main File - Specify in the FILENAME parameter.
2. Internal Files - Automatically generated by following the pattern below. Note that "TEXT" and "RAW" are the choices of the OUTPUTTYPE parameter. For this node, the "containertype" is always "Vector", the "datatype" is either "float" or "complex_float", and the "colsize" is the size of the data mapped to one key.

TEXT: FILENAME + Map key + {contianertype} + {datatype} + "_col"{colsize} + .txt
RAW: FILENAME + Map key + {contianertype} + {datatype} + "_col"{colsize} + . raw

As indicated above, the difference causing to the filename by selecting TEXT or RAW in the OUTPUTTYPE parameter is only the file extension. An example of file names are given as a sample contents of the Main file below.

Contents of the output files:

1. Main file

Lists of the "Internal files" per frame. When one frame consists of data mapped to multiple different keys, each data will be saved in the file for the corresponding key accordingly so multiple file names will be shown for one frame. Every time data of one frame is saved in a file, the file name is to append to the list followed the frame number. The frame number starts with 0.

An example contents of the "Main file" after data of 6397 frames were processed is given below assuming the related parameter values are set to as follows.

Parameter name	Value
FILENAME	Savedata.map
OUTPUTTYPE	TEXT

Note that the numbers in gray on the left are the line numbers just to help the readers to understand easily how actually file names are written in the "Main file" and so those line numbers are not written in a real file.

Sample contents of the Main file:

```

1 0 savedata.map_0000_Vector_complex_float_col257.txt
  savedata.map_0001_Vector_complex_float_col257.txt
  savedata.map_0002_Vector_complex_float_col257.txt
  savedata.map_0003_Vector_complex_float_col257.txt
  savedata.map_0004_Vector_complex_float_col257.txt
  savedata.map_0005_Vector_complex_float_col257.txt
  savedata.map_0006_Vector_complex_float_col257.txt
  savedata.map_0007_Vector_complex_float_col257.txt

2 1 savedata.map_0000_Vector_complex_float_col257.txt
  savedata.map_0001_Vector_complex_float_col257.txt
  savedata.map_0002_Vector_complex_float_col257.txt
  savedata.map_0003_Vector_complex_float_col257.txt
  savedata.map_0004_Vector_complex_float_col257.txt
  savedata.map_0005_Vector_complex_float_col257.txt
  savedata.map_0006_Vector_complex_float_col257.txt
  savedata.map_0007_Vector_complex_float_col257.txt

:
:
6397 6396 savedata.map_0000_Vector_complex_float_col257.txt
      savedata.map_0001_Vector_complex_float_col257.txt
      savedata.map_0002_Vector_complex_float_col257.txt

```

```

savedata.map_0003_Vector_complex_float_col257.txt
savedata.map_0004_Vector_complex_float_col257.txt
savedata.map_0005_Vector_complex_float_col257.txt
savedata.map_0006_Vector_complex_float_col512.txt
savedata.map_0007_Vector_complex_float_col512.txt

```

When an empty container in the `Map<int, ObjectRef>` type is received, only the frame number will be written in the file for the frame. Below is a sample snippet of the Main file indicating that such empty container was received for several frames. The FILENAME parameter value is "file".

```

1 0
2 1
3 2
:
8 7
9 8 file_0000_Vector_float.txt
10 9 file_0000_Vector_float.txt
11 10 file_0000_Vector_float.txt file_0001_Vector_float.txt
12 11 file_0000_Vector_float.txt file_0001_Vector_float.txt
:

```

Note that the numbers in gray on the left indicate the line numbers just to help the readers to understand easily how actually file names are written in the "Main file" and so those line numbers are not written in a real file.

2. Internal files

Table ?? shows the file format when TEXT is selected in the OUTPUTTYPE parameter. Each data is separated by space and data for each frame are separated by a line feed.

Table 6.127: File Format for TEXT in the OUTPUTTYPE parameter

Datatype	File Format
Float	data[0] data[1] data[2] ... data[n] data[0] data[1] data[2] data[n]
Complex float	data[0].real() data[0].imag() data[1].real() data[1].imag() ... data[n].real() data[n].imag()

Where n is the size of the Vector which is the `ObjectRef` of the input `Map<int, ObjectRef>` in which data of frames are stored.

Vector Size

Vector Size is the size of the `Vector<ObjectRef>` of the `Map<int, Vector<ObjectRef>>` received as the input. Here, `ObjectRef` is either `float` or `complex<float>` for this node. In the sample network above (Figure ??), it will be 257 due to MultiFFT connected in the prior to `SaveMapFrames`. In MultiFFT, the Vector size is calculated using formula, $\text{Length}/2+1$. Window length in MultiFFT is 512 by default. Applying the formula will give the result of 257.

Saving Pattern

For each frame, `SaveMapFrames` saves data of a frame in separate files by keys to which each data is mapped. Figure ?? shows an example wherein data of all 6397 frames are mapped to all keys from 0000 to 0007 and thus data in a 257 sized Vector is being written in each file.

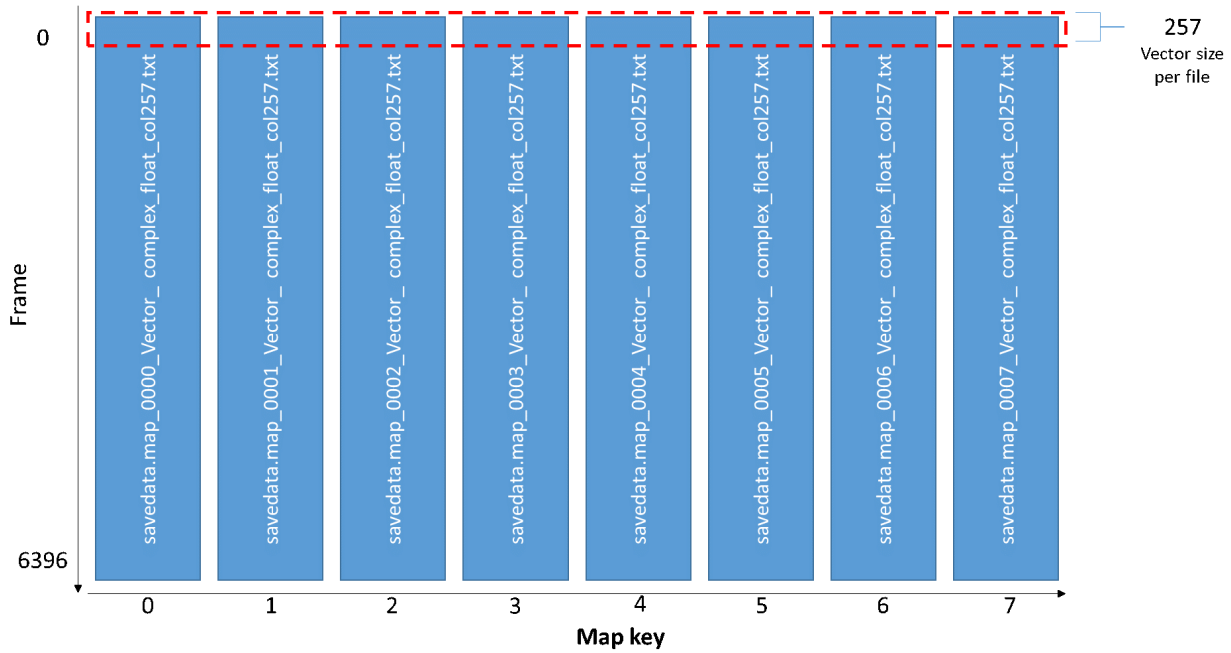


Figure 6.130: Saving pattern for Map with all data present in all frames

In the sample network (Figure ??), the input of `SaveMapFrames` is containers in the `Map<int, ObjectRef>` type, the output of `GHDSS`. Each `Map<int, ObjectRef>` is a pair of a separated sound source ID and the complex spectrum. The Internal files will be generated as many as the number of sound sources. When the data of frames for the input are the results of the sound source separation, it is possible to happen that data of frames are partially not found if all sources do not make any sounds simultaneously or have some duration time. An example of such case is given below in Figure ?. At frame [0], data in a 257 sized Vector is written in files of 0000, 0001 and 0002 sequentially. At frame [100], data in a 257 sized Vector is written in files of 0000, 0002, 0003 and 0004 sequentially.

Where n is the number of sound sources.

Sample contents of the Internal file:

Figure ?? shows how data are written in a TEXT file format described in Table ?. This example assumes that the input data has 6397 frames, Vector size of 512, and the input datatype is `Map<int, Vector<float>>`. These information will be automatically detected internally, no user input will be required. Note that the Frame numbers on the left side of Figure ?? and Figure ?? are not in the files.

Figure ?? gives an example when the input data is in the `Map<int, Vector<complex<float>>>` type, the input data has 6397 frames, and the Vector size is 257 just like in the sample network (Figure ??).

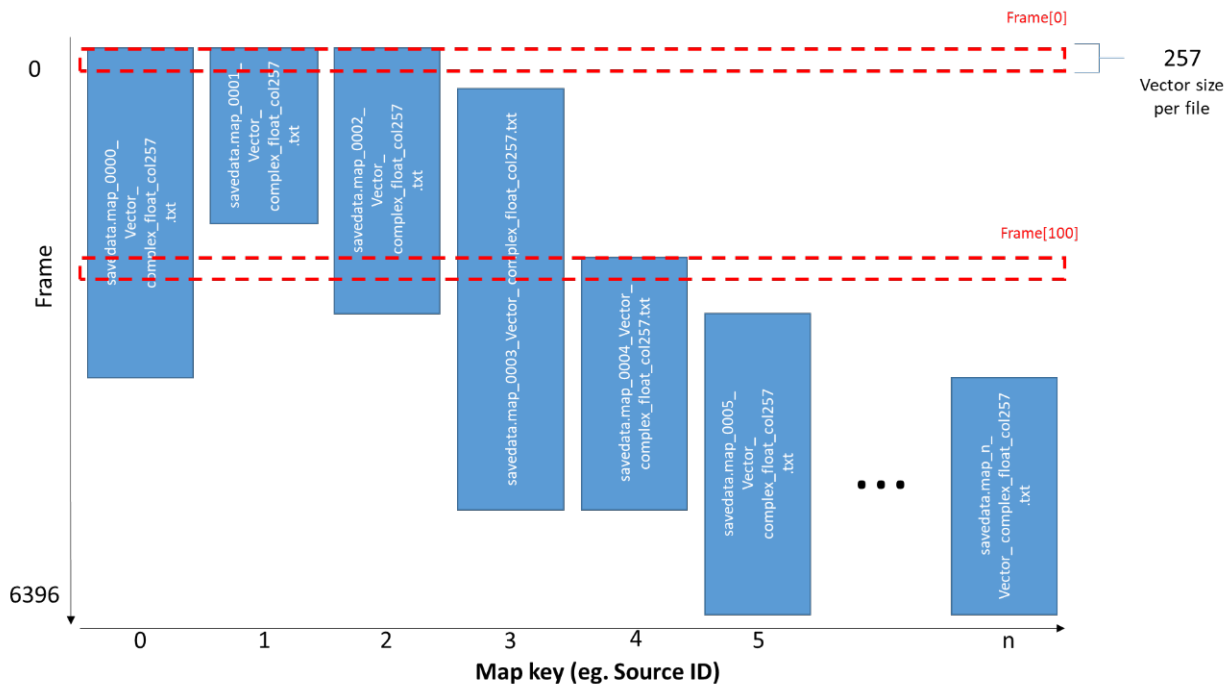


Figure 6.131: Saving pattern for Map with sources that made sounds at different time and duration

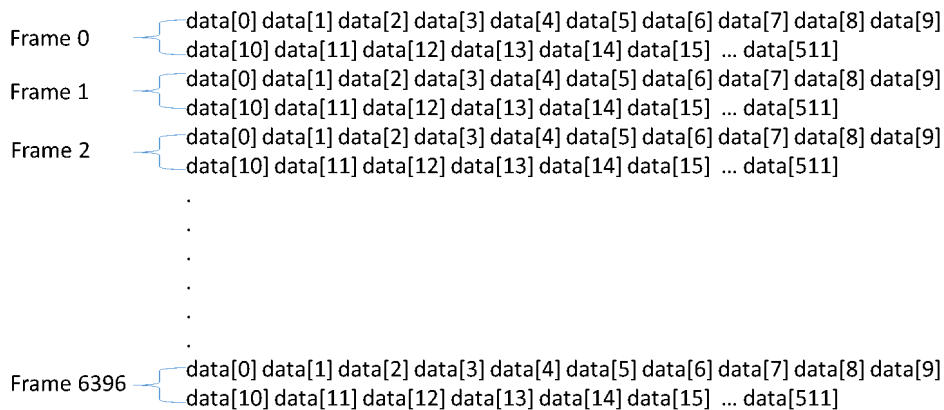


Figure 6.132: Sample SaveMapFrames Internal file content for TEXT float type

Frame 0 { data[0].real() data[0].imag() data[1].real() data[1].imag() data[2].real()
data[2].imag() data[3].real() data[3].imag() data[4].real() data[4].imag()
data[5].real() data[5].imag() data[6].real() data[6].imag() data[7].real()
data[7].imag() ... data[256].real() data[256].imag()

Frame 1 { data[0].real() data[0].imag() data[1].real() data[1].imag() data[2].real()
data[2].imag() data[3].real() data[3].imag() data[4].real() data[4].imag()
data[5].real() data[5].imag() data[6].real() data[6].imag() data[7].real()
data[7].imag() ... data[256].real() data[256].imag()

...

Frame 6396 { data[0].real() data[0].imag() data[1].real() data[1].imag() data[2].real()
data[2].imag() data[3].real() data[3].imag() data[4].real() data[4].imag()
data[5].real() data[5].imag() data[6].real() data[6].imag() data[7].real()
data[7].imag() ... data[256].real() data[256].imag()

Figure 6.133: Sample SaveMapFrames Internal file content for TEXT complex float type

6.7.27 SaveMatrixFrames

Outline of the node

Saves data of frames in the `Matrix<ObjectRef>` type in a file. Note that the supported `ObjectRef` are the `float` type and the `complex<float>` type only.

Necessary file

No files are required.

Usage

When to use

This node is used to save data stored in the `Matrix<ObjectRef>` type for multiple frames in one file.

Typical connection

Figure ?? below shows an example network using `SaveMatrixFrames` where `MultiFFT` outputs data of frames in the `Matrix<complex<float> >` type.

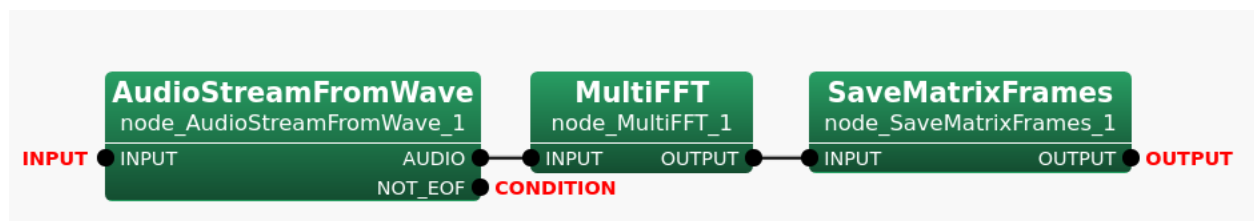


Figure 6.134: Connection example of `SaveMatrixFrames`

Input-output and property of node

Input

INPUT : Data of frames in the `Matrix<ObjectRef>` type. The supported types are the `Matrix<float>` type and the `Matrix<complex<float> >` type.

Output

OUTPUT : A file in which data in the `Matrix<ObjectRef>` type for multiple frames were saved.

Parameter

FILENAME : `string` type. The file name for the output file.

OUTPUTTYPE : `string` type. The file format options for the output file. Select `TEXT` for text files or `RAW` for binary files. The default value is `TEXT`.

TEXTFORMAT : `string` type. The file format options when the `OUTPUTTYPE` is set to `TEXT`. Select `FIXED` or `SCIENTIFIC`.

Table 6.128: Parameter list of SaveMatrixFrames

Parameter name	Type	Default value	Unit	Description
FILENAME	string			The file name of the output file.
OUTPUTTYPE	string	TEXT		The file format options for the output file. Select TEXT for a text file or RAW for a binary file.
TEXTFORMAT	string	FIXED		The file format options when the OUTPUTTYPE is set to TEXT. Select FIXED or SCIENTIFIC.

Details of the node

SaveMatrixFrames takes data in either the `Matrix<float>` type or the `Matrix<complex<float> >` type for multiple frames as the input and then saves all data in one file. The file format is to be specified in the OUTPUTTYPE parameter. When the OUTPUTTYPE parameter is set to RAW, data of frames will be written in IEEE 754 32-bit single-precision floating-point number format by little- endian order. The default value of this parameter, TEXT, will output human-readable data unlike RAW.

Table ?? shows the details of how data will be formatted in the file when TEXT is selected in the OUTPUTTYPE parameter. Each data is separated by space and the data of frames are separated per frame in the size of the Matrix by a line feed.

Table 6.129: File Format for TEXT OUTPUTTYPE

Datatype	File Format
Float	data[0][0] data[0][1] data[1][0] data[1][1] ... data[m][n] data[0][0] data[0][1] data[1][0] data[1][1] ... data[m][n]
Complex float	data[0][0].real() data[0][0].imag() data[0][1].real() data[0][1].imag() ... data[m][n].real() data[m][n].imag()

Where n and m are the column size and the row size of the Matrix for the input in which data of frames were stored.

SaveMatrixFrames checks the size of rows and columns of the Matrix when it receives the data for each frame. In the sample network above (Figure ??), MultiFFT outputs `Matrix<complex<float> >`. When the input matrix size is $M \times L$, the output matrix size is $M \times L/2+1$ for MultiFFT where M stands for the number of channels and L stands for the sample number of waveforms. Window Length in MultiFFT is 512 by default. Applying the formula for L will give 257 for the column size. When the number of channels is 8, the row size will be 8. For more details on MultiFFT output, refer to MultiFFT Node Reference.

Suppose the input data were stored in the `Matrix<float>` type where the column size is 512 and row size is 8, and there are 6397 frames in total, the contents of the output file will be like one shown in Figure ?. It is assumed that the data were saved in the TEXT file format described in the Table ?.

Note that the Frame numbers written on the left side of Figure ? and Figure ? are not written in the files.

Figure ? gives sample contents of the output file when the input data were in the `Matrix<complex<float> >` type like ones in the sample network given by Figure ?. The row size and the column size are 8 and 257, respectively. There are 6397 frames in total. It is also assumed that the data were saved in the TEXT file format described in the Table ?.

Filename:

An option to use a special pattern, {tag:format}, with the data properties as known as a format string is available when specifying the value in the FILENAME parameter. The format string allows the users to set a filename which

```

Frame 0      { data[0][0] data[0][1] data[0][2] data[0][3] ... data[0][511] data[1][0] data[1][1]
               data[1][2] data[1][3] ... data[1][511] data[2][0] data[2][1] data[2][2] data[2][3] ...
               data[2][511] data[3][0] data[3][1] data[3][2] data[3][3] ... data[3][511] data[4][0]
               data[4][1] data[4][2] data[4][3] ... data[4][511] ... data[7][0] data[7][1] data[7][2]
               data[7][3] ... data[7][511]
Frame 1      { data[0][0] data[0][1] data[0][2] data[0][3] ... data[0][511] data[1][0] data[1][1]
               data[1][2] data[1][3] ... data[1][511] data[2][0] data[2][1] data[2][2] data[2][3] ...
               data[2][511] data[3][0] data[3][1] data[3][2] data[3][3] ... data[3][511] data[4][0]
               data[4][1] data[4][2] data[4][3] ... data[4][511] ... data[7][0] data[7][1] data[7][2]
               data[7][3] ... data[7][511]
.
.
.
Frame 6396   { data[0][0] data[0][1] data[0][2] data[0][3] ... data[0][511] data[1][0] data[1][1]
               data[1][2] data[1][3] ... data[1][511] data[2][0] data[2][1] data[2][2] data[2][3] ...
               data[2][511] data[3][0] data[3][1] data[3][2] data[3][3] ... data[3][511] data[4][0]
               data[4][1] data[4][2] data[4][3] ... data[4][511] ... data[7][0] data[7][1] data[7][2]
               data[7][3] ... data[7][511]

```

Figure 6.135: Sample contents of the output data file in the TEXT file format where the data are in the float type

```

Frame 0      { data[0][0].real() data[0][0].imag() data[0][1].real() data[0][1].imag() data[0][2].real()
               data[0][2].imag() ... data[0][256].real() data[0][256].imag() data[1][0].real()
               data[1][0].imag() data[1][1].real() data[1][1].imag() data[1][2].real() data[1][2].imag() ...
               data[1][256].real() data[1][256].imag() ... data[7][0].real() data[7][0].imag()
               data[7][1].real() data[7][1].imag() data[7][2].real() data[7][2].imag() ...
               data[7][256].real() data[7][256].imag()
Frame 1      { data[0][0].real() data[0][0].imag() data[0][1].real() data[0][1].imag() data[0][2].real()
               data[0][2].imag() ... data[0][256].real() data[0][256].imag() data[1][0].real()
               data[1][0].imag() data[1][1].real() data[1][1].imag() data[1][2].real() data[1][2].imag() ...
               data[1][256].real() data[1][256].imag() ... data[7][0].real() data[7][0].imag()
               data[7][1].real() data[7][1].imag() data[7][2].real() data[7][2].imag() ...
               data[7][256].real() data[7][256].imag()
.
.
.
Frame 6396   { data[0][0].real() data[0][0].imag() data[0][1].real() data[0][1].imag() data[0][2].real()
               data[0][2].imag() ... data[0][256].real() data[0][256].imag() data[1][0].real()
               data[1][0].imag() data[1][1].real() data[1][1].imag() data[1][2].real() data[1][2].imag() ...
               data[1][256].real() data[1][256].imag() ... data[7][0].real() data[7][0].imag()
               data[7][1].real() data[7][1].imag() data[7][2].real() data[7][2].imag() ...
               data[7][256].real() data[7][256].imag()

```

Figure 6.136: Sample SaveMatrixFrames Output File Content for TEXT Complex Float Type

contains the information of the input data and so convenient for the later use. The available tags are listed in the Table ??.

Table 6.130: Tag list of SaveMatrixFrames

Tag	Description	Unit
datatype	Data Type (float or complex float)	String
rowsize	Row size of Matrix	Integer
colsize	Column size of Matrix	Integer
dim	Data Dimension (Fixed value. 2 for Matrix)	Integer

The examples of the format string and the output are given in the Table ??.

Table 6.131: Examples of the format string and the output for `Matrix<complex<float> >` where the row size is 8 and the column size is 512

Format String (FILENAME Parameter value)	Output (Formatted Filename)
<code>samplefile.dat</code>	<code>samplefile.dat</code>
<code>samplefile_{datatype}.txt</code>	<code>samplefile_complex_float.txt</code>
<code>samplefile_row{rowsize}.raw</code>	<code>samplefile_row8.raw</code>
<code>samplefile_col{colsize}.dat</code>	<code>samplefile_col512.dat</code>
<code>samplefile_dim{dim}.dat</code>	<code>samplefile_dim2.dat</code>
<code>samplefile_{datatype}_row{rowsize}_col{colsize}_dim{dim}.dat</code>	<code>samplefile_complex_float_row8_col512_dim2.dat</code>

It is required to specify the column size, the row size, and the data type in parameters when using `LoadMatrixFrames` to load a file outputted by `SaveMatrixFrames` into the `Matrix<ObjectRef>` type. Hence, having data information in the filename will be useful for the later use.

As indicated in the Table ??, there is no rules on the file name extension. It can be any as user specify. It allows to have no file name extensions as well.

The value for "format" in {tag:format} is optional. The "format" can be used to specify the number of digits, for instance, 03d where the format specifiers are the same as the ones of `printf` in C.

6.7.28 SaveVectorFrames

Outline of the node

Saves data of frames in the `Vector<ObjectRef>` type in a file. Note that the supported `ObjectRef` are the `float` type and the `complex<float>` type only.

Necessary file

No files are required.

Usage

When to use

This node is used to save data stored in the `Vector<ObjectRef>` type for multiple frames in one file.

Typical connection

Figure ?? shows a connection example using `SaveVectorFrames` in a network. In this network, `SaveVectorFrames` takes data outputted in the `SPECTRUM` terminal, the hidden output of `LocalizeMUSIC`, as the input. The data is power of the `MUSIC` spectrum for every direction. For future use or for documentation, `SaveVectorFrames` can be used to save the data in a file. For more information on `LocalizeMUSIC` hidden output `SPECTRUM`, see `LocalizeMUSIC Node Reference`, specifically `LocalizeMUSIC`'s 6.2.14.4 Input-output and property of the node.

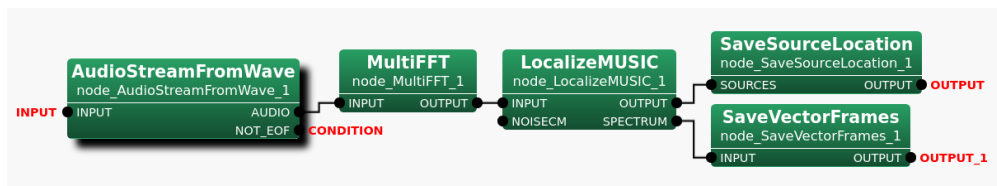


Figure 6.137: Connection example of `SaveVectorFrames`

Input-output and property of node

Input

INPUT : Data of frames in the `Vector<ObjectRef>` type. The supported types are the `Vector<float>` type and the `Vector<complex<float>` > type.

Output

OUTPUT : A file in which data in the `Vector<ObjectRef>` type for multiple frames were saved.

Parameter

FILENAME : `string` type. The file name for the output file.

OUTPUTTYPE : `string` type. The file format options for the output file. Select `TEXT` for text files or `RAW` for binary files. The default value is `TEXT`.

Table 6.132: Parameter list of SaveVectorFrames

Parameter name	Type	Default value	Unit	Description
FILENAME	string			The file name of the output file.
OUTPUTTYPE	string	TEXT		The file format options for the output file. Select TEXT for a text file or RAW for a binary file.

Details of the node

SaveVectorFrames takes data in either the `Vector< float >` type or the `Vector<complex<float> >` type for multiple frames as the input and then saves all data in one file. The file format is to be specified in the OUTPUTTYPE parameter. When the OUTPUTTYPE parameter is set to RAW, data of frames will be written in IEEE 754 32-bit single-precision floating-point number format by little- endian order. The default value of this parameter, TEXT, will output human-readable data unlike RAW.

Table ?? shows the details of how data will be formatted in the file when TEXT is selected in the OUTPUTTYPE parameter. Each data is separated by space and the data of frames are separated per frame in the Vector size by a line feed.

Table 6.133: File Format for TEXT OUTPUTTYPE

Datatype	File Format
Float	data[0] data[1] data[2] ... data[n] data[0] data[1] data[2] data[n]
Complex float	data[0].real() data[0].imag() data[1].real() data[1].imag() ... data[n].real() data[n].imag()

Where n is based on the Vector size of the input data.

SaveVectorFrames checks the size of the Vector when it receives the data. In the sample network above (Figure ??), LocalizeMUSIC outputs `Vector< float >`. When the transfer function specified in the LocalizeMUSIC generated from 72 sound sources, the expected vector size will be 72. A transfer function is a model of sound propagation. The number of sound sources of a transfer function is one of the factors that affects the vector size and it is decided during transfer function creation. For more information on transfer function, please see a video on "HARK Transfer Function Tutorial" at <https://www.hark.jp/document/harktv/>. The detailed documents for generating Transfer Function can be found at <https://www.hark.jp/document/transfer-function-generation-manuals/> as well.

Suppose the input data were stored in the `Vector<float>` type where the column size is 72 and there are 6397 frames in total, the contents of the output data file will be like one shown in Figure ?. It is assumed that the data were saved in the TEXT file format described in the Table ?.

Note that the Frame numbers written on the left side of Figure ? and Figure ? are not written in the files.

Figure ? gives sample contents of the output file when the input data were in the `Vector<complex<float> >` type like ones in the sample network given by Figure ?. The column size is 71 for each frame. There are 6397 frames in total. It is also assumed that the data were saved in the TEXT file format described in the Table ?.

Filename:

An option to use a special pattern, {tag:format}, with the data properties as known as a format string is available when specifying the value in the FILENAME parameter. The format string allows the users to set a filename which contains the information of the input data and so convenient for the later use.

The available tags are listed in the Table ?.

```

Frame 0  { data[0] data[1] data[2] data[3] data[4] data[5] data[6] data[7] data[8] data[9]
          data[10] data[11] data[12] data[13] data[14] data[15] ... data[71]
Frame 1  { data[0] data[1] data[2] data[3] data[4] data[5] data[6] data[7] data[8] data[9]
          data[10] data[11] data[12] data[13] data[14] data[15] ... data[71]
Frame 2  { data[0] data[1] data[2] data[3] data[4] data[5] data[6] data[7] data[8] data[9]
          data[10] data[11] data[12] data[13] data[14] data[15] ... data[71]
          .
          .
          .
          .
          .
Frame 6396 { data[0] data[1] data[2] data[3] data[4] data[5] data[6] data[7] data[8] data[9]
            data[10] data[11] data[12] data[13] data[14] data[15] ... data[71]

```

Figure 6.138: Sample SaveVectorFrames Output File Content for TEXT Float Type

Table 6.134: Tag list of SaveVectorFrames

Tag	Description	Unit
datatype	Data Type (float or complex float)	String
colsize	Column size of Vector	Integer
dim	Data Dimension (Fixed value. 1 for Vector)	Integer

The examples of the format string and the output are given in the Table ??.

It is required to specify the column size and the data type in parameters when using LoadVectorFrames to load a file outputted by SaveVectorFrames into the Vector<ObjectRef> type. Hence, having data information in the filename will be useful for the later use.

```

Frame 0  { data[0].real() data[0].imag() data[1].real() data[1].imag() data[2].real()
          data[2].imag() data[3].real() data[3].imag() data[4].real() data[4].imag()
          data[5].real() data[5].imag() data[6].real() data[6].imag() data[7].real()
          data[7].imag() ... data[71].real() data[71].imag()
Frame 1  { data[0].real() data[0].imag() data[1].real() data[1].imag() data[2].real()
          data[2].imag() data[3].real() data[3].imag() data[4].real() data[4].imag()
          data[5].real() data[5].imag() data[6].real() data[6].imag() data[7].real()
          data[7].imag() ... data[71].real() data[71].imag()
          .
          .
          .
          .
          .
Frame 6396 { data[0].real() data[0].imag() data[1].real() data[1].imag() data[2].real()
            data[2].imag() data[3].real() data[3].imag() data[4].real() data[4].imag()
            data[5].real() data[5].imag() data[6].real() data[6].imag() data[7].real()
            data[7].imag() ... data[71].real() data[71].imag()

```

Figure 6.139: Sample SaveVectorFrames Output File Content for TEXT Complex Float Type

Table 6.135: Examples of the format string and the output for `Vector<complex<float> >` where the Vector size is 20.

Format String (Parameter FILENAME value)	Output (Formatted Filename)
samplefile.dat	samplefile.dat
samplefile_{datatype}.txt	samplefile_complex_float.txt
samplefile_col{colsize}.raw	samplefile_col20.raw
samplefile_dim{dim}.dat	samplefile_dim1.dat
samplefile_{datatype}_col{colsize}_dim{dim}.dat	samplefile_complex_float_col20_dim1.dat

As indicated in the Table ??, there is no rules on the file name extension. It can be any as user specify. It allows to have no file name extensions as well.

The value for "format" in {tag:format} is optional. The "format" can be used to specify the number of digits, for instance, 03d where the format specifiers are the same as the ones of printf in C.

6.7.29 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.

Typical connection

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 .

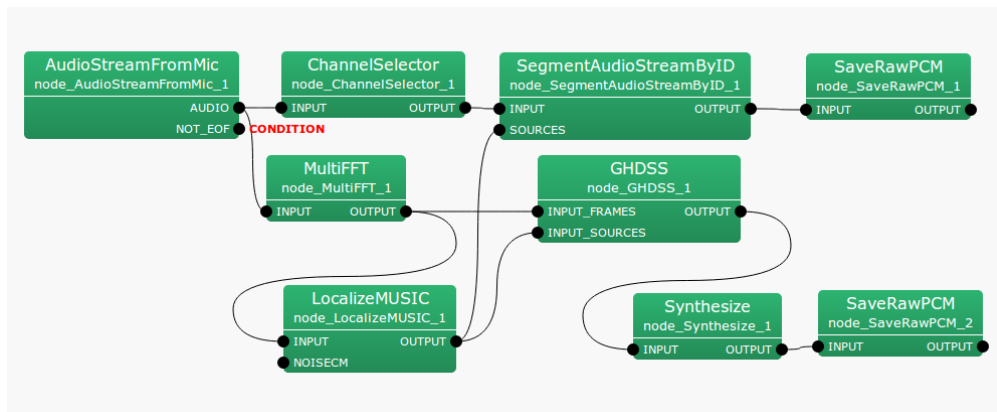


Figure 6.140: Example of a connection of SegmentAudioStreamByID

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.

6.7.30 SourceSelectorByDirection

Outline of the node

This node is a filtering node that filters input source localization results specified angle range in a horizontal and vertical direction (azimuth and elevation).

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 ?? is to a network to extract only those directions specified from the log files of the source localization results.



Figure 6.141: 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

Table 6.136: Parameters of SourceSelectorByDirection

Parameter name	Type	Default value	Unit	Description
MIN_AZIMUTH	float	-20.0	[deg]	Minimum value of a sound source azimuth to be passed.
MAX_AZIMUTH	float	20.0	[deg]	Maximum value of a sound source azimuth to be passed.
MIN_ELEVATION	float	-90.0	[deg]	Minimum value of a sound source elevation to be passed.
MAX_ELEVATION	float	90.0	[deg]	Maximum value of a sound source elevation to be passed.

MIN_AZIMUTH, MAX_AZIMUTH : float type. The angles indicate right and left directions (azimuth angle) of the sound source to be passed.

MIN_ELEVATION, MAX_ELEVATION : float type. The angles indicate vertical directions (elevation 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.

6.7.31 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. 0 ID are given to stationary noise specified by ConstantLocalization node. This node is used to filter the sounds other than this.

Necessary files

No files are required.

Usage

When to use

In case it is known that stationary noise exists, for example, performing sound source separation under conditions in which noise (e.g. the sound of the fan) is known and stationary, only separated sound other than stationary noise can be extracted by specifying the ID of the stationary noise in ConstantLocalization . In the ConstantLocalization node, the sound source ID can be controlled in its property, MIN_ID. So, the sound source ID for the stationary noise can be fixed as 0 by setting MIN_ID to 0 when the number of sound source of stationary noise is 1. The sound source ID of the separated sound corresponding to the stationary noise can be always set to 0 by connecting ConstantLocalization combined with LocalizeMUSIC , SourceTracker , CombineSource to GHDSS as shown in the figure ?? . Only sound sources correspondig to separated sound other than stationary noise can be chosen since sound sources whose ID is 0 will be ignored if setting the MIN_ID to 1 after connecting SourceSelectorByID to GHDSS .

Typical connection

Figure ?? shows a example of a connection, in which the node is connected to the posterior half of GHDSS .

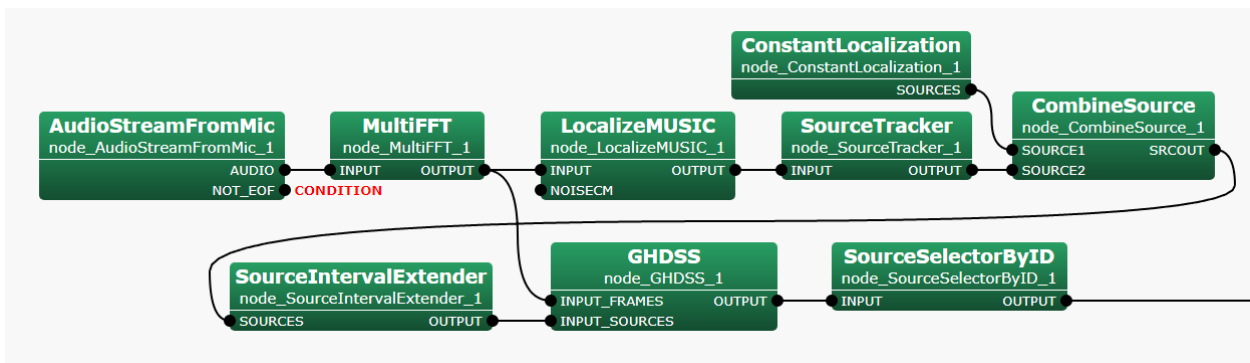


Figure 6.142: Example of connection of SourceSelectorByID

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 6.137: Parameters of `SourceSelectorById`

Parameter name	Type	Default value	Unit	Description
<code>MIN_ID</code>	<code>int</code>	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.

6.7.32 SourceSelectorBySourceInfo

Outline of the node

A filtering node which filters the sound source localization results given as the input to obtain the result that satisfy the conditions specified by parameters based on the information (ID, power, azimuth angle, or elevation angle). Regarding ID or power of a sound source, output the sound source whose ID or power is the minimum, the maximum, or within a certain range specified. Regarding the direction of the sound source, output the sound source whose direction is in the nearest to the specified angle, or angle within a certain range specified.

Necessary file

No files are required.

Usage

When to use

This node is used to obtain the sound source localization results when the information on ID, power, or direction of the sound source is available, for instance, when it is known that there is no sound source expect in front. Alternatively, this node can be used to remove the sound source localization results when information on a sound source to be removed is known.

Typical connection

Figure ?? shows a connection example. Mainly connect sound source localization results obtained with Constant-Localization , LoadSourceLocation , LocalizeMUSIC , etc.

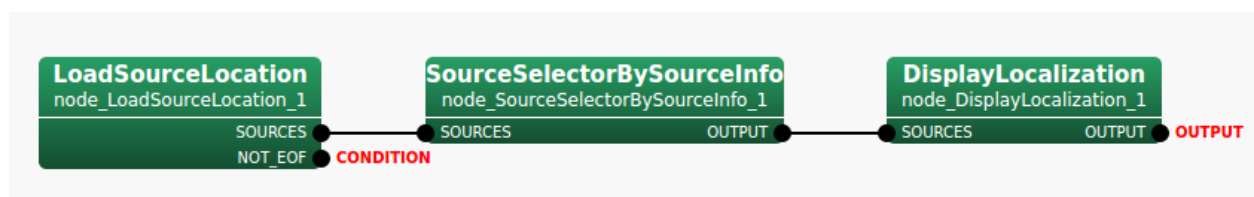


Figure 6.143: Connection example of SourceSelectorBySourceInfo is an Iterator subnetwork

Input-output and property of the node

Input

SOURCES : Vector<ObjectRef> type. The sound source localization results. The ObjectRef refers to data in the Source type indicating the sound source information with an ID.

Output

OUTPUT : Vector<ObjectRef> type. The sound source localization results after filtering. The ObjectRef refers to data in the Source type indicating the sound source information with an ID.

Parameter

Table 6.138: Parameter list of SourceSelectorBySourceInfo

Parameter list	Type	Default value	Unit	Description
SELECTION_TYPE	string	ID		The filter condition for outputting data. Select ALL, ID, POWER, DIRECTION_AZIMUTH, or DIRECTION_ELEVATION.
ID_SELECTION_TYPE	string	LATEST		The type of the sound source ID. Select LATEST 罫修 OLDEST, or BETWEEN.
ID_RANGE_MIN	int	0		The lower limit of the range for the sound source ID.
ID_RANGE_MAX	int	0		The upper limit of the range for the sound source ID.
POWER_SELECTION_TYPE	string	HIGHEST		The type of the sound source power. Select HIGHEST, LOWEST, or BETWEEN.
POWER_RANGE_MIN	float	0		The lower limit of the range for the sound source power.
POWER_RANGE_MAX	float	40.0		The upper limit of the range for the sound source power.
AZIMUTH_SELECTION_TYPE	string	BETWEEN		The type of the azimuth of a sound source. Select NEAREST or BETWEEN.
AZIMUTH	float	0	[deg]	The azimuth angle.
AZIMUTH_RANGE_MIN	float	0	[deg]	The lower limit of the range for the azimuth angle of a sound source.
AZIMUTH_RANGE_MAX	float	360.0	[deg]	The upper limit of the range for the azimuth angle of a sound source.
ELEVATION_SELECTION_TYPE	string	BETWEEN		The type of the elevation of a sound source. Select NEAREST or BETWEEN.
ELEVATION	float	0	[deg]	The elevation angle.
ELEVATION_RANGE_MIN	float	0	[deg]	The lower limit of the range for the elevation angle of a sound source.
ELEVATION_RANGE_MAX	float	90.0	[deg]	The upper limit of the range for the elevation angle of a sound source.

SELECTION_TYPE : string type. The filter condition for outputting data. Select ALL, ID, POWER, DIRECTION_AZIMUTH, or DIRECTION_ELEVATION. Selecting ALL outputs all results unconditionally. When selecting ID, POWER, or DIRECTION_AZIMUTH/DIRECTION_ELEVATION, output results that satisfy the conditions specified by the subsequent parameters for the sound source ID, the sound source power, or the sound source direction, respectively.

ID_SELECTION_TYPE : string type. The type of the sound source ID available when ID is selected in the SELECTION_TYPE parameter. Select LATEST, OLDEST, or BETWEEN. Selecting LATEST or OLDEST outputs the results of the sound source with the latest ID or the oldest ID. Selecting BETWEEN outputs the results of which ID are in between the values specified in ID_RANGE_MIN and ID_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of the multiple sound sources.

ID_RANGE_MIN : int type. The lower limit of the range for the sound source ID available when BETWEEN is selected in the ID_SELECTION_TYPE parameter.

ID_RANGE_MAX : int type. The upper limit of the range for the sound source ID available when BETWEEN is selected in the ID_SELECTION_TYPE parameter.

POWER_SELECTION_TYPE : string type. The type of the sound source power available when POWER is selected in the SELECTION_TYPE parameter. Select HIGHEST, LOWEST, or BETWEEN. Selecting HIGHEST or LOWEST outputs the results of the sound source with the highest power or the lowest power. Selecting

BETWEEN outputs the results whose power is in between the values specified in POWER_RANGE_MIN and POWER_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources.

POWER_RANGE_MIN : float type. The lower limit of the range for the sound source power available when BETWEEN is selected in the POWER_SELECTION_TYPE parameter.

POWER_RANGE_MAX : float type. The upper limit of the range for the sound source power available when BETWEEN is selected in the POWER_SELECTION_TYPE parameter.

AZIMUTH_SELECTION_TYPE : string type. The type of the azimuth of a sound source available when DIRECTION_AZIMUTH is selected in the SELECTION_TYPE parameter. Select NEAREST or BETWEEN. Selecting NEAREST outputs the result whose azimuth is located the nearest to the azimuth specified in the AZIMUTH parameter. Selecting BETWEEN outputs the results whose azimuth is in between the specific values specified in AZIMUTH_RANGE_MIN and AZIMUTH_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources.

AZIMUTH : float type. The azimuth angle available when NEAREST is selected in the AZIMUTH_SELECTION_TYPE parameter.

AZIMUTH_RANGE_MIN : float type. The lower limit of the range for the azimuth angle of a sound source available when BETWEEN is selected in the AZIMUTH_SELECTION_TYPE parameter.

AZIMUTH_RANGE_MAX : float type. The upper limit of the range for the azimuth angle of a sound source available when BETWEEN is selected in the AZIMUTH_SELECTION_TYPE parameter.

ELEVATION_SELECTION_TYPE : string type. The type of the elevation of a sound source available when DIRECTION_ELEVATION is selected in the SELECTION_TYPE parameter. Select NEAREST or BETWEEN. Selecting NEAREST outputs the result whose elevation is located in the nearest to the elevation specified in the ELEVATION parameter. Selecting BETWEEN outputs the results whose elevation is in between the values specified in ELEVATION_RANGE_MIN and ELEVATION_RANGE_MAX parameters. When multiple sound sources satisfy the condition, output the results of multiple sound sources.

ELEVATION : float type. The elevation angle available when NEAREST is selected in the ELEVATION_SELECTION_TYPE.

ELEVATION_RANGE_MIN : float type. The lower limit of the range for the elevation angle of a sound source available when BETWEEN is selected in the ELEVATION_SELECTION_TYPE.

ELEVATION_RANGE_MAX : float type. The upper limit of the range for the elevation angle of a sound source available when BETWEEN is selected in the ELEVATION_SELECTION_TYPE.

6.7.33 SourceTransformer

Outline of the node

Edit values for information (ID, power, azimuth, elevation) of the sound source localization results given as the input. Replace each value with a constant value specified or performs a basic arithmetic operation using the specified value on each value.

Necessary file

No files are required.

Usage

When to use

This node is used to edit values of ID, power, azimuth, or elevation which is information of the sound source localization results. Replace each value with a constant value specified or performs a basic arithmetic operation using the specified value on each value.

Typical connection

Figure ?? shows a connection example. Mainly connect sound source localization results obtained with Constant-Localization , LoadSourceLocation , LocalizeMUSIC , etc.

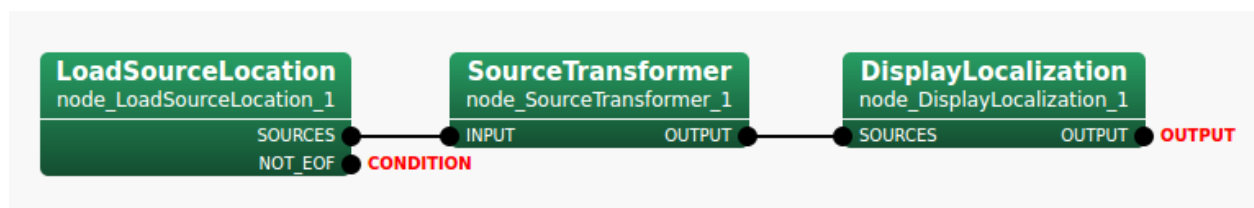


Figure 6.144: Example of connection of SourceTransformer in an Iterator subnetwork

Input-output and property of the node

Input

INPUT : Vector<ObjectRef> type. The sound source localization results. The ObjectRef refers to data in the Source type indicating the sound source information to which an ID is assigned.

Output

OUTPUT : Vector<ObjectRef> type. The sound source localization results. The ObjectRef refers to data in the Source type indicating the sound source information to which an ID is assigned.

Parameter

Table 6.139: Parameter list of SourceTransformer

Parameter name	Type	Default value	Unit	Description
ENABLE_ID	bool	false		Enable or disable to edit sound source ID.
ID_OPERATION_TYPE	string	CONST		The operation type (CONST, ADD, SUB, MUL, DIV) to edit sound source ID.
ID	Object	Vector<int>		The value used to edit the sound source ID.
ENABLE_POWER	bool	false		Enable or disable to edit sound source power.
POWER_OPERATION_TYPE	string	CONST		The operation type (CONST, ADD, SUB, MUL, DIV) to edit sound source power.
POWER	Object	Vector<float>		The value used to edit sound source power.
ENABLE_AZIMUTH	bool	false		Enable or disable to edit azimuth of sound source localization results.
AZIMUTH_OPERATION_TYPE	string	CONST		The operation type (CONST, ADD, SUB, MUL, DIV) to edit azimuth of sound source localization results.
AZIMUTH	Object	Vector<float>		The value used to edit azimuth of sound source localization results.
ENABLE_ELEVATION	bool	false		Enable or disable to edit elevation of sound source localization result.
ELEVATION_OPERATION_TYPE	string	CONST		The operation type (CONST, ADD, SUB, MUL, DIV) to edit elevation of sound source localization results.
ELEVATION	Object	Vector<float>		The value used to edit elevation of sound source localization result.

ENABLE_ID : bool type. Setting the value to true enables editing sound source ID of the localization results given as the input. The default value is false.

ID_OPERATION_TYPE : string type. The operation type to edit sound source ID of the localization results given as the input available when the ENABLE_ID parameter is true. The operation uses the value specified in the ID parameter. Selecting CONST replaces sound source ID with the ID parameter value. Selecting ADD, SUB, MUL, or DIV performs an arithmetic operation on the sound source ID such that add the ID parameter value to the sound source ID, subtract the ID parameter value from the sound source ID, multiply the sound source ID by the ID parameter value, or divide the sound source ID by the ID parameter value.

ID : Object type. The value used to edit sound source ID. When a single value is given, use the value to all sound source localization results. Use Vector<int> when it is desired to specify different value per sound source localization result. However, the last element of the Vector<int> is used for the rest of sound source localization results when the number of Vector<int> elements is less than the number of sound source localization results.

ENABLE_POWER : bool type. Setting the value to true enables editing sound source power of the localization results given as the input. The default value is false.

POWER_OPERATION_TYPE : string type. The operation type to edit sound source power of the localization results given as the input available when the ENABLE_POWER is set to true. The operation uses the value specified in the POWER parameter. Selecting CONST replaces sound source power with the POWER parameter value. Selecting ADD, SUB, MUL, or DIV performs an arithmetic operation on sound source power such that add the POWER parameter value to the sound source power, subtract the POWER parameter value from the

sound source power, multiply the sound source power by the **POWER** parameter value, or divide the sound source power by the **POWER** parameter value.

POWER : **Object** type. The value used to edit sound source power. When a single value is given, use the value to all sound source localization results. Use **Vector<float>** when it is desired to specify different value per sound source localization result. However, the last element of the **Vector<float>** is used for the rest of sound source localization results when the number of **Vector<float>** elements is less than the number of sound source localization results.

ENABLE_AZIMUTH : **bool** type. Setting the value to **true** enables editing azimuth of the sound source localization results given as the input. The default value is **false**.

AZIMUTH_OPERATION_TYPE : **string** type. The operation type to edit azimuth of the localization results given as the input available when the **ENABLE_AZIMUTH** is set to **true**. The operation uses the value specified in the **AZIMUTH** parameter. Selecting **CONST** replaces the azimuth of the sound source localization results with the **AZIMUTH** parameter value. Selecting **ADD**, **SUB**, **MUL**, or **DIV** performs an arithmetic operation on the azimuth of the sound source localization results such that add the **AZIMUTH** parameter value to the azimuth of the sound source localization results, subtract the **AZIMUTH** parameter value from the azimuth of the sound source localization results, multiply the azimuth of the sound source localization results by the **AZIMUTH** parameter value, or divide the azimuth of the sound source localization results by the **AZIMUTH** parameter value.

AZIMUTH : **Object** type. The value used to edit azimuth. When a single value is given, use the value to all sound source localization results. Use **Vector<float>** when it is desired to specify different value per sound source localization result. However, the last element of the **Vector<float>** is used for the rest of sound source localization results when the number of **Vector<float>** elements is less than the number of sound source localization results.

ENABLE_ELEVATION : **bool** type. Setting the value to **true** enables editing the elevation of sound source localization results given as the input. The default value is **false**.

ELEVATION_OPERATION_TYPE : **string** type. The operation type to edit elevation of the localization results given as the input available when the **ENABLE_ELEVATION** is set to **true**. The operation uses the value specified in the **ELEVATION** parameter. Selecting **CONST** replaces the elevation of the sound source localization results with the **ELEVATION** parameter value. Selecting **ADD**, **SUB**, **MUL**, or **DIV** performs an arithmetic operation on the elevation of the sound source localization results such that add the **ELEVATION** parameter value to the elevation of the sound source localization results, subtract the **ELEVATION** parameter value from the elevation of the sound source localization results, multiply the elevation of the sound source localization results by the **ELEVATION** parameter value, or divide the elevation of the sound source localization results by the **ELEVATION** parameter value.

ELEVATION : **Object** type. The value used to edit elevation. When a single value is given, use the value to all sound source localization results. Use **Vector<float>** when it is desired to specify different value per sound source localization result. However, the last element of the **Vector<float>** is used for the rest of sound source localization results when the number of **Vector<float>** elements is less than the number of sound source localization results.

6.7.34 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

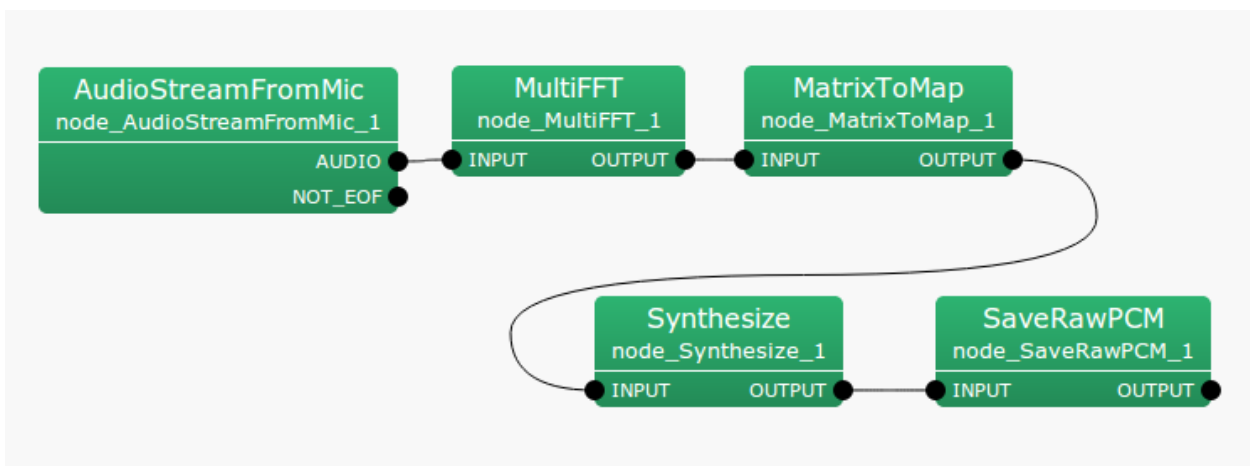


Figure 6.145: Example of a connection of Synthesize

Input-output and properties of the node

Table 6.140: 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:

$$delay = \begin{cases} \lfloor LENGTH/ADVANCE \rfloor - 1, & \text{if } LENGTH \bmod ADVANCE \neq 0, \\ LENGTH/ADVANCE, & \text{otherwise.} \end{cases} \quad (6.176)$$

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>

6.7.35 TextConcatenate

Outline of the node

Concatenate strings of String types.

Necessary files

No files are required.

Usage

When to use

Concatenate inputs of multiple string type texts and outputs them as a single string type text. In other words, you can treat JSON text generated by TextConverter node, a string generated by Constant node when set to parameter type as string, or other nodes that output string type as an input. Output is connected to a node corresponding to input of string type such as HarkDataStreamSender or HARK-Python's PyCodeExecuter.

Typical connection

Figure ?? shows connection examples using the Constant node and TextConverter node.

The sample network in Figure ?? combines character strings generated by Constant node and TextConverter node, and sends multiple sound source information with HarkDataStreamSender.

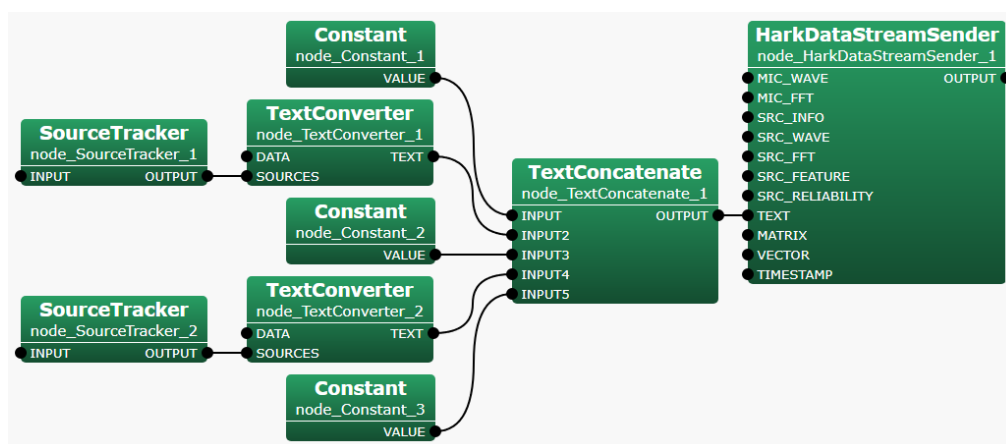


Figure 6.146: Connection example of TextConcatenate to Constant and/or TextConverter

Input-output and property of the node

Input

INPUT : string type. Any number of input terminals can be added.

Output

OUTPUT : string type.

Parameter

Table 6.141: Parameter list of TextConcatenate

Parameter name	Type	Default value	Unit	Description
SEPARATOR	string			Specifies the text separator string. The specified character string is inserted between each INPUT.
ENABLE_DEBUG	bool	false		Select whether to output the concatenated character string to standard output.

SEPARATOR : string type. Specifies the text separator string. The specified character string is inserted between each INPUT. In the default value is blank, there is no separator, and if you enter a white space character, a white space character is used as separator.

ENABLE_DEBUG : bool type. The default value is false. Setting the value to true outputs the concatenate string to the standard output.

Details of the node

<example>

```

INPUT1:  "["
INPUT2:  "{ 'SOURCE': { '1': { 'x': [0.9578, 0, 0.2874], 'power': 29.8}}}"
INPUT3:  ", "
INPUT4:  "{ 'SOURCE': { '1': { 'x': [0.9433, 0.1663, 0.2874], 'power': 28.9}}}"
INPUT5:  ", "
INPUT6:  "{ 'SOURCE': { '2': { 'x': [0.9001, 0.3276, 0.2874], 'power': 27.6}}}"
INPUT7:  ", "
INPUT8:  "{ 'SOURCE': { '4': { 'x': [0.8295, 0.4789, 0.2874], 'power': 25.1}}}"
INPUT9:  ", "
INPUT10: "{ 'SOURCE': { '3': { 'x': [0.7337, 0.6157, 0.2874], 'power': 22.3}}}"
INPUT11: "]"
        ↓
        "[{'SOURCE': {'1': {'x': [0.9578, 0, 0.2874], 'power': 29.8}}}
        , {'SOURCE': {'1': {'x': [0.9433, 0.1663, 0.2874], 'power': 28.9}}}
        , {'SOURCE': {'2': {'x': [0.9001, 0.3276, 0.2874], 'power': 27.6}}}
        , {'SOURCE': {'4': {'x': [0.8295, 0.4789, 0.2874], 'power': 25.1}}}
        , {'SOURCE': {'3': {'x': [0.7337, 0.6157, 0.2874], 'power': 22.3}}}]"
```

Note: Line breaks and double quotes are not included in the actual string.

6.7.36 TextConverter

Outline of the node

Converts any HARK supported data type to JSON text of the `string` type.

Necessary files

No files are required.

Usage

When to use

This node is used to connect to a node (eg `HarkDataStreamSender` or HARK-Python's `PyCodeExecuter`) that can accept input of `string` type. This node can convert any type supported by HARK as JSON text so, it can be handled easily when using it with JSON text understandable languages (eg Python, JavaScript etc). For example, in HARK-Python's `PyCodeExecuter` node, you can easily convert to Python object by using `'import json'` and `'json.load()'` in your script which processes data received with `string` type. Note that although the output of many nodes existing in HARK can be used as an input for this node, exception cases exists. This will be improved with future updates.

Typical connection

Figure ?? and ?? shows connection examples for the `TextConverter` .

The sample network in Figure ?? is a network that transmits the sound source localization result and separated sound using the `HarkDataStreamSender` node. `node_HarkDataStreamSender_2` is a connection example in the case where the information being transmitted on `node_HarkDataStreamSender_1` is transmitted after converting it to JSON text with the `TextConverter` node.

The sample network in Figure ?? is a network used to send the sound source localization result and separated sound to the HARK-Python's `PyCodeExecuter` node.

Input-output and property of the node

Input

DATA : any type. For the currently supported types, see the table ??. Any number of input terminals can be added.

The name of the input terminal is used as the name of the output JSON object (key in map, hash, associative array etc).

Output

TEXT : `string` type.

Parameter

Table 6.142: Parameter list of `TextConverter`

Parameter name	Type	Default value	Unit	Description
<code>ENABLE_DEBUG</code>	<code>bool</code>	<code>false</code>		Select whether to output the converted JSON text to standard output.

ENABLE_DEBUG : `bool` type. The default value is `false`. Setting the value to `true` outputs the converted JSON text to the standard output.

Details of the node

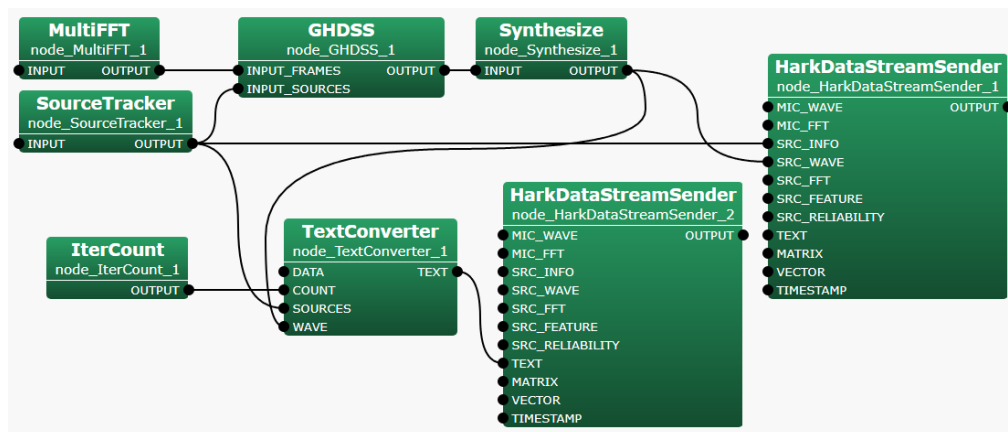


Figure 6.147: Connection example of TextConverter to HarkDataStreamSender

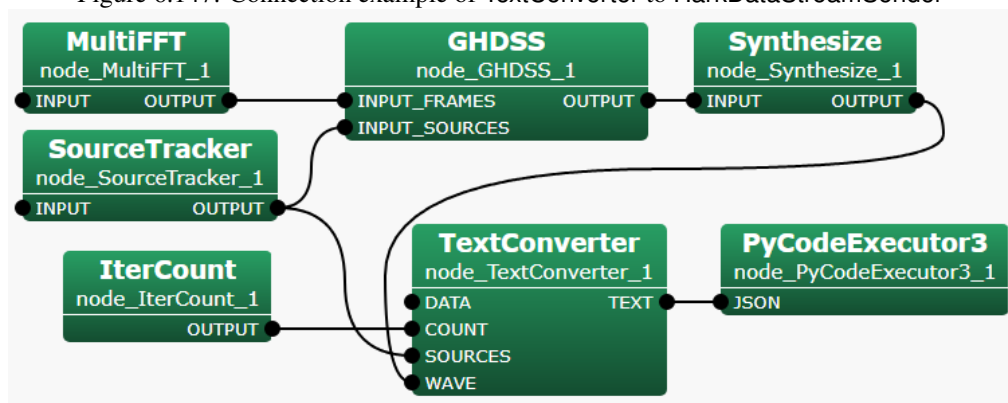


Figure 6.148: Connection example of TextConverter to HARK-Python's PyCodeExecutor

Table 6.143: Conversion table of TextConverter

INPUT	OUTPUT
input object type	output object type
input example	output result
bool	string
<i>true</i>	{'DATA': true}
TrueObject	string
–	{'DATA': true}
FalseObject	string
–	{'DATA': false}
int	string
1	{'DATA': 1}
Int	string
1	{'DATA': 1}
float	string
1.1	{'DATA': 1.1}
Float	string
1.1	{'DATA': 1.1}
Complex	string
$1.1 - 2.2i$	{'DATA': (1.1, -2.2)}
Vector<int>	string
$\langle 1 \ -2 \ 3 \ -4 \ 5 \ -6 \rangle$	{'DATA': [1, -2, 3, -4, 5, -6]}
Vector<float>	string
$\langle 1.1 \ -2.2 \ 3.3 \ -4.4 \ 5.5 \ -6.6 \rangle$	{'DATA': [1.1, -2.2, 3.3, -4.4, 5.5, -6.6]}
Vector<complex<float> >	string
$\langle 1.1 - 2.2i \ 3.3 + 4.4i \ -5.5 - 6.6i \rangle$	{'DATA': [(1.1, -2.2), (3.3, 4.4), (-5.5, -6.6)]}
Matrix<int>	string
$\begin{bmatrix} 1 & -2 \\ -3 & 4 \\ 5 & 6 \end{bmatrix}$	{'DATA': [[1, -2], [-3, 4], [5, 6]]}
Matrix<float>	string
$\begin{bmatrix} 1.1 & -2.2 \\ -3.3 & 4.4 \\ 5.5 & 6.6 \end{bmatrix}$	{'DATA': [[1.1, -2.2], [-3.3, 4.4], [5.5, 6.6]]}
Matrix<complex<float> >	string
$\begin{bmatrix} 1.1 - 2.2i & 3.3 + 4.4i \\ -5.5 + 6.6i & -7.7 - 8.8i \end{bmatrix}$	{'DATA': [[(1.1, -2.2), (3.3, 4.4)], [(-5.5, 6.6), (-7.7, -8.8)]]}
Map<int, Vector<int> >	string
$\{0, \langle 1 \ -2 \ 3 \ -4 \ 5 \ -6 \rangle\}$	{'DATA': {0: [1, -2, 3, -4, 5, -6]}}
Map<int, Vector<float> >	string
$\{0, \langle 1.1 \ -2.2 \ 3.3 \ -4.4 \ 5.5 \ -6.6 \rangle\}$	{'DATA': {0: [1.1, -2.2, 3.3, -4.4, 5.5, -6.6]}}
Map<int, Vector<complex<float> > >	string
$\{0, \langle 1.1 - 2.2i \ 3.3 + 4.4i \ 5.5 - 6.6i \rangle\}$	{'DATA': {0: [(1.1, -2.2), (3.3, 4.4), (5.5, -6.6)]}}
Map<int, Matrix<int> >	string
$\{0, \begin{bmatrix} 1 & -2 \\ -3 & 4 \\ 5 & 6 \end{bmatrix}\}$	{'DATA': {0: [[1, -2], [-3, 4], [5, 6]]}}
Map<int, Matrix<float> >	string
$\{0, \begin{bmatrix} 1.1 & -2.2 \\ -3.3 & 4.4 \\ 5.5 & 6.6 \end{bmatrix}\}$	{'DATA': {0: [[1.1, -2.2], [-3.3, 4.4], [5.5, 6.6]]}}
Map<int, Matrix<complex<float> > >	string
$\{0, \begin{bmatrix} 1.1 - 2.2i & 3.3 + 4.4i \\ -5.5 + 6.6i & -7.7 - 8.8i \end{bmatrix}\}$	{'DATA': {0: [[(1.1, -2.2), (3.3, 4.4)], [(-5.5, 6.6), (-7.7, -8.8)]]}}

6.7.37 VADZC

Outline of the node

This node performs Voice Activity Detection (VAD) on multichannel speech waveform data using the ZeroCross method.

It can be used, for example, to detect speech segments independently and simultaneously on each microphone channel of an audio device with multiple handheld microphones.

It can also be used to extract a single microphone channel from a microphone array and compare its speech recognition performance with that of the isolated sound.

Necessary files

No files are required.

Usage

When to use

This node can know the Voice Activity of each channel of the multichannel audio waveform data to some extent. ZeroCross method is used as the detection for the voice activity. The voice activity detection result output from this node (provided as the same Source type as the localization result) is used for feature extraction in the latter stage.

Typical connection

Figure ?? shows an example of connection of VADZC ,

The input is multi-channel acoustic signal data of type `Matrix<float>` , output from `AudioStreamFromWave` , `AudioStreamFromMic` , etc. The output is the detected acoustic signal data of type `Map<int, ObjectRef>` (`ObjectRef` is of type `Vector<float>`) and the result of the audio segment detection of type `Vector<ObjectRef>` (`ObjectRef` is of type `Source`). Acoustic signal data is an input to `MultiFFT` or `SaveWavePCM`, sound source information is an input to `Client` or `SaveWavePCM` etc.

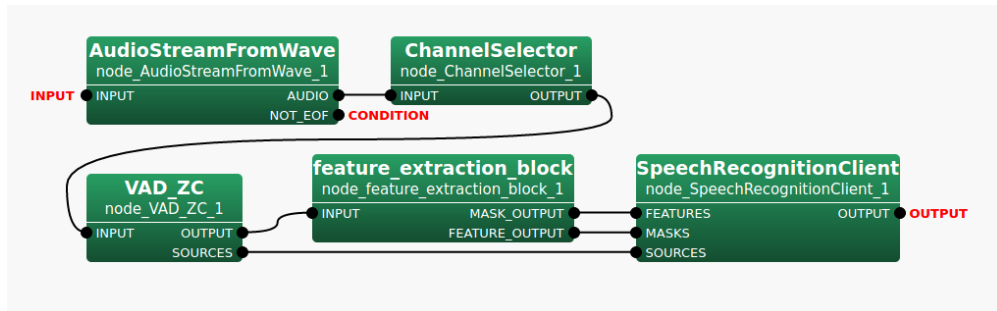


Figure 6.149: Example of a connection of VADZC

Input-output and property of the node

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 : Map<int, ObjectRef> type. The PCM data detected in the audio segment. The Object part will be of type Vector<float> .

SOURCES : Vector<ObjectRef> type. The ID (channel number and audio index) that is uniquely assigned to the audio segment detection result. ObjectRef is Source and $numberofmicrophones \times voiceindex + microphoneindex$ is stored in the ID. The rest of the values are dummies.

Parameter

Table 6.144: Parameter list of VADZC

Parameter name	Type	Default value	Unit	Description
ADVANCE	int	160	[pt]	Shift length of frame.
LENGTH	int	512	[pt]	Number of samples of the frame.
CHANNEL_COUNT	int	0	[ch]	Number of input channels.
SAMPLING_RATE	int	16000	[Hz]	Sampling frequency.
STRIP_ZERO	bool	true	[pt]	Enable / disable of Amplitude 0 frame removal function.
ZERO_MEAN	bool	false		Enable / disable Direct Current (DC) component removal function.
LEVEL_THRESHOLD	float	2000		Threshold of soundless section cut(amplitude level)
ZEROCROSS_THRESHOLD	int	60		Threshold of soundless section cut(Zero crossing number)
HEAD_MARGIN	float	300	[ms]	Margin at the beginning of the audio section.
TAIL_MARGIN	float	400	[ms]	Margin at the end of the audio section.

ADVANCE : int type. Default value is 160. Specify the shift length of the frame.

LENGTH : int type. Default value is 512. Specify the number of samples of frame. It must be equal to the number of columns of Matrix<float> input to the INPUT pin. If it does not match, an exception is raised.

CHANNEL_COUNT : int type. Default value is 0. Specify the number of channels. It must be equal to the number of rows of Matrix<float> input to the INPUT pin. If it does not match, an exception is raised.

SAMPLING_RATE : int type. Default value is 16000. Specify the sampling rate.

STRIP_ZERO : bool type. Default value is true. Specify Enables(true)/disables(false) the function to exclude frames with a continuous amplitude of 0 in the audio waveform. Currently not used.

ZERO_MEAN : bool type. Default value is false. Enables(true) or disables(false) the function to remove the direct current (DC) component from the audio waveform. If the function is enabled by true, the DC component is calculated from the samples in the first frame and the DC offset of each frame is removed. It is processed before the speech segment detection (on the input buffer).

LEVEL_THRESHOLD : float type. Default value is 2000. Specify the silence cut threshold according to the amplitude level. The default value of 2000 is set as a guideline when a 16 bit signed integer of 0 - 32767 is input. To take into account the scaling of floating point numbers by MultiGain nodes etc., it is possible to set them as float .

ZEROCROSS_THRESHOLD : int type. Default value is 60. Specify the silence cut threshold in zero-crossings per second.

HEAD_MARGIN : int type. Default value is 300. Specify the margin at the beginning of the audio section in milliseconds.

TAIL_MARGIN : int type. Default value is 400. Specify the margin at the end of the audio section in milliseconds.

Details of the node

About the condition to detect the voice section:

A frame in which both the amplitude level and the number of zero crossings of the acoustic signal exceed a threshold specified by a parameter is defined as a voice section start. In addition, as shown in Figure ??, margins can be provided at the beginning and end of the section.

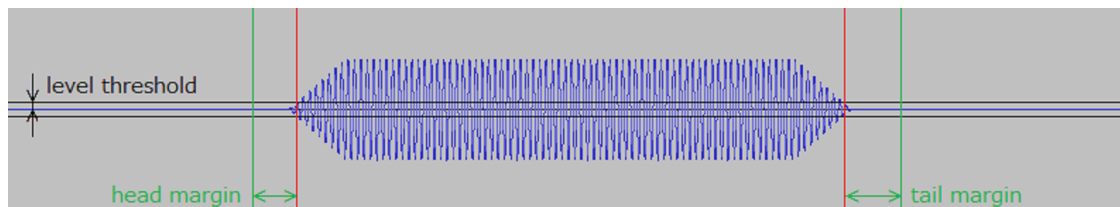


Figure 6.150: Voice section detection

As an example, when `LEVEL_THRESHOLD` is 2000 and `ZEROCROSS_THRESHOLD` is 60, the audio interval is not detected under the following two conditions.

Figure ?? shows an example in which a 440 [Hz] sine wave with an amplitude level of about ± 1600 fades in and out. It indicates that the number of zero crossings is sufficient, but it is not detected because the amplitude level is less than `LEVEL_THRESHOLD`.

Figure ?? is an example of a 22 [Hz] sine wave with an amplitude level of about ± 26000 fading in and out. It shows that the amplitude level is sufficient, but it is not detected because the number of zero crossings is less than `ZEROCROSS_THRESHOLD`. (When the Sin wave is less than 30 [Hz], the number of zero crossings is less than 60).

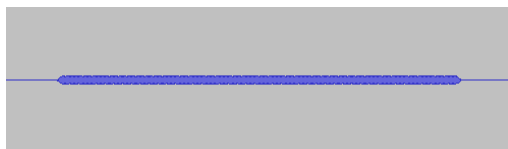


Figure 6.151: Example of undetected due to insufficient amplitude level

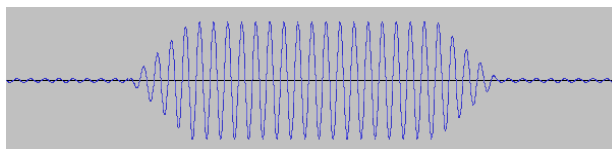


Figure 6.152: Example of undetected due to insufficient number of zero crossings

About the Remove Function for Amplitude 0 Frames:

The `STRIP_ZERO` parameter for the Amplitude 0 Frame Removal function exists, but is not used and the setting value is ignored. The `STRIP_ZERO` parameter may be removed in a future version.

About Direct Current (DC) component removal function:

Figure ?? shows an example of removing a direct current (DC) component when the input acoustic signal contains a direct current (DC) component.

The upper part of the figure shows the input acoustic signal. The acoustic signal is a fade-in and fade-out of a 440 [Hz] Sin wave with an amplitude level of about ± 16000 , but it is an example in which a direct current (DC) component is constantly included around + 6500.

The middle part of the figure shows the direct current (DC) component to be removed, and the direct current (DC) component of about + 6500 calculated using the first frame after the input of the acoustic signal is started is continuously removed in the subsequent frames.

The lower part of the figure shows the acoustic signal of the object for which voice section detection is actually performed, and it is shown that the zero crossing number can be correctly obtained by removing the DC component.

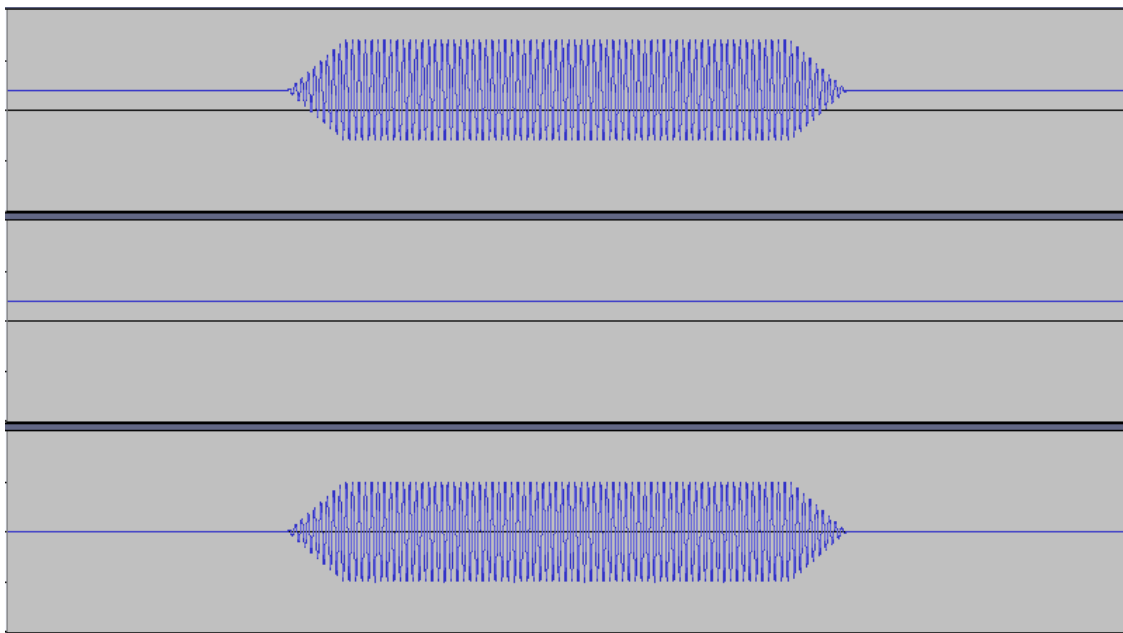


Figure 6.153: Removal of the DC component (top: input acoustic signal, middle: DC component to be removed, bottom: acoustic signal to be VAD)

6.7.38 VectorToMap

Outline of the node

Convert the `Vector<ObjectRef>` type to the `Map<int, ObjectRef>` type when the `ObjectRef` of the `Vector<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the `VecObj` type to the `Map<int, ObjectRef>` type when the `ObjectRef` of the `Vector<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type. Taking a `Vector<float>` as the input will output a `Map<int, Vector<float>>`. Taking a `Vector<complex<float>>` as the input will output a `Map<int, Vector<complex<float>>>`.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are the `Vector<float>` type and the `Vector<complex<float>>` type.

Output

OUTPUT : `Map<int, Vector<float>>`, or `Map<int, Vector<complex<float>>>` of the `Map<ObjectRef>` type.

Parameter

Table 6.145: Parameter list of VectorToMap

Parameter name	Type	Default value	Unit	Description
OUTPUT_TYPE	string	map_of_vector		The data type of the output data. Select map_of_vector or map_of_vectors.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

OUTPUT_TYPE : string type. The data type of the output data. Select `map_of_vector` or `map_of_vectors`. Selecting `map_of_vector` outputs a `Map` whose key is 0 and whose `ObjectRef` is the `Vector` given as the input. Selecting `map_of_vectors` outputs `Map` as many as the number of elements of the `Vector` given as the input. The each `Map` 鐸€s key is the index of the `Vector` and the each `ObjectRef` is a `Vector` consisting of an element corresponding to the index. The default value is `map_of_vector`.

DEBUG : bool type. Setting the value to `true` outputs the conversion status to the standard output. The default value is `false`.

Table 6.146: Conversion table of VectorToMap

INPUT		OUT.TYPE	OUTPUT	
type	size		type	size
Vector<float>	N	map_of_vector	Map<int , Vector<float> >	{N}x1 (1)
		map_of_vectors		{1}xN (2)
Vector<complex<float> >		map_of_vector	Map<int , Vector<complex<float> > >	{N}x1
		map_of_vectors		{1}xN

Details of the node

<example>

INPUT:

< 1 2 3 4 5 >

OUTPUT(1):

{ 0, < 1 2 3 4 5 > }

OUTPUT(2):

{ 0, < 1 > }, { 1, < 2 > }, { 2, < 3 > }, { 3, < 4 > }, { 4, < 5 > }

6.7.39 VectorToMatrix

Outline of the node

Convert the `Vector<ObjectRef>` type to the `Matrix<ObjectRef>` type when the `ObjectRef` of the `Vector<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the `Vector<ObjectRef>` type to the `Matrix<ObjectRef>` type when the `ObjectRef` of the `Vector<ObjectRef>` given as the input is either the `float` type or the `complex<float>` type. Taking a `Vector<float>` as the input will output a `Matrix<float>`. Taking a `Vector<complex<float>>` as the input will output a `Matrix<complex<float>>`.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are the `Vector<float>` type and the `Vector<complex<float>>` type.

Output

OUTPUT : any type. Note that the supported data types are the `Matrix<float>` type and the `Matrix<complex<float>>` type.

Parameter

Table 6.147: Parameter list of VectorToMatrix

Parameter name	Type	Default value	Unit	Description
OUTPUT_TYPE	string	row_vector		The type of the Matrix to output. Select row_vector or column_vector.
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

OUTPUT_TYPE : string type. The type of the `Matrix` to output. Select `row_vector` or `column_vector`. Selecting `row_vector` will output a matrix with one row and `N` columns where `N` is the size of the `Vector` given as the input. Selecting `column_vector` will output a matrix with `N` rows and one column where `N` is the size of the `Vector` given as the input. The default value is `row_vector`.

DEBUG : bool type. Setting the value to `true` outputs the conversion status to the standard output. The default value is `false`.

Table 6.148: Conversion table of VectorToMatrix

INPUT		OUTPUT_TYPE	OUTPUT		
type	size		type	size	
Vector<float>	N	row_vector	Matrix<float>	1 x N	(1)
		column_vector		N x 1	(2)
Vector<complex<float> >		row_vector	Matrix<complex<float> >	1 x N	
		column_vector		N x 1	

Details of the node

<example>

INPUT: $\langle 1 \ 2 \ 3 \ 4 \ 5 \rangle$

OUTPUT(1): $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$

OUTPUT(2): $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$

6.7.40 VectorToVector

Outline of the node

Convert the data type of the ObjectRef of the Vector<ObjectRef> given as the input, between the Vector<float> type and the Vector<complex<float> > type.

Necessary file

No files are required.

Usage

When to use

This node is used to convert the data type of the ObjectRef of the Vector<ObjectRef> given as the input; the Vector<float> type to the Vector<complex<float> > type, Vector<complex<float> > to Vector<float> .

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are the Vector<float> type and the Vector<complex<float> > type.

Output

OUTPUT : any type. Note that the supported data types are the Vector<float> type and the Vector<complex<float> > type.

Parameter

Table 6.149: Parameter list of VectorToVector

Parameter name	Type	Default value	Unit	Description
METHOD_COMPLEX_TO_FLOAT	string	magnitude		The method to convert Vector<complex<float> > to Vector<float> . Select magnitude, real, or imaginary. Output the absolute value, the real part, and the imaginary part, which are of the complex number, respectively.
METHOD_FLOAT_TO_COMPLEX	string	zero		The method to convert Vector<float> to Vector<complex<float> > . Select zero or hilbert. For the imaginary part, output 0 or the absolute value of the real part after converted to complex .
DEBUG	bool	false		Enable or disable to output the conversion status to standard output.

METHOD_COMPLEX_TO_FLOAT : string type. The method to convert the Vector<complex<float> > type to the Vector<float> type. Select magnitude, real, or imaginary. Output the absolute value, the real part, and the imaginary part, which are of the complex number, respectively. The default value is magnitude.

METHOD_FLOAT_TO_COMPLEX : string type. The method to convert the `Vector<float>` type to the `Vector<complex<float> >` type. Select zero or hilbert. For the imaginary part, output 0 by selecting zero or output the absolute value of the real part after converted to complex by selecting hilbert. The default value is zero.

DEBUG : bool type. Setting the value to true outputs the conversion status to the standard output. The default value is false.

Details of the node

Table 6.150: Conversion table of VectorToVector

INPUT	OUTPUT	Parameter to use
<code>Vector<float></code>	<code>Vector<complex<float> ></code>	<code>METHOD_FLOAT_TO_COMPLEX</code>
<code>Vector<complex<float> ></code>	<code>Vector<float></code>	<code>METHOD_COMPLEX_TO_FLOAT</code>

6.7.41 VectorValueOverwrite

Outline of the node

Overwrite a specified subvector of a `Vector<ObjectRef>` with a specified value.

Necessary file

No files are required.

Usage

When to use

This node is used to overwrite a specified subvector of a `Vector<ObjectRef>` with a specified value. The data type of the specified value will be converted to the suitable one depending on the `ObjectRef` of the `Vector<ObjectRef>` given as the input.

Input-output and property of the node

Input

INPUT : any type. Note that the supported data types are `Vector<int>`, `Vector<float>`, and `Vector<complex<float>>`.

Output

OUTPUT : any type. Note that the supported data types are `Vector<int>`, `Vector<float>`, and `Vector<complex<float>>`.

Parameter

Table 6.151: Parameter list of `VectorValueOverwrite`

Parameter name	Type	Default value	Unit	Description
OVERWRITTEN_MIN	int	0		The element index of the <code>Vector</code> given as the input to specify the first element of the subvector on which overwrite.
OVERWRITTEN_MAX	int	0		The element index of the <code>Vector</code> given as the input to specify the last element of the subvector on which overwrite.
OVERWRITE_VALUE_REAL	float	0		The value with which overwrite a specified subvector. The data type will be converted to the suitable one depending on the <code>ObjectRef</code> of the <code>Vector<ObjectRef></code> given as the input.
OVERWRITE_VALUE_IMAG	float	0		The value for the imaginary part with which overwrite a specified subvector when the <code>ObjectRef</code> of the <code>Vector<ObjectRef></code> given as the input is the complex type.
DEBUG	bool	false		Enable or disable to output the overwriting status to standard output.

OVERWRITTEN_MIN : int type. The element index of the `Vector<ObjectRef>` given as the input to specify the first element of the subvector on which overwrite. The default value is 0.

OVERWRITTEN_MAX : int type. The element index of the `Vector<ObjectRef>` given as the input to specify the last element of the subvector on which overwrite. The default value is 0.

OVERWRITE_VALUE_REAL : float type. The value with which overwrite a specified subvector. When the INPUT is `Vector<int>`, the type will be converted to the int type. When the INPUT is `Vector<complex<float>>`, the value will be for the real part. The default value is 0.

OVERWRITE_VALUE_IMAG : float type. The value for the imaginary part with which overwrite a specified subvector when the `ObjectRef` of the `Vector<ObjectRef>` given as the input is the complex type. The default value is 0.

DEBUG : bool type. Setting the value to `true` outputs the overwrite status to the standard output. The default value is `false`.

Details of the node

<example>

PARAMETER:

OVERWRITTEN_MIN:1,
OVERWRITTEN_MAX:2,
OVERWRITE_VALUE_REAL:9

INPUT:

< 1 2 3 4 >, < 3 4 5 6 >, < 5 6 7 8 >

OUTPUT:

< 1 9 9 4 >, < 3 9 9 6 >, < 5 9 9 8 >

6.7.42 WhiteNoiseAdder

Outline of the node

This node adds white noise to input signals.

Necessary files

No files are required.

Usage

When to use

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.

Typical connection

An example is shown in the figure.

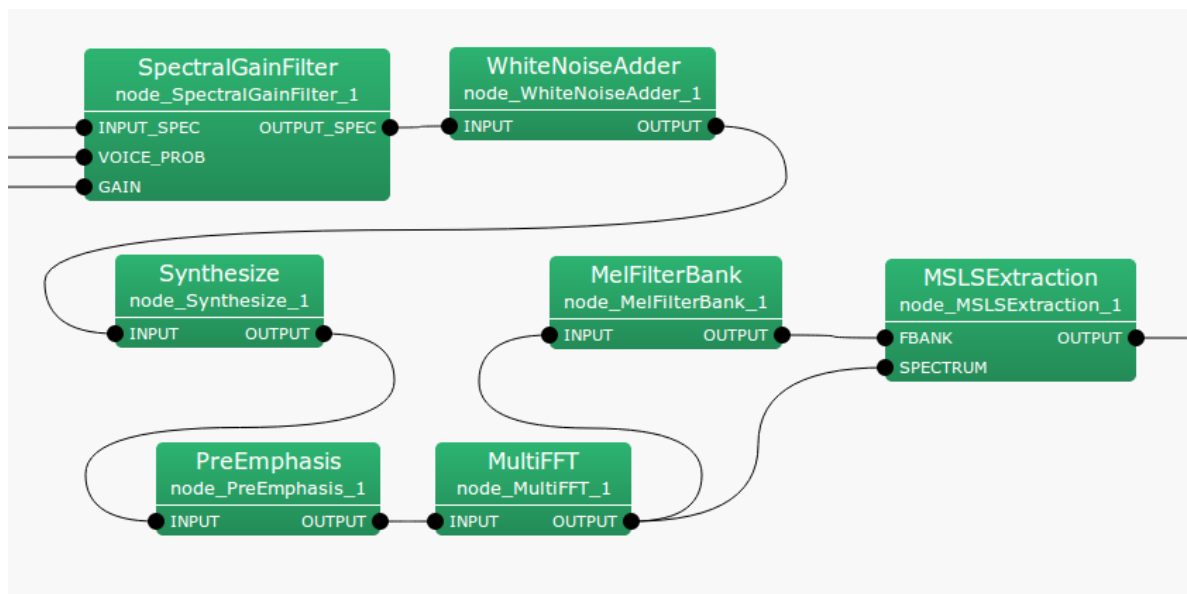


Figure 6.154: Example of connection of WhiteNoiseAdder

Input-output and properties of the node

Table 6.152: 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.

$$\frac{\sqrt{\text{LENGTH}}}{2} \cdot \text{WN_LEVEL} \cdot e^{2\pi jR} \quad (6.177)$$

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 Python:Kivy category

6.8.1 plotQuickMUSICSpecKivy

Outline of the node

This node displays the output obtained by the MUSIC method in GUI using Kivy.

Necessary file

None.

Usage

When to use

This node is used when you want to check the spectrogram of the sound source localization result.

Typical connection

Connect to the spectrum localized by LocalizeMUSIC .

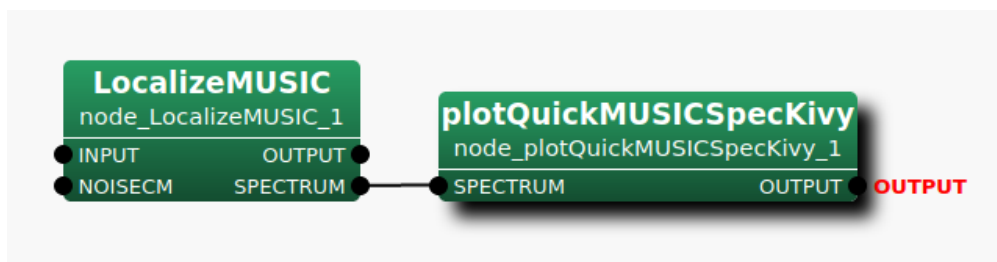


Figure 6.155: connection example of plotQuickMUSICSpecKivy

Input-output and property of the node

Inputs

SPECTRUM : Enter the vector(Vector<float> type) that represents the spectrogram. The number of dimensions is equal to the number of bins in the frequency direction.

Outputs

OUTPUT : Always 0.

Parameters

No parameters.

Details of the node

Figure ?? shows the execution screen of the plotQuickMUSICSpecKivy function.

- A spectrogram is displayed from the center to the upper left of the screen. The horizontal axis corresponds to the time(frame), and the vertical axis corresponds to the frequency(bin). The spectrogram is drawn in gray scale according to the power.

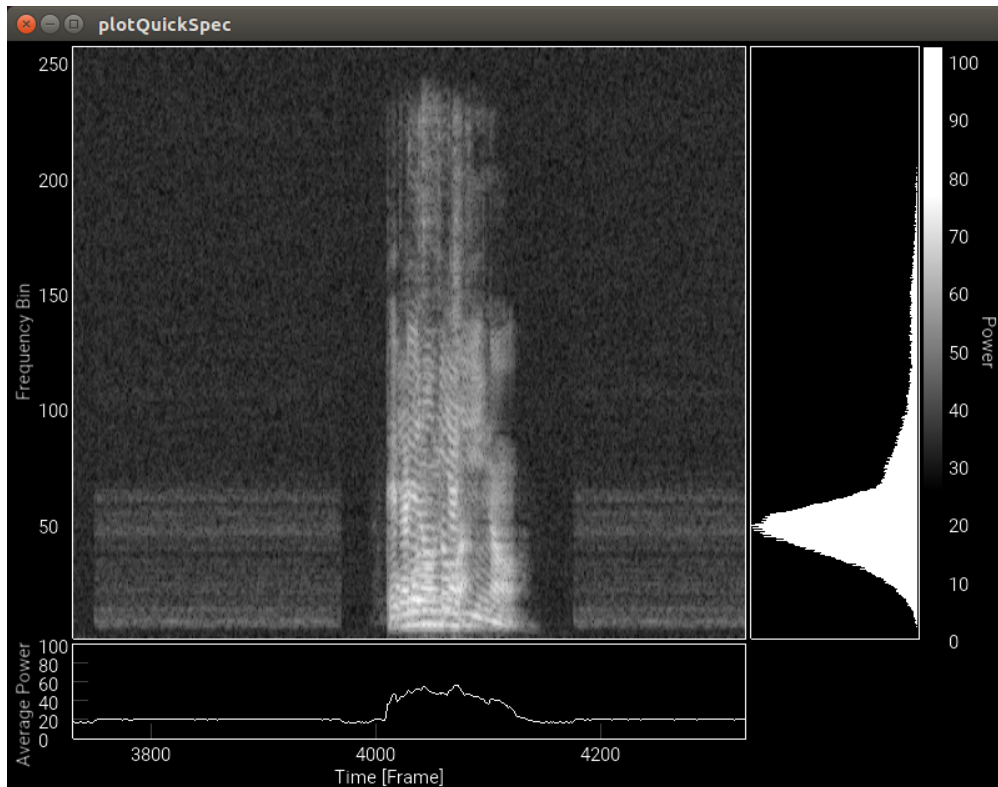


Figure 6.156: Execution screen of plotQuickMUSICSpecKivy

- The average power at each time is displayed at the bottom of the screen. The horizontal axis corresponds to the time(frame) and the vertical axis corresponds to the power.
- A histogram of the power is displayed on the right part of the screen. The vertical axis corresponds to the power and the horizontal axis to the frequency.
- By dragging the power histogram part with the mouse, you can specify the range of power to be displayed in grayscale in the spectrogram part. In the range of the specified power, the spectrogram is drawn with the luminance corresponding to the actual power value. If the power is out of the specified range, it will be drawn with a brightness value of 0 or 1.
- By dragging the spectrogram area with the mouse, you can specify the frequency range for the average power. The average power in the specified range will be displayed at the bottom of the screen.

6.8.2 plotQuickSourceKivy

Outline of the node

This node displays the results of sound source localization in a GUI using Kivy.

Necessary file

None.

Usage

When to use

This node is used when you want to visually confirm the result of sound source localization.

Typical connection

Connect to the localization result that assigned ID by SourceTracker .



Figure 6.157: connection example of plotQuickSourceKivy

Input-output and property of the node

Inputs

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

Outputs

OUTPUT : Always 0.

Parameters

No parameters.

Details of the node

Figure ?? shows the execution screen of plotQuickSourceKivy function.

- The horizontal axis corresponds to the time(frame) and the vertical axis corresponds to the source direction(angle).
- From the appearance of a sound source to its disappearance, it is plotted at a position corresponding to the direction of the source.

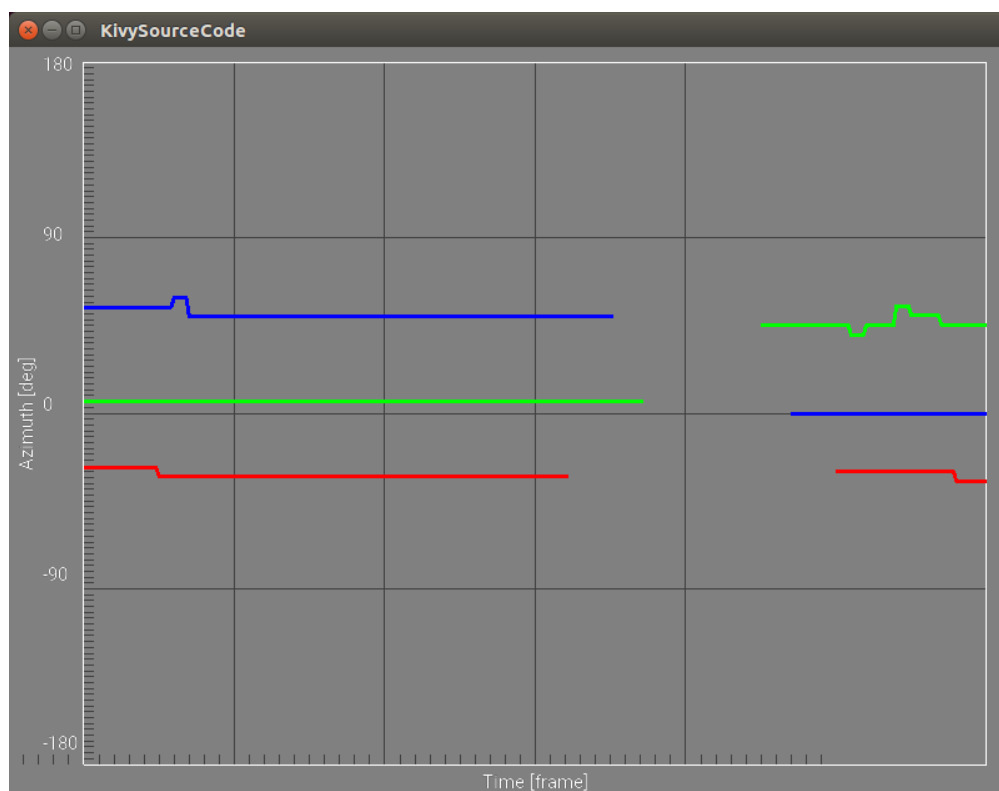


Figure 6.158: Execution screen of plotQuickSourceKivy

6.8.3 plotQuickSpecKivy

Outline of the node

This node displays spectrograms obtained from speech signals by FFT in a GUI using Kivy.

Necessary file

None.

Usage

When to use

This node is used when you want to visually confirm the spectrum converted from the audio signal.

Typical connection

Input the spectrum converted by MultiFFT .

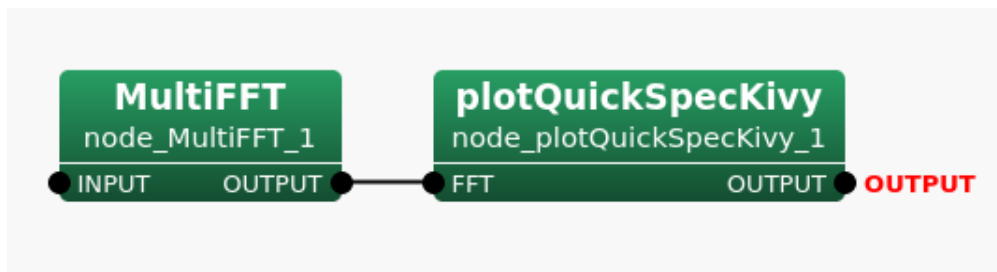


Figure 6.159: connection example of plotQuickSpecKivy

Input-output and property of the node

Inputs

FFT : `Vector<complex<float> >` type. The input is a vector(`Vector<complex<float> >`) that represents the FFT coefficients. The number of dimensions is equal to the number of FFT bins.

Outputs

OUTPUT : Always 0.

Parameters

No parameters.

Details of the node

Since the execution screen is the same as that of `plotQuickMUSICSpecKivy` , we omit the explanation.

6.8.4 plotQuickWaveformKivy

Outline of the node

This node displays waveforms of audio signal in a GUI using Kivy.

Necessary file

None.

Usage

When to use

This node is used when you want to visually check the audio signal waveform.

Typical connection

Input the audio waveform output from `AudioStreamFromMic` or `AudioStreamFromWave` .

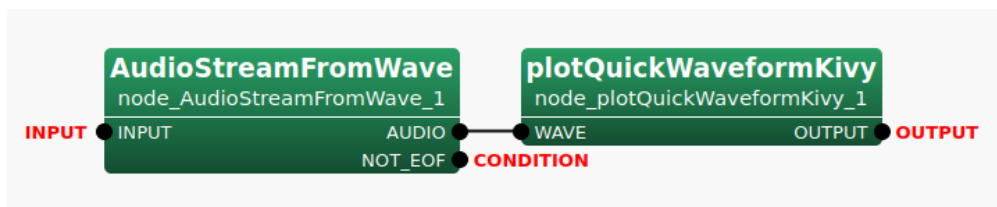


Figure 6.160: connection example of `plotQuickWaveformKivy`

Input-output and property of the node

Inputs

WAVE : `Matrix<float>` type. Input a matrix(`Matrix<float>` type) that consists of numerical values representing PCM waveforms. The rows(first dimension) correspond to the channels and the columns(second dimension) correspond to the samples.

Outputs

OUTPUT : Always 0.

Parameters

No parameters.

Details of the node

Figure ?? shows the execution screen of `plotQuickWaveformKivy` function.

- The horizontal axis corresponds to the time(frame) and the vertical axis corresponds to the channel.

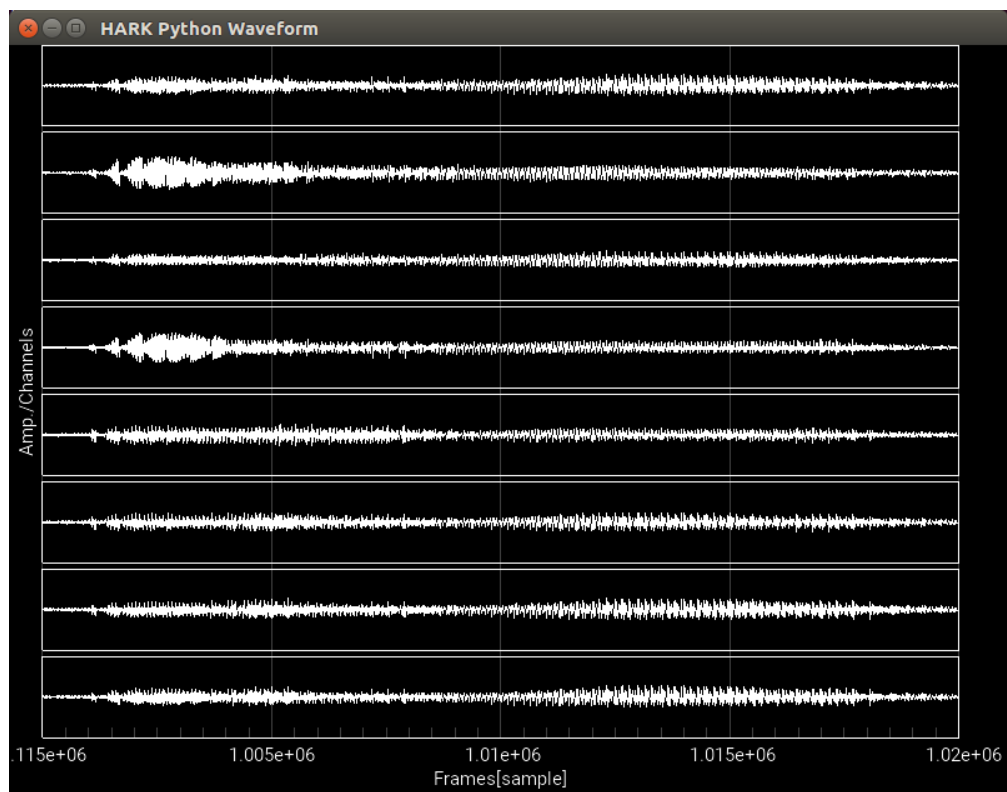


Figure 6.161: Execution screen of plotQuickWaveformKivy

6.9 TF category

6.9.1 EstimateTF

Outline of the node

This node creates the estimated transfer function file using the input signal.

This transfer function is calculated by the following steps.

1. record the input signal from the sound source which generates noise such as white noise.
2. calculate relative transfer function between the microphones for the source position.

Here, localization of the source is estimated by the node for localization such as `LocalizeMUSIC`. These steps should be executed for every direction around the microphone array.

Necessary file

The transfer function file which is used for the base, or the intermediate file group which consists of microphone position files, source position file, transfer function files for each direction and the intermediate file.

Usage

When to use

This node creates the estimated transfer function files using the input signal. The estimated transfer function will be used for the localization node and sound source separation node, such as `LocalizeMUSIC` and `GHDSS`.

Typical connection

Figure ?? shows a typical connection example.

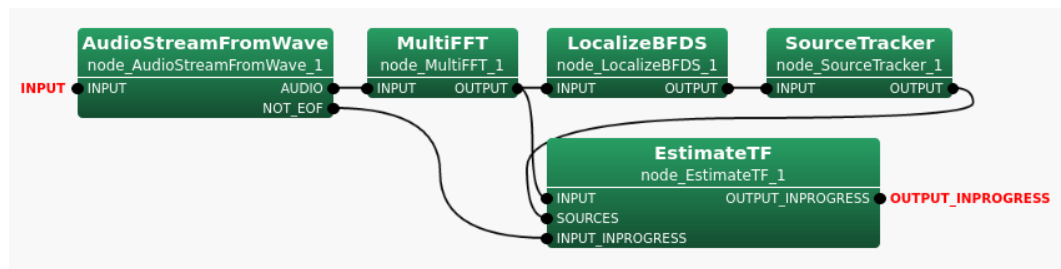


Figure 6.162: Connection example of EstimateTF

Figure ?? shows a typical connection example for the online processing of transfer function. `PyCodeExecutor3` placed in front of `EstimateTF` receives the localization processing result and outputs the sources and the count number. `EstimateTF` updates the transfer function using the source and the spectrum data corresponding to target count, and outputs it to the subsequent stage. `LocalizeMUSIC` updates the transfer function in the node with the transfer function data output from `EstimateTF` and executes localization. The result of localization is transmitted using `PyCodeExecutor3` via `SourceTracker`, and received by `PyCodeExecutor3` in the previous stage of `EstimateTF`. This allows `LocalizeMUSIC` to always use the latest transfer function data to execute localization.

Input-output and property of the node

Input

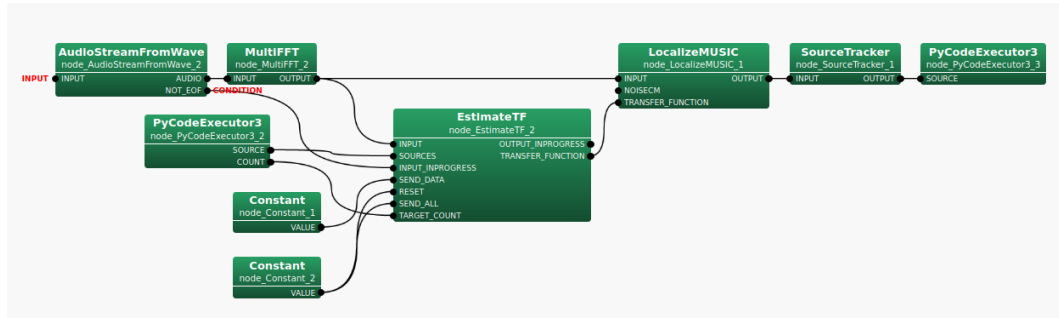


Figure 6.163: Connection example of EstimateTF (Online)

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

SOURCE : Source position (direction) is expressed as Vector<ObjectRef> type. ObjectRef is a Source and is a structure which consists of the source position.

INPUT_INPROGRESS : bool, shows the flag for the progressing. When this input becomes false, this node creates the transfer function file.

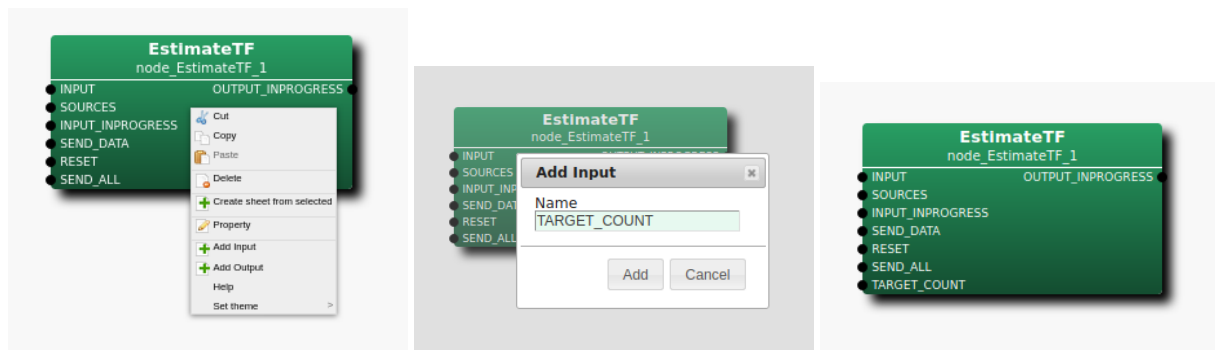
SEND_DATA : bool, Flag for data transfer. If 1/true, transfers transfer function data.

RESET : bool, Flag for data reset. If 1/true, resets and transfers all transfer function data.

SEND_ALL : bool, Flag for data transfer all. If 1/true, transfers all transfer function data.

TARGET_COUNT : int, Target count for estimation the transfer function. This input terminal is not displayed by default.

Refer to Figure ?? for the addition method of hidden input.



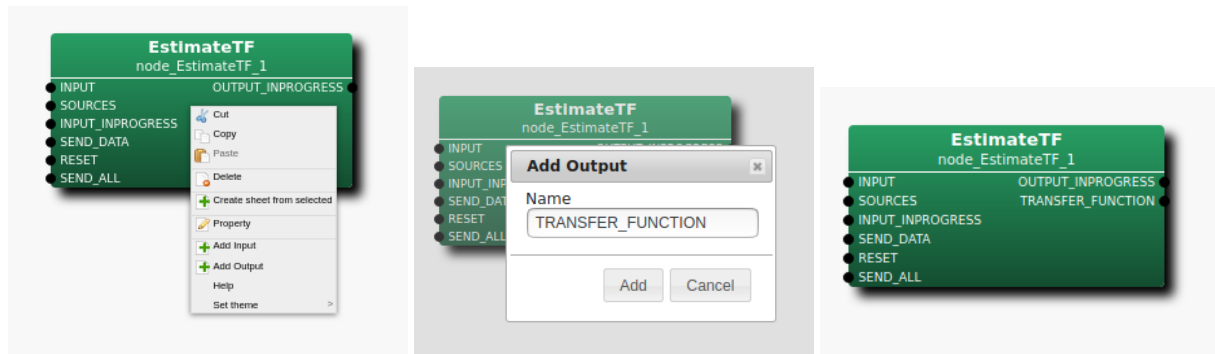
Step 1: Right-click EstimateTF and Step 2: Enter TARGET_COUNT in Step 3: The TARGET_COUNT input click Add Input. the input, then, click Add. terminal is added to the node.

Figure 6.164: Usage example of hidden inputs : Display of TARGET_COUNT terminal

Output

OUTPUT_INPROGRESS : bool, shows the flag for the progressing. The output value is same as INPUT_INPROGRESS.

TRANSFER_FUNCTION : TransferFunction, Transfer function data stream. This output terminal is not displayed by default.



Step 1: Right-click EstimateTF and click Add Output. Step 2: Enter TRANSFER_FUNCTION in the input, output terminal is added to the node. Step 3: The TRANSFER_FUNCTION terminal, then, click Add.

Figure 6.165: Usage example of hidden outputs : Display of TRANSFER_FUNCTION terminal

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

Parameter

Table 6.153: Parameter list of CSP

Parameter name	Type	Default value	Unit	description
EST_TF_FILENAME	string			Estimated transfer function file name
LENGTH	int	512	[pt]	FFT points (<i>NFFT</i>)
SAMPLING_RATE	int	16000	[Hz]	Sampling rate
INPUT_TYPE	string	TF		Input type for the base transfer function
BASE_TF_FILENAME	string			Transfer function file name for the base
IMPORT_INTERMEDIATE_DATA	string			Directory name of the intermediate file group for the base
EXPORT_INTERMEDIATE_DATA	string			Directory name of the intermediate file group for export
EXPORT_HISTORY_FILE	bool	false		ON/OFF for history file output
HISTORY_FILE_FRAMES	int	10000		Number of frames for exporting the history file
HISTORY_FILE_PERIOD	int	1		Period for exporting the history file
INITIAL_TF_OUTPUT	string	ALL		Whether to output all data at the first
LOOK_BACK_COUNT	int	10		The number of frames for look-back
DEBUG	bool	false		ON/OFF of debug output

EST_TF_FILENAME : string type. There is no default value. The file name of the estimated transfer function file is designated. When INPUT_INPROGRESS becomes false, this node creates the estimated transfer function file.

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.

INPUT_TYPE : string type. TF is the default value. The input type for the base transfer function is designated by TF or INTERMEDIATE. In the case of TF, BASE_TF.FILENAME should be designated for execution. In

the case of `INTERMEDIATE`, `IMPORT_INTERMEDIATE_DATA` should be designated, and `microphones.xml` and `source.xml` should be existed there.

BASE_TF_FILENAME : string type. There is no default value. The file name of the base transfer function file is designated. This parameter is shown when `INPUT_TYPE = TF`.

IMPORT_INTERMEDIATE_DATA : string type. There is no default value. The directory name of the intermediate file group for base is designated. `microphones.xml` and `source.xml` should be existed in this directory. This parameter is shown when `INPUT_TYPE = INTERMEDIATE`.

EXPORT_INTERMEDIATE_DATA : string type. There is no default value. The directory name of the intermediate file group for export is designated. When `EXPORT_HISTORY_FILE = true`, this parameter should be designated.

EXPORT_HISTORY_FILE : bool type. `false` is the default value. ON/OFF of the history file output. When `EXPORT_HISTORY_FILE = true`, this node outputs the history file to `EXPORT_INTERMEDIATE_DATA`.

HISTORY_FILE_FRAMES : bool type. 10000 is the default value. Number of the frames for exporting history. This parameter is shown when `EXPORT_HISTORY_FILE = true`.

HISTORY_FILE_PERIOD : int type. 1 is default value. The cycle for output the history. This parameter is shown when `EXPORT_HISTORY_FILE = true`.

INITIAL_TF_OUTPUT : string type. Whether to output all data at the first. "ALL" outputs all data, and "PARTIAL" outputs only update date. The default value is ALL.

LOOK_BACK_COUNT : int type. The number of frames for look-back. The default value is 10.

DEBUG : bool type. ON/OFF of the debug output and the format of the debug output are as follows. First, the updated source position is outputted. Then, the updated transfer function is shown.

Details of the node

Estimation of the transfer function :

The transfer function is estimated by averaging amplitude and phase of the input signal for each direction which is estimated by sound source localization.

The transfer function is estimated by following steps.

1. normalize the base transfer function
2. normalize the input spectrum
3. update the transfer function with normalized input spectrum by linear averaging

Normalization is calculated as $norm of amplitude = 1$ and $average of phase = 0deg$.

Intermediate file group :

Intermediate file group is consists of following file.

- `whatisthis.txt...htransfer functionh`
- `source.xml...source position file`
- `microphones.xml...microphone position file`
- `localization/tf#####.mat...transfer function files for localization`
- `separation/tf#####.mat...transfer function files for separation`
- `intermediate.mat...intermediate file`

- history.mat...history file

When the base transfer function is input by the intermediate file group, source.xml and microphones.xml should be existed.

About the intermediate file :

The intermediate file is used for additional update for the transfer function.

The file format of this file is the Matrix binary format, which type is float. The row data of the matrix means the information for each source position.

- Column 1 - 5 : TFIDCposition(x,y,z), Number of frames for the position
- Column 6 - $(M \times LENGTH \times 2) + 5$: Averaged value(real part and imaginary part) of the estimated transfer function. These values are stored in the order of *Microphone* \times *Frequency*.
- Column $(M \times LENGTH \times 2) + 6 - (M \times LENGTH \times 2) \times 2 + 5$: Varianced value(real part and imaginary part) of the estimated transfer function. These values are stored in the order of *Microphone* \times *Frequency*.

About the history file :

The history file is used for the analysis of estimating the transfer function. This files shows the alteration of the averaged value and the varianced value of the transfer function according to update. The cycle of writing the history is designated by HISTORY_FILE_PERIOD. When the written size of history data is over the HISTORY_FILE_FRAMES, the oldestst history data is overwritten by the new history data.

The file format of this file is the Matrix binary format, which type is float. The row data of the matrix means the information for each history data.

- Column 1 - 6 : Index of frame, TFIDCposition(x,y,z), Number of frames for the position
- Column 7 - $(M \times LENGTH \times 2) + 6$: Averaged value(real part and imaginary part) of the estimated transfer function. These values are stored in the order of *Microphone* \times *Frequency*.
- Column $(M \times LENGTH \times 2) + 7 - (M \times LENGTH \times 2) \times 2 + 6$: Varianced value(real part and imaginary part) of the estimated transfer function. These values are stored in the order of *Microphone* \times *Frequency*.

6.10 Modules independent of FlowDesigner

6.10.1 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 Julius³ 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

³<http://www.furui.cs.titech.ac.jp/mband-julius/>

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.

- **-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 `adinet` 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_mask_to_...` (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 each `gconst` 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 ?? only once just after the socket connection. The sound source information is expressed in a `SourceInfo` structure (Table ??) 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 6.154: 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 6.155: 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 ?? as one frame. It is assumed inside the JuliusMFT that the dimension number of feature vectors and mask vectors are same.

Table 6.156: Data to be transmitted for every frame

Size [byte]	Type	Data to be transmitted
4	int	$N1 = (\text{dimension number of feature vector}) \times \text{sizeof}(\text{float})$
N1	float [N1]	Array of feature vector
4	int	$N2 = (\text{dimension number of mask vector}) \times \text{sizeof}(\text{float})$
N2	float [N2]	Array of mask vector

4. **Completing process** Finishing transmitting features for one sound source, data (table ??) that indicate completion are transmitted and the socket is closed.

Table 6.157: 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. 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 divided 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 s h 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 s h 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 s h 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 ??.

6.10.2 KaldiDecoder

Outline

KaldiDecoder is an acoustic model decoder software developed for HARK . This particular decoder was designed using libraries from Kaldi ⁴ , a deep learning speech recognition toolkit. Up until HARK version 2.2.0, JuliusMFT (a large vocabulary speech recognition decoder system based on Julius) had been the core of HARK 's speech recognition. In response to the recent trends in speech recognition software design, we have decided to provide a Kaldi-based decoder, KaldiDecoder , beginning with HARK version 2.3.0.

Comparing this new decoder with other standard Kaldi -based decoders, the following features are available:

1. Connectivity with HARK modules (same handling as JuliusMFT)

- Compatible with both MSLS and MFCC feature data input via the network (mfcnet)
- Supports the addition of source location info (SrcInfo)
- Supports the recognition of simultaneous speech (mutual exclusion)

The connections with HARK can be made through SpeechRecognitionClient (or through SpeechRecognitionSMNClient), identical to JuliusMFT .

2. Compatibility with JuliusMFT .

- Compatible with JuliusMFT output emulation (in both module-mode and standard output formats)

KaldiDecoder replicates JuliusMFT output as closely as possible, such that modification to the JuliusMFT -based sound system (in its demonstration system or sound scoring system) should be minimal.

3. Additional Kaldi functionality

- Implementation of online decoding for nnet1 models

Kaldi 's standard decoder provides only offline nnet1 model decoding.

4. Functions to be implemented

- Missing feature recognition (except for the already implemented mfcnet-masked data structure recognition)

The features and functions described in numbers 1, 2, and 3 above have been implemented without any change to Kaldi .

The following section explains the method of installing and using KaldiDecoder , with a step-by-step procedure for connecting to HARK in FlowDesigner.

Start up and setting

Execution of KaldiDecoder is performed as follows assuming a settings file named kaldi.conf is used.

```
> kaldidecoder --config=kaldi.conf (for Ubuntu OS)
> kaldidecoder.exe --config=kaldi.conf (for Windows OS)
```

After starting KaldiDecoder in online mode, the socket connection in HARK 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 kaldi.conf is a text file that describes settings for KaldiDecoder . The content of the setting file consists basically of argument options that begin with "--", and the user can also specify arguments directly as KaldiDecoder options when starting. Moreover, descriptions that come after # are treated as comments. Confirm the options used for KaldiDecoder by executing the following command:

⁴<http://kaldi-asr.org/>


```
> kaldidecoder --help (for Ubuntu OS)
> kaldidecoder.exe --help (for Windows OS)
```

The minimum required settings for using nnet1 models in KaldiDecoder are the following seven items:

- `--filename-words=<YOUR_PATH>/words.txt`
- `--filename-align-lexicon=<YOUR_PATH>/align_lexicon.int`
- `--filename-feature-transform=<YOUR_PATH>/final.feature_transform`
- `--filename-nnet=<YOUR_PATH>/final.nnet`
- `--filename-mdl=<YOUR_PATH>/final.mdl`
- `--filename-class-frame-counts=<YOUR_PATH>/ali_train_pdf.counts`
- `--filename-fst=<YOUR_PATH>/HCLG.fst`

The minimum required settings for using nnet3 models in KaldiDecoder are the following four items:

- `--filename-words=<YOUR_PATH>/words.txt`
- `--filename-align-lexicon=<YOUR_PATH>/align_lexicon.int`
- `--filename-mdl=<YOUR_PATH>/final.mdl`
- `--filename-fst=<YOUR_PATH>/HCLG.fst`

In the case of the chain model, in addition to the setting of nnet3, the following setting is necessary:

- `--frame-subsampling-factor=3`

If iVector was used in nnet3/chain model learning, the following setting is additionally required:

- `--ivector-extraction-config=<YOUR_PATH>/ivector_extractor.conf`

The offline decoding mode requires an additional setting to specify the list of features to be evaluated: Notes: From HARK version 2.5.0, parallel decoding became possible also in the case of offline decoding, so it is recommended to restrict the number of the decoder instances which are simultaneously activated with the “`--max-tasks=<Number of the Cores>`” options. As in HARK version 2.4.0 and earlier, if you wish to guarantee the order of the recognition results to be the same as of the features list, specify 1 as shown below.

- `--filename-features-list=<YOUR_PATH>/features_list.txt`
- `--max-tasks=1`

If the above setting is not given, KaldiDecoder will automatically execute in the online decoding mode (mfcnet input mode).

Modify the online decoding mode mfcnet input port or result output port by changing the following options (default port number values are shown):

- `--port-mfcnet=5530`
- `--port-result=10500`

1. Basic Configurations (Input Files and Modes Settings)

- **`--nnet-type=model type number`**

This is to set the nnet model type in Kaldi. The default setting value is “1”. Depending on the model you have, you can change the settings as follows:

Set the value to “1” when using a nnet1 model created using Karel Vesely’s method. Change it to “3”, if you use a nnet3/chain model created using Daniel Povey’s method.

- **`--filename-words=word list file name`**

This setting specifies the word list file. You can set it as a path relative to the current directory or as an absolute path. The file format of the word list file is the same as that of the `words.txt` file included in the lexicon directory used in Kaldi for training/validation or in the language model used for evaluation. This setting is mandatory for the output of recognition results.

- **--filename-phones=phoneme list file name**
This is to set the phoneme list file. You can set it as a path relative to the current directory or as an absolute path. The file format of the phoneme list file is the same as the `phones.txt` file included in the lexicon directory used in Kaldi for training/validation or in the language model used for evaluation. This setting is optional, as it is only necessary when you conduct phoneme alignment.
- **--filename-align-lexicon=lexicon file name**
This is to set the lexicon file. You can set it as a path relative to the current directory or as an absolute path. The file format of the lexicon file is the same as that of the `align_lexicon.int` file included in the lexicon directory used in Kaldi for training/validation or in the language model used for evaluation. If you create a lang directory using `prepare_lang.pl`, the lexicon file will be output to `lang/phones/aligned_lexicon.int`. This setting is mandatory for the output of recognition results.
- **--filename-feature-transform=FeatureTransform file name**
This is to set the FeatureTransform file, which is output to the path of the trained DNN as `exp/<model_dir>/final.feature_transform`. This setting is mandatory when you use KaldiDecoder in nnet1 model decoding, as it is separate from the acoustic model.
- **--filename-nnet=nnet file name**
This is to set the nnet file, which is output to the path of the trained DNN as `exp/<model_dir>/final.nnet` (note: this path is a symbolic link). This setting is mandatory when you use KaldiDecoder in nnet1 model decoding, as it is separate from the acoustic model.
- **--filename-mdl=mdl file name**
This is to set the mdl file, which is output to the path of the trained DNN as `exp/<model_dir>/final.mdl` (note: this path is a symbolic link). This setting is mandatory, as the acoustic model is required for the output of recognition results.
- **--filename-class-frame-counts=class-frame-counts file name**
This is to set the class-frame-counts file, which is output to the path of the trained DNN as `exp/<model_dir>/ali_train_pdf.counts`. This setting is mandatory when you use KaldiDecoder in nnet1 model decoding, as it is separate from the acoustic model.
- **--filename-fst=FST file name**
This is to set the FST file. If you create the graph directory using `mkgraph.sh`, the FST file will be output to `graph*/HCLG.fst`. This setting is mandatory, as it is required for the output of recognition results.
- **--filename-features-list=features list file name**
When using KaldiDecoder in offline decoding mode, it is required to specify a text list containing paths to the feature file(s) to be evaluated (note: it is the name of the list; not of the feature file(s)). If this option is not set, KaldiDecoder runs in the online decoding mode.
- **--ivector-extraction-config=ivector extraction config file name**
This is to set the ivector extraction config file, which is output to the path of the trained DNN as `exp/<model_dir>/ivector*/conf/ivector_extractor.conf`. This setting is mandatory when you use iVectors in nnet3/chain model learning.
- **--frame-subsampling-factor=frame subsampling factor**
This is to set the frame subsampling factor, which is output to the path of the trained DNN as `exp/<model_dir>/frame-subsampling-factor` (Only the integer value written in this file is necessary). This setting is mandatory when you use a chain model. It must be set as follows.
`--frame-subsampling-factor=3`
- **--frames-per-chunk=frames per chunk**
This is to set the number of frames in each chunk that is separately evaluated by the neural net. Measured before any subsampling, if the `--frame-subsampling-factor` option is used. (i.e. counts input frames) See the `step/nnet3/decode.sh` or `steps/nnet3/decode_looped.sh` scripts used by Kaldi's decode recipe for this option. The default setting value is 20, but it is set to 50 in the above decoding recipe.
- **--port-mfcnet=port number**
This is to set the port number that receives the acoustic features and masks transmitted via network by SpeechRecognitionClient (or SpeechRecognitionSMNClient). It is similar to the "-adport" input port setting for mfcnet input mode in JuliusMFT. The same default port number as in JuliusMFT, 5530, will be used if none is provided. This option is valid only in the online decoding mode.
- **--port-result=port number**

This is to set the connection port that transmits the recognition results through the network. It is similar to the “-module” output port setting for module mode in JuliusMFT . The same default port number as in JuliusMFT , 10500, will be used if none is provided.

- **--host-mfcnet=host name**

This option sets the IP address or host name of the server that listens to the port configured with the --port-mfcnet option. The default setting value is “localhost” .

- **--host-result=host name**

This option sets the IP address or host name of the server that listens to the port configured with the --port-result option. The default setting value is “localhost” .

- **--lm-name**

This is to set the language model name. When this setting is active, the LMNAME attribute is provided in module mode output. It has no default value.

2. Decoder Tuning Configuration (Weighting and Pruning)

The items described in this section are options inherited from the features implemented in the decoder (classes) in Kaldi .

- **--acoustic-scale=acoustic scale value**

This is to set the scaling factor for acoustic log-likelihoods. The default setting value is “0.1”, which is generally the inverse of the best LM weight obtained in scoring.

Reference: <https://sourceforge.net/p/kaldi/discussion/1355348/thread/924c555b/>

However, for chain models, “1.0” is the best setting.

Detail, please refer to http://kaldi-asr.org/doc/chain.html#chain_decoding .

- **--max-active**

This is to set the decoder’s maximum number of active states. Although increasing the value provides more accurate results, it also significantly affects the decoding speed. The default setting value is “2147483647” (maximum value of an int32 type). We empirically recommend setting it between 2000 and 5000 to obtain better performance.

- **--min-active**

This is to set the decoder’s minimum number of active states. The default setting value is “200”.

- **--beam=beam width**

This is to set the decoding beam width. Although increasing the value provides more accurate results, it also considerably affects the decoding speed. The default setting value is “16.0”. (> 0.0) For details, refer to the quotation in Table ??.

- **--beam-delta=beam delta**

This is to set the decoding beam delta. This parameter is obscure and relates to a speed-up in the way in which the “max-active” constraint is applied. Increasing the value provides more accurate results. The default setting value is “0.5”. (> 0.0) For details, refer to the quotation in Table ??.

- **--delta=delta**

This is the tolerance value used in determinization, which is set as “0.000976562” by default. For details, refer to the quotation in Table ??.

- **--hash-ratio**

This is a ratio value to control the hash behavior in decoding process. The default setting value is “2.0”. (>= 1.0)

- **--prune-interval=frame count**

This is to set the frame interval at which tokens are to be pruned. The default setting value is “25”. (> 0)

- **--splice=splice count**

This is to set the DNN input splice count. It is the number of frames around the current frame. The default setting value is “3”, which means that there are 3 frames both before and after the current frame.

The following article in the Table ?? is quoted from the Kaldi website (<http://www.danielpovey.com/kaldi-docs/decoders.html>).

3. Lattice Configuration

The items described in this section are options that have been inherited from the features implemented in the decoder (classes) in Kaldi .

Table 6.158: Quotation from “FasterDecoder: a more optimized decoder”

The code in FasterDecoder as it relates to cutoffs is a little more complicated than just having the one pruning step. The basic observation is this: it's pointless to create a very large number of tokens if you are only going to ignore most of them later. So the situation in ProcessEmitting is: we have "weight_cutoff" but wouldn't it be nice if we knew what the value of "weight_cutoff" on the next frame was going to be? Call this "next_weight_cutoff". Then, whenever we process arcs that have the current frame's acoustic likelihoods, we could just avoid creating the token if the likelihood is worse than "next_weight_cutoff". In order to know the next weight cutoff we have to know two things. We have to know the best token's weight on the next frame, and we have to know the effective beam width on the next frame. The effective beam width may differ from "beam" if the "max_active" constraint is limiting, and we use the heuristic that the effective beam width does not change very much from frame to frame. We attempt to estimate the best token's weight on the next frame by propagating the currently best token (later on, if we find even better tokens on the next frame we will update this estimate). We get a rough upper bound on the effective beam width on the next frame by using the variable "adaptive_beam". This is always set to the smaller of "beam" (the specified maximum beam width), or the effective beam width as determined by max_active, plus beam_delta (default value: 0.5). When we say it is a "rough upper bound" we mean that it will usually be greater than or equal to the effective beam width on the next frame. The pruning value we use when creating new tokens equals our current estimate of the next frame's best token, plus "adaptive_beam". With finite "beam_delta", it is possible for the pruning to be stricter than dictated by the "beam" and "max_active" parameters alone, although at the value 0.5 we do not believe this happens very often.

Povey, Daniel:

Citing Sources: [http://www.danielpovey.com/kaldi-docs/decoders.html#decoders_faster]: para. 3: [December 6, 2016]

- **--determinize-lattice**

This is to determinize the lattices. It keeps only the best probability distribution function (p.d.f.) sequence for each word sequence.

- **--lattice-beam=beam width**

This is to set the beam width in lattice generation. Increasing the value gives deeper lattices, which also significantly affects the decoding speed. The default setting value is “10”.

- **--max-mem=maximum memory allocation size**

This is to set the maximum approximate size of memory allocated when determinizing the lattices. However, the actual usage may be higher than the specified value because the allocation may occur more than once.

- **--minimize**

When this option is given, minimize the lattices after determinization.

- **--phone-determinize**

When this option is given, do an initial pass of determinization on both phonemes and words. See also the article on “--word-determinize”.

- **--word-determinize**

When this option is given, do a second pass of determinization on words only. See also the article on “--phone-determinize”.

4. Others

- **--config=configuration file name**

This setting specifies the config file, which can be specified repeatedly.

- **--enable-debug**

This option enables the debugging output. The default setting is “disabled”.

- **--help**

This is to display the help menu. When this option is given, all other options are ignored.

- **--print-args**

When this option is enabled, the command line arguments are sent to the standard output. The default setting is “enabled”. Set it as “--print-args=false” to disable it.

- **--verbose=log level**

This is to set detail level of log information. Increasing the value gives more detailed log output. The default setting value is “0”.

The functionality called “module mode” in the original Julius or JuliusMFT is also available in KaldiDecoder . Selecting the online decoding mode automatically enables it. In addition, the standard output is not deactivated as in Julius or JuliusMFT ; both standard and socket (network) outputs can be used in KaldiDecoder ’s online decoding mode.

Detailed description

mfcnet communication specification

In order to use mfcnet as an acoustic input source, the argument “--filename-features-list” must not be given when starting up KaldiDecoder as mentioned in the options description. In this case, KaldiDecoder acts as a TCP/IP communications server, starting up in the listening state and waiting for input. Moreover, the HARK modules SpeechRecognitionClient and SpeechRecognitionSMNClient work as a client to transmit acoustic features and Missing Feature Mask to KaldiDecoder. The client connects to KaldiDecoder for every utterance and closes the connection immediately after the transmission is complete. 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 the mfcnet communication port in KaldiDecoder.

2. **Communication initialization (data transmitted once at the beginning)**

The client transmits information on the sound source that is going to be transmitted, as shown in Table ??, immediately after the socket connection. The sound source information is expressed in a SourceInfo structure (Table ??) and has a sound source ID, sound source direction and time of transmission start. The time is indicated in a timeval structure defined in `<sys/time.h>` and is the elapsed time from the starting time point (January 1, 1970 00:00:00) in the system time zone. The time indicates the elapsed period from the starting point thereafter.

3. **Data transmission (data transmitted at every frame)**

Acoustic features and Missing Feature Mask are transmitted. Features of one utterance are transmitted as frames, shown in Table ??, repeatedly until the end of the speech section. It is assumed inside the KaldiDecoder that the dimension number of feature vectors and mask vectors are the same.

4. **Connection end (data transmitted once at the end)**

After transmitting features for one sound source, data (Table ??) that indicate completion is transmitted. KaldiDecoder will return to the listening state to receive the next sound source data until either the data indicating completion is received or its socket connection is severed. It is therefore possible to resume and continue data reception in an environment with a relatively unstable connection.

5. Socket disconnection

After the ending process, the sockets are closed. If they close without the ending process, it executes exception tasks; thus, the output of the recognition results may be delayed. Likewise, any data transmitted after the ending process is ignored regardless of the sockets being open or closed.

Table 6.159: Data to be transmitted only once at the beginning (acoustic source information)

Size[byte]	Type	Data to be transmitted
4	int	28 (= sizeof(SourceInfo))
28	SourceInfo	Sound source information of features that are going to be transmitted

Table 6.160: 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 16 bytes long)

Table 6.161: Data to be transmitted for every frame (features, masks data and dimensions information)

Size[byte]	Type	Data to be transmitted
4	int	N1=(dimension number of feature vector) \times sizeof(float)
N1	float [N1]	feature vector (float array)
4	int	N2=(dimension number of mask vector) \times sizeof(float)
N2	float [N2]	mask vector (float array)

Table 6.162: Data to be transmitted only once at the end (data to indicate completion)

Size[byte]	Type	Data to be transmitted
4	int	0

Module mode communication specification

When setting the online decoding, KaldiDecoder operates similarly to the module mode in Julius . In the module mode, KaldiDecoder works as a TCP/IP communication server and provides recognition results to clients such as jcontrol. The character encoding for Japanese text depends on that of the language model used. An XML-like format is used just like in Julius for data representation, and a “.” (period) is transmitted to indicate the data completion for each and every message. As an additional feature of KaldiDecoder , it can also output results in the standard XML format without the “.” (period) mark. The meaning of the most common tags transmitted by KaldiDecoder is as follows.

- INPUT tag

This tag represents information related to inputs and has STATUS and TIME as attributes. The values for STATUS are LISTEN, STARTREC or ENDREC. LISTEN indicates that KaldiDecoder is ready to receive speech. STARTREC indicates that the reception of features has started. ENDREC indicates that the last feature of the sound source being received has arrived. TIME indicates the time at that instant.

- SOURCEINFO tag

This tag represents information related to sound sources and is an original tag of KaldiDecoder . It has ID, AZIMUTH, ELEVATION, SEC and USEC as attributes. The SOURCEINFO tag is transmitted when starting the recognition process. Its ID indicates a sound source ID given by HARK (not the speaker ID but numbers uniformly given to each sound source). AZIMUTH and ELEVATION indicate horizontal and vertical direction (degrees), respectively, seen from the microphone array coordinate system for the first frame of the sound source. SEC and USEC indicate the time of the first frame of the sound source. SEC indicates seconds and USEC indicates the microseconds fraction.

- RECOGOUT tag

This tag represents recognition results, and its sub-element is either a gradual output or the final output. For gradual output, it has the PHYPO tag as a sub-element, and for the final output, it has the SHYPO tag as a sub-element. In the case of the final output, only SHYPO tags for the number of candidates specified in the parameters are output.

- PHYPO tag

This tag represents gradual candidates and it has vectors of WHYPO tags for candidate words as sub-elements. It has PASS, SCORE, FRAME and TIME as attributes. PASS indicates the number of decoding passes and is always 1. SCORE indicates the accumulated score of this candidate. FRAME indicates the number of frames that have been processed in order to output this candidate. TIME indicates time (sec) at that instant.

- SHYPO tag

This tag represents a sentence hypothesis and it has vectors of WHYPO tags for candidate words as sub-elements. It has PASS, RANK, SCORE, AMSCORE and LMSCORE as attributes. PASS indicates the number of decoding passes and, when available, is always set to 1. RANK indicates the rank order of a hypothesis. SCORE indicates the logarithmic likelihood of this hypothesis, AMSCORE indicates a logarithmic acoustic likelihood and LMSCORE indicates a logarithmic language probability.

- WHYPO tag

This tag represents word hypotheses and has WORD, CLASSID, PHONE and CM as attributes. WORD indicates notations, CLASSID indicates the word that is the key in a statistical language model, PHONE indicates phoneme sequences and CM indicates the confidence for the word. Word confidence is included only to maintain compatibility with the Julius -based decoder output, and its value, fixed at 1.0, is irrelevant to the actual performance.

- SYSINFO tag

This tag represents the status of the system and it has 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 that speech recognition can be performed. When PROCESS is SLEEP, it indicates that speech recognition is halted.

Whether or not these tags and attributes are output depends on the arguments set when starting KaldiDecoder . The SOURCEINFO tag is always output, and the others are the same as those of the original Julius and therefore users are recommended to refer to Argument Help of the original Julius .

When comparing to the original Julius , two changes were made to KaldiDecoder .

- Addition of items related to the SOURCEINFO tag for information on source localization as described above, and also the embedding of sound source ID (SOURCEID) to the following tags: STARTRECOG, ENDRECOG, INPUTPARAM, GMM, RECOGOUT, REJECTED, RECOGFAIL, GRAPHOUT, SOURCEINFO
- Changes were made to the format of the module mode in order to reduce the delay caused by mutual exclusion when processing simultaneous utterances. Concretely, mutual exclusion used to be performed utterance-wise, but now the output is divided so that the exclusion control can be performed tag-wise. Also, modifications were made to the output of the following one-time tags.

<< Tags separated by start-tag / end-tag >>

- <RECOGOUT> ... </RECOGOUT>
- <GRAPHOUT> ... </GRAPHOUT>
- <GRAMINFO> ... </GRAMINFO>
- <RECOGPROCESS> ... </RECOGPROCESS>

<< One-line tags that are internally split and output multiple times >>

- <RECOGFAIL ... />
- <REJECTED ... />
- <SR ... />

Example output of KaldiDecoder

1. Example output of standard output mode

```
source_id = 0, azimuth = 0.000000, elevation = 16.700001, sec = 1466144473, usec = 169637
### Recognition: 2nd pass (RL heuristic best-first)
STAT: 00
sentence1: ORDER PLEASE
wseq1: ORDER PLEASE
phseq1: ao ao ao r r r r r d d d er er er er er er p p p l l l iy iy iy iy iy
       iy iy z z z
cmscore1: 1.000 1.000
score1: 260.002472 ( AM: 274.768372, LM: -14.765888 )
```

2. Example output of socket output mode (module mode)

```
<SOURCEINFO SOURCEID="0" AZIMUTH="0.000000" ELEVATION="16.700001" SEC="1466144473"
USEC="169637"/>
.
<STARTRECOG SOURCEID="0"/>
.
<ENDRECOG SOURCEID="0"/>
.
<RECOGOUT SOURCEID="0">
  <SHYPO RANK="1" SCORE="260.002472" AMSCORE="274.768372" LMSCORE="-14.765888">
    <WHYPO WORD="ORDER" CLASSID="ORDER" PHONE="" CM="1.000"/>
    <WHYPO WORD="PLEASE" CLASSID="PLEASE" PHONE="" CM="1.000"/>
  </SHYPO>
</RECOGOUT>
.
```

Notice

- Restraint of the PHONE tag

Although JuliusMFT supports PHONE tag output for each WORD, the same feature is not implemented in KaldiDecoder because of Kaldi's structural reasons: it causes performance degradation. Therefore, in the socket output mode, the phoneme output for each WHYPO tag is not supported. In the standard output mode, only output with no pipes ("|") between words is supported.

- Known issue in the Windows version

There were confirmed issues of corrupted characters when outputting recognition results in a multi-byte encoding to the standard output on Windows. The default character set for Ubuntu terminal's standard output is UTF-8, so the issue occurs when using the same language model on Windows. In other words, a mismatch between the character sets used in the operating system console and language model causes this problem. To avoid this, start KaldiDecoder with the output redirection "> filename", and open the output recognition text file with an appropriate text editor. The reason for this restriction is that, in Julius, it was possible to convert the character set at the output time using the "iconv" library or the internal implementation "libjcode" as needed; however, Kaldi does not have a character set conversion feature, and it has not been implemented in KaldiDecoder.

Installation method

- Using apt

If the HARK apt repository is registered, installation can be done as follows.

```
> sudo apt install kaldidecoder-hark
```

- Installing from source

Since KaldiDecoder uses libraries from Kaldi, Kaldi must be built in advance. However, Kaldi libraries are not packaged in Ubuntu, so it has to be compiled from source by executing the following commands.

```
> sudo apt update
> sudo apt install git automake autoconf libtool cmake cmake-extras build-essential
> sudo apt install libopenblas-base libopenblas-dev
> cd ~/
> mkdir <YOUR_DIR>
> cd <YOUR_DIR>
> git clone https://github.com/kaldi-asr/kaldi.git
> git checkout <COMMIT_ID>
> cd kaldi/tools
> make
> cd ../src
> ./configure --mathlib=OPENBLAS --openblas-root=/usr
> make clean -j <CORES>
> make depend -j <CORES>
> make -j <CORES>
> cd ../
> wget http://archive.hark.jp/harkrepos/dists/<DISTRO>/non-free/source/kaldidecoder-
hark_<HARK_VER>.tar.xz
> tar -Jxvf kaldidecoder-hark_<HARK_VER>.tar.xz
> cd kaldidecoder3
> mkdir build
> cd build
> cmake .. -DCMAKE_BUILD_TYPE=None -DOPENBLAS_ROOT_DIR:STRING=/usr -DCMAKE_VERBOSE_
MAKEFILE:BOOL=TRUE
> make
> sudo make install

** <YOUR_DIR> : Your work directory -- e.g.) kaldi_build
** <COMMIT_ID> : Git commit ID of Kaldi version to build -- e.g.) 4571f47f84
    Please read the KaldiDecoder's README to check
    which Kaldi commit ID it was based on...
** <CORES> : How many cores do you have -- e.g.) 4
** <DISTRO> : Ubuntu distribution -- e.g.) xenial
** <HARK_VER> : HARK version -- e.g.) 2.5.0-openblas
```

Since it is installed in /usr/local/bin by default, the “-DCMAKE_INSTALL_PREFIX” must be set as follows in order to install in /usr/bin like the package version.

```
> cd kaldidecoder3
> mkdir build
> cd build
> cmake .. -DCMAKE_INSTALL_PREFIX=/usr -DCMAKE_BUILD_TYPE=None -DOPENBLAS_ROOT_DIR:
STRING=/usr -DCMAKE_VERBOSE_MAKEFILE:BOOL=TRUE
> make
> sudo make install
```

If the output of “kaldidecoder --help” is as shown below, the KaldiDecoder installation was successful.

```
> kaldidecoder --help
usage: If you requests need use ONLINE decoding with nnet1 model.
       (ONLINE mode is default)
...

```

With the above, installation is complete.

- For the installation method on Windows OS, please refer to Section ??.

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, and the other is by using Commas alone.

See [HARKTOOL5-GUI documentation](#) for the former and [HARKTOOL5 documentation\(Japanese\)](#) for the latter.

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 and (2) the RASP series. See ?? 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

wios accepts three types of options: mode (Record, Playback and Synchronized playback and recording), device (ALSA and RASP), and general, which can be freely combined. You can run **wios** with no options to see the complete help.

For example, if you want to record 2 seconds from an ALSA device `plughw:1,0` at a sampling rate 44.1kHz and store it into `voice.wav`, the command is:

```
wios -r -x 0 -a plughw:1,0 -f 44100 -t 2 -o voice.wav
```

If you want to playback to an RASP device `192.168.33.24` and store it into `tsp.wav`, the command is:

```
wios -p -x 1 -a 192.168.33.24 -i tsp.wav
```

When playing `tsp.wav` with RASP device `192.168.33.24` and saving it to `rec.wav` (Synchronized recording and playback), it is as follows. In this case, `-a` option is the same device, it can be omitted.

```
wios -s -x 1 -d 192.168.33.24 -i tsp.wav -o rec.wav
```

When playing the `tsp.wav` using the ALSA device `plughw:1,0` and saving it to the `rec.wav` using the ALSA device `plughw:2,0` (Synchronized recording and playback) is as follows. In this case, `-d` and `-a` is another device and can not be omitted.

```
wios -s -x 0 -d plughw:1,0 -a plughw:2,0 -i tsp.wav -o rec.wav
```

When playing the `tsp.wav` using the RASP device `192.168.33.24` and saving it to the `rec.wav` using the ALSA device `plughw:2,0` (Synchronized recording and playback) is as follows. Since the types of playback device and recording device are different, `-y` and `-z` are used instead of `-x`.

```
wios -s -y 1 -d 192.168.33.24 -z 0 -a plughw:2,0 -i tsp.wav -o rec.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` or `--ad-file`. 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` or `--da-file`. 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` or `--da-file` and the file to record using `-o` or `--ad-file`. **wios** will then play the specified file and simultaneously record to another specified file.

Device options

The device is specified using two options representing the device type and the device name. Use `-x` or `--dev-type` to specify the device type. To specify the D/A device and the A/D device separately in the `--sync` mode, use `-y` or `--da-dev-type` (D/A device) and `-z` or `--ad-dev-type` (A/D device).

ALSA : The device type of ALSA is 0. That is, specify `-x 0`, `-y 0` or `-z 0`. The device name to be used is specified by `-d` or `--da-dev-name` option (D/A: when playback) or `-a` or `--ad-dev-name` option (A/D: when recording). The default device name is `plughw:0,0`.

RASP : The device type of RASP is 1. That is, specify `-x 1`, `-y 1` or `-z 1`. The IP address of the device to be used is specified by `-d` or `--da-dev-name` option (D/A: when playback) or `-a` or `--ad-dev-name` option (A/D: when recording). The default IP address is `192.168.33.24`.

In addition to these 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.