HARK Document
Version 3.0.0. (Revision: 9272)

奥乃 博
中臺 一博
高橋 徹
武田 龍
中村 圭佑
水本 武志
吉田 尚水
大塚 琢馬
柳楽 浩平
糸原 達彦
坂東 宣昭

[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

目 次

第1章　はじめに

本ドキュメントは，ロボット聴覚ソフトウエア HARK (HRI-JP Audition for Robots with Kyoto Univ., hark は listen を意味する中世英語) に関する情報の集大成である．第 1 章では，HARK の設計思想，設計方針，個々の技術の概要，HARK の応用について述べるとともに，HARK を始めとするロボット聴覚ソフトウエア，ロボット聴覚機能が切り開く新しい地平について概観する．

1.1　ロボット聴覚ソフトウエアは総合システム

人は，色々な音が聞こえる多様な環境で音を「聞き分けて」処理を行い，人とコミュニケーションを行ったり，TV，音楽，映画などを楽しんだりしている．このような聞き分ける処理を提供するロボット聴覚機能は，実環境で聞こえる多様な音を様々なレベルで処理するための機能を包含する必要があり，ロボットビジョンの機能と同様に一言で定義できない．実際，オープンソース画像処理ソフトウエア OpenCV が膨大な処理モジュールの集合体であるように，ロボット聴覚ソフトウエアも最低限必要な機能を含んだ集合体を成していることが不可欠である．

ロボット聴覚ソフトウエア HARK は『聴覚の OpenCV』を目指したシステムである．OpenCV のように「聞き分ける」ために必要なモジュールをデバイスレベルから信号処理アルゴリズム，測定ツール，GUI まで包含するだけでなく，さらに，オープンソースとして公開をしている．

音情報を基に音環境を理解する音環境理解（Computational Auditory Scene Analysis）研究での 3 つの主要課題は，音源定位 (sound source localization)，音源分離 (sound source separation)，及び，分離音声の音声認識 (automatic speech recognition) である．HARK 第 1 版は，これらの研究の成果として開発してきた．現在，研究用にはオープンソースとして無償公開 [1] を行っている．

以下，第 2 節で HARK の設計思想について述べ，HARK がミドルウエアとして利用している HARK middleware について概説する．第 3 節で HARK のモジュール群について概説する．第 4 節で今後の開発予定を述べる．

1.2　HARK の設計思想

ロボット聴覚ソフトウエア HARK の設計思想を以下にまとめる．

1. **入力から音源定位・音源分離・音声認識までの総合機能の提供**：ロボットに装備するマイクロフォンからの入力，マルチチャネル信号処理による音源定位，音源分離，雑音抑制，分離音認識にわたる総合性能の保証，
2. **ロボットの形状への対応**：ユーザの要求するマイク配置への対応と信号処理への組込，
3. **マルチチャネル A/D 装置への対応**：価格帯・機能により多様なマルチチャネル A/D 装置をサポート，

[1] https://www.hark.jp/

4. **最適な音響処理モジュールの提供と助言**：信号処理アルゴリズムはそれぞれアルゴリズムが有効な前提を置いており，同一機能に対して複数のアルゴリズムを開発し，その使用経験を通じて最適なモジュールを提供，

5. **実時間処理**：音を通じたインタラクションや挙動を行うためには不可欠である．

このような設計思想の下に，オープンソースとして HARK の公開を行ってきた．改良，機能向上，バグフィックスには， HARK Forum [2] に寄せられたユーザからの声が多く反映されている．

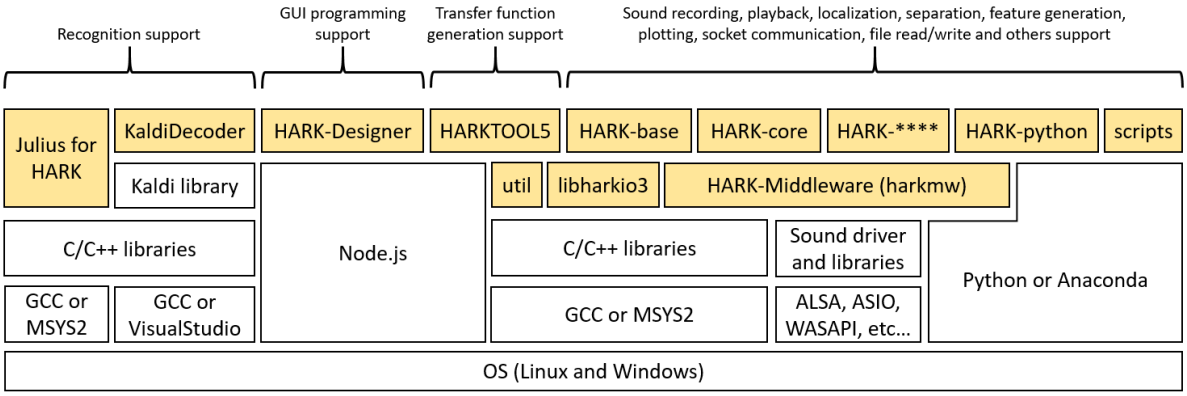


図 1.1: ロボット聴覚ソフトウエア HARK とミドルウエア FlowDesigner，OS との関係

HARK は，図 1.1 に示すように，音声認識部 (Julius, Kaldi) やサポートツールを除き，HARK middleware をミドルウエアとして用いている (なお，ver 2.5 までは， FlowDesigner [2] を利用していた).

ミドルウエア HARK Designer

ロボット聴覚では，音源定位データを基に音源分離し，分離した音声に対して音声認識を行うことが多い．各処理は，アルゴリズムが部分的に置換できるよう複数モジュールで構成する方が柔軟である．このため，効率のよいモジュール間統合が可能なミドルウェアの導入が不可欠である．しかし，統合するモジュール数が多くなると，モジュール間接続の総オーバヘッドが増大し，実時間性が損なわれる．モジュール間接続時にデータのシリアライズを必要とする CORBA (Common Object Request Broker Architecture) のような一般的な枠組みではこうした問題への対応は難しい．実際，HARK の各モジュールでは，同じ時間フレームであれば，同じ音響データを用いて処理を行う．この音響データを各モジュールがいちいちメモリコピーを行って使っていたのでは，速度的にもメモリ効率的にも不利である．

このような問題に対応できるミドルウエアとして，我々は，データフロー指向のミドルウェア HARK middleware を開発した．HARK middleware は，それまでミドルウェアとして用いていた FlowDesigner [2] の機能向上版であるが，中身はスクラッチから再実装している．HARK middleware は，従来の FlowDesigner と同様に，ROS や CORBA 等汎用的なモジュール統合のフレームワークと比較して，処理が高速で軽い．

FlowDesigner は，単一コンピュータ内の利用を前提とすることで [3]，[4]．高速・軽量なモジュール統合を実現するフレームワークであったが，HARK middleware は，複数台のコンピュータを使って分散処理を行うことも考慮した実装となっている．実装言語は，FlowDesigner は C++ のみで実装されていたのに対し，HARK

[2] https://wp.hark.jp/forums/

[3] コンピュータをまたいだ接続は，HARK における音声認識との接続部のようにネットワーク接続用のモジュールを作成することで実現可能である．

[4] FlowDesigner のオリジナルは，http://flowdesigner.sourceforge.net/ から，FlowDesigner 0.9.0 の機能向上版は，https://www.hark.jp/ からそれぞれダウンロードできる．

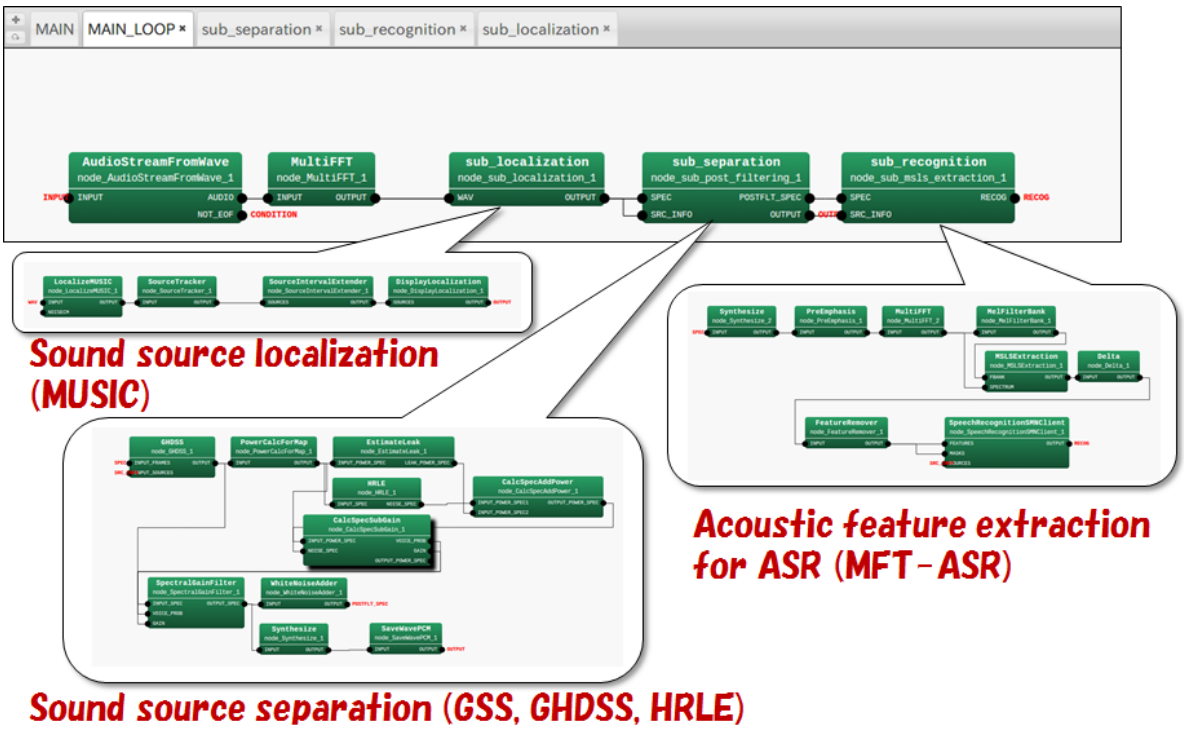


図 1.2: HARK を用いた典型的なロボット聴覚のモジュール構成

middleware は，C++ と pyhton の組み合わせとなっている．FlowDesigner と比較すると，オーバヘッドは増大しているものの，最小限に抑えるよう実装している．また，モジュールの実装については，完全に FlowDesigner とコンパチブルであり，これらのクラスは，共通のスーパークラスを継承した C++ のクラスとして実現される（HARK-Python を用いれば pyhton でモジュールを記述することも可能）．このため，モジュール間のインタフェースは自然と共通化される．モジュール間接続は，各クラスの特定メソッドの呼び出し（関数コール）で実現されるため，オーバヘッドが小さい．データは，参照渡しやポインタで受け渡されるため，前述の音響データのような場合でも，高速にかつ少ないリソースで処理できる．つまり， HARK middleware の利用によって，モジュール間のデータ通信速度とモジュール再利用性の両立が可能である．

HARK を用いた典型的なロボット聴覚に対する HARK middleware 用のネットワークを図 1.2 に示す．ファイル入力によりマルチチャネル音響信号を取得し，音源定位・音源分離を行う．得られた分離音から音響特徴量を抽出し，ミッシングフィーチャマスク (MFM) 生成を行い，これらを音声認識 (ASR) に送る．各モジュールの属性は，属性設定画面で設定することができる（図 1.3 は GHDSS の属性設定画面の例）．

HARK で現在提供している HARK モジュールと外部ツールを表 1.1 に示す．次節では，各モジュールの概要をその設計方針とともに説明をする．

入力装置

HARK では複数のマイク (マイクアレイ) をロボットの耳として搭載して処理を行う．ロボットの耳の設置例を図 4 に示す．この例では，いずれも 8 チャネルのマイクアレイを搭載しているが，HARK では，任意のチャネル数のマイクアレイが利用可能である．HARK がサポートするマルチチャネル A/D 変換装置は，下記のとおりである．

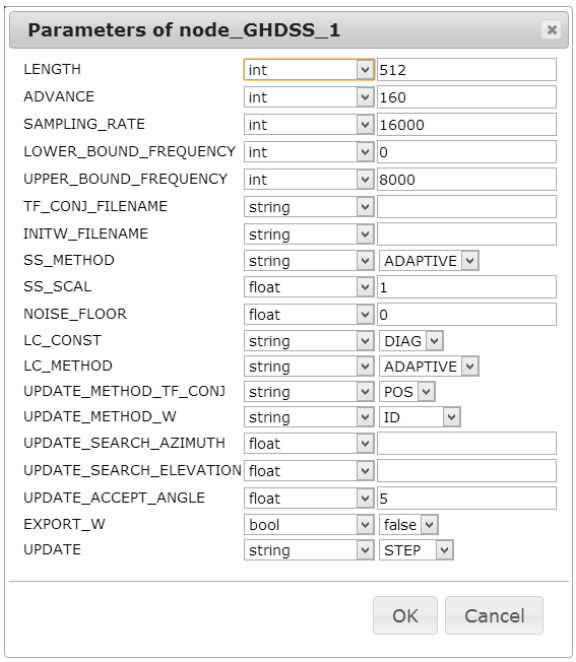


図 1.3: GHDSS の属性設定画面の例


SIG2 (whole body)


Robovie-R2


HRP-2 (8ch)

図 1.4: 3 種類のロボットの耳（マイクロフォン配置）

- システムインフロンティア社製，RASP シリーズ，
- ALSA ベースの A/D 変換装置，例えば，RME 社製 Hammerfall DSP シリーズ，Multiface AE.
- Microsoft Kinect
- Sony PS-EYE
- Dev-Audio Microcone

これらの A/D 装置は入力チャネル数が異なるが，HARK での内部パラメータを変更することで対応できる．ただし，チャネル数が増加すれば，処理速度は低下する．また，量子化ビット数は 16 ビット, 24 ビットの両方に対応している．HARK の想定するサンプリングレートは，16kHz であるので，48KHz サンプリングデータに対しては，ダウンサンプリングモジュールが用意されている．なお，東京エレクトロンデバイス社製 TD-BD-16ADUSB（USB インタフェース）は，サポートするカーネルのバージョンが古いため，HARK 1.2 からサポート対象外となっている．

マイクは，安価なピンマイクで構わないが，ゲイン不足解消のため，プリアンプがあった方がよい．RME 社製からは OctaMic II が販売されている．ヤマハ製のマイクロフォンアンプの方が，収録音のノイズが少ないようである．TD-BD-16ADUSB や RASP は，プリアンプおよび，プラグインパワー対応の電源供給機能を有しているので，使い勝手がよい．

1.3 HARK のモジュール群

音源定位

音源定位には，これまでの経験から最も性能が良かった MUltiple SIgnal Classification (MUSIC) 法を提供している．MUSIC 法は，音源位置と各マイク間のインパルス応答 (伝達関数) を用いて，音源定位を行う手法で

表 1.1: Nodes and Tools provided by HARK 1.2

機能	カテゴリ名	モジュール名	説明
音声入出力	AudioIO	AudioStreamFromMic	マイクから音を取得
		AudioStreamFromWave	ファイルから音を取得
		SaveRawPCM	音をファイルに格納
		SaveWavePCM	音を WAV 形式でファイルに格納
		HarkDataStreamSender	音をソケット通信で送信
音源定位・追跡	Localization	ConstantLocalization	固定定位値を出力
		DisplayLocalization	定位結果の表示
		LocalizeMUSIC	音源定位
		LoadSourceLocation	定位情報をファイルから取得
		NormalizeMUSIC	LocalizeMUSIC のスペクトルを正規化
		SaveSourceLocation	定位情報をファイルに格納
		SourceIntervalExtender	追跡結果を前方に延長
		SourceTracker	音源追跡
		CMLoad	相関行列ファイルの読み込み
		CMSave	相関行列ファイルの保存
		CMChannelSelector	相関行列のチャネル選択
		CMMakerFromFFT	相関行列の生成
		CMMakerFromFFTwithFlag	相関行列の生成
		CMDivideEachElement	相関行列の成分ごとの除算
		CMMultiplyEachElement	相関行列の成分ごとの乗算
		CMConjEachElement	相関行列の共役
		CMInverseMatrix	相関行列逆行列演算
		CMMultiplyMatrix	相関行列の乗算
		CMIdentityMatrix	単位相関行列の出力
音源分離	Separation	BGNEstimator	背景雑音推定
		BeamForming	音源分離
		CalcSpecSubGain	ノイズスペクトラム減算＆最適ゲイン係数推定
		CalcSpecAddPower	パワースペクトラム付加
		EstimateLeak	チャネル間リークノイズ推定
		GHDSS	GHDSS による音源分離
		HRLE	ノイズズスペクトラム推定
		PostFilter	音源分離後ポストフィルター処理
		SemiBlindICA	事前情報を用いた ICA による音源分離
		SpectralGainFilter	音声スペクトラム推定
特徴量抽出	FeatureExtraction	Delta	Δ 項計算
		FeatureRemover	項の削除
		MelFilterBank	メルフィルタバンク処理
		MFCCExtraction	MFCC 抽出
		MSLSExtraction	MSLS 抽出
		PreEmphasis	プリエンファシス
		SaveFeatures	特徴量を格納
		SaveHTKFeatures	特徴量を HTK 形式で格納
		SpectralMeanNormalization	スペクトル平均正規化
ミッシングフィーチャマスク	MFM	DeltaMask	Δ マスク項計算
		DeltaPowerMask	Δ パワーマスク項計算
		MFMGeneration	MFM 生成
ASR との通信	ASRIF	SpeechRecognitionClient	ASR に特徴量を送る
		SpeechRecognitionSMNClient	同上，特徴量 SMN 付
その他	MISC	ChannelSelector	チャネル選択
		DataLogger	データのログ出力
		HarkParamsDynReconf	ネットワーク経由の動的パラメータ設定
		MatrixToMap	`Matrix` → `Map` 変換
		MultiGain	マルチチャネルのゲイン計算
		MultiDownSampler	ダウンサンプリング
		MultiFFT	マルチチャネル FFT
		PowerCalcForMap	`Map` 入力のパワー計算
		PowerCalcForMatrix	行列入力のパワー計算
		SegmentAudioStreamByID	ID による音響ストリームセグメント選択
		SourceSelectorByDirection	方向による音源選択
		SourceSelectorByID	ID による音源選択
		MapSelectorBySource	Source による分離結果選択
		Synthesize	波形変換
		WhiteNoiseAdder	白色雑音追加

機能	カテゴリ	ツール名	説明
データ生成	外部ツール	harktool4	データ可視化・各種設定ファイル作成
		wios	伝達関数作成用録音ツール

ある．インパルス応答は，実測値もしくは，マイクロフォンの幾何的位置を用いて計算により求めることができる．

HARK 0.1.7 では，音源定位として ManyEars [3] のビームフォーマが利用可能であった．このモジュールは，2D 極座標空間 (3D 極座標空間で方向情報が認識できるという意味で「2D」となっている) で，マイクアレイから 5 m 以内，かつ，音源間が 20° 以上離れていれば，定位誤差は約 1.4° であると報告されている．しかし，ManyEars のモジュール全体がもともと 48 kHz サンプリング用に作成されており，HARK で利用している 16 kHz サンプリングと合致しないこと，マイクロフォン配置からインパルス応答をシミュレーションする時にマイクロフォンが自由空間に配置されていることが前提となっており，ロボットの身体の影響を考慮できないこと，MUSIC のような適応ビームフォーマの方が一般的なビームフォーマよりも音源定位精度が高いことなどの理由から HARK 1.0.0 では，MUSIC 法のみをサポートしている．

HARK 1.1 では，MUSIC 法における部分空間に分解するアルゴリズムを拡張した GEVD-MUSIC と GSVD-MUSIC[7] のサポートを新たに行った．本拡張により，既知の雑音（ロボットのファン雑音等）を白色化をした上で音源定位を行うことができ，ロボットの自己雑音を初めとする，大きな雑音下においてもロバストに音源定位ができるようになった．

HARK 1.2 では，さらに 3 次元音源定位を行うことができるように拡張を行った．

音源分離

音源分離には，これまでの使用経験から種々の音響環境で最も総合性能の高い Geometric-Constrained High-order Source Separation (GHDSS) [8]，及び，ポストフィルタ PostFilter とノイズ推定法 Histogram-based Recursive Level Estimation HRLE を HARK 1.0.0 では提供している．現在，最も性能がよく，様々な音環境で安定しているのは，GHDSS と HRLE の組合せである．

これまでに，適応型ビームフォーマ (遅延和型，適応型)，独立成分分析 (ICA)，Geometric Source Separation (GSS) など様々な手法を開発し，評価実験を行ってきた．HARK で提供してきた音源分離手法を下記にまとめる：

1. HARK 0.1.7 で提供した遅延和型ビームフォーマ，
2. HARK 0.1.7 で外部モジュールとしてサポートした ManyEars Geometric Source Separation (GSS) と PostFilter の組合せ [4]，
3. HARK 1.0.0 プレリリースで提供した独自設計の GSS と PostFilter の組合せ [5]，
4. HARK 1.0.0 で提供する GHDSS と HRLE の組合せ [6, 8].

HARK 0.1.7 で利用していた ManyEars の GSS は，音源からマイクへの伝達関数を幾何制約として使用し，与えられた音源方向から到来する信号の分離を行う手法である．幾何学的制約は，音源から各マイクへの伝達関数として与えらると仮定し，マイク位置と音源位置との関係から伝達関数を求めている．本伝達関数の求め方ではマイク配置が同じでもロボットの形状が変わると伝達関数が変わるという状況においては，性能劣化の原因となっていた．

HARK 1.0.0 プレリリースでは，GSS を新たに設計し直し，実測の伝達関数を幾何学的制約として使用できるように拡張し，ステップサイズを適応的に変化させて分離行列の収束を早める等の改良を行った．さらに，GSS の属性設定変更により，遅延和型ビームフォーマが構成できようにもなった．このため，HARK 0.1.7 で提供されていた遅延和型ビームフォーマ DSBeamformer は廃止された．

音源分離一般に当てはまるのだが，音源分離手法の大部分は，ICA を除き，分離すべき音源の方向情報をパラメータとして必要とする．もし，定位情報が得られない場合には，分離そのものが実行されないことにな

る．一方，ロボット定常雑音は，方向性音源としての性質が比較的強いので，音源定位ができれば，定常雑音を除去することができる．しかし，実際にはそのような雑音に対する音源定位がうまく行かないことが少なからずあり，その結果，定常雑音の分離性能が劣化する場合があった．HARK 1.0.0 プレリリースの GSS および GHDSS には，特定方向に常に雑音源を指定する機能が追加され，定位されない音源でも常に分離し続けることが可能となっている．

一般に，GSS や GHDSS のような線形処理に基づいた音源分離では分離性能に限界があるので，分離音の音質向上のためにポストフィルタという非線形処理が不可欠である．ManyEars のポストフィルタを新たに設計し直し，パラメータ数を大幅に減らしたポストフィルタを HARK 1.0.0 プレリリース版および確定版で提供している．

ポストフィルタは，上手に使えばよく切れる包丁ではあるが，その使い方が難しく，下手な使い方をすれば逆効果になる．ポストフィルタの設定すべきパラメータ数は，PostFilter においても少なからずあるので，それらの値を適切に設定するのが難しい．さらに，ポストフィルタは確率モデルに基づいた非線形処理を行っているので，分離音には非線形スペクトラム歪が生じ，分離音に対する音声認識率の性能がなかなか向上しない．

HARK 1.0.0 では，HRLE (Histogram-based Recursive Level Estimation) という GHDSS に適した定常ノイズ推定法を提供している．GHDSS 分離アルゴリズムを精査して開発したチャンネル間リークエネルギーを推定する EstimateLeak と HRLE とを組み合わせて使用すると，従来よりも音質の向上した分離音が得られる．

MFT-ASR: MFT に基づく音声認識

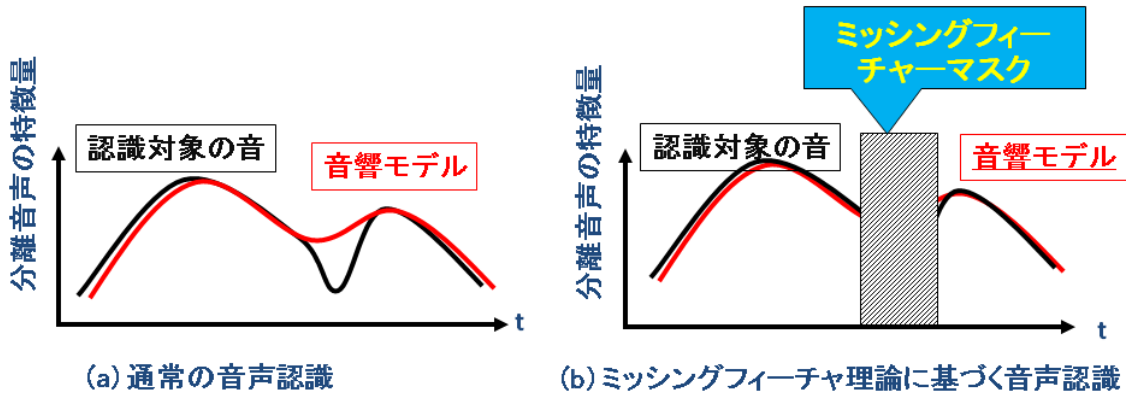


図 1.5: ミッシングフィーチャ理論による音声認識の概念図

混合音や分離など様々な要因によって引き起こされるスペクトル歪は，従来の音声認識コミュニティで想定されている以上のものであり，それに対処するためには，音源分離と音声認識とをより密に結合する必要がある．HARK では，ミッシングフィーチャ理論 (Missing Feature Theory，MFT) に基づいた音声認識 (MFT-ASR) により対処をしている [4]．

MFT-ASR の概念を図 1.5 に示す．図中の黒い線は分離音の音響特徴量の時間変化を，赤い線は ASR システムが保持する対応する発話の音響モデルの時間変化を示す．分離音の音響特徴量は歪によりシステムのそれと大きく異なっている箇所がある (図 1.5(a))．MFT-ASR では，歪んでいる箇所をミッシングフィーチャマスク (MFM) でマスクすることにより，歪みの影響を無視する (図 1.5(b))．MFM とは，分離音の音響特徴量に対応する時間信頼度マップであり，通常は 2 値のバイナリーマスク (ハードマスクとも呼ばれる) が使用される．0〜1 の連続値をとるマスクはソフトマスクと呼ばれる．HARK では，MFM はポストフィルタから得られる定常雑音とチャネル間リークのエネルギーから求めている．

MFT-ASR は，一般的な音声認識と同様に隠れマルコフモデル (Hidden Markov Model，HMM) に基づいているが，MFM が利用できるよう HMM から計算する音響スコア (主に出力確率計算) に関する部分に変更を加えている．HARK では，東京工業大学古井研究室で開発されたマルチバンド Julius を MFT-ASR と解釈し直して使用している [13].

HARK 1.0.0 では，Julius 4 系のプラグイン機能を利用し，MFT-ASR の主要部分は Julius プラグインとして提供している．プラグインとして提供したことで，Julius のバージョンアップによる新しい機能を，そのまま利用できる．また，MFT-ASR は FlowDesigner から独立したサーバ/デーモンとして動き，HARK の音声認識クライアントからソケット通信で送信された音響特徴量とその MFM に対し，結果を出力する．

音響特徴量抽出と音響モデルの雑音適用

スペクトル歪を特定の音響特徴量だけに閉じ込めて，MFT の有効性を高めるために，音響特徴量には，メルスケール対数スペクトル特徴量 (Mel Scale Log Spectrum，MSLS) [4] を使用している．HARK では，音声認識で一般的に使用されるメル周波数ケプストラム係数 (Mel-Frequency Cepstrum Coefficient，MFCC) も提供しているが，MFCC では，歪がすべての特徴に拡散するので，MFT との相性が悪い．同時発話が少ない場合には，MFCC を用いて音声認識を行う方が認識性能がよい場合もある．

HARK 1.0.0 では，MSLS 特徴量で，新たに Δ パワー項を利用するためのモジュールを提供する [6]．Δ パワー項は，MFCC 特徴量でもその有効性が報告されている．各 13 次元の MSLS と Δ MSLS，及び，Δ パワーという 27 次元 MSLS 特徴量を使用した方が，HARK 0.1.7 で使用していた MSLS，Δ MSLS 各 24 次元の計 48 次元 MSLS 特徴量よりも性能がよいことを確認している．

HARK では，上述の非線形分離による歪の影響を，少量の白色雑音を付加することで緩和している．クリーン音声と白色雑音を付加した音声とを使ったマルチコンディション学習により音響モデルを構築するとともに，認識音声にも分離後に同量の白色雑音を付加してから 音声認識を行う．これにより，一話者発話では，S/N が -3 dB 程度でも，高精度な認識が可能である [6].

1.4 HARK の応用

我々は，これまでに２本のマイクロフォンを使用した両耳聴によるロボット聴覚機能を開発し，３話者同時発話認識を一種のベンチマークとして使用してきた．SIG や SIG2 という上半身ヒューマノイドロボット上でのロボット聴覚では，1m 離れた所から 30 度間隔に立つ３話者の同時発話認識がそれなりの精度で認識が可能となった [16]．しかし，このシステムは事前知識量や事前処理量が多く，どのような音環境でも手軽に使えるロボット聴覚として機能を備えるのは難しいと判断せざるを得なかった．この性能限界を突破するために，マイクロフォンの本数を増やしたロボット聴覚の研究開発を開始し，HARK が開発されたわけである．

したがって，HARK がベンチマークとして使用してきた 3 人が同時に料理の注文をするのを聞き分けるシステムに応用するのは必然であった．現在，Robovie-R2，HRP-2 等のロボット上で動いている．３話者同時発話認識の変形として，３人が口で行うじゃんけんの勝者判定を行う審判ロボットも Robovie-R2 上で開発を行った [17].

また，ロボットの応用ではないが，実時間で取得したデータ，あるいは，アーカイブされたデータに対して，HARK が定位・分離した音を可視化するシステムを開発してきた．音の提示において，多くの環境で正確な「音に気づかない」状況がしばしば見受けられる．この問題を，**聴覚的アウエアネス** (音の気づき) の欠如によるものと捉え，聴覚的アウエアネスを改善するために，音環境理解の支援を行う 3 次元音環境可視化システムを設計し，HARK を用いて実装を行った [18, 19].

1.4.1　3 話者同時発話認識

a) Robovie が注文をたずねる．　b) 3 人が同時に料理の注文を行う．c) 1.9 秒後に Robovie が注文を反復し，合計金額を答える．

図 1.6: 3 人が料理を同時に注文するのを聞き分ける Robovie-R2

3 話者同時発話認識は，マイクロフォン入力，音源定位，音源分離，ミッシングフィーチャマスク生成，および，自動音声認識の一連の処理により，話者それぞれの発話認識結果を返す．この FlowDesigner でのモジュールネットワークは図 1.2 に示したものである．対話管理モジュールは，

1. ユーザの発話を聞き，注文依頼だと判定すると，次の処理を行う．
2. ロボット聴覚の一連の処理 – 音源定位・音源分離・ポストフィルタ処理・音響特徴量の抽出・ミッシングフィーチャマスク生成 – を行う．
3. 発話人数分の 音響特徴量とミッシングフィーチャマスクを音声認識エンジンに送り，音声認識結果を受け取る．
4. 音声認識結果を分析し，料理の注文である場合には，注文を復唱し，料理の金額の合計額を答える．
5. さらに注文を受け付ける．

音声認識での音響モデルは，不特定話者対象としている．言語モデルは文脈自由文法で記述しているので，文法を工夫すれば,「ラーメン　大盛り」や「ラーメン　ピリ辛　大盛り」,「ラーメン　ライス大盛り」なども可能である．

3 人の実話者全員が話し終えてから認識終了までに従来のファイル経由ベースの処理では，約 7.9 秒を要していたが，HARK の使用により，応答が約 1.9 秒に短縮された [5]．応答が速いため，全員の注文終了後，直ちにロボットがそれぞれの注文を復唱し，合計金額を答えるように感じられる．なお，モジュールの設定にも依存するが，ファイル入力の場合には，発話終了時が明確であるので，発話終了から認識を終え，ロボットが応答を始めるまでの遅延時間は 0.4 秒程度である．

また，復唱の時に，ロボットが発話者の方へ顔を振り向けることも可能である．HRP-2 では挙動付きの応答を行っている．ただし，身振り手振りを入れるとその準備のためにどうしても応答が遅れ，間の抜けた挙動となってしまうので，注意が必要である．

1.4.2　口ジャンケンの審判

3 話者が同時に料理を注文するのは，デモとして不自然であるとのご意見があったので，同時発話が不可欠なゲームを対象とした．ジャンケンを言葉で行う「口ジャンケン」である.「口ジャンケン」の面白さは，相手に顔を見せずにジャンケンができたり，暗闇でもジャンケンができることにあるものの，問題を誰が勝ったの

[5] デモは http://winnie.kuis.kyoto-u.ac.jp/SIG/

かがすぐに分からないことである．ロボット聴覚機能のついたロボットに，口ジャンケンの審判をさせようと言うわけである [17].

口ジャンケン審判のプログラムは，前述の 3 話者同時発話認識と対話戦略のところだけが異なっている．ジャンケンが正しく発話されたか，つまり，後出しをしたプレーヤはいないか，をチェックしてから，誰が勝ったのか，あるいは，勝負がアイコだったのか，の判定を行い，結果を知らせる．もし，勝負がつかない場合には，再度ジャンケンを行うようにプレーヤに指示をする．(ニュースサイエンティスト誌の記事を参照)

本システムの詳細は，ICRA-2008 の論文 [17] に書かれているので，興味のある方はそちらを参照していただきたい．

1.4.3 CASA 3D Visualizer

一般に，音声は，時間的・場所的空間を共有する人間同士のコミュニケーションメディアとして，根源的な役割を果たしており，我々は様々な環境で音声を通じて情報のやり取りを行っている．しかし，いろいろな音を聴き逃していることも多く，また，録音を高忠実に再生しても，そのような聞き逃しを回避することは難しい．これは，人生のすべてを記録しようというライフログで，音の再生上大きな問題となろう．このような問題の原因の 1 つは，録音からは音の気づき (アウエアネス) が得られない，すなわち聴覚的アウエアネスの欠如であると考えられる．

高忠実再生技術は，聴覚的アウエアネスを現実世界以上に改善するわけではない．現実世界で聞き分けられないものが，高忠実再生になったから解決できるとは考えられない．実際，心理物理学の観点から人は 2 つ以上の音を同時に認識することは難しい [20] とされており，複数話者など同時に複数の音が発生する時には，音を聞き分けて提示する等の施策が不可欠である．

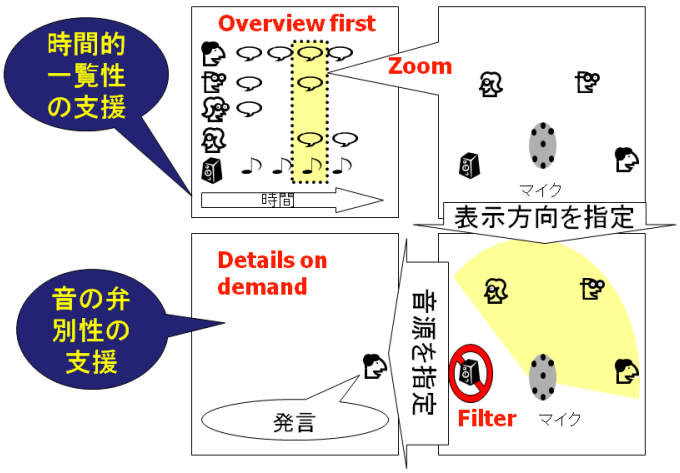


図 1.7: CASA 3D Visualizer: Visual Information-Seeking Matra “Overview first，zoom and filter, then details on demand” に従った HARK 出力の可視化

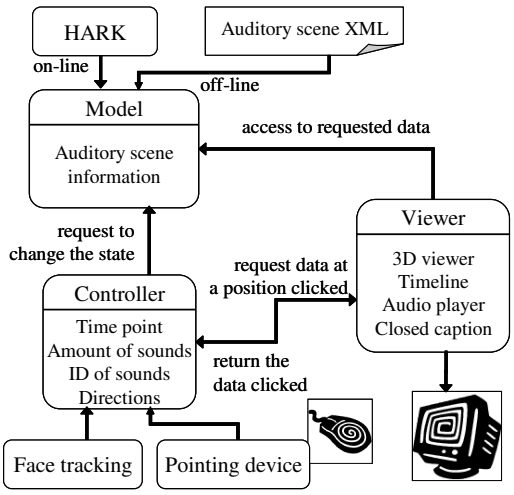


図 1.8: CASA 3D Visualizer の MVC (Model-View-Control) モデルを使用した実装法

我々は，**聴覚的アウエアネス** (音の気づき) の改善にするために，HARK を応用して，音環境理解の支援を行う 3 次元音環境可視化システムを設計し，実装を行った [18, 19]． GUI には Schneiderman が提唱した情報視覚化の指針 “overview first，zoom and filter，then details on demand” (図 1.7) を音情報提示に解釈し直し，以下のような機能を設計した．

1. Overview first: まず概観を見せる．

2. Zoom: ある特定の時間帯を詳しく見せる．

3. Filter: ある方向の音だけを抽出して，聞かせる．

4. Details on Demand: 特定の音だけ聞かせる．

このような GUI により，従来音情報を取り扱う上での課題であった時間的一覧性の支援と音の弁別性の支援の解決を図った．また，実装に関しては，Model-View-Control (MVC) モデルに基づいた設計 (図 1.8) をした．HARK から得られる情報は，まず AuditoryScene XML に変換される．次に，AuditoryScene XML 表現に対して，3D 可視化システムが表示を行う．

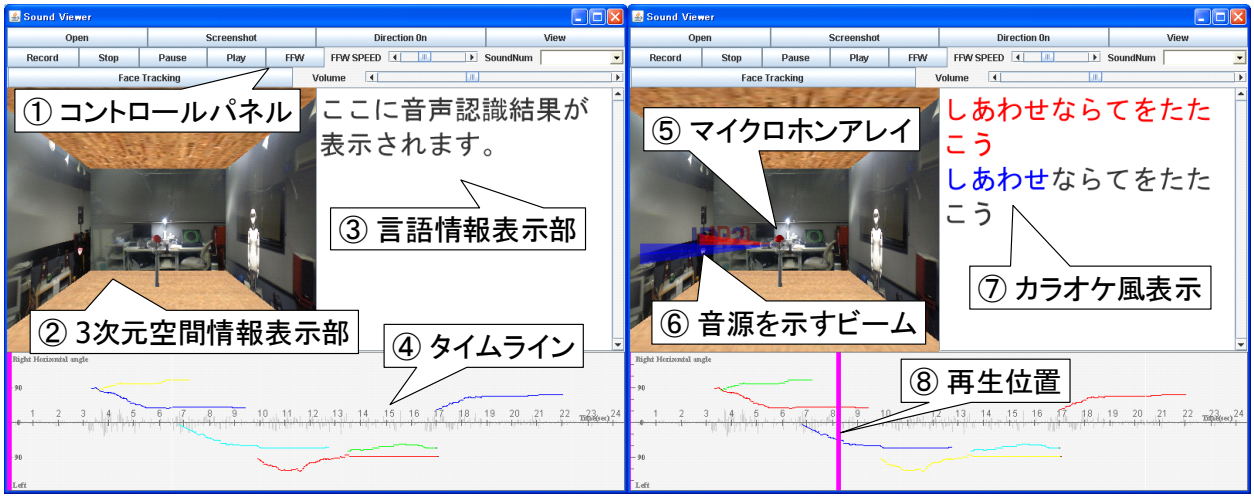


図 1.9: CASA 3D Visualizer の GUI

図 1.9 に表示画面を示す．3 次元空間情報表示では，拡大・縮小，回転が行える．音の再生時には，音源方向を示すビームが ID とともに表示される．また，矢印の大きさは音量の大きさに対応している．言語情報表示部には，音声認識結果が表示される．音声の再生時には対応する字幕がカラオケ風に表示される．タイムラインには，音源の定位の変化の overview 情報が表示され，音の再生時には，再生位置が表示される．表示と音響データとは対応付けが行われているので，ビームあるいはタイムラインの音源をマウスでクリックすると，対応する分離音が再生される．また，再生については早送りモードも提供されている．このように，音情報を見せることにより，聴覚的アウエアネスの改善を試みた．

HARK 出力の可視化のさらなる応用として次のようなシステムも試作されている．

1. ユーザの顔の動きに従って，GUI の表示や音の再生を変更 [18],

2. Visualizer の結果をヘッドマウントディスプレィ (HMD) に表示 [21].

上記で説明した GUI は，3D 音環境を鳥瞰する外部観察者のモードである．それに対して，1 番目の応用は，3D 音環境の満真中にいる没入モードの提供である．この 2 つの表示法は，Google Map のアナロジーをとると，鳥瞰モードと street view モードに相当する．没入モードでは，顔を近づけると音量が大きくなり，顔を遠ざけるとすべての音が聞こえてくる．また，顔を上下左右に移動すると，そちらから聞こえる音が聞こえてくる，等の機能が提供されている．

2 番目の応用は，CASA 3D Visualizer を HMD に表示することで，音源方向を実時間で表示するとともに，その下部には，字幕を表示している．字幕の作成は音声認識ではなく，iptalk という字幕作成用ソフトウエアを使用している．聴覚障害者が字幕を頼りに講義を受ける場合，視線は字幕と黒板の板書をいったりきたりす

ることになる．これは，非常に負担が大きい上に，話が進んでいることに気がつかずに重要なことを見逃したりする場合が少なからず生ずる．本システムを利用すると，ディスプレイに音源の方向が表示されるので，話題の切り替えへの聴覚的アウエアネスが補強されると期待される．

1.4.4 テレプレゼンスロボットへの応用

2010 年の 3 月に，米国 Willow Garage 社のテレプレゼンスロボット Texai に，HARK と音環境を可視化するシステムを移植し，遠隔ユーザが音源方向をカメラ映像に表示し，特定方向の音源の音だけを聞く機能を実現した [6]．テレプレゼンスロボットでの音情報提示の設計は，前節で説明をした「聴覚的アウエアネスがキーテクノロジである」というこれまでの経験に基づいている．

図 1.10: Texai (中央) を通じて，remote operator が 2 人の話者と， 1 台の Texai とインタラクションを行う．なお，場所はカリフォルニア州であるが， 左側の Texai はインディアナ州から遠隔操作中.

具体的な HARK の移植と Texai への HARK 関連モジュールの開発は次の 2 工程に分けられる．

1. Texai へのマイクロフォン搭載，インパルス応答の測定及び HARK の移植,
2. Texai 制御プログラムが走る ROS (Robot Operating System) への HARK インタフェースとモジュールの実装.

図 1.11 に最初に設置したマイクロフォンの設置状況を示す．このロボットを使用する講義室と大食堂に置き，それぞれ 5 度間隔でインパルス応答を測定し，音源定位の性能を測定した．次に，見栄え，さらには，マイクロフォン間のクロストークを減少させるために Texai に頭を付けることを検討した．具体的には，雑貨店で見つけた竹製のサラダボールである．最初に付けたものとほぼ同じ直径になる辺りに MEMS マイクロフォンを設置した (図 1.11)．同様にインパルス応答を測定し，音源定位性能について評価を行った．その結果，両者の性能はそれほど変わらないことが判明した．

GUI については，Visual Information-seeking matra の，overview と filter を実装した．図 1.13 に示した Texai 自身の斜め下の全方位の画像の中央から出ている矢印が，話者の音源方向である．矢印の長さは音量を表している．図中では 3 名の話者がしゃべっていることが分かる．Texai のもう 1 つのカメラの画像が右下に，リモートオペレータの画像が左下に示されている．図中の円弧は，filter で通過させる範囲を示す．この円弧内にある方位から届いた音は，リモートオペレータに送られる．データは図 1.14 に示したように The Internet を通じて行われる．

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

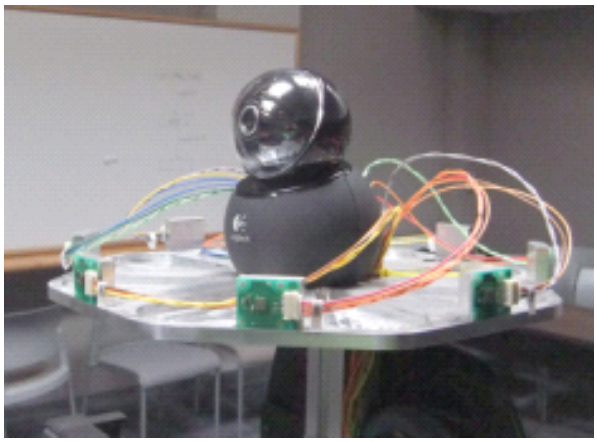


図 1.11: Texai の最初の頭部の拡大: 8 個の MEMS マイクロフォンを円盤上に設置

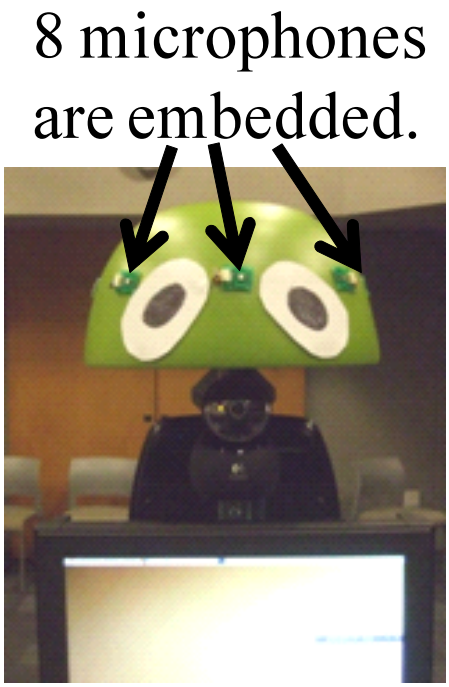


図 1.12: Texai の頭部の拡大: 8 個の MEMS マイクロフォンを円周状に設置

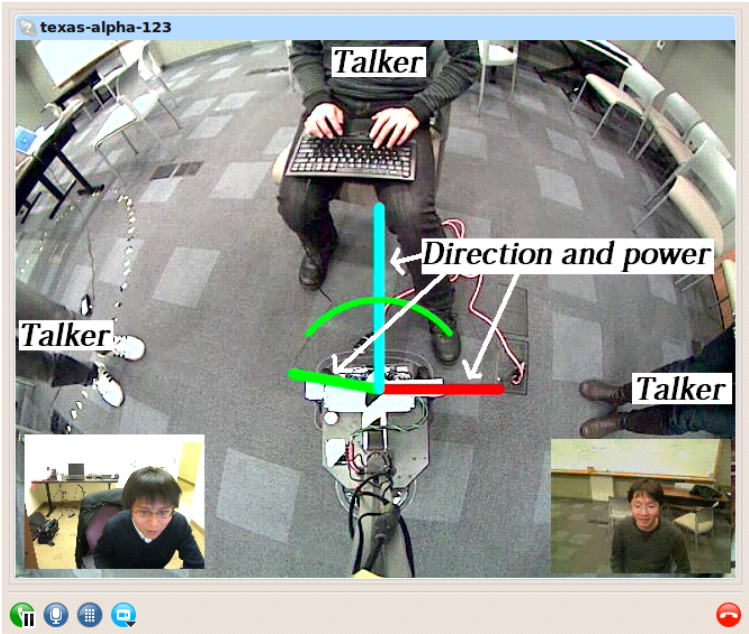


図 1.13: Texai を通じて，remote operator に見える画面

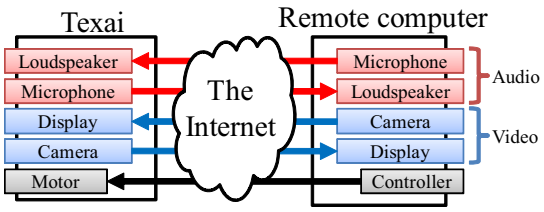


図 1.14: Texai の Teleoperation の方法

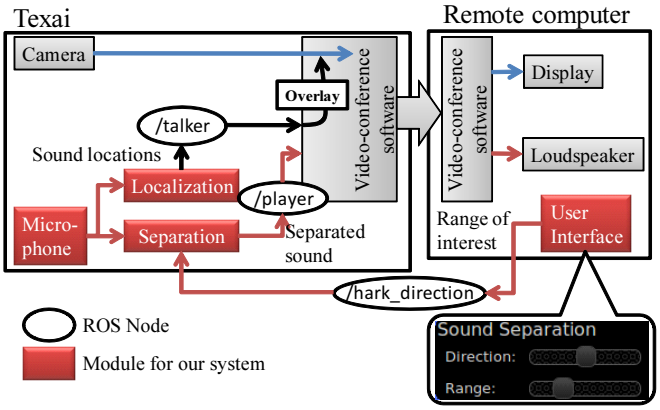


図 1.15: Texai への HARK の組込方法

GUI と，リモートオペレータ用の操作コマンド群はすべて ROS モジュールとして実装されるので，図 1.15 に示した方法で HARK を組み込むようにした．図中の茶色が HARK システムである．ここで開発したモジュールは，ROS の Web サイトから入手可能である．

これら一連の作業は頭部の加工，インパルス応答の測定，予備実験，GUI と操作コマンド群の設計を含めて１週間で終了できた．HARK や ROS の高いモジュール性が，生産性向上に寄与したと考えられる．

1.5 まとめ

以上，HARK 1.0.0 の概要を報告した．ミドルウエア FlowDesigner を使って，音環境理解の基本機能である音源定位，音源分離，分離音認識をモジュールとして実現し，ロボットの耳への応用について概説した．

HARK 1.0.0 は，ロボット聴覚研究をさらに展開するための機能を提供している．例えば，移動音源処理に向けた機能，音源分離の各種パラメータの詳細設定機能，設定データ可視化・作成ツールなどである．また，Windows のサポート，OpenRTM へのインタフェースなども進行中である．

HARK は，ダウンロードし，インストールするだけでもある程度の認識は可能であるものの，個々のロボットの形状や使用環境に合わせたチューニングを行えば，さらに音源定位，音源分離，分離音認識の性能が向上する．このようなノウハウの顕在化には，HARK コミュニティの形成が重要である．本稿がロボット聴覚研究開発者のクリティカルマスを超えるきっかけとなれば幸いである．

第2章　ロボット聴覚とその課題

本章では，HARK の開発のきっかけとなったロボット聴覚研究，およびその課題について述べる．

2.1　ロボット聴覚は聞き分ける技術がベース

鉄腕アトム大事典（沖光正著，晶文社）によると鉄腕アトムには「スイッチひとつで聴力が千倍になり，遠くの人の声もよく聞こえ，さらに 2 千万ヘルツの超音波も聞きとる」サウンドロケータが装備されているという [1]．サウンドロケータは，1953 年に Cherry が発見した選択的に音声を聞き分ける「カクテルパーティ効果」を実現するスーパーデバイスなのであろう．

聴覚障害者や耳の聞こえが悪くなった高齢者からは「スーパーデバイスでなくても，常時同時発話が聞き分けられる機能じゃだめなの」という素朴な疑問がわく．日本書紀推古紀には,「一聞十人訴，以勿失能辨」とあり，同時に 10 人の訴えを聞き分けて裁いたという「聖徳太子」の逸話が紹介されている．動物や草木の言葉が聞こえるという「聞き耳頭巾」の昔話は子供たちの想像力をかき立てる．このような聞き分け機能をロボットに持たせることができれば，人との共生が大きく前進すると期待される．(日本書紀推古紀によれば,「一聞十人訴以勿失能辨兼知未然」豊聡耳厩戸皇子)

日常生活で最も重要なコミュニケーション手段が話声や歌声などを含めた音声であることは論を俟たない．音声コミュニケーションは，言葉獲得，非音声によるバックチャネルなどを包含し，その機能は極めて多彩である．実際，自動音声認識 (ASR，Automatic Speech Recognition) 研究の重要性は高く認識され，過去 20 年以上に渡り膨大な資金と労力が投入された．一方，ロボット自身に装着されたマイクロフォンで音を聞き分け，音声認識をするシステムの研究は麻生らの仕事を除き，ほとんど取り組まれてこなかった．

筆者らの研究スタンスは，事前知識最小の音の処理方式を開発することであった．そのために，音声だけでなく，音楽，環境音，さらにはそれらの混合音の処理を通じて音環境を分析理解する音環境理解の研究が重要であると考えた．この立場から，単一音声入力を仮定する現行の ASR がロボット学で重要な役割を果たせ切れていないことの説明が付く．

2.2　音環境理解をベースにしたロボット聴覚

音声に加えて音楽や環境音さらには混合音を含めた音一般を扱う必要があるという立場から，音環境理解 (Computational Auditory Scene Analysis) [9] 研究を進めてきた．音環境理解研究での重要な課題は，混合音の処理である．話者の口元に設置した接話型マイクロフォンを使用して混合音の問題を回避するのではなく，入力は混合音との立場から，混合音処理に直球で立ち向うのが音環境理解である．

音環境理解の主たる課題は，音源方向認識の**音源定位** (sound source localization)，**音源分離** (sound source separation)，**分離音の音声認識** (automatic speech recognition) の 3 つである．個々の課題に対してはこれまでに多種多様な技術が研究開発されている．しかし，いずれの技術もその能力を最大限発揮するためには何らかの条件を前提としている．ロボット聴覚でこれらの技術を組合せ，能力を最大限発揮させるためには，個別技術のインタフェース，すなわち，前提条件をうまく揃えて，システム化することが不可欠である．このためには，

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

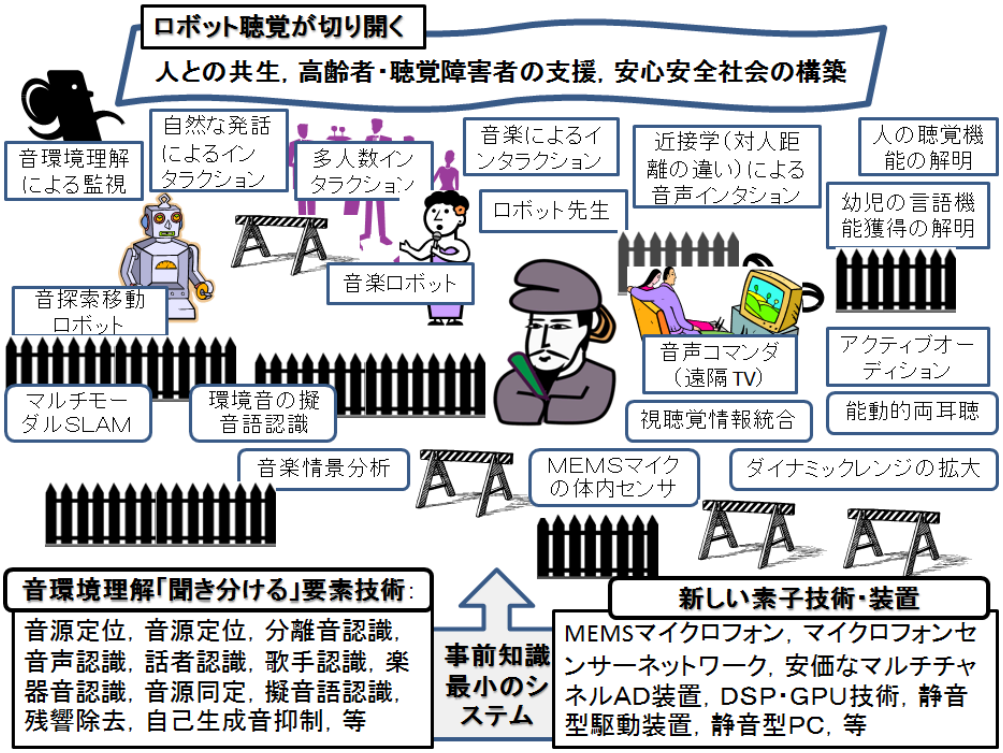


図 2.1: 音環境理解をベースとしたロボット聴覚の展開

ドベネックの桶 (リービッヒの最小律) ではないが，バランスの良い組合せを効率よく提供できるミドルウエアも重要となる．

ロボット聴覚ソフトウエア **HARK** は，FlowDesigner というミドルウエアの上に構築されており，8 本のマイクロフォンを前提として，音環境理解の機能を提供している．HARK は，事前知識を極力減らすという原則で設計されおり，“音響処理の OpenCV” を目指したシステムである．実際，3 人の料理の注文を聞き分けるロボットや口によるじゃんけんの審判ロボットなどが複数のロボットで実現されている．

一般には画像や映像が主たる環境センサとなっているものの，見え隠れや暗い場所には対応できず，必ずしも万能というわけではない．音情報を使って，画像や映像での曖昧性を解消し，逆に，音響情報での曖昧性を画像情報を使って解消する必要がある．例えば，2 本のマイクロフォンによる音源定位では，音源が前か後ろかの判断は極めて難しい．

2.3　人のように 2 本のマイクロフォンで聞き分ける

人や哺乳類は 2 つの耳で聞き分けを行っている．ただし，頭を固定した実験では高々2 音しか聞き分けれないことが報告されている．人の音源定位機能のモデルとしては，両耳入力に遅延フィルタをかけて和を取る Jeffress モデルと，両耳間相互相関関数によるモデルがよく知られている．中臺と筆者らは，ステレオビジョンにヒントを得て，調波構造を両耳で抽出し，同じ基本周波数の音に対して，両耳間位相差と両耳間強度差を求めて，音源定位を行っている [11, 12]．一対の参照点を求めるのに，ステレオビジョンではエピポーラ幾何を使用し，我々の方法は調波構造を使用する．

2 本のマイクロフォンによる混合音からの音源定位では，定位が安定せず大きくぶれることが少なからずあり，また，前後問題，とくに，真正面と真後ろにある音源を区別するのが難しい．中臺らは視聴覚情報統合に

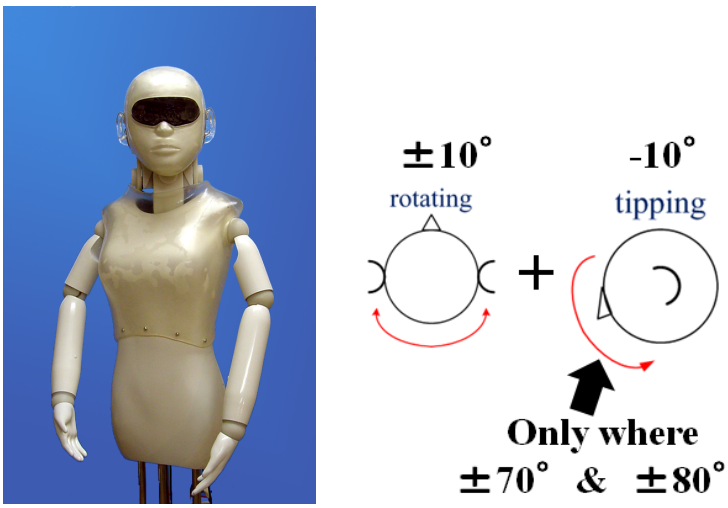


図 2.2: SIG2 のアクティブオーディション：周辺部の音に対しては首を左右と下に動かして前後問題の曖昧性を解消する．

より安定した音源定位を実現するとともに，SIG というロボットで呼びかけられたら振り向くロボットを実現している [14, 15, 27]．前後問題の曖昧性解消は百聞一見に如かず，というわけである．

金と奥乃らは，SIG2 というロボットに頭を動かすことにより音源定位の曖昧性の解消するシステムを実現している．単純に頭を左右に 10 度動かすだけでなく，音源が 70 度〜80 度にある時には，下向きに 10 度頷きを入れるとよい．実際，正面の音源同定では 97.6%と 1.1%の性能向上に過ぎないのに対して，後ろの音源同定では 75.6%と 10%大幅に性能が向上する (図 2.2)．これは， Blauert が “Spatial Hearing” で報告している人の前後問題の解消時の頭の動きとよく一致している．曖昧性の解消のために挙動を用いる方法は**アクティブオーディション**の 1 形態である．

公文のグループや中島のグループは，様々な耳介を用いて頭や耳介自身を動かすことで音源定位の性能向上に取り組んでいる [12]．ちょうど，ウサギの耳が通常は垂れ下がって広範囲な音を聞いており，異常音がすると耳が立ちあがり，特定方向の音を聞くために指向性を高める．このようなアクティブオーディションの実現法の基礎研究である．これが，ロボットだけでなく，様々な動物の聴覚機能の構成的解明に応用できると，新たなロボットの耳の設計開発につながっていくと期待される．とくに，両耳聴は，ステレオ入力装置がそのまま使えるので，高性能の両耳聴機能が実現できると，工学的な貢献が大きいと考えられる．

2.4 自己生成音抑制機能

アクティブオーディションでは，モータが動くことにより発生するモータ自身の音に加えてロボット自身の体の軋みから音が発生することがある．ロボットの動きに伴って発生する音は，小さい音であっても音源がマイクロフォンの近くにあるので，逆 2 乗則から外部の音源と比較して相対的に大きな音となる．

モデルベースによる自己生成音抑制

中臺らはロボット SIG の頭部内部にマイクロフォンを 2 本設置し，自己生成音の抑制を試みている．モータ音や機械音について簡単なテンプレートを持ち，モータの稼働中でテンプレートに合うような音が発生すると，ヒューリスティクスを用いて破壊されやすいサブバンドを破棄する．本手法を用いた理由は，FIR フィルタに基づくアクティブノイズキャンセラでは，左右の耳が別々に処理されるので両耳間位相差を正しく求めること

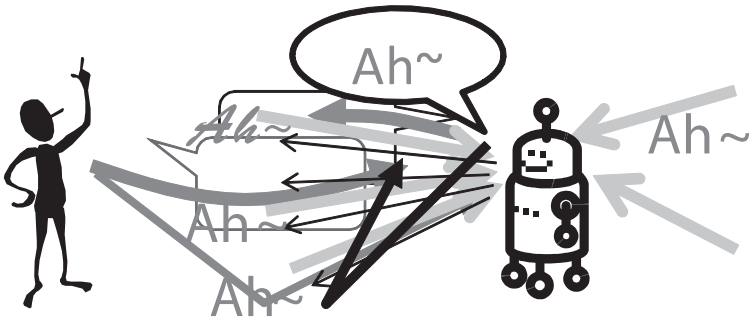


図 2.3: 自分の話声が残響を伴って自分の耳に入り，さらに，相手の割り込み発話 (バージイン) も聞こえる

ができないからであり，さらに，バースト性雑音の抑制に FIR フィルタがあまり効果がなかったからである．なお，SIG2 では，マイクロフォンが人の外耳道モデルに埋め込まれており，モータも静音型かなので，雑音抑制処理は行っていない．ソニーの QRIO でも体内に 1 本マイクロフォンを設置し，外部を向いた 6 本のマイクロフォンを使用して自分の出す雑音を抑制している．

Ince らは，自分の動きから生じる自己生成雑音を，関節角の情報から予測し，スペクトルサブトラクション法により削減する方法を開発している [12]．中臺らは，特定の方向からのモータ雑音を棄却する機能を HARK に組み込んでいる [12]．Even らは，体内に設置した 3 個の振動センサを使って，体表から放射される音の方向を推定し，その放射音方向と話者方向が一致しないように線形マイクロフォンアレイの角度を調節し，自己生成音の抑制を行っている [12]．

ロボットが人とインタラクションを取るときには，自己生成音の影響，環境による音への影響を勘案して，最もよく聞こえる位置に移動したり，体の向きを変えるといった「よりよく聞くための戦略」の開発が不可欠である．

セミブラインド分離による自己生成音抑制機能

ロボット聴覚では，自己発話信号がロボット自身に既知である点を活用した自己生成音抑制が可能である．武田らは，図 2.3 に示した状況において，自己発話を既知として，その残響成分を推定し，入力混合音から自己発話を抑制し，相手の発話を抽出する自己生成音抑制機能を独立成分分析 (ICA) に基づいたセミブラインド分離技術より開発している [12]．本技術の応用のプロトタイプとしてバージイン許容発話認識と音楽ロボット（後述）が開発されている．

バージイン許容発話とは，ロボットの発話中でも人が自由に発話ができる機能である．ロボットが項目を列挙して情報提供を行っているときに，ユーザが割り込んで「それ」「2 番目の」「アトム」と発話すると，本技術を応用して，発話内容や発話タイミングからどの項目が指定されたか従来よりは高性能で判定することができる．人とロボットが共生していくためには，交互に話すのではなく，いついかなる時でもお互いに自由に話すことができる混合主導型のインタラクションが不可欠であり，本自己生成音抑制機能によってそのような機能が容易に実現できる．

セミブラインド分離技術は，自己生成音が耳まで入るが，分離されると捨てられ，高次処理の対象となっていない．本庄の『言葉をきく脳しゃべる脳』によると，成人では自分の声が側頭葉の一次聴覚野までは入るが，大脳皮質の連合聴覚野には送られず，聞き流していることが観測されている．上述のセミブラインド分離による自己生成音抑制は一次聴覚野止まりの処理の工学的実現ととらえることもできよう．

2.5　視聴覚情報統合による曖昧性解消

ロボット聴覚は**要素技術ではなく，プロセスであり，複数のシステムから構成される**．構成部品となる要素技術は多数あり，しかも，構成部品の性能にはばらつきがあるので，プロセスではすべてがうまくかみ合って機能する必要がある．しかも，このかみ合わせがしっかりするほど，プロセスはうまく機能する．音響処理だけでは曖昧性が解消できないので，視聴覚情報統合がかみ合わせの重要な鍵となる．

情報統合のレベルには，時間的，空間的，メディア間，システム間があり，さらに，各レベル内でも，レベル間でも階層的な情報統合が必要である．中臺らは次のような視聴覚情報統合を提案している．最下位レベルでは音声信号と唇の動きから話者を検出する．その上のレベルでは，音素 (phoneme) 認識 と口形素 (viseme) 認識とを統合する．その上位レベルは，話者位置と顔の 3D 位置との統合である．最上位は，話者同定・検証と顔同定・検証との統合である．もちろん，同一レベルの情報統合だけでなく，ボトムアップ処理やトップダウン処理の相互作用が考えられる．

一般に混合音処理は不良設定問題であり，より完全な解を得るためには，何らかの前提，例えばスパースネスの仮定が必要となる．時間領域でのスパースネス，周波数領域でのスパースネス，3D 空間でのスパースネス，さらには特徴空間でのスパースネスなどが考えられる．情報統合の成否は，スパースネスの設計だけでなく，個々の要素技術の性能にも依存することに注意する必要がある．

2.6　ロボット聴覚が切り開くキラーアプリケーション

ロボット聴覚機能が充実しても，それは，個々の信号処理モジュールの統合であり，それからどのような応用が見えてくるのかは明らかでない．実際，音声認識は IT 事業の中でも非常に低い地位しか与えられていない．そのような現状から，本当に不可欠な応用を見つけるためには，まず，使えるシステムを構築し，経験を積んでいく必要があろう．

近接学によるインタラクション

インタラクションの基本原理として，対人距離に基づく近接学 (Proxemics) が知られている．すなわち，親密距離 (〜0.5 m)，個人距離 (0.5 m〜1.2 m)，社会距離 (1.2 m〜3.6 m)，公共距離 (3.6 m〜) に分け，各距離ごとにインタラクションの質が変っている．

近接学に対するロボット聴覚の課題は，マイクロフォンのダイナミックレンジが拡大することである．複数人インタラクションにおいて，個々の話者が同じ音量で話すとすると，遠方の話者の声は逆 2 乗則に従って小さくなる．従来の 16 ビット入力では不足し，24 ビット入力で対応することが不可欠である．システム全体を 24 ビット化するのは，計算資源や既存ソフトウェアとの整合性から難しい．荒井らは，情報欠損の少ない 16 ビットへのダウンサンプリング法を提案している [12]．また，マルチチャネル A/D 装置や携帯電話用 MEMS マイクロフォンなど，新しい装置の出現にも対応していく必要もある．

音楽ロボット

音楽を聴けば自然と体が動き，インタラクションが円滑になるので，音楽インタラクションへの期待は大きい．ロボットが音楽を扱えるようになるには,「聞き分ける」機能が不可欠である．テストベッドとして開発した音楽ロボット処理の流れを示す．

1. 自己生成音を入力音（混合音）から抑制あるいは分離,
2. 分離音のビート追跡からテンポ認識と次テンポ推定,

3. テンポに合わせて挙動（歌を歌う，動作）を実行.

ロボットは，スピーカから音楽が鳴るとすぐにテンポに合わせて足踏みを始め，音楽がなり終わると足踏みを終える.

自分の歌声を残響の影響を含めて入力混合音から分離するのために自己生成音抑制機能を使用している．ビート追跡やテンポ推定では誤りが避けられない．音楽ロボットでは，テンポ推定誤りから生ずる楽譜追跡時の迷子からいかに早く，かつ，スマートに合奏や合唱に復帰するかが重要であり，人とのインタラクションで不可欠な機能となっている.

視聴覚統合型 SLAM

佐々木・加賀美（産総研）らは，32 チャネルマイクロフォンアレイを装着した移動ロボットを開発し，室内の音環境理解の研究開発に取り組んでいる．事前に与えられたマップを使い，いくつかのランドマークをたどりながら定位とマップ作成を同時に行う SLAM (Simultaneous Localization And Mapping) の音響版である [1]. 従来の SLAM では，画像センサ，レーザレンジセンサ，超音波センサなどが使われるものの，マイクロフォン，つまり，可聴帯域の音響信号は使用されてこなかった．佐々木らの仕事は，従来の SLAM では扱えていなかった音響信号を SLAM に組み込む研究であり，重要な先駆的な研究である．これにより，見えないけれども音がする場合にも，SLAM あるいは音源探索が可能となり，真の情景理解 (Scene anaysis) や環境理解への道筋が開かれたことになると考えられる.

2.7 まとめ

ロボットが自分自身の耳で聞くという**ロボット聴覚研究** の筆者の考え方を述べるとともに，今後の展開への期待を述べた．ロボット聴覚研究は，ほとんど 0 からの立ち上げであったために，自分たちの研究だけでなく，当該研究の振興を図るべく浅野 (産総研，以下敬称略)，小林 (早大)，猿渡 (奈良先端大) らのアカデミア，NEC，日立，東芝，HRI-JP などのロボット聴覚を展開する企業，さらには，カナダ Sherbrooke 大学，韓国 KIST，フランス LAAS，ドイツ HRI-EU などの海外研究機関からの協力を得て，IEEE/RSJ IROS でこれまでに 6 年間ロボット聴覚 organized session を組み，ロボット学会学術講演会でも 5 年間特別セッションを組んでいる．さらに，2009 年には IEEE 信号処理部門の国際会議 ICASSP-2009 でロボット聴覚スペシャルセッションを開催した．このような研究コミュニティの育成により，世界的に徐々に研究者が増加し，その中でも日本のロボット聴覚研究のレベルの高さが輝いている．今後斯学の益々の発展を通じ，聖徳太子ロボットが聴覚障害者や高齢者の支援，安心できる社会の構築に寄与していくことを期待したい.

六十而耳順 (「論語・為政」)

60 にして耳に順う，というが，聴覚器官は加齢あるいは酷使されると高域周波数の感度が落ち，人の話が聞こえなくなり，耳に順いたくとも，順えなくなる.

関連図書

[1] 中臺，光永，奥乃 (編): ロボット聴覚特集，日本ロボット学会誌，Vol.28，No.1 (2010 年 1 月).

[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] 奥乃，中臺: ロボット聴覚オープンソフトウエア HARK, 日本ロボット学会誌，Vol.28，No.1 (2010 年 1 月) 6–9，日本ロボット学会.

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

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

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

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

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

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

[12] 人工知能学会 AI チャレンジ研究会資料．Web より入手可能: http://winnie.kuis.kyoto-u.ac.jp/AI-Challenge/

[13] 西村 義隆，篠崎 隆宏，岩野 公司，古井 貞熙: 周波数帯域ごとの重みつき尤度を用いた音声認識の検討，日本音響学会 2004 年春季研究発表会講演論文集，日本音響学会，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] 徳田 浩一，駒谷 和範，尾形 哲也，奥乃 博: 音源定位結果と音声認識結果を HMD に統合呈示する聴覚障害者向け音環境理解支援システム，情報処理学会第 70 回全国大会，5ZD-7，Mar. 2008.

[22] 奥乃 博，中臺 一博: ロボット聴覚の課題と現状，情報処理，Vol.44，No.11 (2003) pp.1138–1144，情報処理学会.

[23] 奥乃 博，溝口 博: ロボット聴覚のための情報統合の現状と課題，計測と制御，Vol.46，No.6 (2007) pp.415–419，計測自動制御学会.

[24] 奥乃 博，山本 俊一: 音環境理解コンピューティング，人工知能学会誌，Vol.22，No.6 (2007) pp.846–854，人工知能学会.

[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, 人工知能学会論文誌，Vol.20，No.3 (Mar. 2005) pp.209–219，人工知能学会.

[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.

第3章　はじめての HARK

この章では，はじめて HARK を使う人を対象に，ソフトウェアの入手方法，インストール方法について述べ，基本的な操作について説明する．

3.1　ソフトウェアの入手方法

HARK の Web サイト (https://www.hark.jp/) に入手方法が解説されている。Windows 版はこの URL からインストーラをダウンロードできる。Linux 版はこの URL にある手順に従ってリポジトリを登録し、インストールを行う。リポジトリを登録すればソースコードのダウンロードもできる。

はじめてインストールする人は，パッケージファイルをダウンロードして，インストールする方法を強く推奨する．　ソースコードをダウンロードして，インストールする方法は，上級者向けであり，本ドキュメントでは扱わない．

3.2　ソフトウェアのインストール方法

3.2.1　Linux 版のインストール方法

本項の説明で，インストール完了までの説明に，すべて作業例を示す．行頭の > はコマンドプロンプトを表す．作業例の太字の部分は，ユーザの入力を，イタリック部分は，システムからのメッセージを表す．例えば，

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

という作業例で，1 行目の先頭 > は，コマンドプロンプトを表している．作業環境によってプロンプトの表示が異なるので，各自の環境に合せて読み換える必要がある．1 行目のプロンプト以降の太字部分は，ユーザが実際に入力する部分である．ここでは， echo Hello World! の 17 文字（スペースを含む）がユーザの入力部分である．行末では，Enter キーを入力する．2 行目の斜字体部分は，システムの出力である． 1 行目の行末で Enter キーの入力後，表示される部分である．

ユーザの入力やシステムからのメッセージの一部には，バージョン番号やリリース番号が含まれている．そのため，実際にインストールするバージョンやリリースに応じて，読み換えて作業を進める必要がある．また，具体的な作業例で表示されるシステムからのメッセージは，オプションでインストール可能なライブラリの有無により，異なる．メッセージの内容が完全に一致しなくてもエラーメッセージが表示されない限り，作業を進めてよい．

For Ubuntu

Ubuntu 使用者は，パッケージからのインストールを利用できる．パッケージの配布サーバを設定ファイルに加えた後に，パッケージのインストールを行う．リポジトリへの追加方法は HARK installation instructions のページを参照。
次にパッケージのインストールを行う． 端末で以下のコマンドを実行する．

```
> 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
```

その他の環境を使っている場合は、ソースコードからコンパイルする必要がある。ここでは扱わないので、次のページ HARK installation instructions でソースコードを入手し、コンパイルを行う。

3.2.2 Windows 版のインストール方法

まず最初に,HARK に必要なソフトウェアのインストールを行う．必要なソフトウェアは HARK installation instructions を参照. Windows ではインストーラーによりインストールを行う．このインストーラーでは，次のソフトウェアがインストールされる。

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

HARK Web ページからダウンロードしたインストーラを実行する．インストーラーが起動すると，HARK のインストールが開始される．最初にライセンス文が表示されるので，それを読み,「使用許諾契約の条項に同意します (A)」を選択すると「次へ (N)」ボタンが表示されるのでクリックして次へ進む. セットアップタイプの選択画面が表示されるのですべての項目をインストールする場合は「完全 (C)」をクリックする. インストールしたい項目が限られている場合は「カスタム (S)」をクリックする.「カスタム (S)」を選択した場合，カスタムセットアップ画面が表示されるのでインストールしない項目についてはチェックを外し,「インストール (I)」をクリックする.

図 3.1: 使用許諾契約画面

3.2.3 Windows 版のアンインストール方法

Windows 版のアンインストールは以下のいずれかの方法で行う.

1. 「スタート」→「HARK」→「アンインストール」
2. 「コントロールパネル」→「プログラムのアンインストール」
3. インストーラを実行

3.3 HARK Designer

HARK バージョン 1.9.9 からは、従来システム構築の GUI として使用されていた FlowDesigner に変わって、Web ブラウザから使用できる GUI, **HARK Designer** が新たに導入された。使用方法は基本的には FlowDeisnger と同じになるように設計されている。HARK Designer の使用方法については別ドキュメントを参照。

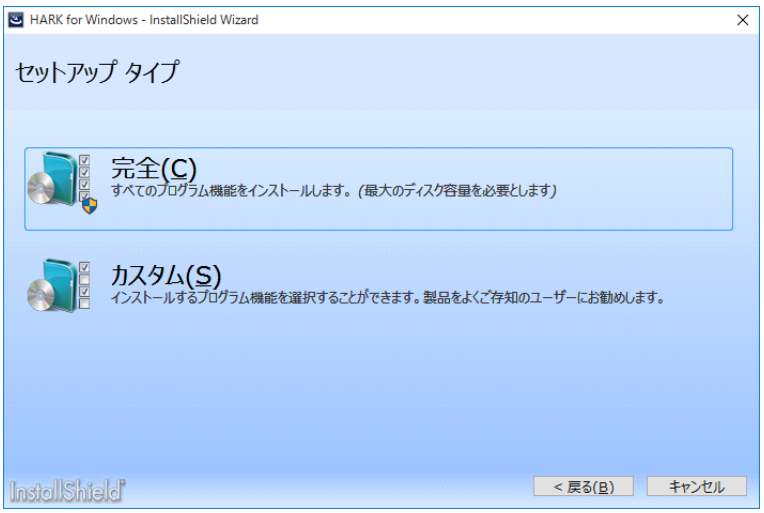


図 3.2: インストールタイプの選択

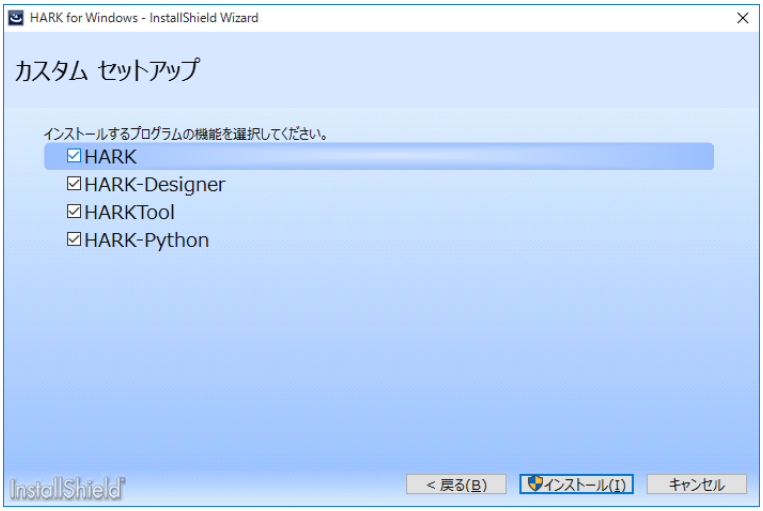


図 3.3: インストールモジュールの選択

3.3.1 Linux 版

図 3.4 に HARK Designer の概観を示す．以下のコマンドで HARK Designer を起動できる．

```
> hark_designer
```

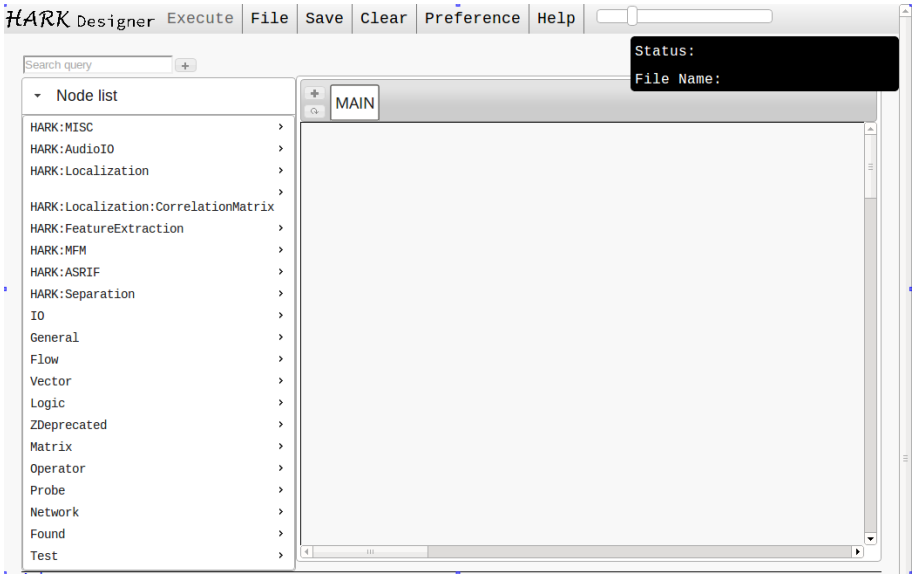


図 3.4: HARK Designer の概観

3.3.2 Windows 版

Windows 版の HARK Designer ではデスクトップ上のアイコン，または [スタート] → [プログラム] → [HARK] から HARK Designer を起動する．

図 3.5: HARK Designer アイコン

第4章　データ型

本章では，HARK のノード群で使用するデータ型について述べる．HARK でデータ型を意識する必要があるのは，以下の 2 つのケースである．

- ノードのプロパティ設定
- ノード同士の接続（ノード間通信）

ノードのプロパティ設定で用いるデータ型

ノードのプロパティとして現状で指定できるデータ型は，以下の 5 種類である．

型	意味	データ型レベル
int	整数型	基本型
float	単精度浮動小数点型	基本型
string	文字列型	基本型
bool	論理型	基本型
Object	オブジェクト型	HARK 固有型
subnet_param	サブネットパラメータ型	HARK 固有型

int , float , string , bool , については，C++ の基本データ型をそのまま利用しているので，仕様は C++ に準じる．Object ， subnet_param については，HARK 固有のデータ型である．Object は, HARK 内部の Object 型を継承しているクラスとして定義されるデータ型の総称となっている．HARK では，プロパティとして指定できる Object は，Vector もしくは Matrix であるが，後述のように基本型以外は Object 型を継承しているため，Vector や Matrix のようにテキスト形式での記述が実装されていればプロパティとして指定することができる．基本型であっても，Object 型を継承したオブジェクトを利用することで (例えば <Int 1 > など)， Object として指定することも可能である．subnet_param は，複数のノード間で一つのパラメータをラベルを用いて共有する際に用いられる特殊なデータ型である．

ノード同士の接続の際に用いるデータ型

ノードの接続（ノード間通信）は，異なるノードのターミナル（ノードの左右に黒点として表示される）を HARK Designer の GUI 上で線で結ぶことによって，実現される．この際に用いられるデータ型は，以下の通りである．

型	意味	データ型レベル
any	Any 型	HARK 固有型
int	整数型	基本型
float	単精度浮動小数点実数型	基本型
double	倍精度浮動小数点実数型	基本型
complex<float>	単精度浮動小数点複素数型	基本型
complex<double>	倍精度浮動小数点複素数型	基本型
char	文字型	基本型
string	文字列型	基本型
bool	論理型	基本型
Vector	配列型	HARK オブジェクト型
Matrix	行列型	HARK オブジェクト型
Int	整数型	HARK オブジェクト型
Float	単精度浮動小数点実数型	HARK オブジェクト型
String	文字列型	HARK オブジェクト型
Complex	複素数型	HARK オブジェクト型
TrueObject	論理型（真）	HARK オブジェクト型
FalseObject	論理型（偽）	HARK オブジェクト型
nilObject	オブジェクト型 (nil)	HARK オブジェクト型
ObjectRef	オブジェクト参照型	HARK 固有型
Map	マップ型	HARK 固有型
Source	音源情報型	HARK 固有型

any はあらゆるデータ型を含む抽象的なデータ型であり，HARK 固有で定義されている．int , float , double , complex<float> , complex<double> , char , string , bool は，C++ の基本データ型を利用している．これらの仕様は対応する C++のデータ型の仕様に準ずる．基本型を，Object のコンテキストで使おうとすると自動的に GenericType<T> に変換され，Int , Float のように，Object を継承した先頭が大文字になったクラスとして扱うことができる．ただし，String , Complex は，GenericType ではなく，それぞれ std:string, std:complex に対するデファインとして定義されているが，同様に string , complex を Object 型として使う際に用いられる．このように基本型に対して，HARK の Object を継承する形で定義されているデータ型を HARK オブジェクト型と呼ぶものとする．TrueObject , FalseObject , nilObject もそれぞれ，true, false, nil に対応する Object として定義されている．HARK オブジェクト型で最もよく使われるものは，Vector , Matrix , Map であろう．これらは，C++ の STL を継承した HARK オブジェクト型であり，基本的には C++ の STL の対応するデータ型の仕様に準ずる．

ObjectRef は，オブジェクト型へのスマートポインタとして実現されている HARK 固有のデータ型であり，Vector , Matrix , Map の要素として用いられることが多い．

Source は，音源情報型として定義される HARK 固有のデータ型である．

ノードのターミナルの型　HARK Designer で，ノードのターミナル同士は，データ型が同じである，もしくは受け側のターミナルが，送り側のターミナルのデータ型を包含していれば，正常に接続することができ，黒線で表示される．この条件を満たさないターミナル同士を接続しようとすると、エラーになり接続できない．

以降の節では，上述について基本型，HARK オブジェクト型，HARK 固有型に分けて説明を行う．

4.1 基本型

int，float，double，bool，char，string，complex (complex<float>，complex<double>) は，前述のように C++ データ型を引き継いだ基本型である．HARK では，番号など，必ず整数であると分かっているもの (音源数や FFT の窓長など) は int が，それ以外の値（角度など）にはすべて float が用いられる．フラグなど，真偽の 2 値のみが必要な場合は bool が用いられる．ファイル名などの文字列が必要な場合は string が使われる．HARK はスペクトル単位の処理や特定長の時間ブロック（フレーム）ごとの処理を行うことが多いため，ノードのターミナルのデータ型としては，直接基本型を用いる場合は少なく，Matrix や Vector，Map の要素として用いることが普通である．complex<float> も，単独で用いることは少なく，スペクトルを表現するために，Vector，Matrix の要素として用いることが多い．倍精度の浮動小数点 (double 型) は，HARK では，Source を除いて利用していない．

4.2 HARK オブジェクト型 (FlowDesigner 互換)

Int , Float , String , Complex はそれぞれ，int , float , string , complex の Object 型である．TrueObject , FalseObject は bool 型の true, false に対応する Object であり，nilObject は，nil に対応する Object である．これらの説明は省略する．C++ の標準テンプレートライブラリ (STL) を継承する形で HARK 内で Object 型として再定義されているものとして，Vector , Matrix が挙げられる．これらに関して以下で説明する．

4.2.1 Vector

データの配列を格納する型を表す．Vector には何が入っていてもよく代表的には ObjectRef を要素にもつ Vector< Obj > ，値 (int ， float) を要素にもつ Vector< int > ，Vector< float > などが使われる．

複数の値を組にして用いるので，ConstantLocalization での角度の組の指定や，LocalizeMUSIC の出力である定位結果の組を表すのに用いられる．

以下に，Vector 型の定義を示す．BaseVector は，HARK 用のメソッドを実装した型である．下に示すように，Vector 型は STL の vector 型を継承している．

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

Conversion カテゴリにある ToVect ノードは，int ， float などの入力をとり，入力された値を 1 つだけ要素に持つ Vector を出力する．

また，ノードのパラメータとして Vector を使いたい時は，パラメータのタイプを Object にし，以下のようにテキストで入力することができる．

例えば，二つの要素 3，4 を持つ int 型の Vector をパラメータに入力したい時は，図 4.1 に示すように文字列を入力すればよい．ただし，文字列は初めの文字 < の次に，スペースを開けずに Vector を書く必要があるなどの注意が必要である．

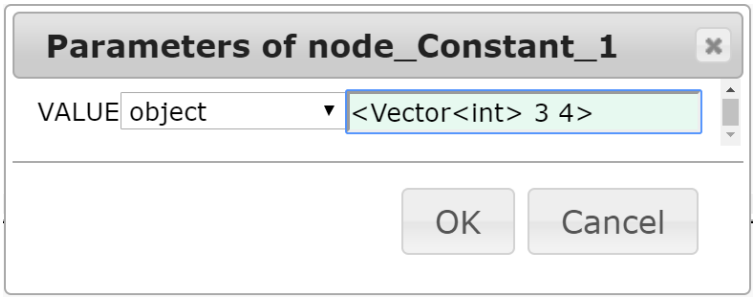


図 4.1: Vector の入力例

4.2.2 Matrix

行列を表す．代表的な型は Matrix<complex<float> > 型と Matrix<float> 型である．それぞれ，複素数を要素に持つ行列と，実数を要素に持つ行列である．

Matrix 型は，以下のように定義されている．ただし，Base Matrix は HARK 用のメソッドを実装したクラスである．

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

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

`Matrix` をノード間通信に用いるノードとしては，MultiFFT (周波数解析)，LocalizeMUSIC (音源定位) などが挙げられるなお，HARK を用いた典型的な音源定位・追従・分離と音声認識の機能を有するロボット聴覚システムでは，音源追従 (SourceTracker) で，音源に ID が付与されるので，それ以前の処理では，ノード間通信に `Matrix` が用いられ，それ以降の処理では，`Map` が用いられることが多い．

4.3 HARK 固有型 (FlowDesigner 互換)

HARK 固有型には，any , ObjectRef , Object , subnet_param が挙げられる．

4.3.1 any

any はあらゆるデータ型の総称となっており，ノードのターミナルが any 型である場合は，他のノードのターミナルがどんな型であっても警告なしに（黒線で）接続することができる．ただし，実際に通信が可能かどうかはノード内部の実装に左右されてしまうため，自ら実装を行う際には極力利用しないことが望ましい．

HARK では，MultiFFT , DataLogger , SaveRawPCM , MatrixToMap といった汎用的に用いるノードに使用を限定している．

4.3.2 ObjectRef

HARK 内で定義されている Object 型を継承するデータ型への参照を表すデータ型である．

具体的には，Object 型へのスマートポインタとして定義されている．HARK オブジェクト型，HARK 固有型 HARK 固有型はいずれも Object を継承したデータ型であるので，これらのデータ型はどれでも参照することができる．基本データ型も，前述のように，ObjectRef に代入しようとした際に最終的に NetCType<T> で変換され，Object のサブクラスとなるため，利用可能である．

4.3.3 Object

主にノードのプロパティで用いられるデータ型である．HARK では，ChannelSelector などで用いられている．基本的には，事前に用意されている int , float , string , bool , subnet_param 以外のデータ型をプロパティとして設定する際に用いるデータ型である．4.3.2 節で述べたように，基本データ型を含めて Object 型として利用可能なため，原理的には，すべてのデータ型を Object として指定できることになるが，実際に入力できるのはテキストでの入出力が実装されているデータ型に限られる．Vector や, Matrix も指定できるように実装されているが，Map はテキスト入出力を実装していないため，Object として入力することは現時点ではできない．

入力のフォーマットについては以下の表 4.1, 4.2, 4.3 を参照してください．

表 4.1: Primitive `Object` Sample

型	値	入力	備考
Char	a	<Char a> <Char a >	通常 Char 型は使用しない
`Int`	1	<Int 1> <Int 1 >	`int` 型入力 = 1
`Float`	1.0	<Float 1.0> <Float 1.0 >	`float` 型入力 = 1.0
Double	1.0	<Double 1.0> <Double 1.0 >	通常 Double 型は使用しない
`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) >	通常 Complex<double> 型は使用しない
`Bool`	false true	<Bool 0> <Bool 0 > <Bool 1> <Bool 1 >	`bool` 型入力 = false `bool` 型入力 = true
`String`	”Hello World!” ”Hello World! ”	<String Hello World!> <String Hello World! >	`string` 型入力 = ”Hello World!” スペースは無視しない
`nilObject`		<NilObject >	

表 4.2: Vector `Object` Sample

型	値	入力	備考
`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> 入力不要
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> >	

表 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> 入力不要
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

ノードのプロパティで用いられるデータ型である．subnet の複数のノードに同じパラメータをプロパティとして設定する際に，subnet_param を指定して，共通のラベルを記述すれば，MAIN(subnet) 上で，このラベルの値を書き換えることによって，該当の箇所すべての値を修正することができる．

例えば，Iterator ネットワークを作成して (名前を LOOP0 とする）その上に LocalizeMUSIC , GHDSS といったサンプリング周波数を指定する必要のあるノードを配置した際に，これらのプロパティである SAMPLING_RATE の型を subnet_param とし，“SAMPLINGRATE” というラベルを指定しておけば，MAIN(subnet) 上に仮想ネットワーク LOOP0 を配置すると，そのプロパティに SAMPLINGRATE が現れる．このプロパティを int 型にして 16000 を記入しておけば，LocalizeMUSIC , GHDSS の SAMPLING_RATE は常に同一であることが保証できる．

また，別の使い方として，MAIN(subnet) 上のノードのプロパティを subnet_param 型し，名前を ARGx (x は引数の番号) にすると，そのパラメータをバッチ実行の際に引数として指定することができるようになる（例えば，名前を ARG1 として，subnet_param を指定すると，バッチ実行の際の第一引数として利用できる）．

4.4 HARK 固有型

HARK が独自に定義しているデータ型は，Map 型と Source 型である．

4.4.1 Map

Map 型は，キーと ObjectRef 型をセットにしたデータ型である．ObjectRef は，Matrix，Vector，Source といった Object を継承するデータ型へのポインタを指定する．HARK では，音声認識機能も提供しているため，発話単位で処理を行うことが多い．この際に，発話単位の処理を実現するため，発話 ID（音源 ID）をキーとした Map<int, ObjectRef> を用いている．例えば，GHDSS（音源分離）の出力は，Map<int, ObjectRef> となっており，発話 ID がキーであり，ObjectRef には，分離した発話のスペクトルを表す Vector< complex > へのポインタが格納されている．

こうしたノードと，通常，音源追従処理より前に使う Matrix ベースで通信を行うノードを接続するために，MatrixToMap が用意されている．

4.4.2 Source

音源定位情報を表す型であり，HARK では，LocalizeMUSIC（出力），SourceTracker（入出力），GHDSS（入力）という音源定位から音源分離に至る一連の流れの中で Map<int, ObjectRef> の ObjectRef が指し示す情報として用いられる．

Source 型は，次のような情報を持っている．

1. **ID :** int 型．音源の ID
2. **パワー:** float 型．定位された方向のパワー．
3. **座標:** float 型の長さ 3 の配列．音源定位方向に対応する，単位球上の直交座標．
4. **継続時間:** double 型．定位された音源が終了するまでのフレーム数，対応する音源が検出されなければ時間とともに減っていき，この値が 0 になった場合，その音源は消滅する．この変数は，SourceTracker でのみ使用される内部変数である．
5. **TF インデックス:** int 型. 定位された方向が伝達関数ファイル中の何番目に該当するのかを表す。

Problem

MFCCExtraction や SpeechRecognitionClient などのノードの入出力に使われているデータ型「Map < ·, · >」について知りたいときに読む．

Solution

Map 型は，キーとそのキーに対応するデータの組からなる型である．例えば 3 話者同時認識を行う場合，音声認識に用いる特徴量は話者毎に区別する必要がある．そのため，特徴量がどの話者の何番目の発話に対応するのかを表した ID をキーとし，そのキーとデータをセットにして扱うことで話者・発話を区別する．

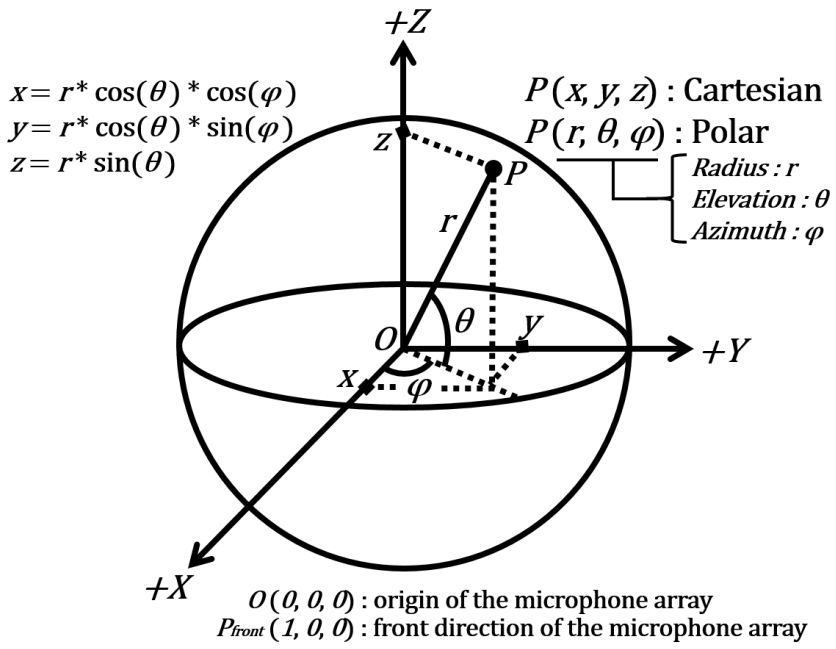


図 4.2: HARK 標準座標系

4.5 HARK 標準座標系

HARK で用いる座標は，指定した原点を中心とし（通常はマイクロホンアレイの中心），x 軸の正方向が正面，y 軸の正方向が左，z 軸の正方向が上になるようにしている．単位はメートルで記述する．また，角度は正面を 0[deg] として反時計回りとすることを前提にしている．(例えば，左方向が 90[deg]．) また，仰角は，その座標の方向ベクトルが XY 平面となす角として定義する．

第5章　ファイルフォーマット

本節では，HARK で利用するファイルの種類およびその形式について述べる．従来の HARK では多様なファイルフォーマットが使用されており、全体像の把握が困難であった。特にバイナリ形式の伝達関数ファイルはフォーマットが複雑で解析が困難であった。そこで、 HARK 2.1 より、 従来の種類の多いファイルフォーマットをよりシンプルな形式に整理した。設計方針は次の 2 点である。

1. できるだけ標準に使用されているフォーマットを使い、独自形式は減らす。
2. ファイル入出力 API を提供するライブラリを充実させる。

本方針に従い、独自のファイルフォーマットは 行列を表現する Matrix バイナリのみで、その他はすべて標準的なフォーマットとそれの組み合わせとした。また、ファイル入出力をサポートするライブラリ libharkio3 を提供し、ファイルの操作を容易にした。

HARK の独自ファイルフォーマットは、以下の 3 種類である。

1. **XML:** 位置を表現するファイルに使用。拡張子は .xml
2. **Matrix バイナリ:** 行列を表現するファイルに使用。拡張子は .mat
3. **Zip:** 伝達関数など、上記のファイルから構成される複雑なファイル形式に使用。拡張子は .zip

他のファイルは上記 3 種類に統合、あるいは標準フォーマットを使用するように変更される。

HARK では、ノードの入出力やプロパティ設定でファイルを指定することができる．表 5.1 に一覧を示す。

以降では，HARK で使用する 3 種類のフォーマットについて説明する。なお、Julius 形式については Julius のファイルフォーマットに基本的に準ずる．オリジナルの Julius との違いに関しては JuliusMFT の説明を参照されたい。Raw Audio Format, PCM Wave Format については、標準フォーマットに準ずるのでその説明を参照されたい。

5.1　XML 形式

マイク位置や音源定位結果など、位置を表現する種類の情報の保存に使用するフォーマット。図 5.1 に示すサンプルのように、ルート要素が `hark_xml`、その子要素に `config`, `positions`, `neighbors`, `channels` がある。以下では、各要素について詳細に説明していく。

5.1.1　hark_xml

この要素が表現する情報

HARK の XML ファイルの起点となる。HARK で使用されるすべての XML フォーマットにはこの要素をルート要素としている。

表 5.1: ファイル入出力が関係する HARK ノード一覧

ノード名	使用箇所	ファイル種類	新ファイル形式	旧形式
SaveRawPCM	出力	Raw Audio ファイル	Raw Audio Format	同じ
SaveWavePCM	出力	Wave ファイル	PCM Wave Format	同じ
LocalizeMUSIC	プロパティ設定	音源定位伝達関数ファイル	Zip	HGTF バイナリ
SaveSourceLocation	出力	音源定位結果ファイル	XML	定位結果テキスト
LoadSourceLocation	プロパティ	音源定位結果ファイル	XML	定位結果テキスト
GHDSS	プロパティ設定	音源分離伝達関数ファイル	Zip	HGTF バイナリ
	プロパティ設定	マイクロホン位置ファイル	XML	HARK テキスト
	プロパティ設定	定常ノイズ位置ファイル	XML	HARK テキスト
	プロパティ設定	初期分離行列ファイル	Zip	HGTF バイナリ
	出力	分離行列ファイル	Zip	HGTF バイナリ
SaveFeatures	出力	特徴量ファイル	Matrix バイナリ	`float` バイナリ
SaveHTKFeatures	出力	特徴量ファイル	HTK 形式	同じ
DataLogger	出力	`Map` データファイル	XML	`Map` テキスト
CMSave	プロパティ	相関行列ファイル	Zip	相関行列テキスト
CMLoad	出力	相関行列ファイル	Zip	相関行列テキスト
JuliusMFT	起動時引数	設定ファイル	jconf (テキスト)	同じ
	設定ファイル中	音響モデル・音素リスト	julius 形式	同じ
	設定ファイル中	言語モデル・辞書	julius 形式	同じ
harktool	harktool	音源位置リストファイル	XML	srcinf テキスト
	harktool	インパルス応答ファイル	PCM Wave Format	`float` バイナリ

属性とその意味

hark_xml は属性 version をもつ。現在のバージョンは ”1.3” である。

子要素

次節以降で説明する config, positions, neighbors が子要素となる。いずれの要素もオプショナルであり、あってもなくても良い。

5.1.2 config

この要素が表現する情報

XML ファイルの一般的な属性を表現する。

属性とその意味

属性はない。

```
<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>
```

図 5.1: XML フォーマットのサンプル

子要素

下記の要素を子要素にもつ。ただしすべてオプショナルであり、あってもなくても良い。comment 以外は基本的に伝達関数ファイルで使用される。

comment ファイルの説明が入る。任意の文字列をいれてよい。

SynchronousAverage 伝達関数計測用の信号 (TSP 信号) の再生回数を表す。自然数が入る。

TSPpath 伝達関数計測用の信号 (TSP 信号) のパスを表す。文字列が入る。/home/tsp.wav

TSPOffset 伝達関数を計算する際のオフセットに使用される。自然数が入る。

PeakSearch 2 つの属性 `from` と `to` をもち、伝達関数を計算する際の直接音のピークを検索する範囲に用いられる。これらの属性は必須であり、自然数が入る。

nfft 伝達関数を計算する際に行うフーリエ変換の解析長を表す。自然数が入る。

samplingRate 収録された伝達関数計測用の信号のサンプリング周波数が入る。自然数が入り、通常は 16000 を使用する。

signalMax 収録された伝達関数計測用の信号の振幅の最大値が入る。自然数が入り、Wave ファイルが 16bit の場合は 32767 を使用する。

TSPLength 伝達関数計測用の信号 (TSP 信号) の 1 回分のサンプル数を表す。自然数が入り、通常は 16384 を使用する。

5.1.3 positions

この要素が表現する情報

XML ファイル中で位置の集合を表現する際に用いられる。

属性とその意味

3 種類の属性をもつ

type 必須。この要素が表す位置の意味を指定する。現時点では、以下の 5 種類の type を許容する。

- noise: 雑音位置を表す
- microphone: マイク位置を表す。
- source: 音源定位された音源の位置を表す
- tsp: TSP 信号を計測した位置を表す。
- impulse: インパルス応答を計測した位置を表す。

coordinate 必須。座標系を表す。直交座標系なら `cartesian`、極座標系なら `polar` が入る。

frame オプショナル。この positions が何らかのフレーム番号に対応するとき、その値が入る。

子要素

個々の位置を表す要素 `position` が 0 以上の任意の個数入る。
positions の属性は、まずは固定かつ必須の属性がある。

id : positions 内で一意となる整数。

path : positions が対応するファイルへのパス。

次に、親要素となる positions の coordinate 属性によって、座標の属性が異なる

coordinate = ”cartesian” の場合 三次元座標を表す x, y, z の値。単位はミリメートル。

coordinate = ”polar” の場合 極座標を表す azimuth, elevation, radius の値。azimuth/elevation の単位は degree、radius の単位はミリメートル。

5.1.4 neighbors

この要素が表現する情報

兄弟要素である positions の隣接関係を表す要素。

属性とその意味

必須の属性 `algorithm` をもつ。この属性は隣接関係を計算するアルゴリズムを表しており、現状ではユークリッド距離の近い順に隣接関係を求める `NearestNeighbor` のみが実装されている。

子要素

個々の位置 (position) に対する隣接関係を表す `neighbor` 要素を子要素に持つ。それぞれ 2 つの必須の属性を持つ。

id 隣接関係を表す position の id。 整数が入る。

ids id と隣接する position の id がセミコロンで区切られて入る。自分自身の id を含む。

例えば、 id が 1 の position 要素が、 id が 2 の position 要素と隣接している場合、neighbor 要素は以下のように表現される。

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

5.2 Matrix バイナリ形式

ある方向の伝達関数など、行列を表現するためのフォーマット。図 5.2 に概要を示す。

最初の 32 バイトに、 HARK の Matrix バイナリ形式で表すことを示す文字列が入り、次の 32 バイトにデータ型を表す文字列が入る。データ型には、 `int32`, `float32`, `complex` があり、それぞれ 4 バイト整数、 4 バイト浮動小数点数、 4 バイト浮動小数点数を実数・虚数にそれぞれ持つ複素数を意味する。続いて、行列の次元数を表す 4 バイト整数 (現時点ではテンソルは表現しないので 2 で固定) があり、行・列の順にサイズが 4 バイト整数で入る。その後、1 行 1 列目の要素、 1 行 2 列目の要素。。。 という順で行列の内容が保存される。

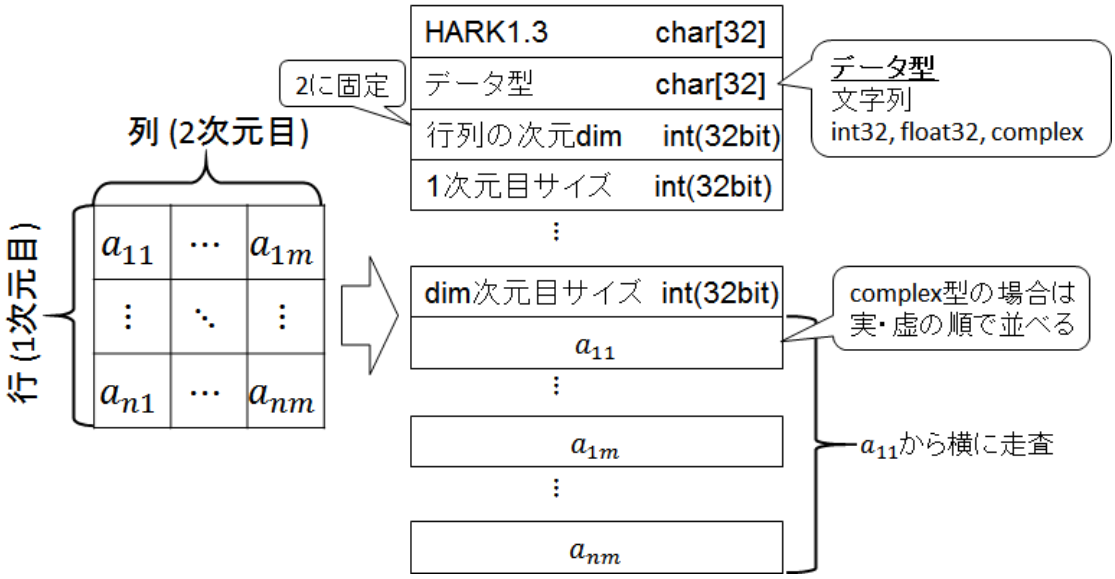


図 5.2: Matrix バイナリ形式の概要

5.3 Zip 形式

複数の方向の伝達関数を表す定位用伝達関数など、複雑な情報を表現するためのフォーマット。要素はテキストファイル、XML ファイル、Matrix バイナリなどのシンプルで独立したファイルで表し、それらの構造は zip ファイル内のディレクトリ構造で表す。

Zip 形式は任意のファイル構造を表現できるが、現時点では 1 種類の構造のみをサポートしており、それを伝達関数ファイル用の形式と、 GHDSS ノードの Export_W 使用時の分離行列、CMLoad CMSave ノードなどで利用する相関行列 に使用している。

5.3.1 伝達関数ファイルのディレクトリ構造

伝達関数ファイルのディレクトリ構造は下記のようになっている。なお、 / で終わる名前はディレクトリを表す。

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

ルートディレクトリは transferFunction である。まず、ファイルの種類を表す `whatisthis.txt` があり、内容は `transfer function` である。また、伝達関数に対応するマイク位置を表す microphone.xml と、伝達関数の計測位置を表す source.xml がある。定位用伝達関数は localization/ ディレクトリ以下に、分離用伝達関数は separation/ ディレクトリ以下に保存される。各ディレクトリ内のファイルは全てフォーマット文字列で使用する `tf%05d.mat` の形式になる。つまり、伝達関数ファイルの名前は、 source.xml 内の id に対応する数字を 5 桁で表した文字列 (0 詰め) となる。

なお、localization/ と separation/ は、いずれか、あるいは両方が空でも構わない。例えば、 localization/ 以下が空の場合、旧ファイルフォーマットで使用した分離用伝達関数に相当する。また、 localization/ と separation/ の両方に Matrix 形式のバイナリファイルがある場合、従来の分離用伝達関数ファイルと定位用伝達関数ファイルを 1 ファイルに統合したものに相当する。

5.3.2 GHDSS 分離行列のディレクトリ構造

分離行列ファイルのディレクトリ構造は下記のようになっている。なお、 / で終わる名前はディレクトリを表す。

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

ルートディレクトリは (この場合伝達関数ではないが) transferFunction である。まず、ファイルの種類を表す `whatisthis.txt` があり、内容は `separation matrix` である。また、分離行列に対応するマイク位置を表す microphone.xml と、分離行列計算時の音源定位結果を表す source.xml がある。

separation/ ディレクトリ以下に、分離行列（マイク数 × 周波数ビン数のサイズの複素数行列）が保存される。分離用伝達関数は separation/ ディレクトリ以下に保存され、そのファイル名は対応する音源の ID に対応している。ファイルの命名規則はフォーマット文字列で使用する `tf%05d.mat` の形式になる。つまり、分離行列ファイルの名前は、 source.xml 内の id に対応する数字を 5 桁で表した文字列 (0 詰め) となる。なお、localization/ は常に空ディレクトリである。

5.3.3 CMSave/CMLoad 定位用相関行列ファイルのディレクトリ構造

定位用相関行列ファイルのディレクトリ構造は下記のようになっている。なお、 / で終わる名前はディレクトリを表す。

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

ルートディレクトリは (この場合伝達関数ではないが) transferFunction である。まず、ファイルの種類を表す `whatisthis.txt` があり、内容は `correlation matrix` である。定位用相関行列ファイルであるため, microphone.xml は空ファイルである．source.xml は便宜上，localization/ ディレクトリ以下のファイル名と対応した個数の source が格納されている．

localization/ ディレクトリ以下に、相関行列（マイク数 × マイク数のサイズの正方複素数行列）が保存される。定位用相関行列は localization/ ディレクトリ以下に保存され、そのファイル名は対応する周波数ビンに対応している。ファイルの命名規則はフォーマット文字列で使用する `tf%05d.mat` の形式になる。つまり、定位用相関行列ファイルの名前は、 source.xml 内の id に対応する数字を 5 桁で表した文字列 (0 詰め) となる。なお、separation/ は常に空ディレクトリである。

第6章　ノードリファレンス

本章では，各ノードの詳細な情報を示す．はじめに，ノードリファレンスの読み方について述べる．

ノードリファレンスの読み方

1. **ノードの概要:** そのノードが何の機能を提供しているのかについて述べる．大まかに機能を知りたいときに読むとよい．
2. **必要なファイル:** そのノードを使用するのに要求されるファイルについて述べる．このファイルは 5 節 の記述とリンクしているので，ファイルの詳しい内容は 5 節で述べる．
3. **使用方法:** どんなときにそのノードを使えばよいのかと，具体的な接続例について述べる．とにかく当該ノードを使って見たいときは，その例をそのまま試してみるとよい．
4. **ノードの入出力とプロパティ:** ノードの入力ターミナルと出力ターミナルの型と意味について述べる．また，設定すべきパラメータを表に示している．詳しい説明が必要なパラメータについては表の後にパラメータごとに解説を加えている．
5. **ノードの詳細:** そのノードの理論的背景や実装の方法を含む詳細な解説を述べる．詳しく当該ノードを知りたいときはこの部分を読むとよい．

記号の定義

本ドキュメントで用いる記号を表 6.1 の通り定義する．また，暗黙的に，次のような表記を用いる．

- 小文字は時間領域，大文字は周波数領域を意味する．
- ベクトルと行列は太字で表記する．
- 行列の転置は ^{T} ，エルミート転置は ^{H} で表す．(X^T, X^H)
- 推定値にはハットをつける．(例： x の推定値は $\hat{x}$)
- 入力は x ，出力は y を用いる．
- 分離行列は $\boldsymbol{W}$ を，伝達関数行列は $\boldsymbol{H}$ を用いる．
- チャネル番号は下付き文字で表す． (例: 3 チャネル目の信号源は s_3)
- 時間，周波数は () 内に書く． (例: $X(t, f)$)

表 6.1: 記号のリスト

変数名	説明
m	マイクロホンのインデックス
M	マイクロホンの数
$m_1, \cdots, m_M$	各マイクロホンを示す記号
n	音源のインデックス
N	音源の数
$s_1, \cdots, s_N$	各音源を示す記号
i	周波数ビンのインデックス
K	周波数ビンの数
$k_0 \cdots k_{K-1}$	周波数ビンを示す記号
$NFFT$	FFT ポイント数
$SHIFT$	シフト長
$WINLEN$	窓長
π	円周率
j	虚数単位

6.1 AudioIO カテゴリ

6.1.1 AudioStreamFromMic

モジュールの概要

マイクロホンアレーからマルチチャネル音声波形データを取り込む．サポートするオーディオインタフェースデバイスは，システムインフロンティア製 RASP シリーズ，東京エレクトロンデバイス製 TD-BD-16ADUSB, ALSA ベースのデバイス (例，RME Hammerfall DSP Multiface シリーズ) である．また，デバイス以外にも，IEEE Float 形式のマルチチャネルの音響信号の raw データを TCP/IP ソケット接続で受信することも可能である．各種デバイスの導入は，第 **??** 章を参照のこと．

必要なファイル

無し．

使用方法

どんなときに使うのか

このモジュールは，HARK システムへの入力として，マイクロホンアレイから得られた音声波形データを用いる場合に使用する．

典型的な接続例

図 6.1 , 6.2 に AudioStreamFromMic モジュールの使用例を示す．

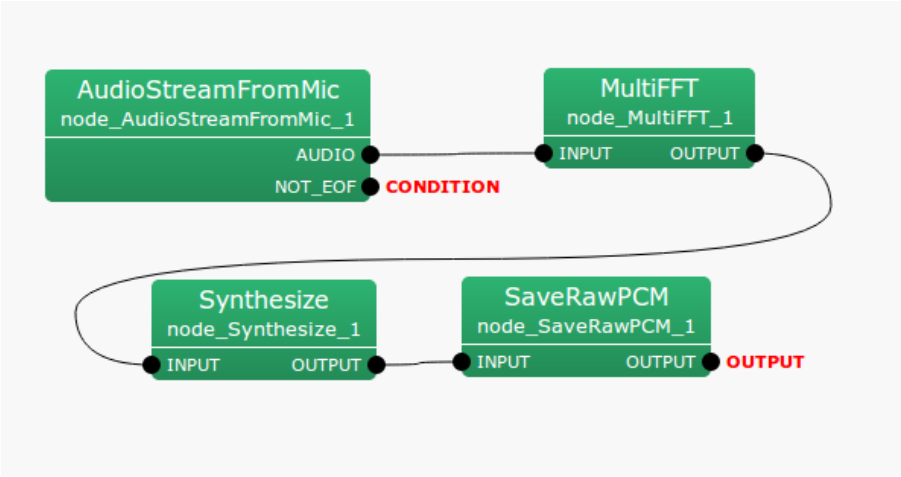


図 6.1: AudioStreamFromMic の接続例: LOOP0 の内部

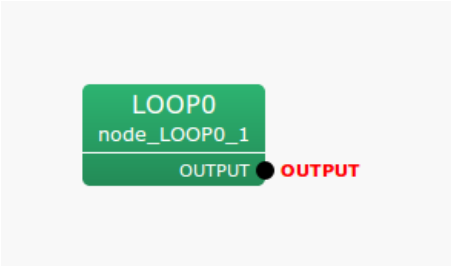


図 6.2: AudioStreamFromMic の接続例: MAIN

デバイスの写真

AudioStreamFromMic モジュールがサポートするデバイスのうち，以下を写真で紹介する．

1. 無線 RASP
2. RME Hammerfall DSP シリーズ Multiface (ALSA 対応デバイス)

１．無線 RASP　図 6.3 は無線 RASP の外観である．HARK システムとの接続には，無線 LAN による Ethernet を通じて行う．無線 RASP への電源供給は，付属の AC アダプタで行う．

無線 RASP は，プラグインパワーに対応しており，プラグインパワー供給のマイクロホンをそのまま端子に接続できる．マイクロホンプリアンプを使用せずに手軽に録音ができる利点がある．

2．RME Hammerfall DSP Multiface シリーズ　図 6.4，6.5 は RME Hammerfall DSP シリーズ Multiface の外観である．32bit CardBus を通じてホスト PC と通信を行う．6.3 mm TRS 端子を通じてマイクロホンを接続できるが，入力レベルを確保するために，別途マイクロホンアンプを使用する (図 6.5)．例えば，マイクロホンを RME OctaMic II に接続し，OctaMic II と Multiface を接続する．OctaMic II は，ファンタム電源供給をサポートしており，ファンタム電源を必要とするコンデンサマイクロホン（例えば，DPA 社 4060-BM）を直接接続可能である．

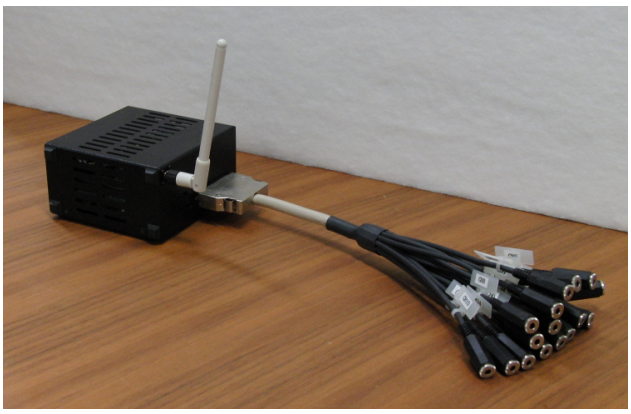


図 6.3: 無線 RASP

しかし，プラグインパワー供給機能がないため，プラグインパワー供給型のマイクロホンを接続するためには，別途プラグインワパー用の電池ボックスが必要である．例えば，電池ボックスは Sony EMC-C115 や audio-technica AT9903 に附属している．

図 6.4: RME Hammerfall DSP Multiface の正面

図 6.5: RME Hammerfall DSP Multiface の背面

モジュールの入出力とプロパティ

入力

無し.

出力

表 6.2: AudioStreamFromMic のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	処理を行う基本単位となるフレームの長さ.
ADVANCE	int	160	[pt]	フレームのシフト長.
CHANNEL_COUNT	int	8	[ch]	使用するデバイスのマイクロホン入力チャネル数.
SAMPLING_RATE	int	16000	[Hz]	取り込む音声波形データのサンプリング周波数.
DEVICETYPE	string	WS		使用するデバイスの種類.
GAIN	string	0dB		RASP を使用する場合のゲイン.
DEVICE	string	127.0.0.1		デバイスへのアクセスに必要な文字列. ”plughw:0,1” などのデバイス名や，RASP を使用する時は IP アドレスなど.

AUDIO ：`Matrix<float>` 型. 行番号がチャネルのインデックス, 列番号がサンプルのインデックスである, マルチチャネル音声波形データ. 列の大きさはパラメータ LENGTH に等しい.

NOT_EOF ：`bool` 型．まだ波形の入力があるかどうかを表す．録音波形に対する繰り返し処理の終了フラグとして用いる．`true` のとき，波形の取り込みを続行し，`false` のとき，読み込みを終える．常に `true` を出力する．

パラメータ

LENGTH ：`int` 型．512 がデフォルト値．処理の基本単位であるフレームの長さをサンプル数で指定する．値を大きくすれば，周波数解像度が上がる一方，時間解像度は下がる．音声波形の分析には，20 ~ 40 [ms] に相当する長さが適切であると言われている．サンプリング周波数が 16000 [Hz] のとき，デフォルト値は 32 [ms] に相当する．

ADVANCE ：`int` 型．160 がデフォルト値．フレームのシフト長をサンプル数で指定する．サンプリング周波数が 16000 [Hz] のとき，デフォルト値はフレーム周波数 10 [ms] に相当する．

CHANNEL_COUNT ：`int` 型．使用するデバイスのチャネル数.

SAMPLING_RATE ：`int` 型．16000 がデフォルト値．取り込む波形のサンプリング周波数を指定する．処理の中で ω [Hz] までの周波数が必要な場合，サンプリング周波数は 2ω [Hz] 以上の値を指定する．サンプリング周波数を大きくとると，一般にデータ処理量が増えるので，実時間処理が困難になる．

DEVICETYPE ：`string` 型．ALSA, RASP, WS, TDBD16ADUSB, RASP24-16, RASP24-32, RASP-LC, NETWORK から選択する．ALSA ベースのドライバをサポートするデバイスを使用する場合には ALSA を選択する．RASP-2 を使用する場合には RASP を選択する．無線 RASP を使用する場合は WS を選択する．TD-BD-16ADUSB を使用する場合には TDBD16ADUSB を選択する．RASP-24 を 16bit 量子化ビット数の録音モードで使用する場合には RASP24-16 を選択する．RASP-24 を 24bit 量子化ビット数の録音モードで使用する場合には RASP24-32 を選択する．RASP-LC を PC と無線 LAN による接続で使用する場合には RASP-LC を選択する.（RASP-LC を PC に直接 USB 接続する場合は ALSA で構わない.）TCP/IP 接続を介して IEEE float 形式の raw データを受信したい場合は NETWORK を選択する．

GAIN ：`string` 型．0dB がデフォルト値．マイクの録音ゲインを設定する．0dB, 12dB, 24dB, 36dB, 48dB の中から選択する．RASP-24 を録音デバイスとして利用する時のみ有効になる．

DEVICE : `string` 型．入力に用いるデバイスへのアクセスに必要な識別名. DEVICETYPE 毎に入力内容が異なるため，以下の説明を参考のこと．

モジュールの詳細

HARK がサポートするオーディオデバイスは，以下の通り．

1. システムインフロンティア製 RASP シリーズ
   - RASP-2
   - 無線 RASP
   - RASP-24
   - RASP-LC
2. 東京エレクトロンデバイス製 TD-BD-16ADUSB.
3. ALSA ベースのデバイス．以下は例.
   - Microsoft 製 Kinect Xbox
   - Sony 製 PlayStation Eye
   - Dev-Audio 製 Microcone
   - RME Hammerfall DSP シリーズ Multiface
4. TCP/IP ソケット接続で送られる音響信号 (IEEE float wav 形式)

以下ではそれぞれのデバイスを用いる際のパラメータ設定を記す.

RASP シリーズ:

- **RASP-2** のパラメータ設定

CHANNEL_COUNT	8
DEVICETYPE	WS
DEVICE	**RASP-2** の IP アドレス

- **無線 RASP** のパラメータ設定

CHANNEL_COUNT	16
DEVICETYPE	WS
DEVICE	**無線 RASP** の IP アドレス
備考	RASP シリーズはモデルによって 16 チャネル中マイクロホン入力とライン入力が混在しているものがある．混在する場合には，ChannelSelector モジュールを，AudioStreamFromMic モジュールの AUDIO 出力に接続しマイクロホン入力チャネルだけを選択する必要がある．

- **RASP-24** のパラメータ設定

CHANNEL_COUNT	9 の倍数
DEVICETYPE	RASP24-16 または RASP24-32
DEVICE	**RASP-24** の IP アドレス
備考	録音の量子化ビット数を 16bit にする場合は DEVICETYPE=RASP24-16 を，量子化ビット数を 24bit にする場合は DEVICETYPE=RASP24-32 を指定する．CHANNEL_COUNT は，9 の倍数を指定する．録音チャネルは 0 番目〜 7 番目のチャネルはマイクロホン入力，8 番目のチャネルはライン入力となる．マイクアレイ処理には，ChannelSelector モジュールを，AudioStreamFromMic モジュールの AUDIO 出力の後段に接続しマイクロホン入力チャネルだけを選択する必要がある．

- **RASP-LC** のパラメータ設定

CHANNEL_COUNT	8
DEVICETYPE	ALSA または RASP-LC
DEVICE	DEVICETYPE=ALSA に指定した場合は plughw:a,b と指定する．設定の詳細は下記の ALSA 対応デバイスを参照．DEVICETYPE=RASP-LC に指定した場合は RASP-LC の IP アドレスを指定する．
備考	RASP-LC の USB インターフェイスを直接 PC に接続する場合は DEVICETYPE=ALSA に設定する．RASP-LC を 無線 LAN を通じて PC と接続する場合は DEVICETYPE=RASP-LC に設定する．録音チャネルは全てマイクロホン入力となる．

東京エレクトロンデバイス製デバイス:

- **TD-BD-16ADUSB** のパラメータ設定

CHANNEL_COUNT	16
DEVICETYPE	TDBD16ADUSB
DEVICE	TDBD16ADUSB

ALSA 対応デバイス:

ALSA 対応デバイスの場合，DEVICE パラメータは plughw:a,b と指定する．a と b には正の整数が入る．a には，arecord -l で表示されるカード番号を入れる．音声入力デバイスが複数接続されている場合には，カード番号が複数表示される．使用するカード番号を入れる．b には arecord -l で表示されるサブデバイス番号を入れる．サブデバイスが複数あるデバイスの場合，使用するサブデバイスの番号を入れる．サブデバイスが複数ある場合の例としては，アナログ入力とディジタル入力を持ったデバイスが該当する．

- **Kinect Xbox** のパラメータ設定

CHANNEL_COUNT	4
DEVICETYPE	ALSA
DEVICE	plughw:a,b

- **PlayStation Eye** のパラメータ設定

CHANNEL_COUNT	4
DEVICETYPE	ALSA
DEVICE	plughw:a,b

- **Microcone** のパラメータ設定

CHANNEL_COUNT	7
DEVICETYPE	ALSA
DEVICE	plughw:a,b

- **RME Hammerfall DSP シリーズ Multiface** のパラメータ設定

CHANNEL_COUNT	8
DEVICETYPE	ALSA
DEVICE	plughw:a,b

ソケット接続 (DEVICETYPE=NETWORK を選択した場合):

DEVICE パラメータは音響信号を送信する側のマシンの IP アドレスを指定する．その他のパラメータは送られてくる音響信号に合わせる必要がある．送信したい音響信号が M チャネルで，１フレーム毎に T サンプルのデータを取得できる場合，以下のように送信すれば良い．

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

ここで，X は IEEE float wav 形式であり，$-1 \leq X \leq 1$ である．

Windows 版 DirectSound 対応デバイス:

Windows 版 HARK では，無線 RASP，RASP-24，ソケット接続に加えて DirectSound に対応したオーディオインタフェースデバイスを使用できる．DEVICE パラメータにデバイス名を入力することでデバイスの指定が可能である．DEVICE パラメータへのマルチバイト文字の入力には対応していない．

デバイス名の確認方法は, デバイスマネージャなど使う方法と，Windows 版 HARK で提供している Sound Device List をつかう方法がある. 後者の場合は, [スタート] → [プログラム] → [HARK] にある Input Sound Device List をクリックすると，図 6.6 に示すように現在接続中のデバイス名が表示される．DEVICE パラメータは部分一致でも設定可能となっているので，図 6.6 の場合は単に ”Hammerfall” を設定するだけでもよい．このとき，複数のデバイス名が部分一致した場合は上位に表示されているデバイスが選択される．

また, Kinect Xbox， PlayStation Eye，Microcone，RASP-ZX に関しては, 正確なデバイス名を入力しなくとも下記に示すパラメータを設定することで利用できる．

ただし，RASP-ZX については Windows7 使用時に限りオートゲインがかかってしまうため注意が必要である．

Windows 版 ASIO 対応デバイス:

ASIO 対応デバイス, たとえば Microcone や RME Hammerfall DSP シリーズ Multiface を利用したい場合は, HARK の ASIO プラグインを HARK ウェブページからダウンロードしてインストールする必要がある．また,

ASIO 対応デバイスは Windows 標準のドライバでは認識しないため，デバイスメーカが配布するドライバをインストールする必要がある．

ASIO 対応デバイスでは AudioStreamFromMic のかわりに AudioStreamFromASIO を使用する．

- **Kinect Xbox** のパラメータ設定

CHANNEL_COUNT	4
DEVICETYPE	DS
DEVICE	kinect

- **PlayStation Eye** のパラメータ設定

CHANNEL_COUNT	4
DEVICETYPE	DS
DEVICE	pseye

- **Microcone** のパラメータ設定

CHANNEL_COUNT	7
DEVICETYPE	ASIO
DEVICE	microcone

- **TAMAGO** のパラメータ設定

CHANNEL_COUNT	8
DEVICETYPE	DS
DEVICE	TAMAGO or tamago

- **RASP-ZX** のパラメータ設定

CHANNEL_COUNT	8 or 16
DEVICETYPE	DS
DEVICE	rasp

- **RME Hammerfall DSP シリーズ Multiface** のパラメータ設定

CHANNEL_COUNT	8
DEVICETYPE	ASIO
DEVICE	ASIO Hammerfall DSP

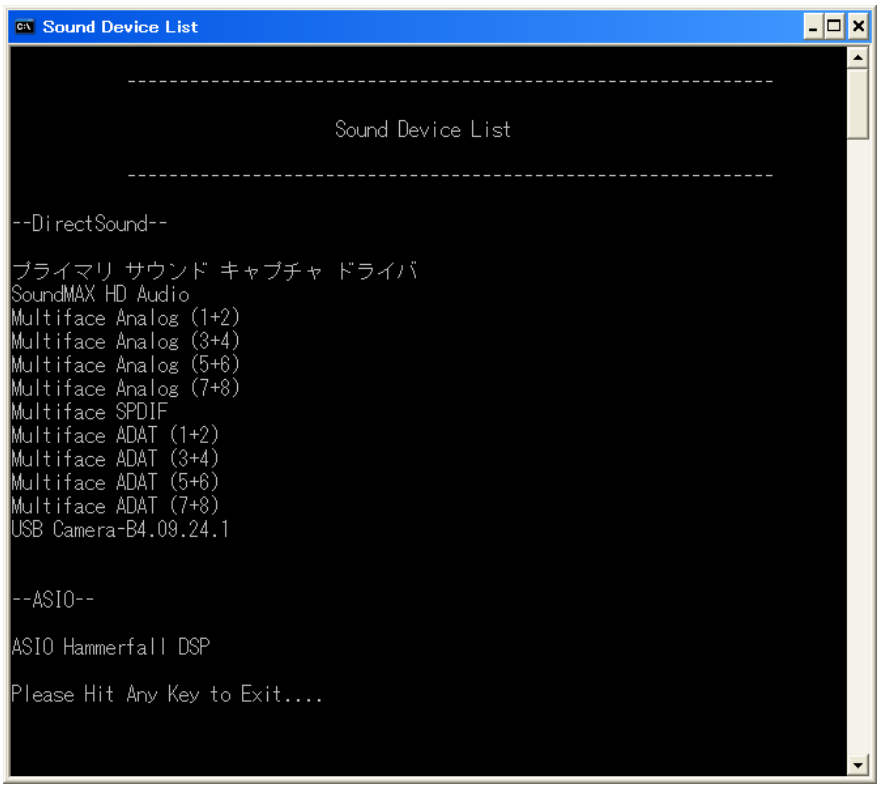


図 6.6: デバイス名の確認

6.1.2 MultiAudioStreamFromMic

モジュールの概要

複数のマイクロホンアレイからマルチチャネル音声波形データを取り込む．このノードは AudioStreamFromMic を拡張したものである．長時間のデータを受信した時に，マイクロホンデバイス間でフレームのずれが発生する可能性がある．MultiAudioStreamFromMic は，そのずれを補正する機能を持つ．

必要なファイル

無し．

使用方法

どんなときに使うのか

このノードは，HARK システムへの入力として，複数のマイクロホンアレイから得られた音声波形データを用いる場合に使用する．複数のマイクロホンアレイは同一の型番あるいは同一の仕様でなければならない．

典型的な接続例

図 6.7 に MultiAudioStreamFromMic モジュールの使用例を示す．

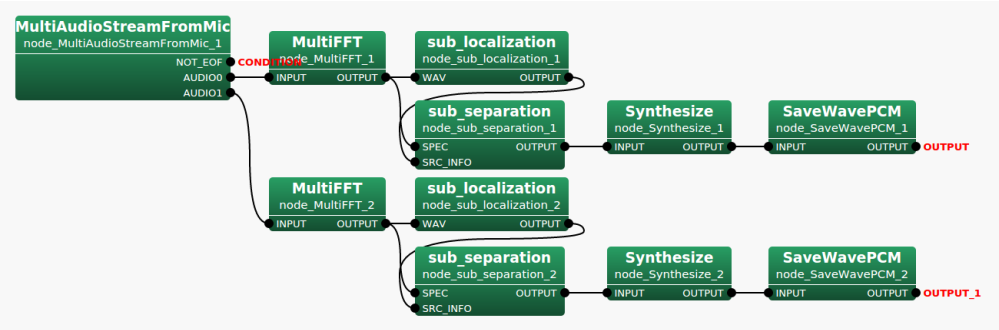


図 6.7: MultiAudioStreamFromMic の接続例: LOOP0 の内部

モジュールの入出力とプロパティ

入力

無し．

出力

AUDIO0 : `Matrix<float>` 型．行番号がチャネルのインデックス，列番号がサンプルのインデックスである，マルチチャネル音声波形データ．列の大きさはパラメータ LENGTH に等しい．本出力端子はデフォルトでは非表示である．パラメータ DEVICE で指定した順に AUDIO0,AUDIO1,... のように，AUDIO の後ろに 0 から始まる通し番号をつけた出力端子を追加する必要がある．

表 6.3: MultiAudioStreamFromMic のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	処理を行う基本単位となるフレームの長さ.
ADVANCE	int	160	[pt]	フレームのシフト長.
CHANNEL_COUNT	int	8	[ch]	使用するデバイスのマイクロホン入力チャネル数.
SAMPLING_RATE	int	16000	[Hz]	取り込む音声波形データのサンプリング周波数.
DEVICETYPE	string	WS		使用するデバイスの種類.
GAIN	string	0dB		RASP を使用する場合のゲイン.
DEVICE	string	/dev/null		デバイスへのアクセスに必要な識別名リスト．複数のデバイスの識別名を空白文字で区切って指定する.
FRAME_COUNT_SKEW_TOLERANCE	float	5.0	[sec]	フレームずれの許容量を秒単位で指定する.

NOT_EOF ：bool 型．まだ波形の入力があるかどうかを表す．録音波形に対する繰り返し処理の終了フラグとして用いる．true のとき，波形の取り込みを続行し，false のとき，読み込みを終える．常に true を出力する.

非表示出力の追加方法は図 6.8 を参照されたい.

パラメータ

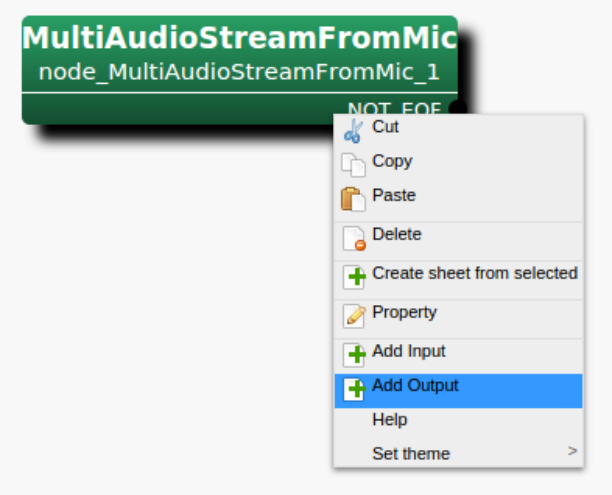
LENGTH, ADVANCE, CHANNEL_COUNT, SAMPLING_RATE, DEVICETYPE, GAIN ：これらのパラメータは AudioStreamFromMic と同じである．これらパラメータ DEVICE にて空白区切りで指定した複数のデバイス識別名に対応させて， AUDIO0、AUDIO1， .... のように昇順に並べた出力端子を追加する.

DEVICE ：string 型. 入力に用いる複数のデバイスへのアクセスに必要な識別名を, 空白文字で区切って指定する. 区切られた個々の識別名は AudioStreamFromMic の DEVICE と同じである.

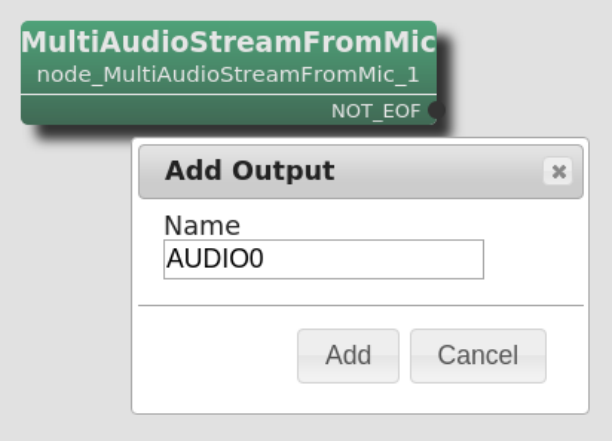
FRAME_COUNT_SKEW_TOLERANCE ：float 型. デバイス間におけるフレーム数の差の許容範囲を秒単位で指定する. この値を超えるフレームずれが生じたタイミングで, 補正処理が行われる.

ノードの詳細

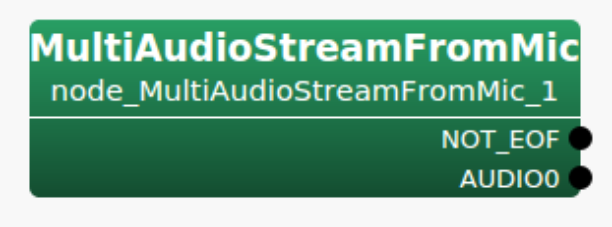
AudioStreamFromMic を複数のデバイスに対応するよう拡張したノードである．パラメータ DEVICE にて空白区切りで指定した複数のデバイス識別名に対応させて, AUDIO0、AUDIO1， .... のように昇順に並べた出力端子を追加する. パラメータ FRAME_COUNT_SKEW_TOLERANCE はデバイス間におけるフレーム数の補正を実行するタイミングを決定する. 複数デバイスから取得した音声波形データのフレーム数の最大値と最小値の差が FRAME_COUNT_SKEW_TOLERANCE に指定した秒数を超えた時点で, その秒数分を最大の音声波形データから削除する.



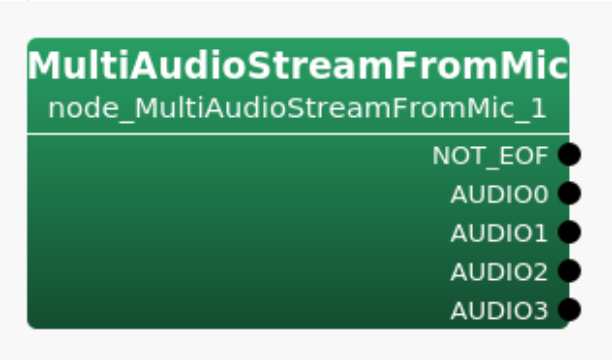
Step 1: MultiAudioStreamFromMic を右クリックし，Add Output をクリック



Step 2: Add Output の入力フォームに AUDIO0 を記入し，Add をクリック



Step 3: ノードに AUDIO0 出力端子が追加される



Step 4: Step 1 〜 Step 3 を繰り返し AUDIO1,... を追加

図 6.8: 出力の追加例 : AUDIO0 端子の表示

6.1.3 AudioStreamFromWave

ノードの概要

音声波形データを WAVE ファイルから読み込む．読み込んだ波形データは，`Matrix<float>` 型で扱われる．行がチャネル，列が波形の各サンプルのインデックスとなる．

必要なファイル

RIFF WAVE フォーマットの音声ファイル．チャネル数，サンプリング周波数に制約はない．量子化ビット数は，16 bit または 24 bit の符号付き整数の，リニア PCM フォーマットを仮定する．

使用方法

どんなときに使うのか

このノードは，HARK システムへの入力として，WAVE ファイルを読み込ませたいときに使う．

典型的な接続例

図 6.9，6.10 に AudioStreamFromWave ノードの使用例を示す．

図 6.9 は，AudioStreamFromWave がファイルから読み込んだ `Matrix<float>` 型のマルチチャネル波形を MultiFFT ノードによって周波数領域に変換している例である．

AudioStreamFromWave でファイルを読み込むには，図 6.10 のように Constant ノード (FlowDesigner の標準ノード) でファイル名を指定し，InputStream ノードでファイルディスクリプタを生成する．そして，InputStream ノードの出力を，AudioStreamFromWave など HARK の各種ノードのネットワークがある iterator サブネットワーク（図 6.10 中の LOAD_WAVE）に接続する．

ノードの入出力とプロパティ

表 6.4: AudioStreamFromWave のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	`int`	512	[pt]	処理を行う基本単位となるフレームの長さ．
ADVANCE	`int`	160	[pt]	イタレーション毎にフレームをシフトさせる長さ．
USE_WAIT	`bool`	`false`		処理を実時間で行うかどうか．

入力

INPUT ：`Stream` 型．FlowDesigner 標準ノードの，IO カテゴリにある InputStream ノードから入力を受け取る．

出力

AUDIO ：`Matrix<float>` 型．行がチャネル，列がサンプルのインデックスである，マルチチャネル音声波形データ．列の大きさはパラメータ LENGTH に等しい．

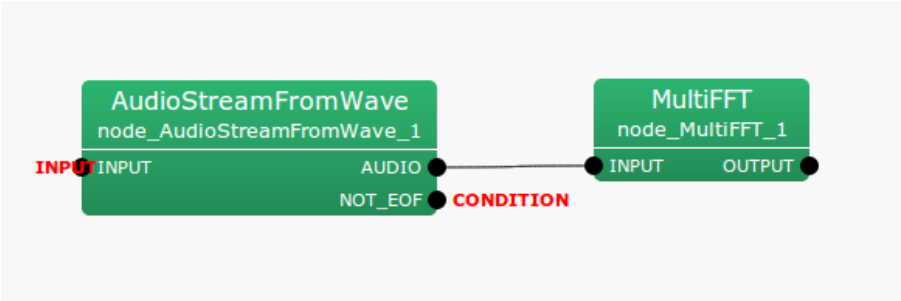


図 6.9: AudioStreamFromWave の接続例: LOAD_WAVE の内部

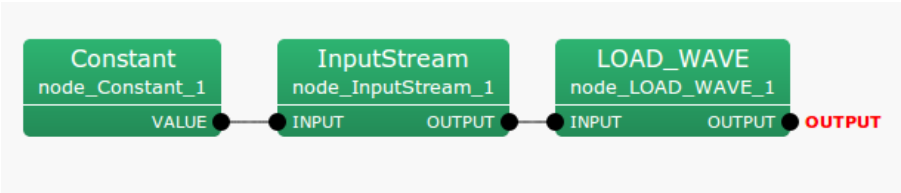


図 6.10: AudioStreamFromWave の接続例: MAIN

NOT_EOF : `bool` 型．まだファイルを読めるかどうかを表す．ファイルに対する繰り返し処理の終了フラグとして用いる．ファイルの終端に達したとき `false` を出力し，それ以外のとき `true` を出力する．

パラメータ

LENGTH : `int` 型．512 がデフォルト値．処理の基本単位であるフレームの長さをサンプル数で指定する．値を大きくすれば，周波数解像度が上がる一方，時間解像度は下がる．音声波形の分析には，20 ~ 40 [ms] に相当する長さが適切であると言われている．サンプリング周波数が 16000 [Hz] のとき，デフォルト値は 32 [ms] に相当する．

ADVANCE : `int` 型．160 がデフォルト値．音声波形に対する処理のフレームを，波形の上でシフトする幅をサンプル数で指定する．サンプリング周波数が 16000 [Hz] のとき，10 [ms] に相当する．

USE_WAIT : `bool` 型．`false` がデフォルト値．通常，HARK システムの音響処理は実時間よりも高速に動作する．処理に “待ち” を加えて，入力ファイルに対して実時間で処理を行いたい場合は `true` に設定する．ただし，実時間よりも遅い場合は，`true` にしても効果はない．

ノードの詳細

対応するファイルフォーマット: RIFF WAVE ファイルを読み込むことができる．チャネル数，量子化ビット数はファイルのヘッダから読み込むが，サンプリング周波数，量子化手法を表すフォーマット ID は無視する．チャネル数，サンプリング周波数は任意の形式に対応する．サンプリング周波数が処理を行う上で必要になる場合は，パラメータとして要求するノードがある (GHDSS ， MelFilterBank など)．量子化手法とビット数は，16 または 24 ビット符号付き整数によるリニア PCM を仮定する．

パラメータの目安: 処理の目的が音声の分析 (音声認識など) の場合，LENGTH には 20 ～ 40 [ms] 程度，ADVANCE には LENGTH の 1/3 ～ 1/2 程度が良いとされている．サンプリング周波数が 16000 [Hz] の時，LENGTH，ADVANCE のデフォルト値はそれぞれ，32，10 [ms] に対応する．

6.1.4 SaveRawPCM

ノードの概要

時間領域の音声データをファイルに保存する．音声データは，Raw PCM 形式に基づき 16 [bit] または 24 [bit] 整数でファイルに書き出される．その際，入力データの型に応じて多チャネル音声データ，または，モノラル音声データとして記録される．

SaveRawPCM ノードにより書き出されるファイルを WAVE ファイルに変換するためには，ヘッダをファイルの先頭に追加すればよい．例えば SoX などのソフトウェアを使用することでヘッダを追加することができる．しかし，SaveWavePCM ノードを使用するとこのヘッダの追加を自動的に行うため，WAVE ファイルが必要な場合は SaveWavePCM ノードを使うとよい．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源分離における性能を確認するために，分離音を聞いてみたい場合や，AudioStreamFromMic ノードと組みわせて，マイクロホンアレイを用いて多チャンネルの音声データ録音を行う場合に用いる．分離音を保存したい場合は，この SaveWavePCM ノードに接続する前に Synthesize ノードで時間領域の音声データにしておく必要がある．

典型的な接続例

図 6.11，6.12 に SaveRawPCM の使用例を示す．図 6.11 は，AudioStreamFromMic からの多チャネル音声データを SaveRawPCM ノードでファイルに保存する例である．この場合，各チャンネルの音声データは別々のファイルにそれぞれモノラル音声データとして保存される．図 6.11 では，MatrixToMap ノードを取り除き，AudioStreamFromMic ノードと SaveRawPCM ノードを直接つなぐことができる．この場合は全チャンネルの音声データが一つのファイルに保存される．

図 6.12 は，分離音を SaveRawPCM ノードによって保存する例である．GHDSS ノードや，分離後のノイズ抑圧を行う PostFilter ノードから出力される分離音は周波数領域にあるので，Synthesize ノードによって時間領域の波形に変換したのち，SaveRawPCM ノードに入力される．WhiteNoiseAdder ノードは，分離音の音声認識率向上のため通例用いられるもので，SaveRawPCM の使用に必須ではない．

ノードの入出力とプロパティ

入力

INPUT : `Map<int, ObjectRef>` または `Matrix<float>` 型．前者は分離音など，音源 ID と音声データの構造体，後者は多チャネルの音声データ行列．

出力

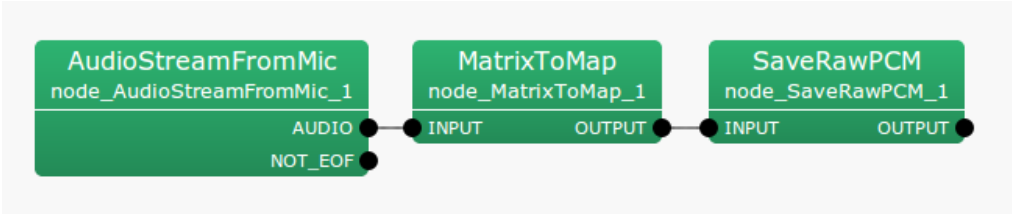


図 6.11: SaveRawPCM の接続例 1

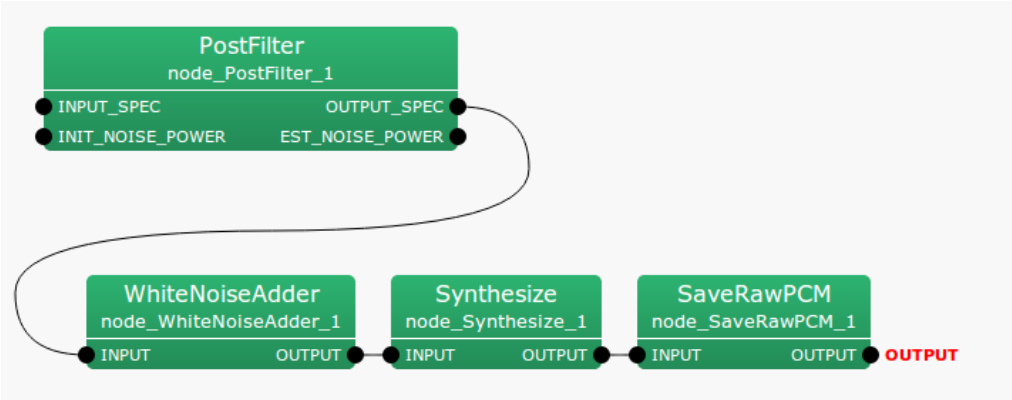


図 6.12: SaveRawPCM の接続例 2

表 6.5: SaveRawPCM のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
BASENAME	string	sep_		保存するファイル名のフォーマット。詳しくは SaveWavePCM 参照.
ADVANCE	int	160	[pt]	ファイルに保存する音声波形の分析フレームのシフト長.
BITS	int	16	[bit]	ファイルに保存する音声波形の量子化ビット数. 16 または 24 を指定可.
INPUT_BITS	string	as_BITS	[bit]	入力音声波形の量子化ビット数.

OUTPUT : `Map<int, ObjectRef>` または `Matrix<float>` 型．入力と同じものが出力される．

パラメータ

BASENAME : `string` 型．デフォルトは sep_．ファイル名のプレフィックスを指定する．出力されるファイル名は，音源 ID が付いている場合は “BASENAME_ID.sw” となる．3 つの混合音を分離した結果の分離音のファイル名は，BASENAME が sep_ のとき，`sep_0.sw`，`sep_1.sw`，`sep_2.sw` などとなる．

ADVANCE : `int` 型．他のノードの ADVANCE の値と揃える必要がある．

BITS : `int` 型．ファイルに保存する音声データの量子化ビット数．16 または 24 を指定可．

INPUT_BITS : `string` 型．入力音声波形の量子化ビット数．16 または 24 を指定可．as_BITS は BITS と同じ値を意味する．

ノードの詳細

保存されるファイルのフォーマット: 保存されるファイルは，ヘッダ情報を持たない Raw PCM 音声データとして記録される．したがって，ファイルを読む際には，適切なサンプリング周波数とトラック数，量子化ビット数を 16 [bit] または 24 [bit] に指定する必要がある．

また，入力の型によって書き出されるファイルは次のように異なる．

`Matrix<float>` 型 このとき書き出されるファイルは，入力の行の数だけチャネルを持った多チャネル音声データファイルとなる．

`Map<int, ObjectRef>` 型 このとき書き出されるファイルは，BASENAME の後に ID 番号が付与されたファイル名で，各 ID ごとにモノラル音声データファイルが書き出される．

6.1.5 SaveWavePCM

ノードの概要

時間領域の音声データをファイルに保存する．SaveRawPCM ノードとの違いは，出力されるのがヘッダーを持つ WAVE 形式のファイルである点である．そのため，例えば audacity や wavesurfer などで読み込む際に簡単である．また，ファイルを AudioStreamFromWave ノードで開きたい場合は，この SaveWavePCM で保存する．

必要なファイル

無し．

使用方法

どんなときに使うのか

SaveRawPCM ノードと同様に，分離音を聞いてみたい場合や，多チャンネルの音声データ録音を行う場合に用いる．

典型的な接続例

使い方は，サンプリング周波数をパラメータとして指定する必要がある以外は SaveRawPCM ノードと同じである．図 6.11，6.12 の例において SaveRawPCM ノードを SaveWavePCM ノードに入れ替えて使うことができる．

ノードの入出力とプロパティ

表 6.6: SaveWavePCM のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
BASENAME	`string`	sep_		保存するファイル名のフォーマット．
ADVANCE	`int`	160	[pt]	ファイルに保存する音声波形の分析フレームのシフト長．
SAMPLING_RATE	`int`	16000	[Hz]	サンプリング周波数．ヘッダを作成するために使用する．
BITS	`string`	int16	[bit]	ファイルに保存する音声波形の量子化ビット数． int16 または int24 を指定可．
INPUT_BITS	`string`	as_BITS	[bit]	入力音声波形の量子化ビット数．

入力

INPUT : `Map<int, ObjectRef>` または `Matrix<float>` 型．前者は分離音など，音源 ID と音声データの構造体，後者は多チャネルの音声データ行列．

SOURCES : `Vector<ObjectRef>` 型．音源定位結果 (`Source` 型の `Vector`) が入力される。なくても良い。

出力

OUTPUT ：`Map<int, ObjectRef>` または `Matrix<float>` 型．入力と同じものが出力される．

パラメータ

BASENAME ：`string` 型．デフォルトは sep_．通常の文字列の場合はファイル名のプレフィックス、あるいはフォーマットを指定する．詳細は次のセクションを参照。

ADVANCE ：`int` 型．他のノードの ADVANCE の値と揃える必要がある．

SAMPLING_RATE ：`int` 型．他のノードの SAMPLING_RATE の値と揃える必要がある．この値はヘッダに書き込むために用いられるだけであり，このパラメータを変更しても A/D 変換におけるサンプリングレートを変更することはできない．

BITS ：`string` 型．ファイルに保存する音声波形の量子化ビット数．int16 または int24 を指定可．

INPUT_BITS ：`string` 型．入力音声波形の量子化ビット数．int16 または int24 を指定可．as_BITS は BITS と同じ値を意味する．

ノードの詳細

保存されるファイルのフォーマット:

保存されるファイルは，ヘッダ情報を持つ WAVE ファイルとして記録される．したがって，ファイルを読む際には，特にサンプリング周波数とトラック数，量子化ビット数を指定しなくてもよい．

また，入力の型によって書き出されるファイルは次のように異なる．

`Matrix<float>` 型 このとき書き出されるファイルは，入力の行の数だけチャネルを持った多チャネル音声データを含む WAVE ファイルとなる．

`Map<int, ObjectRef>` 型 このとき書き出されるファイルは，BASENAME の後に ID 番号が付与されたファイル名で，各 ID ごとにモノラル音声データが書き出される (1 つのファイルには 1 つの ID に対応する音声データのみ).

保存されるファイル名の決定方法：

ファイル名の決定方法は 2 種類ある。

1. プレフィックス方式
   デフォルトの動作は、パラメータ BASENAME をプレフィックスとし、そのあとに音源 ID が接続され、最後に拡張子 `.wav` が接続される方式で名前が決定される。たとえば、BASENAME の値が `sep_` であれば、音源 ID が 0, 1, 2 の音声が入力されると、ファイル名は `sep_0.wav`，`sep_1.wav`，`sep_2.wav` となる。

2. フォーマット文字列方式 (HARK 2.3.1 以降)
   BASENAME に次の特別な文字列パターン`{tag:format}`がある場合、それにしたがってファイル名が決定される。このフォーマットを利用すると、ファイル名にさまざまな情報を追加できる。

   tag には表 6.7 に示す 4 種類が使用できる。音源 ID を表す srcid と実行時の時刻を表す date は制限なく利用できる。方位角と仰角をあらわす azimuth と elevation を利用するためには、SaveWavePCM に SOURCES 入力を追加し、音源定位結果を接続する必要がある。

表 6.7: tag リスト

Tag	Description	Unit
srcid	音源 ID	整数
date	実行時の時刻	時刻文字列 (yyyyMMDD-HHmmss)
azimuth	方位角 (要 SOURCES)	degree(整数、小数点以下四捨五入)
elevation	仰角 (要 SOURCES)	degree(整数、小数点以下四捨五入)

format はオプショナルなのでなくても良い。この部分は、桁数の指定 (`03d`) に利用できる。フォーマットは `printf` などで使われる標準的なフォーマット。

以下に使用例を示す。

音源 ID をファイルの中頃に挿入する場合

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

音源 ID の桁を揃える場合

- FORMAT: `wav_id_{srcid:03d}`
- OUTPUT: `wav_id_000.wav`, `wav_id_001_output.wav` ...

azimuth をファイル名に含める場合

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

だたし、この場合同じ角度の定位結果があれば上書きされるので注意。srcid をファイル名に含めることでファイル名の一意性が確保できる。

6.1.6 HarkDataStreamSender

ノードの概要

以下の音響信号処理結果をソケット通信で送信するノードである．

- 音響信号
- STFT 後の周波数スペクトル
- 音源定位結果のソース情報
- 音響特徴量
- ミッシングフィーチャーマスク
- 任意の文字列
- 任意の行列
- 任意のベクトル

必要なファイル

無し．

使用方法

どんなときに使うのか

上記のデータの中で必要な情報を TCP/IP 通信を用いて，HARK 外のシステムに送信するために用いる．

典型的な接続例

図 6.13 の例では全ての入力端子に接続している．送信したいデータに合わせて入力端子を開放することも可能である．入力端子の接続と送信されるデータの関係については「ノードの詳細」を参照．

ノードの入出力とプロパティ

入力

MIC_WAVE ：`Matrix<float>` 型．音響信号（チャンネル数 × 各チャンネルの STFT の窓長サイズの音響信号）

MIC_FFT ：`Matrix<complex<float> >` 型．周波数スペクトル（チャンネル数 × 各チャンネルのスペクトル）

SRC_INFO ：`Vector<ObjectRef>` 型．音源数個の音源定位結果のソース情報

SRC_WAVE ：`Map<int, ObjectRef>` 型．音源 ID と音響信号の `Vector<float>` 型のデータのペア．

SRC_FFT ：`Map<int, ObjectRef>` 型．音源 ID と周波数スペクトルの `Vector<complex<float> >` 型のデータのペア．

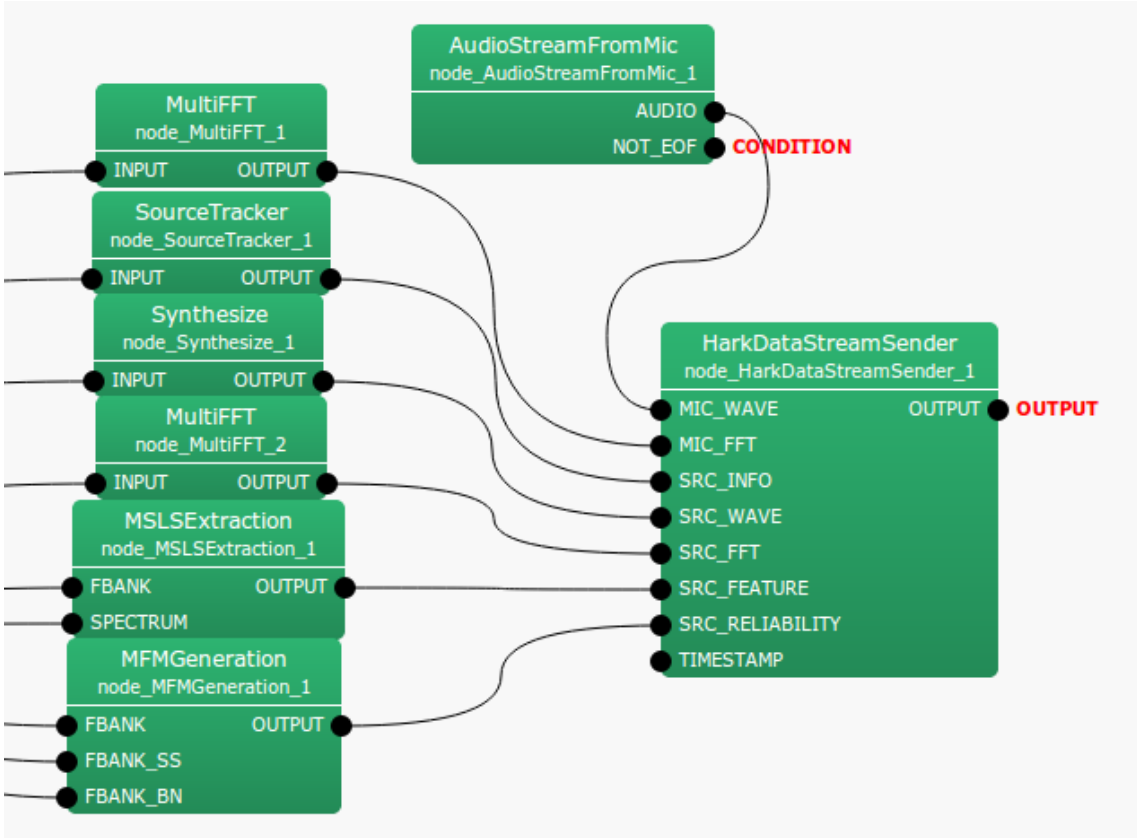


図 6.13: HarkDataStreamSender の接続例

表 6.8: HarkDataStreamSender のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
HOST	string	localhost		データの送信先サーバのホスト名/IP アドレス
PORT	int	5530		ネットワーク送出用ポート番号
ADVANCE	int	160	[pt]	フレームのシフト長
BUFFER_SIZE	int	512		ソケット通信のために確保する float 配列のサイズ
FRAMES_PER_SEND	int	1	[frm]	ソケット通信を何フレームに一回行うか
TIMESTAMP_TYPE	string	GETTIMEOFDAY		送信されるタイムスタンプ
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数
DEBUG_PRINT	bool	false		デバッグ情報出力の ON/OFF
SOCKET_ENABLE	bool	true		ソケット出力をするかどうかを決めるフラグ

SRC_FEATURE ：`Map<int, ObjectRef>` 型．音源 ID と音響特徴量の `Vector<float>` 型のデータのペア．

SRC_RELIABILITY ：`Map<int, ObjectRef>` 型．音源 ID とマスクベクトルの `Vector<float>` 型のデータのペア．

TEXT ：型．任意の文字列．

MATRIX ：`Matrix<float>` もしくは `Matrix<complex<float> >` 型．任意の行列．

VECTOR ：`Vector<float>` もしくは `Vector<complex<float> >` 型．任意のベクトル．

TIMESTAMP ：TimeStamp 型．送信される時刻．

出力

OUTPUT ：`ObjectRef` 型．入力と同じものが出力される．

パラメータ

HOST ：`string` 型．データ送信先ホストの IP アドレス．SOCKET_ENABLED が `false` の場合は無効．

PORT ：`int` 型．ソケット番号． SOCKET_ENABLED が `false` の場合は無効．

ADVANCE ：`int` 型．フレームのシフト長．前段処理と同じ値にする．

BUFFER_SIZE ：`int` 型．ソケット通信のために確保するバッファサイズ．

FRAMES_PER_SEND ：`int` 型．ソケット通信を何フレームに一回行うかを指定する．デフォルトは 1．

TIMESTAMP_TYPE ：`string` 型．送信データにスタンプされる時刻の設定．TIMESTAMP_TYPE=GETTIMEOFDAY なら gettimeofday で得た時刻，TIMESTAMP_TYPE=CONSTANT_INCREMENT なら SAMPLING_RATE から計算されるフレーム時間を毎フレーム加算した時刻とする．

SAMPLING_RATE ：`int` 型．サンプリング周波数．デフォルトは 16000．TIMESTAMP_TYPE=CONSTANT_INCREMENT の時のみ有効．

DEBUG_PRINT ：`bool` 型．デバッグ標準出力の ON/OFF

SOCKET_ENABLE ：`bool` 型．`true` でデータをソケットに転送し，`false` で転送しない．

ノードの詳細

(A) パラメータの説明

HOST は，データを送信する外部プログラムが動作するホストのホスト名，または IP アドレスを指定する．
PORT は，データを送信するネットワークポート番号を指定する．
ADVANCE はフレームのシフト長であり，前段処理の設定値と同じにする．
BUFFER_SIZE はソケット通信のために確保するバッファサイズ．BUFFER_SIZE * 1024 の float 型の配列が初期化時に確保される．送信するデータより大きく確保する．
FRAMES_PER_SEND はソケット通信を何フレームに一回行うかを指定する．通常は 1 で問題ないが，通信量を削減したい時に使用できる．
TIMESTAMP_TYPE は送信データにスタンプされる時刻を設定する．
SAMPLING_RATE はサンプリング周波数を指定する．
DEBUG_PRINT はデバッグ標準出力の表示の可否である．送信データの一部と同じ情報が出力される．表示される内容については表 6.15 の「 Debug 」を参照．
SOCKET_ENABLED が `false` のときは，データを外部システムに送信しない．これは，外部プログラムを動かさずに HARK のネットワーク動作チェックを行うために使用する．

(B) データ送信の詳細

(B-1) データ送信用構造体

データの送信は，各フレーム毎に幾つかに分けられて行われる．データ送信のために定義されている構造体を下記にリストアップする．

- **HD_Header**
  説明 : 送信データの先頭で送信される基本情報が入ったヘッダ
  データサイズ : `3 * sizeof(int) + 2 * sizeof(int64)`

表 6.9: HD_Header のメンバ

変数名	型	説明
type	int	送信データの構造を示すビットフラグ．各ビットと送信データとの関係については表 6.10 参照.
advance	int	フレームのシフト長
count	int	HARK のフレーム番号
tv_sec	int64	タイムスタンプ（秒）
tv_usec	int64	タイムスタンプ（マイクロ秒）

表 6.10: HD_Header の type の各ビットと送信データ

桁数	関係入力端子	送信データ
１桁目	MIC_WAVE	音響信号
２桁目	MIC_FFT	周波数スペクトル
３桁目	SRC_INFO	音源定位結果ソース情報
４桁目	SRC_INFO, SRC_WAVE	音源定位結果ソース情報 + 音源 ID 毎の音響信号
５桁目	SRC_INFO, SRC_FFT	音源定位結果ソース情報 + 音源 ID 毎の周波数スペクトル
６桁目	SRC_INFO, SRC_FEATURE	音源定位結果ソース情報 + 音源 ID 毎の音響特徴量
７桁目	SRC_INFO, SRC_RELIABILITY	音源定位結果ソース情報 + 音源 ID 毎のミッシングフィーチャーマスク
８桁目	TEXT	任意の文字列
９桁目	MATRIX	任意の行列
１０桁目	VECTOR	任意のベクトル

HarkDataStreamSender は入力端子の開放の可否によって送信されるデータが異なり，データ受信側では type によって，受信データを解釈できる．以下に例を挙げる．送信されるデータの更なる詳細については (B-2) に示す．

例 1) MIC_FFT 入力端子のみが接続されている場合，type は２進数で表すと 0000000010 となる．また，送信されるデータはマイク毎の周波数スペクトルのみとなる．

例 2) MIC_WAVE, SRC_INFO, SRC_FEATURE の３つの入力端子が接続されている場合，type は２進数で表すと 0000100101 となる．送信されるデータはマイク毎の音響信号，音源定位結果のソース情報，音源 ID 毎の音響特徴量となる．

注) SRC_WAVE, SRC_FFT, SRC_FEATURE, SRC_RELIABILITY の４つの入力端子については，音源 ID ごとの情報になるため，SRC_INFO の情報が必須である．もし，SRC_INFO を接続せずに，上記４つの入力端子を接続したとしても，何も送信されない．その場合，type は２進数で 0000000000 となる．

- **HDH_MicData**
  説明：２次元配列を送信するための，サイズに関する配列の構造情報
  データサイズ：`3 * sizeof(int)`

表 6.11: HDH_MicData のメンバ

変数名	型	説明
nch	int	マイクチャネル数（送信する２次元配列の行数）
length	int	データ長（送信する２次元配列の列数）
data_bytes	int	送信データバイト数．float 型の行列の場合は `nch * length * sizeof(float)` となる．

- **HDH_SrcInfo**
  説明：音源定位結果のソース情報
  データサイズ：`1 * sizeof(int) + 4 * sizeof(float)`

表 6.12: HDH_SrcInfo のメンバ

変数名	型	説明
src_id	int	音源 ID
x[3]	float	音源３次元位置
power	float	LocalizeMUSIC で計算される MUSIC スペクトルのパワー

- **HDH_SrcData**
  説明：１次元配列を送信するための，サイズに関する配列の構造情報
  データサイズ：`2 * sizeof(int)`

表 6.13: HDH_SrcData のメンバ

変数名	型	説明
length	int	データ長（送信する１次元配列の要素数）
data_bytes	int	送信データバイト数．float 型のベクトルの場合は `length * sizeof(float)` となる．

(B-2) 送信データ

送信データは各フレーム毎に，表 6.14，表 6.15 の (a)-(w) のように，分割されて行われる．表 6.14 に，送信データ (a)-(w) と，接続された入力端子の関係を，表 6.15 に，送信データの説明を示す．

(B-3) 送信アルゴリズム

HARK のネットワークファイルを実行する際に繰り返し演算される部分のアルゴリズムを以下に示す．

表 6.14: 送信順のデータリストと接続入力端子（○ 記号の箇所が送信されるデータ，○* は，SRC_INFO 端子が接続されていない場合は送信されないデータ）

送信データ詳細			入力端子と送信データ								
	型	サイズ	MIC_WAVE	MIC_FFT	SRC_INFO	SRC_WAVE	SRC_FFT	SRC_FEATURE	SRC_RELIABILITY	TEXT	MAT
(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									

```
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)
```

表 6.15: 送信データ詳細

	説明	Debug
(a)	送信データヘッダ．表 6.9 参照．	○
(b)	音響信号の構造（マイク数，フレーム長，送信バイト数）を表す構造体．表 6.11 参照．	○
(c)	音響信号（マイク数 × フレーム長 の float 型の行列）	
(d)	周波数スペクトルの構造（マイク数，周波数ビン数，送信バイト数）を表す構造体．表 6.11 参照．	○
(e)	周波数スペクトルの実部（マイク数 × 周波数ビン数 の float 型の行列）	
(f)	周波数スペクトルの虚部（マイク数 × 周波数ビン数 の float 型の行列）	
(g)	検出された音源個数	○
(h)	音源定位結果のソース．表 6.12 参照．	○
(i)	音源 ID 毎の音響信号の構造（アドバンス長，送信バイト数）を表す構造体．表 6.13 参照．	○
(j)	音源 ID 毎の音響信号（アドバンス長 の short int 型の一次元配列）	
(k)	音源 ID 毎の周波数スペクトルの構造（周波数ビン数，送信バイト数）を表す構造体．表 6.13 参照．	○
(l)	音源 ID 毎の周波数スペクトルの実部（周波数ビン数 の float 型の一次元配列）	
(m)	音源 ID 毎の周波数スペクトルの虚部（周波数ビン数 の float 型の一次元配列）	
(n)	音源 ID 毎の音響特徴量の構造（特徴量次元数，送信バイト数）を表す構造体．表 6.13 参照．	○
(o)	音源 ID 毎の音響特徴量（特徴量次元数 の float 型の一次元配列）	
(p)	音源 ID 毎の MFM の構造（特徴量次元数，送信バイト数）を表す構造体．表 6.13 参照．	○
(q)	音源 ID 毎の MFM （特徴量次元数 の float 型の一次元配列）	
(r)	文字列情報（文字数，送信バイト数）を表す構造体．表 6.13 参照．	○
(s)	文字列（文字数 の char 型の一次元配列）	
(t)	行列の構造（行数，列数，送信バイト数）を表す構造体．表 6.11 参照．	○
(u)	行列データ （float 型の行列）	
(v)	ベクトルの構造（ベクトル次元数，送信バイト数）を表す構造体．表 6.13 参照．	○
(w)	ベクトルデータ（float 型の一次元配列）	

ここで，コード内の (a)-(w) が，表 6.14 と表 6.15 の (a)-(w) に対応している．

6.1.7 PlayAudio

ノードの概要

入力された音声波形データの再生を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

分離音声や収録した音声を試聴する場合に使用する．

典型的な接続例

PlayAudio ノードの接続例を図 6.14 と図 6.15 に示す．

図 6.14 は，分離音声を試聴する場合の接続例である．分離音声は， GHDSS などで分離を行った後， Synthesize で時間領域波形に変換したものを入力する．2 チャンネルのステレオ出力であれば， INPUT1 に入力した波形は L チャンネル， INPUT2 に入力した波形は R チャンネルから出力される．INPUT1 および INPUT2 にマルチチャンネルの音声波形が入力された場合，それぞれの合成音が各出力チャンネルから再生される．

図 6.15 は，収録した音声を試聴する場合の接続例である．AudioStreamFromWave などで得られたマルチチャンネル音声波形を INPUT_MULTI_CHANNEL に入力する．出力チャンネルへのアサインは MULTI_CHANNEL_ASSIGN パラメータを用いて行う．

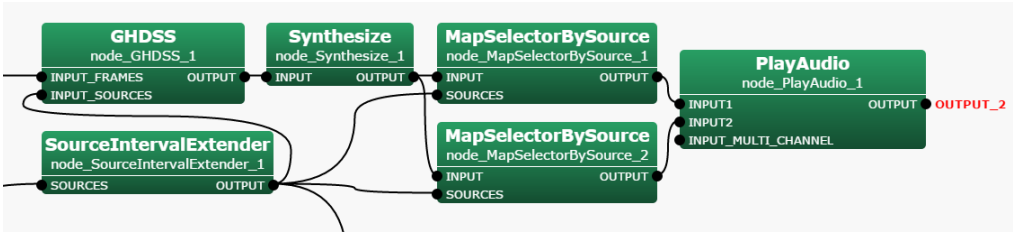


図 6.14: PlayAudio の接続例 1

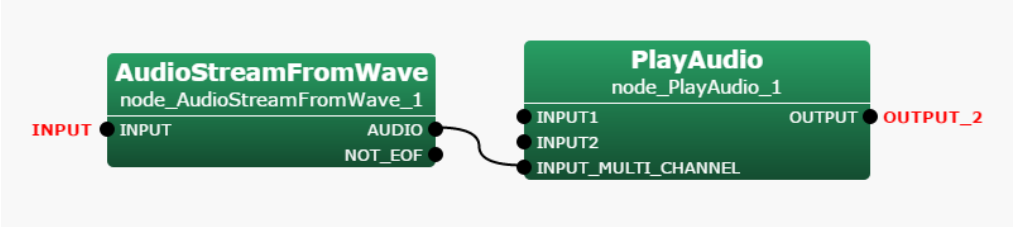


図 6.15: PlayAudio の接続例 2

表 6.16: PlayAudio のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
MASTER_VOLUME	float	0	[dB]	マスターボリュームの設定.
DEVICE	int			デバイスリストの番号でデバイスを指定する. 指定しない場合は OS 標準のデバイスに出力.
CHANNNEL_COUNT	int	2		再生チャンネル数. 最大 2 チャンネルに対応.
MULTI_CHANNEL_ASSIGN	Vector<int>	下記参照		INPUT_MULTI_CHANNEL 使用時のチャンネルアサイン.
LATENCY	int	1000	[msec]	安定再生を図るための遅延時間.
LENGTH	int	512	[pt]	FFT 長.
ADVANCE	int	160	[pt]	シフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリングレート.

ノードの入出力とプロパティ

入力

INPUT1 : 型は Matrix<float>, Map<int, ObjectRef>, または Vector<float> 型. オーディオデバイスのチャンネル 1 から出力する音声データ. 複数チャンネル, 複数音源が入力された場合はミックスして出力する. Map<int, ObjectRef> の ObjectRef は Vector<float>.

INPUT2 : 型は Matrix<float>, Map<int, ObjectRef>, または Vector<float> 型. オーディオデバイスのチャンネル 2 から出力する音声データ. 複数チャンネル, 複数音源が入力された場合はミックスして出力する. Map<int, ObjectRef> の ObjectRef は Vector<float>.

INPUT_MULTI_CHANNEL : 型は Matrix<float> または Map<int, ObjectRef> 型. 出力チャンネルのアサインは MULTI_CHANNEL_ASSIGN で行う.

出力

OUTPUT : Matrix<float> 型. 再生データ. 出力データにパラメータ LATENCY の影響は受けない.

パラメータ

MASTER_VOLUME : float 型. マスターボリュームの設定.

DEVICE : int 型. デバイスリストの番号でデバイスを指定する. 指定しない場合は OS 標準のデバイスに出力する. デバイスリスト番号の確認方法は, ノードの詳細を参照.

CHANNNEL_COUNT : int 型. 再生チャンネル数. 最大 2 チャンネルに対応.

MULTI_CHANNEL_ASSIGN : Vector<int> 型. INPUT_MULTI_CHANNEL 使用時のチャンネルアサイン. 再生チャンネルに対し, 入力データのインデックスまたは音源 ID を 0 ベースで指定する. たとえば, 4 チャンネル収録データの Matrix<float> を入力, そのうちインデックス 1 およびインデックス 2 に対応する音声データを L チャンネルと R チャンネルから出力したい場合, <Vector<int> 1 2> と指定する. 指定したパラメータが入力データサイズおよび CHANNEL_COUNT を上回る場合, 実際の再生パラメータを表示の上, 再生可能な範囲内で再生する. 本パラメータを指定しない場合, 入力データの上位チャンネルから CHANNEL_COUNT で指定したチャンネル数で再生する.

LATENCY : int 型. 安定再生を図るための遅延時間.

LENGTH ：int 型．FFT 長．前段階における値（AudioStreamFromMic，MultiFFT ノードなど）と一致している必要がある．

ADVANCE ：int 型．シフト長．前段階における値（AudioStreamFromMic，MultiFFT ノードなど）と一致している必要がある．

SAMPLING_RATE ：int 型．入力波形のサンプリングレート．

ノードの詳細

基本的には INPUT# または INPUT_MUITI_CHANNEL のどちらかを使用することになるが，両方同時に入力された場合はミックスされた音が出力される．

オーディオデバイスを使用するため，本ノードはネットワークファイルに 1 つのみ設置可能である．

出力チャンネルのアサイン変更方法:

INPUT1， INPUT2 を利用する場合は，ノード同士の接続を変更することでチャンネルのアサインを決定する．INPUT_MULTI_CHANNEL にデータを入力する場合は，マルチチャンネルデータを入力し，ノードのパラメータでチャンネルのアサインを決定する．

デバイスリスト番号の確認方法:

デバイスリスト番号は, Windows 版 HARK で提供している Output Sound Device List を使って確認できる. [スタート] → [プログラム] → [HARK] にある Output Sound Device List をクリックすると，図 6.16 に示すように現在接続中のデバイス名が表示される．

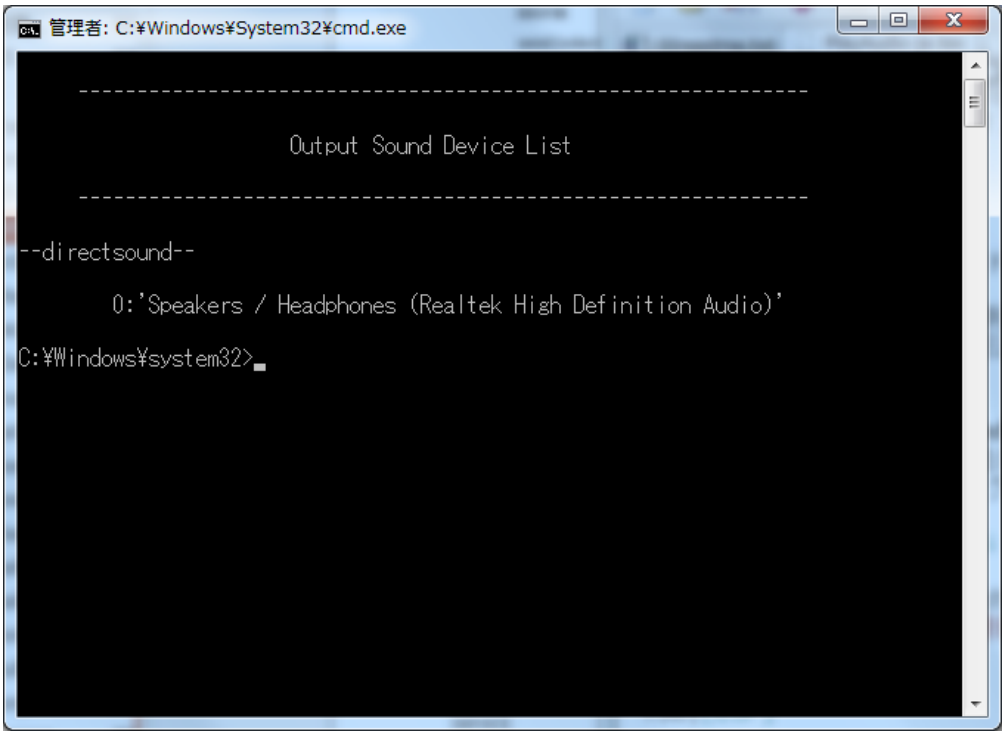


図 6.16: デバイスリスト番号の確認

6.2 Localization カテゴリ

6.2.1 CMLoad

ノードの概要

音源定位のための相関行列をファイルから読み込む.

必要なファイル

CMSave で保存する形式のファイル.

使用方法

どんなときに使うのか

CMSave で保存した音源定位用の相関行列を読み込む時に使用する.

典型的な接続例

図 6.17 に CMLoad ノードの使用例を示す.

- **Version 2.0 以前**
  FILENAMER と FILENAMEI は `string` 型の入力で，それぞれ相関行列の実数部と虚数部の入った読み込みファイル名を表す.
- **Version 2.1 以降**
  zip 形式の相関行列ファイルを読み込む．FILENAMER のみが使用され，FILENAMEI は無視される.

OPERATION_FLAG は `int` 型，または `bool` 型の入力で，相関行列を読み込むタイミングを指定する．使用例では FILENAMER, FILENAMEI, OPERATION_FLAG の全ての入力に対して，Constant ノードを接続しており，実行時のパラメータは不変となっているが，前段のノードの出力を動的にすることで，使用する相関行列を変更することが可能である.

ノードの入出力とプロパティ

表 6.17: CMLoad のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
ENABLE_DEBUG	`bool`	false		デバッグ情報出力の ON/OFF

入力

FILENAMER ：`string` 型．読み込む相関行列の実部のファイル名.

FILENAMEI ：`string` 型．読み込む相関行列の虚部のファイル名.

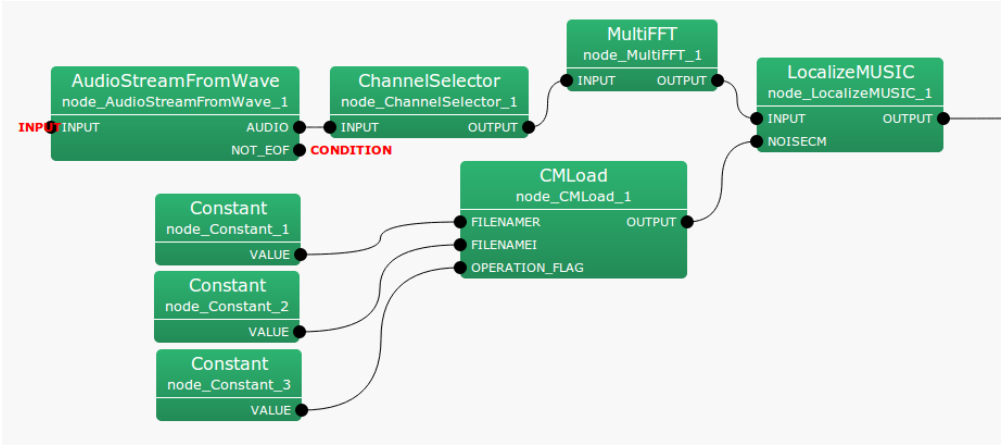


図 6.17: CMLoad の接続例

OPERATION_FLAG ：int 型，または bool 型．本入力端子が 1 もしくは真の時，かつファイル名が変更した時にのみ相関行列が読み込まれる．

出力

OUTPUT ：Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個出力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

パラメータ

ENABLE_DEBUG ：bool 型．false がデフォルト値．true の場合は相関行列が読み込まれる時に標準出力にその旨が出力される．

ノードの詳細

周波数ビン毎の M 次の複素正方行列である相関行列の実部と虚部をそれぞれ二つのファイルから Matrix<float> 形式で読み込む．

周波数ビン数を k とする ($k = NFFT/2+1$) と，読み込みファイルはそれぞれ k 行 M^2 列の行列で構成されている．読み込みは OPERATION_FLAG が 1 もしくは真の時に，ネットワーク動作直後，または読み込みファイル名が変化した時に限り行われる．

6.2.2 CMSave

ノードの概要

音源定位のための相関行列をファイルに保存する．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT や CMMakerFromFFTwithFlag 等から作成した音源定位用の相関行列を保存する時に使用する．

典型的な接続例

図 6.18 に CMSave ノードの使用例を示す．
INPUTCM 入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する．

- **Version 2.0 以前**
  型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している．FILENAMER と FILENAMEI は `string` 型の入力で，それぞれ相関行列の実数部と虚数部の保存ファイル名を表す．

- **Version 2.1 以降**
  zip 形式で保存される．FILENAMER のみが使用され，FILENAMEI は無視される．

OPERATION_FLAG は `int` 型，または `bool` 型の入力で，相関行列を保存するタイミングを指定する（図 6.18 では，一例として Equal ノードを接続しているが，`int` 型や `bool` 型を出力できるノードであれば何でも構わない）．

ノードの入出力とプロパティ

表 6.18: CMSave のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
ENABLE_DEBUG	`bool`	false		デバッグ情報出力の ON/OFF

入力

INPUTCM ：`Matrix<complex<float> >` 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．`Matrix<complex<float> >` の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

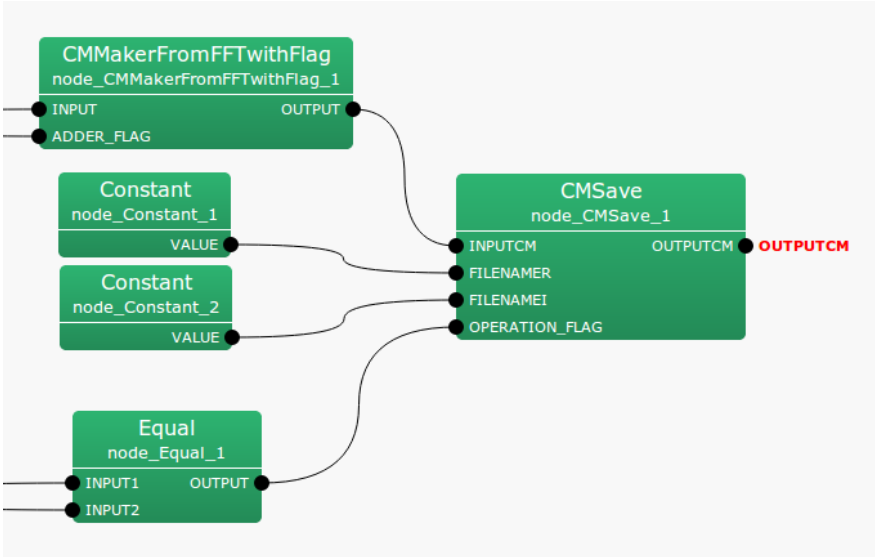


図 6.18: CMSave の接続例

FILENAMER ：`string` 型．保存する相関行列の実部のファイル名．

FILENAMEI ：`string` 型．保存する相関行列の虚部のファイル名．

OPERATION_FLAG ：`int` 型，または `bool` 型．本入力端子が 1 もしくは真の時にのみ相関行列が保存される．

出力

OUTPUTCM ：`Matrix<complex<float> >` 型．INPUTCM に同じ．

パラメータ

ENABLE_DEBUG ：`bool` 型．false がデフォルト値．true の場合は相関行列が保存される時に，標準出力に保存した時のフレーム番号が出力される．

ノードの詳細

- **Version 2.0 以前**
  周波数ビン毎の M 次の複素正方行列である相関行列を `Matrix<float>` 形式に直し，指定したファイル名で保存する．保存ファイルは相関行列の実部と虚部に分割される．周波数ビン数を k とする ($k = NFFT/2+1$) と，保存ファイルはそれぞれ k 行 M^2 列の行列が格納される．

- **Version 2.1 以降**
  周波数ビン毎の M 次の複素正方行列である相関行列を zip 形式で保存する．

保存は OPERATION_FLAG が 1 もしくは真の時に限り行われる．

6.2.3 CMChannelSelector

ノードの概要

マルチチャネルの相関行列から，指定したチャネルのデータだけを指定した順番に取り出す．

必要なファイル

無し．

使用方法

どんなときに使うのか

入力されたマルチチャネルの相関行列の中から，必要のないチャネルを削除したいとき，あるいは，チャネルの並びを入れ替えたいとき．あるいは，チャネルを複製したいとき．

典型的な接続例

図 6.19 に CMChannelSelector ノードの使用例を示す．

入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する（型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している）．

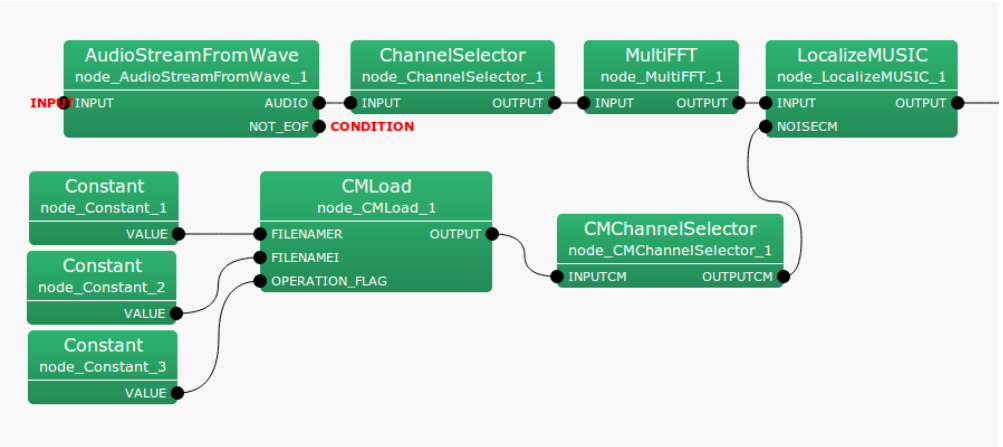


図 6.19: CMChannelSelector の接続例

ノードの入出力とプロパティ

表 6.19: CMChannelSelector のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
SELECTOR	`Vector<int>`	`<Vector<int> >`		出力するチャネルの番号を指定

入力

INPUTCM : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

出力

OUTPUTCM : Matrix<complex<float> > 型．INPUTCM に同じ．

パラメータ

SELECTOR : Vector<int> 型，デフォルト値は無し（<Vector<int> >）．使用するチャネルの，チャネル番号を指定する．チャネル番号は 0 からはじまる．

例: 5 チャネル (0-4) のうち 2，3，4 チャネルだけを使うときは <Vector<int> 2 3 4> のように，3 チャネルと 4 チャネルを入れ替えたい時は <Vector<int> 0 1 2 4 3 5> のように指定する．

ノードの詳細

相関行列が格納された入力データの $k \times M \times M$ 型の複素三次元配列から指定したチャネルの相関行列だけを抽出し，新たな $k \times M' \times M'$ 型の複素三次元配列のデータを出力する．ただし，k は周波数ビン数 ($k = NFFT/2+1$)，M は入力チャネル数，M' は出力チャネル数．

6.2.4 CMMakerFromFFT

ノードの概要

MultiFFT ノードから出力されるマルチチャネル複素スペクトルから，音源定位のための相関行列を一定周期で生成する．

必要なファイル

無し．

使用方法

どんなときに使うのか

LocalizeMUSIC ノードの音源定位において，雑音等の特定の音源を抑圧したい場合は，あらかじめ雑音情報を含む相関行列を用意する必要がある．本ノードは，MultiFFT ノードから出力されるマルチチャネル複素スペクトルから，相関行列を一定周期で生成する．本ノードの出力を LocalizeMUSIC ノードの NOISECM 入力端子に接続することで，一定周期前の情報を常に雑音とみなして抑圧した音源定位が実現できる．

典型的な接続例

図 6.20 に CMMakerFromFFT ノードの使用例を示す．

INPUT 入力端子へは，MultiFFT ノードから計算される入力信号の複素スペクトルを接続する．型は `Matrix<complex<float> >` 型である．本ノードは入力信号の複素スペクトルから周波数ビン毎にチャネル間の相関行列を計算し出力する．出力の型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している．

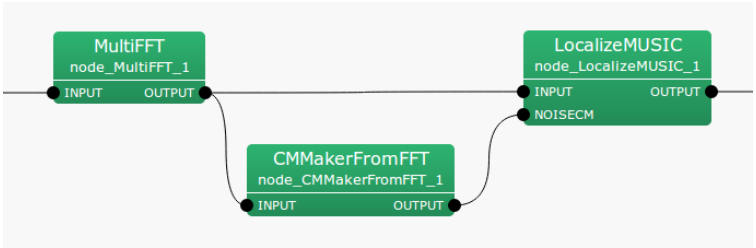


図 6.20: CMMakerFromFFT の接続例

ノードの入出力とプロパティ

入力

INPUT : `Matrix<complex<float> >`, 入力信号の複素スペクトル表現 $M \times (NFFT/2+1)$.

出力

表 6.20: CMMakerFromFFT のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
WINDOW	int	50		相関行列の平滑化フレーム数
PERIOD	int	50		相関行列の更新フレーム周期
WINDOW_TYPE	string	FUTURE		相関行列の平滑化区間
ENABLE_DEBUG	bool	false		デバッグ情報出力の ON/OFF

OUTPUT : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個出力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

OPERATION_FLAG : bool 型．OUPUT から出力される相関行列が更新されている時は true を，それ以外は false を出力する．本出力はデフォルトでは非表示である．表示方法は LocalizeMUSIC の図 6.32 を参照されたい．

パラメータ

WINDOW : int 型．50 がデフォルト値．相関行列計算時の平滑化フレーム数を指定する．ノード内では，入力信号の複素スペクトルから相関行列を毎フレーム生成し，WINDOW で指定されたフレームで加算平均を取ったものが新たな相関行列として出力される．PERIOD フレーム間は最後に計算された相関行列が出力される．この値を大きくすると，相関行列が安定するが計算負荷が高い．

PERIOD : int 型．50 がデフォルト値．相関行列の更新フレーム周期を指定する．ノード内では，入力信号の複素スペクトルから相関行列を毎フレーム生成し，WINDOW で指定されたフレームで加算平均を取ったものが新たな相関行列として出力される．PERIOD フレーム間は最後に計算された相関行列が出力される．この値を大きくすると，相関行列の時間解像度が改善される計算負荷が高い．

WINDOW_TYPE : string 型．FUTURE がデフォルト値．相関行列計算時の平滑化フレームの使用区間を指定する．FUTURE に指定した場合，現在のフレーム f から $f+WINDOW-1$ までが平滑化に使用される．MIDDLE に指定した場合，$f-(WINDOW/2)$ から $f+(WINDOW/2)+(WINDOW\%2)-1$ までが平滑化に使用される．PAST に指定した場合，$f-WINDOW+1$ から f までが平滑化に使用される．

ENABLE_DEBUG : bool 型．false がデフォルト値．true の場合は相関行列が生成される時に，標準出力に生成した時のフレーム番号が出力される．

ノードの詳細

MultiFFT ノードから出力される入力信号の複素スペクトルを以下のように表す．

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

ここで，ω は周波数ビン番号，f は HARK で扱うフレーム番号，M は入力チャネル数を表す．

入力信号 $\boldsymbol{X}(\omega, f)$ の相関行列は，各周波数，各フレームごとに以下のように定義できる．

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

ここで，$()^*$ は複素共役転置演算子を表す．理論上は，この $\boldsymbol{R}(\omega, f)$ をそのまま以降の処理で利用すれば問題はないが，実用上，安定した相関行列を得るため，HARK では，次のように時間方向に平均したものを使用している．

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

平滑化に使用する区間は WINDOW_TYPE パラメータによって変更できる．WINDOW_TYPE=FUTURE の場合，$W_i = 0$, $W_f = \mathrm{WINDOW} - 1$ となる．WINDOW_TYPE=MIDDLE の場合，$W_i = \mathrm{WINDOW}/2$, $W_f = \mathrm{WINDOW}/2 + \mathrm{WINDOW}\%2 - 1$ となる．WINDOW_TYPE=PAST の場合，$W_i = -\mathrm{WINDOW} + 1$, $W_f = 0$ となる．

$\boldsymbol{R}'(\omega, f)$ が CMMakerFromFFT ノードの OUTPUT 端子から PERIOD で指定したフレーム周期ごとに出力される．

6.2.5 CMMakerFromFFTwithFlag

ノードの概要

MultiFFT ノードから出力されるマルチチャネル複素スペクトルから，音源定位のための相関行列を入力フラグで指定した区間で生成する．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT ノードと使用する場面は同じであり，詳細は CMMakerFromFFT ノードの項を参照されたい．相違点は，相関行列の計算区間である．CMMakerFromFFT ノードでは，一定周期 (PERIOD) 毎に相関行列の更新が行われたが，本ノードでは，入力端子から得られるフラグの値に合わせて指定した区間の相関行列を生成することができる．

典型的な接続例

図 6.21 に CMMakerFromFFTwithFlag ノードの使用例を示す．

INPUT 入力端子へは，MultiFFT ノードから計算される入力信号の複素スペクトルを接続する．型は `Matrix<complex<float>>` 型である．ADDER_FLAG は `int` 型，または `bool` 型の入力で，相関行列計算に関するイベントを制御する．イベント制御の詳細はノードの詳細の項に譲る．本ノードは入力信号の複素スペクトルから周波数ビン毎にチャネル間の相関行列を計算し出力する．出力の型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している．

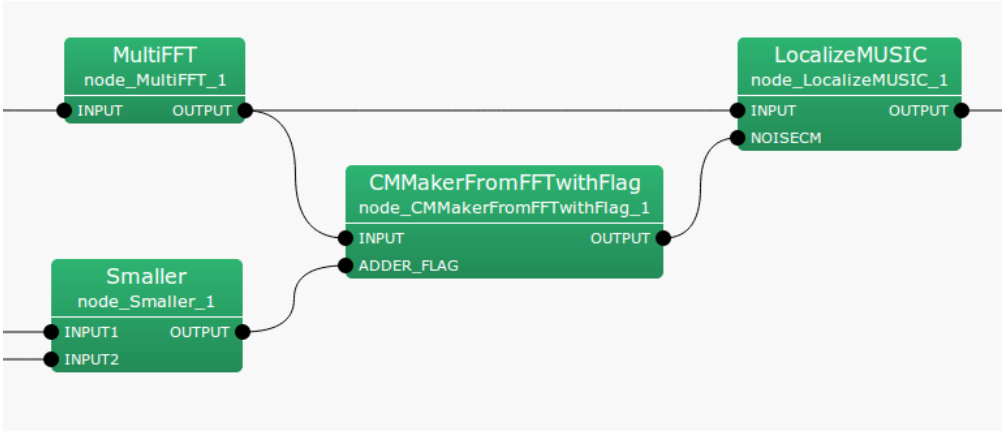


図 6.21: CMMakerFromFFTwithFlag の接続例

表 6.21: CMMakerFromFFTwithFlag のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
DURATION_TYPE	string	FLAG_PERIOD		フラグ値に従うかフレーム周期に従うかの選択
WINDOW	int	50		相関行列の平滑化フレーム数
PERIOD	int	50		相関行列の更新フレーム周期
WINDOW_TYPE	string	FUTURE		相関行列の平滑化区間
MAX_SUM_COUNT	int	100		相関行列の平滑化フレーム数の最大値
ENABLE_ACCUM	bool	false		過去の相関行列との加算平均を取るかの選択
ENABLE_DEBUG	bool	false		デバッグ情報出力の ON/OFF

ノードの入出力とプロパティ

入力

INPUT : Matrix<complex<float> >, 入力信号の複素スペクトル表現 $M \times (NFFT/2+1)$.

ADDER_FLAG : int 型，または bool 型．相関行列計算に関するイベントを制御する．イベント制御の詳細についてはノードの詳細の項を参照されたい．

出力

OUTPUT : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個出力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

OPERATION_FLAG : bool 型．OUPUT から出力される相関行列が更新されている時は true を，それ以外は false を出力する．本出力はデフォルトでは非表示である．表示方法は LocalizeMUSIC の図 6.32 を参照されたい．

パラメータ

DURATION_TYPE : string 型．FLAG_PERIOD がデフォルト値．相関行列を更新する周期と加算平均する区間をフラグ値に従うか，フレーム周期に従うかを選択する．詳細はノードの詳細の項を参照されたい．

WINDOW : int 型．50 がデフォルト値．DURATION_TYPE=WINDOW_PERIOD の時のみ指定する．相関行列計算時の平滑化フレーム数を指定する．ノード内では，入力信号の複素スペクトルから相関行列を毎フレーム生成し，WINDOW で指定されたフレームで加算平均を取ったものが新たな相関行列として出力される．PERIOD フレーム間は最後に計算された相関行列が出力される．この値を大きくすると，相関行列が安定するが計算負荷が高い．

PERIOD : int 型．50 がデフォルト値．DURATION_TYPE=WINDOW_PERIOD の時のみ指定する．相関行列の更新フレーム周期を指定する．ノード内では，入力信号の複素スペクトルから相関行列を毎フレーム生成し，WINDOW で指定されたフレームで加算平均を取ったものが新たな相関行列として出力される．PERIOD フレーム間は最後に計算された相関行列が出力される．この値を大きくすると，相関行列の時間解像度が改善されるが計算負荷が高い．

WINDOW_TYPE : string 型．FUTURE がデフォルト値．相関行列計算時の平滑化フレームの使用区間を指定する．FUTURE に指定した場合，現在のフレーム f から $f+WINDOW-1$ までが平滑化に使用され

る．MIDDLE に指定した場合，$f-(WINDOW/2)$ から $f+(WINDOW/2)+(WINDOW\%2)-1$ までが平滑化に使用される．PAST に指定した場合，$f-WINDOW+1$ から f までが平滑化に使用される．

MAX_SUM_COUNT ：`int` 型．100 がデフォルト値．DURATION_TYPE=FLAG_PERIOD の時のみ指定する．相関行列計算時の平滑化フレーム数の最大値を指定する．本ノードは，ADDER_FLAG によって相関行列の平滑化フレーム数を制御できる．このため，ADDER_FLAG が 常に 1 の場合は，相関行列の加算のみが行われ，いつまでも出力されないことになってしまう．そこで，MAX_SUM_COUNT を正しく設定することで，平滑化フレーム数の上限に来た時に強制的に相関行列を出力することができる．この機能を OFF にするには MAX_SUM_COUNT = 0 を指定すれば良い．

ENABLE_ACCUM ：`bool` 型．false がデフォルト値．DURATION_TYPE=FLAG_PERIOD の時のみ指定する．過去に生成した相関行列も含めて加算平均を取るかどうかを指定できる．

ENABLE_DEBUG ：`bool` 型．false がデフォルト値．true の場合は相関行列が生成される時に，標準出力に生成した時のフレーム番号が出力される．

ノードの詳細

CMMakerFromFFT ノードと相関行列算出のアルゴリズムは同じであり，詳細は CMMakerFromFFT のノードの詳細を参照されたい．CMMakerFromFFT ノードとの相違点は相関行列の平滑化フレームを ADDER_FLAG 入力端子のフラグによって制御できることである．

CMMakerFromFFT ノードでは，PERIOD で指定したフレーム数によって以下の式で相関行列を算出していた．

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

ここで，平滑化に使用する区間は WINDOW_TYPE パラメータによって変更できる．WINDOW_TYPE=FUTURE の場合，$W_i = 0, W_f = \mathrm{WINDOW}-1$ となる．WINDOW_TYPE=MIDDLE の場合，$W_i = \mathrm{WINDOW}/2, W_f = \mathrm{WINDOW}/2+\mathrm{WINDOW}\%2-1$ となる．WINDOW_TYPE=PAST の場合，$W_i = -\mathrm{WINDOW}+1, W_f = 0$ となる．

本ノードでは DURATION_TYPE=FLAG_PERIOD の場合，ADDER_FLAG の値によって次のように相関行列を生成する．

A) ADDER_FLAG が 0 （または偽）から 1 （または真）に変化する時

- 相関行列を零行列に戻し，PERIOD を 0 に戻す．
  $\boldsymbol{R}'(\omega) = \boldsymbol{O}$
  PERIOD = 0
  ただし，$\boldsymbol{O} \in \mathbb{C}^{(NFFT/2+1)\times M\times M}$ は零行列を表す．

B) ADDER_FLAG が 1 （または真）を保持する区間

- 相関行列を加算する．
  $\boldsymbol{R}'(\omega) = \boldsymbol{R}'(\omega) + \boldsymbol{R}(\omega, f+i)$
  PERIOD = PERIOD + 1

C) ADDER_FLAG が 1 （または真）から 0 （または偽）に変化する時

- 加算した相関行列の平均を取って OUTPUT から出力する．
  $\boldsymbol{R}_{out}(\omega, f) = \frac{1}{\mathrm{PERIOD}}\boldsymbol{R}'(\omega)$

D) ADDER_FLAG が 0 （または偽）を保持する区間

- 最後に生成した相関行列を保持する．
  $\boldsymbol{R}_{out}(\omega, f)$

ここで，$\boldsymbol{R}_{out}(\omega, f)$ が OUTPUT 端子から出力される相関行列となる．つまり，新たな相関行列が $\boldsymbol{R}_{out}(\omega, f)$ に格納されるのは C) のフェイズとなる．

また，DURATION_TYPE=WINDOW_PERIOD の場合，ADDER_FLAG が 1 （または真）の時のみ，式 (6.4) が実行され，相関行列の更新が起こる．

6.2.6 CMDivideEachElement

ノードの概要

音源定位のための二つの相関行列を成分ごとに除算する．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT，CMMakerFromFFTwithFlag から作成した音源定位用の相関行列の演算ノードの一つで，成分毎に除算する機能を持つ．

典型的な接続例

図 6.22 に CMDivideEachElement ノードの使用例を示す．

CMA 入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する（型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している）．CMB 入力端子も CMA と同じく相関行列を接続する．除算の際は，CMA ./ CMB が演算される．ただし ./ は成分ごとの割り算を表す．OPERATION_FLAG は `int` 型，または `bool` 型の入力で，相関行列の除算を計算するタイミングを指定する．

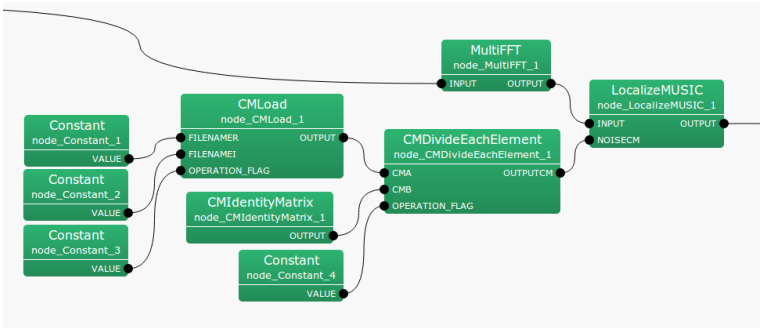


図 6.22: CMDivideEachElement の接続例

ノードの入出力とプロパティ

表 6.22: CMDivideEachElement のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FIRST_FRAME_EXECUTION	bool	false		1 フレーム目だけ演算を実行するかの選択
ENABLE_DEBUG	bool	false		デバッグ情報出力の ON/OFF

入力

CMA : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

CMB : Matrix<complex<float> > 型．CMA に同じ．

OPERATION_FLAG : int 型，または bool 型．本入力端子が 1 もしくは真の時にのみ相関行列の演算が実行される．

出力

OUTPUTCM : Matrix<complex<float> > 型．CMA ./ CMB に相当する除算後の相関行列が出力される．

パラメータ

FIRST_FRAME_EXECUTION : bool 型．false がデフォルト値．true の場合は OPERATION_FLAG が常に 0 または偽であった場合にも 1 フレーム目のみ演算が実行される．

ENABLE_DEBUG : bool 型．false がデフォルト値．true の場合は相関行列が除算される時に，標準出力に計算した時のフレーム番号が出力される．

ノードの詳細

二つの相関行列の成分毎の除算を行う．相関行列の行列としての除算でないことに注意されたい．相関行列は $k \times M \times M$ の複素三次元配列であり，$k \times M \times M$ 回の除算が以下のように行われる．ただし，k は周波数ビン数 ($k = NFFT/2+1$)，M は入力信号のチャネル数である．

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

calculate{

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

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

               ENDFOR
           ENDFOR
       ENDFOR
   ENDIF

}
```

OUTPUTCM 端子から出力される行列は，零行列として初期化され，以降は最後の演算結果を保持する．

6.2.7 CMMultiplyEachElement

ノードの概要

音源定位のための二つの相関行列を成分ごとに乗算する．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT，CMMakerFromFFTwithFlag から作成した音源定位用の相関行列の演算ノードの一つで，成分毎に乗算する機能を持つ．

典型的な接続例

図 6.23 に CMMultiplyEachElement ノードの使用例を示す．

CMA 入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する（型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している）．CMB 入力端子も CMA と同じく相関行列を接続する．乗算の際は，CMA $.*$ CMB が演算される．ただし $.*$ は成分ごとの乗算を表す．OPERATION_FLAG は `int` 型，または `bool` 型の入力で，相関行列の演算を実行するタイミングを指定する．

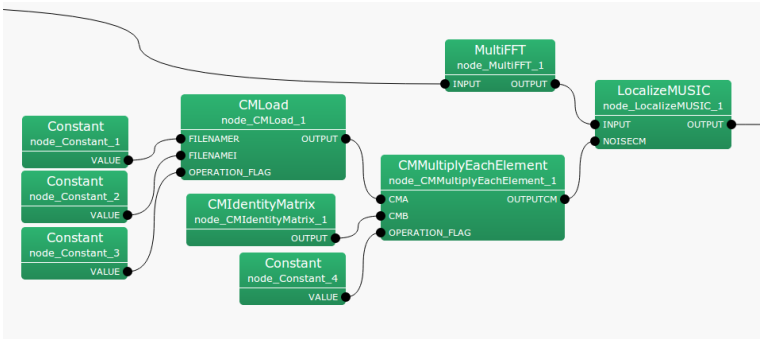


図 6.23: CMMultiplyEachElement の接続例

ノードの入出力とプロパティ

入力

CMA ：`Matrix<complex<float> >` 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．`Matrix<complex<float> >` の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

表 6.23: CMMultiplyEachElement のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FIRST_FRAME_EXECUTION	bool	false		1 フレーム目だけ演算を実行するかの選択
ENABLE_DEBUG	bool	false		デバッグ情報出力の ON/OFF

CMB : Matrix<complex<float> > 型．CMA に同じ．

OPERATION_FLAG : int 型，または bool 型．本入力端子が 1 もしくは真の時にのみ相関行列の演算が実行される．

出力

OUTPUTCM : Matrix<complex<float> > 型．CMA .∗ CMB に相当する乗算後の相関行列が出力される．

パラメータ

FIRST_FRAME_EXECUTION : bool 型．false がデフォルト値．true の場合は OPERATION_FLAG が常に 0 または偽であった場合にも 1 フレーム目のみ演算が実行される．

ENABLE_DEBUG : bool 型．false がデフォルト値．true の場合は相関行列が乗算される時に，標準出力に乗算した時のフレーム番号が出力される．

ノードの詳細

二つの相関行列の成分毎の乗算を行う．相関行列の行列としての乗算でないことに注意されたい（行列としての乗算は CMMultiplyMatrix を参照）．相関行列は $k \times M \times M$ の複素三次元配列であり，$k \times M \times M$ 回の乗算が以下のように行われる．ただし，k は周波数ビン数 ($k = NFFT/2 + 1$)，M は入力信号のチャネル数である．

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

calculate{

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

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

               ENDFOR
           ENDFOR
       ENDFOR
   ENDIF

}
```

OUTPUTCM 端子から出力される行列は，零行列として初期化され，以降は最後の演算結果を保持する．

6.2.8 CMConjEachElement

ノードの概要

相関行列の共役をとる．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT，CMMakerFromFFTwithFlag から作成した音源定位用の相関行列の演算ノードの一つで，成分毎に共役をとる機能を持つ．

典型的な接続例

図 6.24 に CMConjEachElement ノードの使用例を示す．

入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する（型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している）．

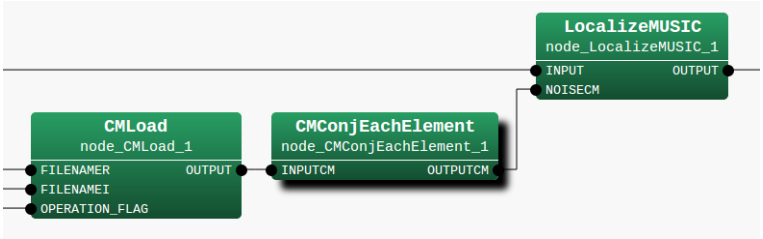


図 6.24: CMConjEachElement の接続例

ノードの入出力とプロパティ

入力

INPUTCM：`Matrix<complex<float> >` 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．`Matrix<complex<float> >` の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

出力

OUTPUTCM：`Matrix<complex<float> >` 型．INPUTCM の共役を取った後の相関行列が出力される．

パラメータ

無し．

ノードの詳細

相関行列の共役を取る．相関行列は $k \times M \times M$ の複素三次元配列であり，$k \times M \times M$ 回の除算が以下のように行われる．ただし，k は周波数ビン数 ($k = NFFT/2 + 1$)，M は入力信号のチャネル数である．

```
calculate{

   FOR i = 1 to k
       FOR i = 1 to M
           FOR i = 1 to M

               OUTPUTCM[i][j][k] = conj(INPUTCM[i][j][k])

           ENDFOR
       ENDFOR
   ENDFOR

}
```

6.2.9 CMInverseMatrix

ノードの概要

音源定位のための相関行列の逆行列を演算する．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT，CMMakerFromFFTwithFlag から作成した音源定位用の相関行列の演算ノードの一つで，相関行列の逆行列を演算する機能を持つ．

典型的な接続例

図 6.25 に CMInverseMatrix ノードの使用例を示す．

INPUTCM 入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する（型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している）．OPERATION_FLAG は `int` 型，または `bool` 型の入力で，相関行列の逆行列を計算するタイミングを指定する．

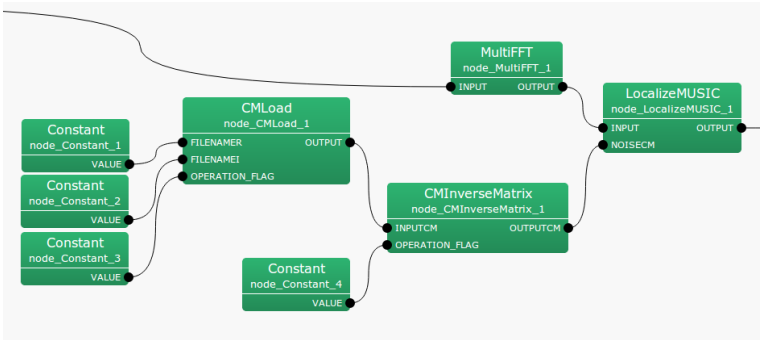


図 6.25: CMInverseMatrix の接続例

ノードの入出力とプロパティ

表 6.24: CMInverseMatrix のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FIRST_FRAME_EXECUTION	bool	false		1 フレーム目だけ演算を実行するかの選択
ENABLE_DEBUG	bool	false		デバッグ情報出力の ON/OFF

入力

INPUTCM : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

OPERATION_FLAG : int 型，または bool 型．本入力端子が 1 もしくは真の時にのみ相関行列の演算が実行される．

出力

OUTPUTCM : Matrix<complex<float> > 型．逆行列演算後の相関行列が出力される．

パラメータ

FIRST_FRAME_EXECUTION : bool 型．false がデフォルト値．true の場合は OPERATION_FLAG が常に 0 または偽であった場合にも 1 フレーム目のみ演算が実行される．

ENABLE_DEBUG : bool 型．false がデフォルト値．true の場合は相関行列が計算される時に，標準出力に計算した時のフレーム番号が出力される．

ノードの詳細

相関行列の逆行列の演算を行う．相関行列は $k \times M \times M$ の複素三次元配列であり，k 回の逆行列演算が以下のように行われる．ただし，k は周波数ビン数 ($k = NFFT/2+1$)，M は入力信号のチャネル数である．

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

calculate{

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

}
```

OUTPUTCM 端子から出力される行列は，零行列として初期化され，以降は最後の演算結果を保持する．

6.2.10 CMMultiplyMatrix

ノードの概要

音源定位のための二つの相関行列を周波数ビン毎に乗算する．

必要なファイル

無し．

使用方法

どんなときに使うのか

CMMakerFromFFT，CMMakerFromFFTwithFlag から作成した音源定位用の相関行列の演算ノードの一つで，周波数ビン毎の相関行列を乗算する機能を持つ．

典型的な接続例

図 6.26 に CMMultiplyMatrix ノードの使用例を示す．

CMA 入力端子へは，CMMakerFromFFT や CMMakerFromFFTwithFlag 等から計算される相関行列を接続する（型は `Matrix<complex<float> >` 型だが，相関行列を扱うため，三次元複素配列を二次元複素行列に変換して出力している）．CMB 入力端子も CMA と同じく相関行列を接続する．乗算の際は，CMA ∗ CMB が周波数ビン毎に演算される．OPERATION_FLAG は `int` 型，または `bool` 型の入力で，相関行列の演算を実行するタイミングを指定する．

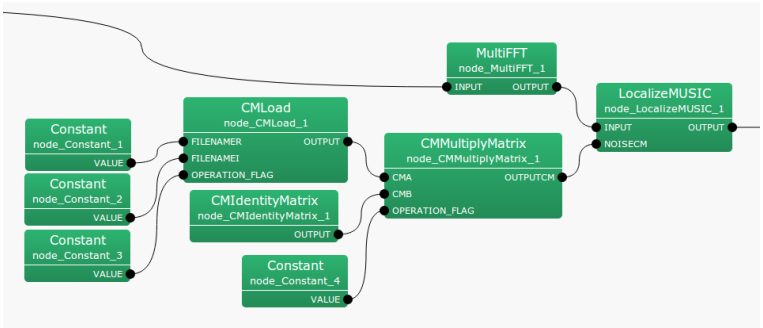


図 6.26: CMMultiplyMatrix の接続例

ノードの入出力とプロパティ

表 6.25: CMMultiplyMatrix のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FIRST_FRAME_EXECUTION	`bool`	false		1 フレーム目だけ演算を実行するかの選択
ENABLE_DEBUG	`bool`	false		デバッグ情報出力の ON/OFF

入力

CMA : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

CMB : Matrix<complex<float> > 型．CMA に同じ．

OPERATION_FLAG : int 型，または bool 型．本入力端子が 1 もしくは真の時にのみ相関行列の演算が実行される．

出力

OUTPUTCM : Matrix<complex<float> > 型．CMA $*$ CMB に相当する乗算後の相関行列が出力される．

パラメータ

FIRST_FRAME_EXECUTION : bool 型．false がデフォルト値．true の場合は OPERATION_FLAG が常に 0 または偽であった場合にも 1 フレーム目のみ演算が実行される．

ENABLE_DEBUG : bool 型．false がデフォルト値．true の場合は相関行列が乗算される時に，標準出力に乗算した時のフレーム番号が出力される．

ノードの詳細

周波数ビン毎の二つの相関行列の乗算を行う．相関行列は $k \times M \times M$ の複素三次元配列であり，k 回の行列の乗算が以下のように行われる．ただし，k は周波数ビン数 ($k = NFFT/2+1$)，M は入力信号のチャネル数である．

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

calculate{

   IF OPERATION_FLAG
       FOR i = 1 to k

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

       ENDFOR
   ENDIF

}
```

OUTPUTCM 端子から出力される行列は，零行列として初期化され，以降は最後の演算結果を保持する．

6.2.11 CMIdentityMatrix

ノードの概要

単位行列が格納された相関行列を出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

LocalizeMUSIC ノードの NOISECM 入力端子に接続することで，LocalizeMUSIC ノードが持つ雑音抑圧機能を OFF にすることができる．

典型的な接続例

図 6.27 に CMIdentityMatrix ノードの使用例を示す．

本ノードは全ての周波数ビンに対して単位行列を持つ相関行列データをノード内で生成するため，入力端子は存在しない．出力端子から生成された相関行列が出力される．

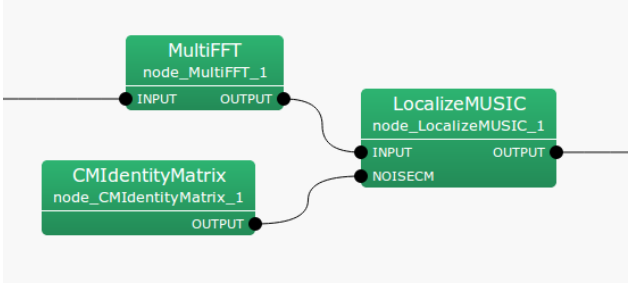


図 6.27: CMIdentityMatrix の接続例

ノードの入出力とプロパティ

表 6.26: CMIdentityMatrix のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
NB_CHANNELS	int	8		入力信号のチャネル数 M
LENGTH	int	512		処理を行う基本単位となるフレームの長さ $NFFT$

入力

無し．

出力

OUTPUT : Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の単位行列である相関行列が $NFFT/2+1$ 個出力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．

パラメータ

NB_CHANNELS : int 型．入力信号のチャネル数．相関行列の次数と等価．前段で使用していた相関行列の次元を合わせる必要がある．8 がデフォルト値．

LENGTH : int 型．512 がデフォルト値．フーリエ変換の際の FFT 点数．前段までの FFT 点数と合わせる必要がある．

ノードの詳細

周波数ビン毎の M 次の複素正方行列である相関行列に対して，単位行列を格納し Matrix<complex<float> > 形式に直して出力する．

6.2.12 ConstantLocalization

ノードの概要

一定の音源定位結果を出力し続けるノード．パラメータは，ANGLES, ELEVATIONS, POWER, MIN_ID であり，それぞれに音源が到来する方位角 (ANGLES)，仰角 (ELEVATIONS)，パワー (POWER)，音源番号 (MIN_ID) を設定する．各パラメータは `Vector` なので，複数の定位結果を出力させることも可能である．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源定位結果が既知の場合の評価をするときに用いる．例えば，音源分離の処理結果を評価する場合に，分離処理に問題があるのか，音源定位誤差に問題があるのかを判断したいときや，音源定位を同じ条件にした状態での音源分離の性能評価をしたい場合など．

典型的な接続例

図 6.28 , 6.29 に接続例を示す．このネットワークでは，一定の定位結果を Iterate のノードパラメータに設定した回数だけ表示する．

ノードの入出力とプロパティ

入力

無し．

出力

SOURCES : `Vector< ObjectRef >` 型．固定の音源定位結果を出力する．`ObjectRef` が参照するのは，`Source` 型のデータである．

パラメータ

表 6.27: ConstantLocalization のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
ANGLES	`Object`	`<Vector<float> >`	[deg]	音源の方位角 (左右の向き)
ELEVATIONS	`Object`	`<Vector<float> >`	[deg]	音源の仰角 (上下の向き)
POWER	`Object`	`<Vector<float> >`	[dB]	音源のパワー
MIN_ID	`int`	0		音源番号

ANGLES : `Vector< float >` 型．音源が到来している方向の，方位角 (左右) を表す．角度の単位は degree.

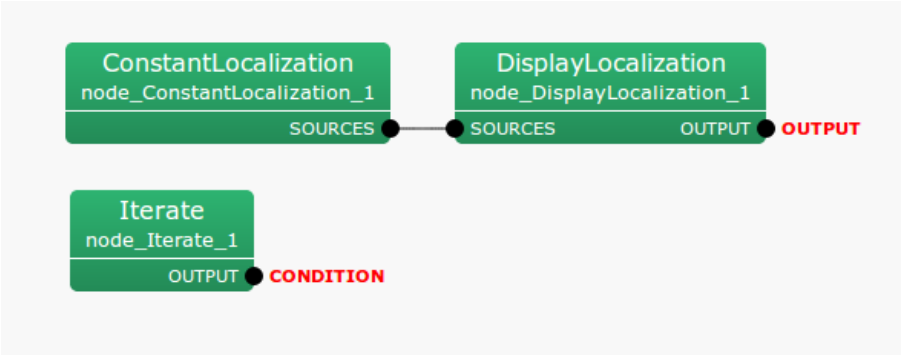


図 6.28: ConstantLocalization の接続例: LOOP0 の内部

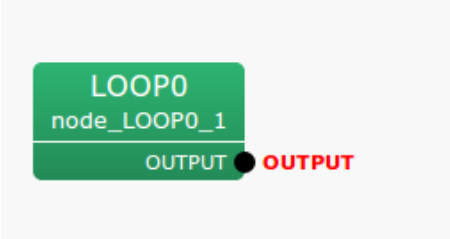


図 6.29: ConstantLocalization の接続例: MAIN

ELEVATIONS : `Vector< float >` 型．音源が到来している方向の，仰角 (上下) を表す．角度の単位は degree.

POWER : `Vector< float >` 型．到来音源のパワーを指定する．LocalizeMUSIC が算出する空間スペクトルと同じ次元であり，単位は [dB]．指定しない場合は自動的に 1.0[dB] が代入される．

MIN_ID : `int` 型．各音源に割り振られる最小音源番号を表す．各音源は後段処理で異なる音源と識別されるよう，音源番号はユニークである必要がある．音源番号は ANGLES と ELEVATIONS で指定した第一成分から MIN_ID を最小番号として順に割り振られる．例えば `MIN_ID = 0` とし，`ANGLES = <Vector<float> 0 30>` とした場合，0 度方向の音源には 0 番が，30 度方向の音源には 1 番が振られる．

ノードの詳細

音源数を N，i 番目の音源の方位角 (ANGLE) を a_i，仰角 (ELEVATION) を e_i とする．このとき，パラメータは以下のように記述する．

ANGLES: `<Vector<float>` a_1 ... a_N `>`

ELEVATIONS: `<Vector<float>` e_1 ... e_N `>`

このように，入力は極座標系で行うが，実際に ConstantLocalization が出力するのは，単位球上の点に対応する，直交座標系の値 (x_i, y_i, z_i) である．極座標系から直交座標系への変換は，以下の式に基づいて行う．

$$x_i = \cos(a_i\pi/180)\cos(e_i\pi/180) \tag{6.5}$$

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

$$z_i = \sin(e_i\pi/180) \tag{6.7}$$

なお, 音源の座標の他に， ConstantLocalization は音源のパワー (POWER で指定．未指定で 1.0 を自動的に代入) と音源番号 (MIN_ID + i) も出力する．

6.2.13 DisplayLocalization

ノードの概要

音源定位結果を GTK ライブラリを使って表示するノードである．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源定位結果を視覚的に確認したいときに用いる．

典型的な接続例

ConstantLocalization や LocalizeMUSIC などの，定位ノードの後に接続する．図 6.30 では，ConstantLocalization からの固定の定位結果を表示し続ける．

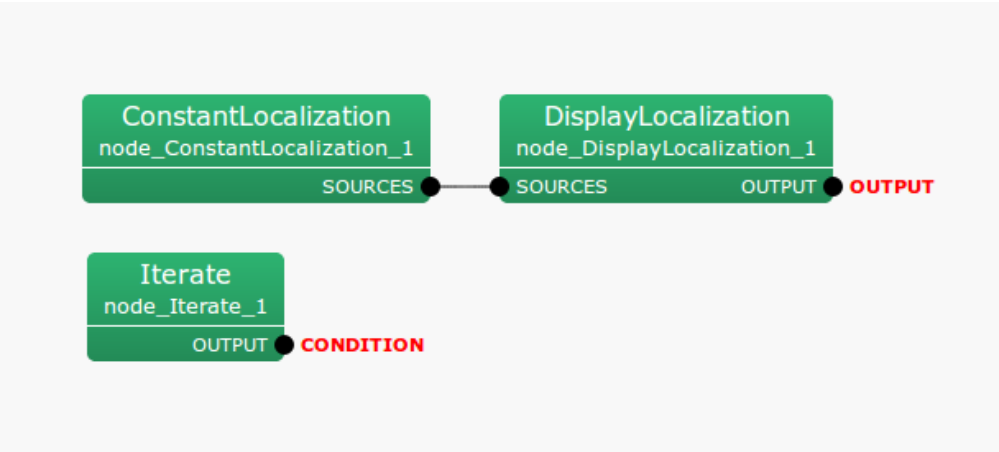


図 6.30: DisplayLocalization の接続例． (ConstantLocalization と同様)

ノードの入出力とプロパティ

入力

SOURCES : `Vector< ObjectRef >` 型．音源位置を表すデータ (`Source` 型) を入力する．

出力

OUTPUT : `Vector< ObjectRef >` 型．入力された値 (`Source` 型) をそのまま出力する．

パラメータ

表 6.28: DisplayLocalization のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
WINDOW_NAME	string	Source Location		音源定位結果を表示するウィンドウ名
WINDOW_LENGTH	int	1000	Frame	音源定位結果を表示するフレーム幅
VERTICAL_RANGE	Vector< int >	下記参照		縦軸に表示する値域の範囲
PLOT_TYPE	string	AZIMUTH		表示するデータの種類

WINDOW_NAME ：string 型．ウィンドウ名．

WINDOW_LENGTH ：int 型．デフォルトは 1000 なので, 表示されるウィンドウの幅は 1000 フレーム．このパラメータを調整することで, 表示する音源定位の時間幅を変えられる．

VERTICAL_RANGE ：Vector< int > 型．縦軸に表示されるデータの値域の範囲．第一成分に最小値，第二成分に最大値を指定する．デフォルトは <Vector<int> -180 180> なので，方位角推定結果を-180 度から 180 度の範囲で表示する．

PLOT_TYPE ：string 型．表示されるデータの種類．AZIMUTH に指定すると方位角推定結果を，ELEVATION に指定すると仰角推定結果を表示する．

ノードの詳細

表示される色は， 赤，緑，青の 3 色．ID の値が 1 増えるごとに順番に色が変わっていく．例えば， ID が 0 なら赤，1 なら緑， 2 なら青，3 なら赤．

6.2.14 LocalizeMUSIC

ノードの概要

マルチチャネルの音声波形データから，MUltiple SIgnal Classification (MUSIC) 法を用いて，マイクロホンアレイ座標系で水平面方向での音源方向を推定する．HARK における音源定位のメインノードである．

必要なファイル

ステアリングベクトルからなる定位用伝達関数ファイルが必要．マイクロホンと音源の位置関係，もしくは，測定した伝達関数に基づき生成する．

使用方法

どんなときに使うのか

本ノードは MUSIC 法 によって，どの方向にどのくらいのパワーの音があるかを推定する．大きなパワーを持つ方向を各フレームで検出することで，音源の方向や，音源数，発話区間などをある程度知ることが可能である．本ノードから出力される定位結果が，後段の音源追跡や音源分離に利用される．

典型的な接続例

典型的な接続例を図 6.31 に示す．

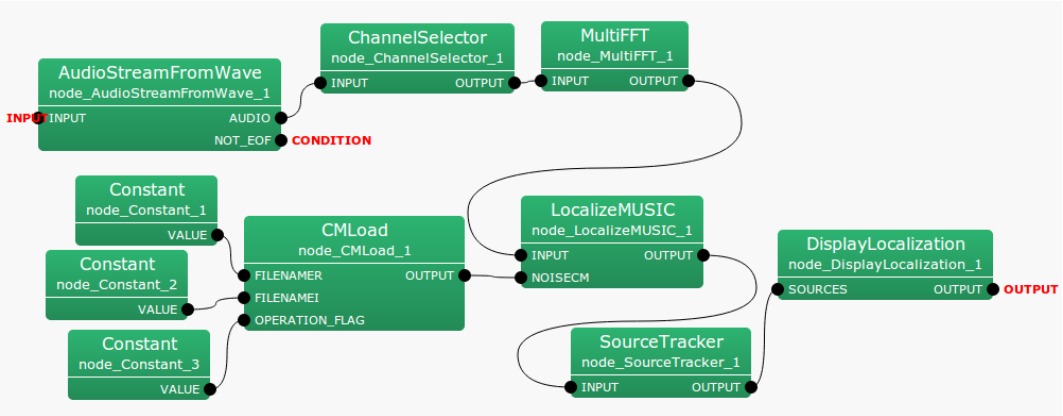


図 6.31: LocalizeMUSIC の接続例

ノードの入出力とプロパティ

入力

INPUT : `Matrix<complex<float> >`, 入力信号の複素周波数表現 $M \times (NFFT/2+1)$.

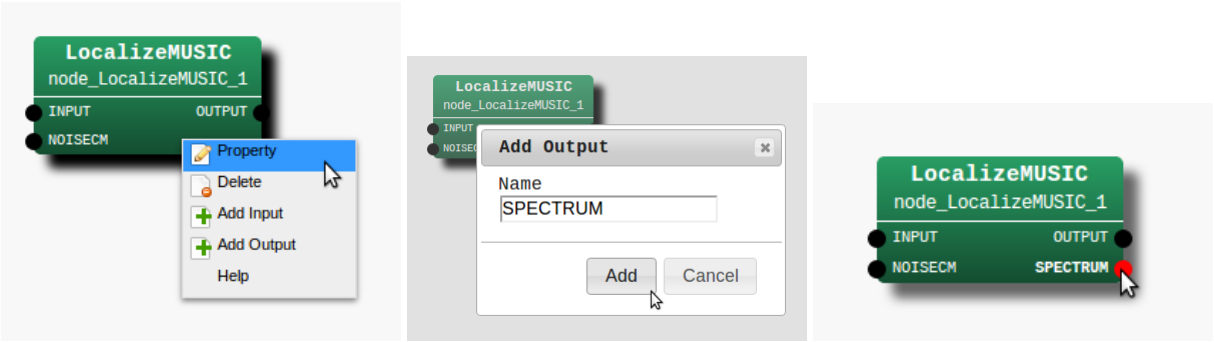
NOISECM ：Matrix<complex<float> > 型．各周波数ビン毎の相関行列．M 次の複素正方行列である相関行列が $NFFT/2+1$ 個入力される．Matrix<complex<float> > の行は周波数 ($NFFT/2+1$ 行) を，列は複素相関行列 ($M*M$ 列) を表す．本入力端子は開放することも可能であり，開放した場合は相関行列に単位行列が用いられる．

出力

OUTPUT ：Vector<ObjectRef> 型で音源位置（方向）を表す．ObjectRef は，Source であり，音源位置とその方向の MUSIC スペクトルのパワーからなる構造体である．Vector の要素数は音源数 (N)．MUSIC スペクトルの詳細については，ノードの詳細を参照されたい．

SPECTRUM ：Vector<float> 型．各方向毎の MUSIC スペクトルのパワー．式 (6.16) の $\bar{P}(\theta)$ に相当する．三次元音源定位の場合は θ が三次元となる．出力形式についてはノードの詳細を参照．本出力端子は，デフォルトでは非表示である．

非表示出力の追加方法は図 6.32 を参照されたい．



Step 1: LocalizeMUSIC を右クリックし，Add Output をクリック

Step 2: Outputs の入力フォームに SPECTRUM を記入し，Add をクリック

Step 3: ノードに SPECTRUM 出力端子が追加される

図 6.32: 非表示出力の使用例 : SPECTRUM 端子の表示

パラメータ

MUSIC_ALGORITHM ：string 型．MUSIC 法において，信号の部分空間を計算するために使うアルゴリズムの選択．SEVD は標準固有値分解を，GEVD は一般化固有値分解を，GSVD は一般化特異値展開を表す．LocalizeMUSIC は，NOISECM 端子から雑音情報を持つ相関行列を入力することで，その雑音を白色化（抑圧）した音源定位ができる機能を持つ．SEVD はその機能がついていない音源定位を実現する．SEVD を選択した場合は NOISECM 端子からの入力は無視される．GEVD と GSVD は共に NOISECM 端子から入力された雑音を白色化する機能を持つが，GEVD は GSVD に比べて雑音抑圧性能が良好だが計算時間がおよそ 4 倍かかる問題を持つ．使用したい場面や計算機環境に合わせて，三つのアルゴリズムを適宜使い分けられる．アルゴリズムの詳細については，ノードの詳細を参照されたい．

TF_CHANNEL_SELECTION ：Vector<int> 型．定位用伝達関数ファイルに格納されているマルチチャネルのステアリングベクトルの中で，指定したチャネルのステアリングベクトルを選択するパラメータである．ChannelSelector と同様に，チャネル番号は 0 から始まる．デフォルトでは 8 チャネルの信号処理を想定し，

表 6.29: LocalizeMUSIC のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
MUSIC_ALGORITHM	string	SEVD		MUSIC のアルゴリズム
TF_CHANNEL_SELECTION	Vector<int>	下記参照		使用チャネル番号
LENGTH	int	512	[pt]	FFT 点数 ($NFFT$)
SAMPLING_RATE	int	16000	[Hz]	サンプリングレート
A_MATRIX	string			定位用伝達関数ファイル名
WINDOW	int	50	[frame]	相関行列の平滑化フレーム数
WINDOW_TYPE	string	FUTURE		相関行列の平滑化区間
PERIOD	int	50	[frame]	定位結果を算出する周期
NUM_SOURCE	int	2		MUSIC で仮定する音源数
MIN_DEG	int	-180	[deg]	ピーク探索方位角の最小値
MAX_DEG	int	180	[deg]	ピーク探索方位角の最大値
LOWER_BOUND_FREQUENCY	int	500	[Hz]	使用周波数帯域の最小値
UPPER_BOUND_FREQUENCY	int	2800	[Hz]	使用周波数帯域の最大値
SPECTRUM_WEIGHT_TYPE	string	Uniform		定位周波数重みの種類
A_CHAR_SCALING	float	1.0		A 特性重みの伸展係数
MANUAL_WEIGHT_SPLINE	Matrix<float>	下記参照		スプライン重みの係数
MANUAL_WEIGHT_SQUARE	Matrix<float>	下記参照		矩形重みの周波数転換点
ENABLE_EIGENVALUE_WEIGHT	bool	true		固有値重みの有無
ENABLE_INTERPOLATION	bool	false		伝達関数補間の有無
INTERPOLATION_TYPE	string	FTDLI		伝達関数補間手法
HEIGHT_RESOLUTION	float	1.0	[deg]	仰角の補間間隔
AZIMUTH_RESOLUTION	float	1.0	[deg]	方位角の補間間隔
RANGE_RESOLUTION	float	1.0	[m]	半径の補間間隔
PEAK_SEARCH_ALGORITHM	string	LOCAL_MAXIMUM		音源探索アルゴリズム
MAXNUM_OUT_PEAKS	int	-1		最大出力音源数
DEBUG	bool	false		デバッグ出力の ON/OFF

<Vector<int> 0 1 2 3 4 5 6 7> と設定されている．本パラメータの成分数 (M) を入力信号のチャネル数と合わせる必要がある．また，INPUT 端子に入力されるチャネルの順序と TF_CHANNEL_SELECTION のチャネル順序を合わせる必要がある．

LENGTH ：int 型．512 がデフォルト値．フーリエ変換の際の FFT 点数．前段までの FFT 点数と合わせる必要がある．

SAMPLING_RATE ：int 型．16000 がデフォルト値．入力音響信号のサンプリング周波数．LENGTH と同様，他のノードとそろえる必要がある．

A_MATRIX ：string 型．デフォルト値はなし．定位用伝達関数ファイルのファイル名を指定する．絶対パスと相対パスの両方に対応している．定位用伝達関数ファイルの作成方法については，harktool4 を参照．

WINDOW ：int 型．50 がデフォルト値．相関行列計算時の平滑化フレーム数を指定する．ノード内では，入力信号の複素スペクトルから相関行列を毎フレーム生成し，WINDOW で指定されたフレームで加算平均を取る．この値を大きくすると，相関行列が安定するが，区間が長い分，時間遅れが長くなる．

WINDOW_TYPE : `string` 型．FUTURE がデフォルト値．相関行列計算時の平滑化フレームの使用区間を指定する．FUTURE に指定した場合，現在のフレーム f から $f + WINDOW - 1$ までが平滑化に使用される．MIDDLE に指定した場合，$f - (WINDOW/2)$ から $f + (WINDOW/2) + (WINDOW\%2) - 1$ までが平滑化に使用される．PAST に指定した場合，$f - WINDOW + 1$ から f までが平滑化に使用される．

PERIOD : `int` 型．50 がデフォルト値．音源定位結果算出の周期をフレーム数で指定する．この値が大きいと，定位結果を得るための時間間隔が大きくなり，発話区間が正しく取りにくくなったり，移動音源の追従性が悪くなる．ただし，小さくすると計算負荷がかかるため，計算機環境に合わせたチューニングが必要となる．

NUM_SOURCE : `int` 型．2 がデフォルト値．MUSIC 法における信号の部分空間の次元数であり，実用上は，音源定位のピーク検出で強調すべき目的音源数と解釈できる．下記のノード詳細では N_s と表わされている．$1 \leq N_s \leq M - 1$ である必要がある．目的音の音源数に合わせておくことが望ましいが，例えば目的音源数が 3 の場合にも，一つ一つの音源が発音している区間が異なるため，実用上はそれより少ない値を選択すれば十分である．

MIN_DEG : `int` 型．-180 がデフォルト値．音源探索する際の最小角度であり，ノード詳細で θ_{min} として表わされている．0 度がロボット正面方向であり，負値がロボット右手方向，正値がロボット左手方向である．指定範囲は，便宜上 ±180 度としてるが，360 度以上の回り込みにも対応しているので，特に制限はない．

MAX_DEG : `int` 型．180 がデフォルト値．音源探索する際の最大角度であり，ノード詳細で θ_{max} として表わされている．その他は，MIN_DEG と同様である．

LOWER_BOUND_FREQUENCY : `int` 型．500 がデフォルト値．音源定位のピーク検出時に考慮する周波数帯域の下限であり，ノードの詳細では，ω_{min} で表わされている．$0 \leq \omega_{min} \leq$ SAMPLING_RATE/2 である必要がある．

UPPER_BOUND_FREQUENCY : `int` 型．2800 がデフォルト値．音源定位のピーク検出時に考慮する周波数帯域の上限であり，下記では，ω_{max} で表わされている．$\omega_{min} < \omega_{max} \leq$ SAMPLING_RATE/2 である必要がある．

SPECTRUM_WEIGHT_TYPE : `string` 型．Uniform がデフォルト値．音源定位のピーク検出時に使用する MUSIC スペクトル の周波数軸方向に対する重みの様式を指定する．Uniform は重みづけを OFF に設定する．A_Characteristic は人間の聴覚の音圧感度を模した重み付けを MUSIC スペクトルに与える．Manual_Spline は，MANUAL_WEIGHT_SPLINE で指定した点を補間点とした Cubic スプライン曲線に合わせた重み付けを MUSIC スペクトルに与える．Manual_Square は，MANUAL_WEIGHT_SQUARE で指定した周波数に合わせた矩形重みを生成し，MUSIC スペクトルに付与する．

A_CHAR_SCALING : `float` 型．1.0 がデフォルト値．A 特性重みを周波数軸方向に伸展するスケーリング項を指定する．A 特性重みは人間の聴覚の音圧感度を模しているため，音声帯域外を抑圧するフィルタリングが可能である．A 特性重みは規格値があるが，雑音環境によっては，雑音が音声帯域内に入ってしまい，うまく定位できないことがある．そこで，A 特性重みを周波数軸方向に伸展し，より広い低周波帯域を抑圧するフィルタを構成する．

MANUAL_WEIGHT_SPLINE : `Matrix<float>` 型．
`<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> >` がデフォルト値．2 行 K 列の `float` 値で指定する．K はスプライン補間で使用するための補間点数に相当する．1 行目は周波数を，2 行目はそれに対応した重みを指定する．重み付けは補間点を通るスプライ

ン曲線に合わせて行われる．デフォルト値では 0 [Hz] から 8000[Hz] までの周波数帯域に対して全て 1 となる重みが付与される．

MANUAL_WEIGHT_SQUARE ：`Vector<float>` 型．`<Vector<float> 0.0 2000.0 4000.0 6000.0 8000.0>` がデフォルト値．MANUAL_WEIGHT_SQUARE で指定した周波数によって矩形重みを生成し，MUSIC スペクトルに付与する．MANUAL_WEIGHT_SQUARE の奇数成分から偶数成分までの周波数帯域は 1 の重みを，偶数成分から奇数成分までの周波数帯域は 0 の重みを付与する．デフォルト値では 2000 [Hz] から 4000[Hz]，6000 [Hz] から 8000[Hz] までの MUSIC スペクトルを抑圧することができる．

ENABLE_EIGENVALUE_WEIGHT ：`bool` 型．true がデフォルト値．true の場合，MUSIC スペクトルの計算の際に，相関行列の固有値分解（または特異値分解）から得られる最大固有値（または最大特異値）の平方根を重みとして，付与する．この重みは，MUSIC_ALGORITHM に GEVD や GSVD を選ぶ場合は NOISECM 端子から入力される相関行列の固有値に依存して大きく変化するため，false にするのが良い．

ENABLE_INTERPOLATION ：`bool` 型．false がデフォルト値．A_MATRIX で指定した伝達関数を補間し，音源定位の解像度を改善したい場合に true にする．補間手法は INTERPOLATION_TYPE で指定したものを使う．補間後の伝達関数の解像度は 仰角は HEIGHT_RESOLUTION で，方位角は AZIMUTH_RESOLUTION で，半径は RANGE_RESOLUTION で指定できる．

INTERPOLATION_TYPE ：`string` 型．FTDLI がデフォルト値．伝達関数の補間手法を指定する．

HEIGHT_RESOLUTION ：`float` 型．1.0[deg] がデフォルト値．伝達関数補間の仰角の間隔を指定する．

AZIMUTH_RESOLUTION ：`float` 型．1.0[deg] がデフォルト値．伝達関数補間の方位角の間隔を指定する．

RANGE_RESOLUTION ：`float` 型．1.0[deg] がデフォルト値．伝達関数補間の半径の間隔を指定する．

PEAK_SEARCH_ALGORITHM ：`string` 型．LOCAL_MAXIMUM がデフォルト値．MUSIC スペクトルのピーク探索に使用するアルゴリズムを選択する．LOCAL_MAXIMUM の場合は，探索点の上下左右を用いて，探索点が最大となる点（極大点）が探索される．HILL_CLIMBING の場合は，まず水平面上の方位角のみでピークを探索し，次に探索されたピーク水平角方向にある仰角を用いてピークが探索される．

MAXNUM_OUT_PEAKS ：`int` 型．-1 がデフォルト値．出力最大音源数を表す．0 の場合は，探索された全てのピークが出力される．MAXNUM_OUT_PEAKS > 0 の場合は，パワーの大きなピークから順に MAXNUM_OUT_PEAKS 個が出力される．-1 の場合は，MAXNUM_OUT_PEAKS = NUM_SOURCE として処理される．

DEBUG ：`bool` 型．デバッグ出力の ON/OFF，デバッグ出力のフォーマットは，以下の通りである．まず，フレームで検出された音源数分だけ，音源のインデックス，方向，パワーのセットがタブ区切りで出力される．ID はフレーム毎に 0 から順番に便宜上付与される番号で，番号自身には意味はない．方向 [deg] は小数を丸めた整数が表示される．パワーは MUSIC スペクトルのパワー値（式 (6.16) の $\bar{P}(\theta)$）がそのまま出力される．次に，改行後，“MUSIC spectrum: ” と出力され，式 (6.16) の $\bar{P}(\theta)$ の値が，すべての θ について表示される．

ノードの詳細

MUSIC 法は，入力信号のチャネル間の相関行列の固有値分解を利用して，音源方向の推定を行う手法である．以下にアルゴリズムをまとめる．

伝達関数の生成:

MUSIC 法では，音源から各マイクロホンまでの伝達関数を計測または数値的に求め，それを事前情報として用いる．マイクロホンアレイからみて，θ 方向にある音源 $S(\theta)$ から i 番目のマイク M_i までの周波数領域での伝達関数を $h_i(\theta, \omega)$ とすると，マルチチャネルの伝達関数ベクトルは以下のように表せる．

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

この伝達関数ベクトルを，適当な間隔 $\Delta\theta$ 毎（非等間隔でも可）に，事前に計算もしくは計測によって用意しておく．HARK では，計測によっても数値計算によっても伝達関数ファイルを生成できるツールとして，harktool4 を提供している．具体的な伝達関数ファイルの作り方に関しては harktool4 の項を参照されたい（harktool4 より三次元の伝達関数に対応した）．LocalizeMUSIC ノードでは，この事前情報ファイル（定位用伝達関数ファイル）を A_MATRIX で指定したファイル名で読み込んで用いている．このように伝達関数は，音源の方向ごとに用意することから，方向ベクトル，もしくは，この伝達関数を用いて定位の際に方向に対して走査を行うことから，ステアリングベクトルと呼ぶことがある．

入力信号のチャネル間相関行列の算出:

HARK による音源定位の処理はここから始まる．まず，M チャネルの入力音響信号を短時間フーリエ変換して得られる周波数領域の信号ベクトルを以下のように求める．

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

ここで，ω は周波数，f はフレームを表す．HARK では，ここまでの処理を前段の MultiFFT ノードで行う．

入力信号 $\boldsymbol{X}(\omega, f)$ のチャネル間の相関行列は，各フレーム，各周波数ごとに以下のように定義できる．

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

ここで $()^*$ は複素共役転置演算子を表す．理論上は，この $\boldsymbol{R}(\omega, f)$ をそのまま以降の処理で利用すれば問題はないが，実用上，安定した相関行列を得るため，HARK では，時間方向に平均したものを使用している．

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

平滑化に使用する区間は WINDOW_TYPE パラメータによって変更できる．WINDOW_TYPE=FUTURE の場合，$W_i = 0$, $W_f = \mathrm{WINDOW} - 1$ となる．WINDOW_TYPE=MIDDLE の場合，$W_i = -\mathrm{WINDOW}/2$, $W_f = \mathrm{WINDOW}/2 + \mathrm{WINDOW}\%2 - 1$ となる．WINDOW_TYPE=PAST の場合，$W_i = -\mathrm{WINDOW} + 1$, $W_f = 0$ となる．

信号と雑音の部分空間への分解 :

MUSIC 法では，式 (6.11) で求まった相関行列 $\boldsymbol{R}'(\omega, f)$ の固有値分解，もしくは特異値分解を行い，M 次の空間を，信号の部分空間と，それ以外の部分空間に分解する．

本節以降の処理は計算負荷が高いため，HARK では計算負荷を考慮し，数フレームに一回演算されるように設計されている．LocalizeMUSIC では，この演算周期を PERIOD で指定できる．

LocalizeMUSIC では，部分空間に分解する方法が MUSIC_ALGORITHM によって指定できる．

MUSIC_ALGORITHM を SEVD に指定した場合，以下の標準固有値分解を行う．

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

ここで，$\boldsymbol{E}(\omega, f)$ は互いに直交する固有ベクトルからなる行列 $\boldsymbol{E}(\omega, f) = [\boldsymbol{e}_1(\omega, f), \boldsymbol{e}_2(\omega, f), \cdots, \boldsymbol{e}_M(\omega, f)]$ を，$\boldsymbol{\Lambda}(\omega)$ は各固有ベクトルに対応する固有値を対角成分とした対角行列を表す．なお，$\boldsymbol{\Lambda}(\omega)$ の対角成分 $[\lambda_1(\omega), \lambda_2(\omega), \ldots, \lambda_M(\omega)]$ は降順にソートされているとする．

MUSIC_ALGORITHM を GEVD に指定した場合，以下の一般化固有値分解を行う．

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

ここで，$\boldsymbol{K}(\omega, f)$ は f フレーム目で NOISECM 端子から入力される相関行列を表す．$\boldsymbol{K}(\omega, f)$ との一般化固有値分解により $\boldsymbol{K}(\omega, f)$ に含まれる雑音由来の大きな固有値を白色化することができるため，雑音を抑圧した音源定位が実現できる．

MUSIC_ALGORITHM を GSVD に指定した場合，以下の一般化特異値分解を行う．

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

ここで，$\boldsymbol{E}(\omega, f), \boldsymbol{E}_r(\omega, f)$ は，それぞれ左特異ベクトル，右特異ベクトルからなる行列を表し，$\boldsymbol{\Lambda}(\omega)$ は各特異値を対角成分とした対角行列を表す．

分解によって得た固有ベクトル空間 $\boldsymbol{E}(\omega, f)$ に対応する固有値（または特異値）は音源のパワーと相関があることから，値が大きな固有値に対応した固有ベクトルを取ることで，パワーの大きな目的音の部分空間のみを選択することができる．すなわち，考慮する音源数を N_s とすれば，$[e_1(\omega), \cdots, e_{N_s}(\omega)]$ が音源に対応する固有ベクトル，$[e_{N_s+1}(\omega), \cdots, e_M(\omega)]$ が雑音に対応する固有ベクトルとなる．LocalizeMUSIC では N_s を NUM_SOURCE として指定できる．

MUSIC スペクトルの算出:

音源定位のための MUSIC スペクトルは，雑音に対応した固有ベクトルのみを用いて次のように計算される．

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

右辺の分母は，入力のうち雑音に起因する固有ベクトルと伝達関数の内積を計算している．固有ベクトルによって張られる空間上では，小さい固有値に対応する雑音の部分空間と大きい固有値に対応する目的信号の部分空間は互いに直交するため，もし，伝達関数が目的音源に対応するベクトルであれば，この内積値は理論上 0 になる．よって，$P(\theta, \omega, f)$ は無限大に発散する．実際には，ノイズ等の影響により，無限大には発散しないが，遅延和などのビームフォーミングと比較すると鋭いピークが観測されるため，音源の抽出が容易になる．右辺の分子は正規化を行うための正規化項である．

$P(\theta, \omega, f)$ は，各周波数ごとに得られる MUSIC スペクトルであるため，以下のようにして周波数方向の統合を行う．

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

ここで，$\omega_{min}, \omega_{max}$ は，それぞれ MUSIC スペクトルの周波数方向統合で扱う周波数帯域の下限と上限を示し，LocalizeMUSIC では，それぞれ LOWER_BOUND_FREQUENCY, UPPER_BOUND_FREQUENCY として指定できる．

また，$W_\Lambda(\omega, f)$ は，周波数方向統合の際の固有値重みであり，最大固有値（または最大特異値）の平方根である．LocalizeMUSIC では，固有値重みの有無を ENABLE_EIGENVALUE_WEIGHT によって選択することができ，false の場合は $W_\Lambda(\omega, f) = 1$，true の場合は $W_\Lambda(\omega, f) = \sqrt{\lambda_1(\omega, f)}$ となる．

また，$W_\omega(\omega, f)$ は，周波数方向統合の際の周波数重みであり，LocalizeMUSIC では SPECTRUM_WEIGHT_TYPE でその種類を以下のように指定できる．

- SPECTRUM_WEIGHT_TYPE が Uniform の場合
  一様重みとなり，全ての周波数ビンに対して，$W_\omega(\omega, f) = 1$ となる．

- SPECTRUM_WEIGHT_TYPE が A_Characteristic の場合
  国際電気標準会議が規格している A 特性重み $\mathcal{W}(\omega)$ となる．図 6.33 に A 特性重みの周波数特性を示す．横軸は ω，縦軸は $\mathcal{W}(\omega)$ を表す．LocalizeMUSIC では，規格の周波数特性に対して，周波数方向の伸展スケーリング項 A_CHAR_SCALING を導入している．A_CHAR_SCALING を α とすれば，実際に使用する周波数重みは $\mathcal{W}(\alpha\omega)$ と表すことができる．図 6.33 では，例として，$\alpha = 1$ の場合と $\alpha = 4$ の場合をプロットしている．最終的に MUSIC スペクトルにかかる重みは $W_\omega(\omega, f) = 10^{\frac{\mathcal{W}(\alpha\omega)}{20}}$ となる．例として，図 6.34 に A_CHAR_SCALING = 1 の時の $W_\omega(\omega, f)$ を示す．

- SPECTRUM_WEIGHT_TYPE が Manual_Spline の場合
  MANUAL_WEIGHT_SPLINE で指定した補間点に対してスプライン補間をした曲線にそった周波数重みとなる．MANUAL_WEIGHT_SPLINE は 2 行 k 列の `Matrix<float>` 型で指定し，一行目は周波数を，二行目はその周波数での重みを表す．補間点数 k は何点でも良い．例として，MANUAL_WEIGHT_SPLINE を，
  `<Matrix<float> <rows 2> <cols 3> <data 0.0 4000.0 8000.0 1.0 0.5 1.0> >`
  とした場合，補間点数は 3 で，周波数軸上の 0, 4000, 8000[Hz] の 3 つの周波数において，それぞれ，1, 0.5, 1 の重みをかけるスプライン曲線ができる．その時の $W_\omega(\omega, f)$ を図 6.35 に示す．

- SPECTRUM_WEIGHT_TYPE が Manual_Square の場合
  MANUAL_WEIGHT_SQUARE で指定した周波数で矩形が切り替わる矩形重みにそった周波数重みとなる．MANUAL_WEIGHT_SQUARE は k 次の `Vector<float>` 型で指定し，矩形を切り替えたい周波数を表す．切替点の個数 k は任意である．例として，MANUAL_WEIGHT_SQUARE を，
  `<Vector<float> 0.0 2000.0 4000.0 6000.0 8000.0>`
  とした場合の矩形重み $W_\omega(\omega, f)$ を図 6.36 に示す．この重みを使うことで，UPPER_BOUND_FREQUENCY と LOWER_BOUND_FREQUENCY だけでは指定できない複数の周波数領域を選択できる．

出力端子 SPECTRUM からは，式 (6.16) の $\bar{P}(\theta, f)$ が一次元配列として出力される．θ は三次元定位の場合は三次元となり，SPECTRUM からは三次元の $\bar{P}(\theta, f)$ が一次元配列として変換されて出力される．`Ne`, `Nd`, `Nr` をそれぞれ，仰角数，方位角数，半径数とすると，変換は以下のようになる．

```
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
```

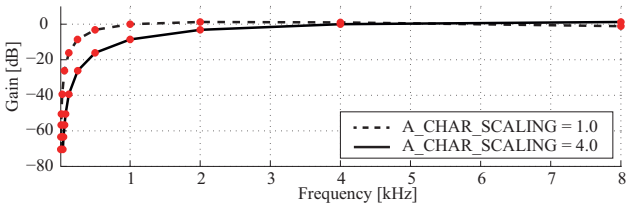


図 6.33: SPECTRUM_WEIGHT_TYPE = A_Charasteristic とした時の A 特性重みの周波数特性

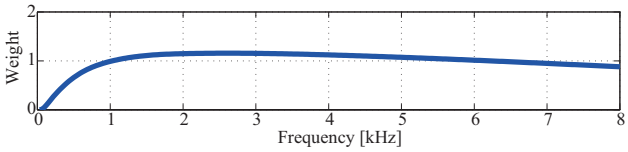


図 6.34: SPECTRUM_WEIGHT_TYPE = A_Charasteristic, A_CHAR_SCALING = 1 とした時の $W_\omega(\omega, f)$

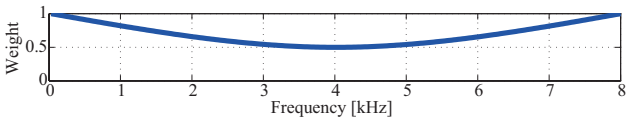


図 6.35: SPECTRUM_WEIGHT_TYPE = Manual_Spline とした時の $W_\omega(\omega, f)$

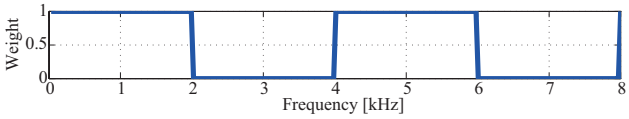


図 6.36: SPECTRUM_WEIGHT_TYPE = Manual_Square とした時の $W_\omega(\omega, f)$

音源の探索:

次に，式 (6.16) の $\bar{P}(\theta, f)$ について θ_{min} から θ_{max} までの範囲からピークを検出し，値の大きい順に，上位 MAXNUM_OUT_PEAKS 個について音源方向に対応する方向ベクトル，および，MUSIC スペクトルのパワーを出力する．また，ピークが MAXNUM_OUT_PEAKS 個に満たない場合は，出力が MAXNUM_OUT_PEAKS 個以下になることもある．ピークの検出アルゴリズムは PEAK_SEARCH_ALGORITHM によって，極大値探索か山登り法かを選択できる．LocalizeMUSIC では，方位角の θ_{min} と θ_{max} をそれぞれ，MIN_DEG と MAX_DEG で指定できる．仰角と半径に関しては全てを用いる．

考察:

終わりに，MUSIC_ALGORITHM に GEVD や GSVD を選んだ場合の白色化が，式 (6.15) の MUSIC スペクトルに与える効果について簡単に述べる．

ここでは，例として，4 人（75 度，25 度，-25 度，-75 度方向）が同時に発話している状況を考える．

図 6.37(a) は，MUSIC_ALGORITHM に SEVD を選択し，雑音を白色化していない音源定位結果である．横軸が方位角，縦軸が周波数，値は式 (6.15) の $P(\theta, \omega, f)$ である．図のように低周波数領域に拡散性雑音と，-150 度方向に方向性雑音があり，正しく 4 話者の方向のみにピークを検出できていないことがわかる．

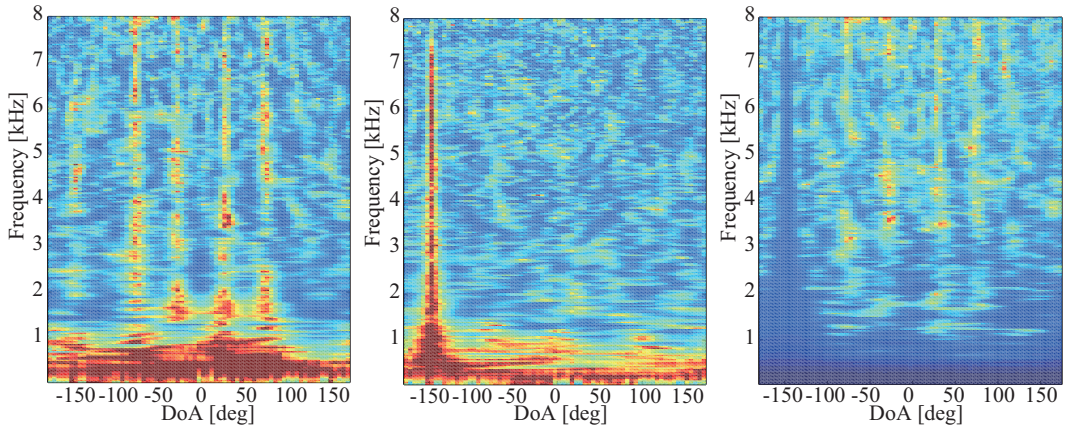
図 6.37(b) は，MUSIC_ALGORITHM に SEVD を選択し，4 話者が発話していない区間の MUSIC スペクトルである．図 6.37(a) で観察された拡散性雑音と方向性雑音が確認できる．

図 6.37(c) は，雑音情報として，図 6.37(b) の情報から $\boldsymbol{K}(\omega, f)$ を生成し，MUSIC_ALGORITHM に GSVD を選択して雑音を白色化した時の MUSIC スペクトルである．図のように，$\boldsymbol{K}(\omega, f)$ に含まれる拡散性雑音と方向性雑音が正しく抑圧され，4 話者の方向のみに強いピークが検出できていることが確認できる．

このように，既知の雑音に対して，GEVD や GSVD を用いることは有用である．

参考文献

(1) Futoshi Asano *et. al*, “Real-Time Sound Source Localization and Separation System and Its Application to Automatic Speech Recognition.” in *Proc. of International Conference on Speech Processing (Eurospeech*



(a) MUSIC_ALGORITHM = SEVD の MUSIC スペクトル（話者数 4）

(b) $\boldsymbol{K}(\omega, f)$ を生成した雑音の MUSIC スペクトル（話者数 0）

(c) MUSIC_ALGORITHM = GSVD の MUSIC スペクトル（話者数 4）

図 6.37: MUSIC スペクトルの比較

2001), pp.1013–1016, 2001.

(2) 大賀 寿郎, 金田 豊, 山崎 芳男, “音響システムとディジタル処理,” 電子情報通信学会.

(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

ノードの概要

SaveSourceLocation ノードで保存された音源定位結果を読み込むノード.

必要なファイル

定位結果を保存したテキストファイル．このファイルを生成するには，例えば SaveSourceLocation を使うとよい.

使用方法

どんなときに使うのか

音源定位した結果を再度使いたいときや, 完全に同じ音源定位結果を用いて複数の音源分離手法を評価するときなどに用いる. ファイル形式が揃えばよいので, 音源定位は別のプログラムで行い, その結果を LoadSourceLocation で渡すことも可能.

典型的な接続例

図 6.38 , 6.39 は，LoadSourceLocation のパラメータ FILENAME で指定した定位結果を読み込んで表示するネットワークである．一行ずつファイルを読み込み，ファイルの最後に到達すると終了する．このように，定位した結果を後で使う場合に用いる.

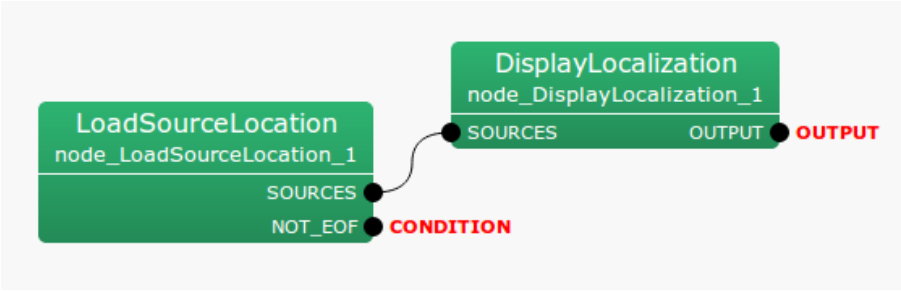


図 6.38: LoadSourceLocation の接続例: LOOP0 の内部

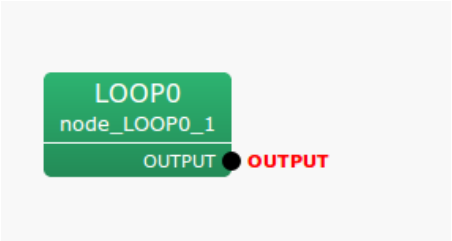


図 6.39: LoadSourceLocation の接続例: MAIN

ノードの入出力とプロパティ

入力

無し.

出力

SOURCES : Vector<ObjectRef> 型．読み込まれた定位結果を，音源定位ノード (LocalizeMUSIC，Constant-Localization など) と同様の形式で出力する．ObjectRef 型が参照するのは， Source 型のデータである．

NOT_EOF : bool 型．ファイルの終端まで読むと false になる出力端子なので，Iterator サブネットワークの終了条件端子に設定するとファイルを最後まで読ませられる．

パラメータ

表 6.30: LoadSourceLocation のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FILENAME	string			読み込むファイルのファイル名

FILENAME : string 型．読み込むファイルのファイル名を設定する．

ノードの詳細

ノードが出力する音源定位結果は次の 5 つのメンバ変数を持ったオブジェクトの Vector である．

1. **パワー:** 100.0 で固定
2. **ID:** ファイルに保存されている，音源の ID
3. **音源位置の** x **座標:** 単位球上の，音源方向に対応する直交座標.
4. **音源位置の** y **座標:** 単位球上の，音源方向に対応する直交座標.
5. **音源位置の** z **座標:** 単位球上の，音源方向に対応する直交座標.

エラーメッセージとその原因は以下のとおり

`FILENAME is empty` ノードパラメータ FILENAME にファイル名が指定されていない.

`Can't open file name` ファイルのオープンに失敗した．原因は読み込み権限が無い，そのファイルが存在しないなど.

6.2.16 NormalizeMUSIC

ノードの概要

LocalizeMUSIC モジュールで計算された MUSIC スペクトルを [0 1] の範囲に正規化し，SourceTracker モジュールによる音源検出を安定化させるモジュール．

必要なファイル

無し．

使用方法

どんなときに使うのか

HARK での音源定位では，各時間・方向 [1] で計算された MUSIC スペクトルに閾値を設定し，閾値を超える MUSIC スペクトルを持つ時間・方向に音源が存在すると判定する．このモジュールは，LocalizeMUSIC モジュールによって計算される MUSIC スペクトルに対して，SourceTracker モジュールで閾値を設定するのが困難な場合に利用することができる．このモジュールは観測音に対する MUSIC スペクトルを [0 1] の範囲に正規化するため，SourceTracker モジュールで設定する閾値は 0.95 などに設定すれば音源定位が安定する．

内部では，正規化用パラメータ推定と，MUSIC スペクトルの正規化処理を行なっている．正規化用パラメータ推定は MUSIC スペクトルがある程度たまると計算され，観測された MUSIC スペクトルから音源が存在する/しない場合の MUSIC スペクトルの分布を得る．推定された正規化用パラメータを利用して，各フレームでの MUSIC スペクトルの正規化処理をパーティクルフィルタによって行う．

典型的な接続例

図 6.40 に NormalizeMUSIC の使用例を示す．図 6.40 では，LocalizeMUSIC モジュールの出力 (OUTPUT: 音源位置や対応する MUSIC スペクトル値などの音源情報，SPECTRUM: 各方向の MUSIC スペクトル) を，NormalizeMUSIC にそれぞれ入力している．NormalizeMUSIC の出力 SOURCES_OUT は，LocalizeMUSIC の出力 OUTPUT と同様に SourceTracker モジュールに接続できる．

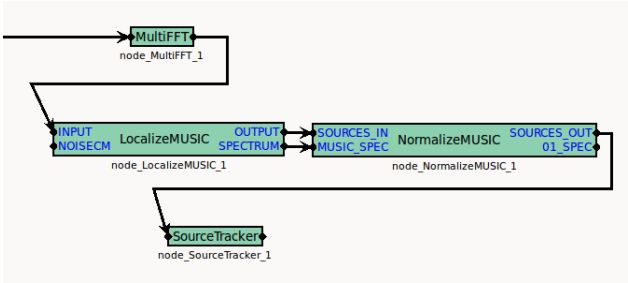


図 6.40: NormalizeMUSIC example

表 6.31: NormalizeMUSIC のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
SHOW_ALL_PARAMETERS	bool	false		INITIAL_THRESHOLD 以外の設定可能なパラメータの表示/非表示を設定.
INITIAL_THRESHOLD	float	30		音源が存在する/しないときの MUSIC スペクトルのおおまかな境目の値．最初の正規化用パラメータ更新にも利用する.
ACTIVE_PROP	float	0.05		正規化用パラメータ更新を行うか否かの閾値.
UPDATE_INTERVAL	int	10		正規化用パラメータ更新の周期，および，更新に利用する MUSIC スペクトルの時間フレーム数.
PRIOR_WEIGHT	float	0.02		正規化用パラメータ更新時の正則化パラメータ.
MAX_SOURCE_NUM	int	2		下記パーティクルフィルタの 1 つのパーティクルが持つ音源数．実際の音源数より少なくても良い.
PARTICLE_NUM	int	100		正規化スペクトル計算時に利用するパーティクルフィルタのパーティクル数.
LOCAL_PEAK_MARGIN	float	0		MUSIC スペクトルが極大になる方向を取得する際，隣接方向同士で無視する MUSIC スペクトルの差.

ノードの入出力とプロパティ

入力

SOURCES_IN ：Vector<ObjectRef> 型．LocalizeMUSIC の OUTPUT と接続．音源情報 (音源位置と対応する MUSIC スペクトル) が格納されている.

MUSIC_SPEC ：Vector<float> 型．LocalizeMUSIC の SPECTRUM と接続．各方向ごとの MUSIC スペクトル値．パラメータ更新や正規化に利用.

出力

SOURCES_OUT ：Vector<ObjectRef> 型．入力の SOURCES_IN と同じ音源情報が入っている．ただし，各音源の MUSIC スペクトルは [0 1] に正規化された値に書き換えられている.

01_SPEC ：Vector<float> 型．入力の MUSIC_SPEC を正規化した値．デバッグなどに使用.

パラメータ

SHOW_ALL_PARAMETERS ：bool 型．デフォルトは false．下記の INITIAL_THRESHOLD 以外のパラメータを変更したい場合は true にして表示させる．多くの場合，INITIAL_THRESHOLD 以外はデフォルト値で問題なく動作する.

[1] 例えば，10 ms の時間解像度，5° おきの方向解像度など.

INITIAL_THRESHOLD：`float` 型．この値は 2 つの役割を果たす．(1) パラメータ更新時に音源が存在する/しない場合の境目の事前知識として利用．(2) 下記 ACTIVE_PROP と併用して最初のパラメータ更新をするかを決定する．

ACTIVE_PROP：`float` 型．デフォルト値は 0.05．MUSIC スペクトルが UPDATE_INTERVAL フレーム集まったとき，その観測値に基づいて正規化用パラメータを更新するかを決める閾値．MUSIC スペクトルが T フレーム，D 方向，合計 TD 個あり，ACTIVE_PROP が θ のとき，INITIAL_THRESHOLD よりも大きな MUSIC スペクトルの値を持つ時間・方向点が θTD 個以上あった場合，正規化用パラメータの更新処理が行われる．

UPDATE_INTERVAL：正規化用パラメータ更新に使う MUSIC スペクトルのフレーム数．ここで指定したフレーム数を周期として更新が行われる．HARK を初期設定で利用した場合，16000 (Hz) サンプリングされた音声信号が MultiFFT モジュールにて，160 (pt)，つまり 0.01 (sec) 間隔で短時間フーリエ変換が行われる．LocalizeMUSIC では，初期設定では 50 フレームおき，つまり 0.5 (sec) おきに MUSIC スペクトルが計算される．したがって，UPDATE_INTERVAL を 10 に設定すると，5 (sec) のデータを使って正規化パラメータの更新が行われる．

PRIOR_WEIGHT：正規化パラメータ更新でのパラメータ計算を安定化させる正則化パラメータ．具体的な意味は下記の **技術的な詳細**，値設定の目安は **トラブルシューティング** をそれぞれ参照されたい．

MAX_SOURCE_NUM：MUSIC スペクトル正規化に用いるパーティクルフィルタで，各パーティクルは音源が存在する方向を仮説として持っている．その時に各パーティクルが持つことのできる音源数の最大値．入力音の実際の音源数にかかわらず，1–3 に設定すると安定する．

PARTICLE_NUM：MUSIC スペクトル正規化に用いるパーティクルフィルタが利用するパーティクル数．経験的には，MUSIC スペクトルが 72 方向 (5° 解像度で水平面 1 周分) の場合，100 程度で十分．より多くの方向 (例えば，仰角方向も考慮し 72×30 方向など) を扱う場合，より多くの粒子数が必要の可能性がある．

LOCAL_PEAK_MARGIN：MUSIC スペクトル正規化のパーティクルフィルタでは，MUSIC スペクトルが極大値をとる方向を利用する．隣接方向の MUSIC スペクトル値と比較する際，無視する MUSIC スペクトルの差を設定する．この値を大きくし過ぎると，どの方向でも MUSIC スペクトルが極大と解釈され，音源の誤検出につながるおそれがある．

ノードの詳細

技術的な詳細: 正規化パラメータの計算や，そのパラメータによる MUSIC スペクトルの正規化の詳細は下記の参考文献を参照されたい．下記の文献中で，正規化パラメータの計算は VB-HMM (変分ベイズ隠れマルコフモデル) の事後分布推定に対応し，MUSIC スペクトルの正規化処理は，パーティクルフィルタによるオンライン確率推定に対応している．

おおまかには，図 6.41 に示すように，観測した MUSIC スペクトルの分布 (図 6.41 中，青い度数分布) から，音源が存在しない場合の分布 (赤) と，音源が存在する場合の分布 (緑) をガウス分布でフィッティングしている．

以下では，この文献中の変数の値と，このノードのパラメータの対応関係を述べる．正規化パラメータ計算用の VB-HMM に対する入力は，フレーム数 T，方向数 D の MUSIC スペクトルである．T は UPDATE_INTERVAL に指定されたフレーム数で，D は LocalizeMUSIC モジュールから出力される方向数である．VB-HMM が用いるハイパーパラメータは $\alpha_0, \beta_0, m_0, a_0, b_0$ がある．これらのうち，α_0, a_0, b_0 は文献と同様に 1 に設定されてい

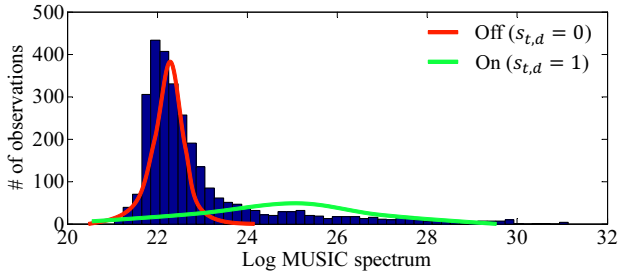


図 6.41: MUSIC スペクトルからのパラメータ学習

る．m_0 としては INITIAL_THRESHOLD を利用し，β_0 は，PRIOR_WEIGHT の値を ε としたとき，$\beta_0 = TD\varepsilon$ とする．

処理の流れ図: 図 6.42 に，NormalizeMUSIC モジュール内の処理の流れずを示す．青線が SOURCES_IN から

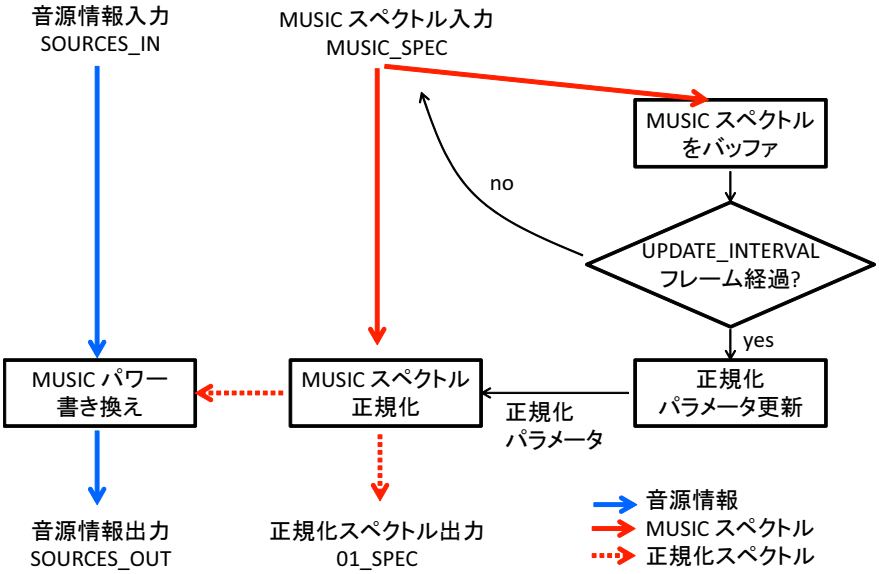


図 6.42: 処理の流れ図

SOURCES_OUT への音源情報のデータの流れ，赤実線が正規化前の MUSIC スペクトル，赤破線が正規化後の MUSIC スペクトルの値の流れを示す．毎フレーム行われる処理は，(1) 最新の正規化パラメータを利用した MUSIC スペクトルの正規化処理と (中央列)，(2) 音源情報内の MUSIC パワー値を正規化したものに置き換える処理 (左列) である．後段の SourceTracker モジュールは，音源情報内の MUSIC パワー値を参照して音源の検出処理を行なっている．

右列は，観測した MUSIC スペクトルからの正規化パラメータ更新処理に相当する．正規化パラメータの更新処理は，次の 2 つの基準が満たされると実行される．

1. MUSIC スペクトルのバッファが UPDATE_INTERVAL フレームだけたまった，かつ
2. 音源が存在する時間・方向点の割合が ACTIVE_PROP を超える．

1. が満たされたとき，フレーム数 T，方向数 D の MUSIC スペクトル $x_{t,d}$ に対して，2. の判定を行う．ACTIVE_PROP の値を θ とする．

最初の更新: プログラムが実行されてから，1 度も正規化パラメータ更新が行われていない場合，$x_{t,d}$ のうち，INITIAL_THRESHOLD を超える時間・方向点の個数が θTD を超える場合，正規化パラメータ更新処理が行われる．
以降の更新: それ以降は，前回の正規化パラメータ更新が観測音の MUSIC スペクトルを反映しているため，正規化された MUSIC スペクトルの合計値が θTD を超える場合に次の正規化パラメータの更新処理が行われる．

トラブルシューティング: ここでは，正しく音源定位・検出が行われない場合のモジュールパラメータ調整の指針を示す．

基本は MUSIC スペクトルを可視化: MUSIC スペクトルが音源が存在するときに高い値，音源が存在しないときは低い値になっているか確認する．可視化の方法は LocalizeMUSIC モジュールのパラメータ DEBUG を true にして，ネットワークを実行した時に標準出力に現れる MUSIC スペクトルの値を適当なツール (例: python+matplotlib，matlab など) で可視化する．HARK クックブック「3.3 うまく定位できない」に同様の説明がある．もし，MUSIC スペクトルの計算結果が信頼出来ない場合，HARK クックブック「8 音源定位」などを参照し，問題点を確認する．LocalizeMUSIC の NUM_SOURCE を 1 にする，LOWER_BOUND_FREQUENCY を 1000 (Hz) に上げることで MUSIC スペクトルの計算が安定化する場合がある．

まず試すこと: パラメータ ACTIVE_PROP，PRIOR_WEIGHT を 0 にし，INITIAL_THRESHOLD を 低め (例: 20 程度) にする．NormalizeMUSIC の SOURCES_OUT を接続した SourceTracker モジュールの THRESH パラメータを 0.95 に設定する．この状態で何も定位できない場合，MUSIC スペクトルが正しく計算されていない可能性が高い．音源が過度に検出される場合，INITIAL_THRESHOLD を 5 刻み程度で上げて調整していく (例: 20 → 25 → 30).

音源が過度に検出される: 考えられる要因は (1) 検出したくない方向の MUSIC スペクトル値が高い，(2) 図 6.41 の音源が存在する緑の分布の平均値が低い，などが考えられる．(1) の場合，雑音相関行列を利用した MUSIC アルゴリズムの利用 (LocalizeMUSIC モジュール参照)，LocalizeMUSIC の LOWER_BOUND_FREQUENCY パラメータを 800–1000 (Hz) に上げるなどで問題が和らぐことがある．後者は特に，音源間が近いため，音源間の MUSIC スペクトル値が高い場合に有用である．(2) の場合，INITIAL_THRESHOLD の値を上げる (例: 30 から 35 に上げてみる)，PRIOR_WEIGHT の値を上げる (例: 0.05–0.1 程度にする),

音源が検出されない: 考えられる要因は (1) 検出するべき方向の MUSIC スペクトル値が低い，(2) INITIAL_THRESHOLD が大きすぎる．(1) の場合，LocalizeMUSIC の調整が必要である．NUM_SOURCES パラメータを実際の音源数と揃える (あるいは $M-1$, $M-2$ など大きめの値を使う，ただし M はマイク数), LOWER_BOUND_FREQUENCY，UPPER_BOUND_FREQUENCY を目的音に適った周波数帯域に設定する．(2) の場合，INITIAL_THRESHOLD を下げてみる．

参考文献

(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) 大塚 琢馬, 中臺 一博, 尾形 哲也, 奥乃 博: 音源定位手法 MUSIC のベイズ拡張, 第 34 回 AI チャレンジ研究会, SIG-Challenge-B102-6, pp.4-25〜4-30, 人工知能学会. [3]

[2] http://winnie.kuis.kyoto-u.ac.jp/members/okuno/Public/Interspeech2011-Otsuka.pdf
[3] http://winnie.kuis.kyoto-u.ac.jp/SIG-Challenge/SIG-Challenge-B102/SIG-Challenge-B102.pdf

6.2.17 SaveSourceLocation

ノードの概要

音源定位結果をファイルに保存するノード．形式は 5.1 に定義されている．

必要なファイル

無し．

使用方法

どんなときに使うのか

定位結果の解析や視覚化などに利用するために，テキストファイルに保存したいときに使う．保存したファイルの読み込みには， LoadSourceLocation ノードを用いる．

典型的な接続例

図 6.43 に典型的な接続例を示す．例で示すネットワークは，ConstantLocalization のノードパラメータに定めた固定の定位結果をファイルに保存する．その他にも，本もジュールは音源定位結果を出力するノードなら何にでも接続できる．例えば LocalizeMUSIC ， ConstantLocalization ， LoadSourceLocation など．

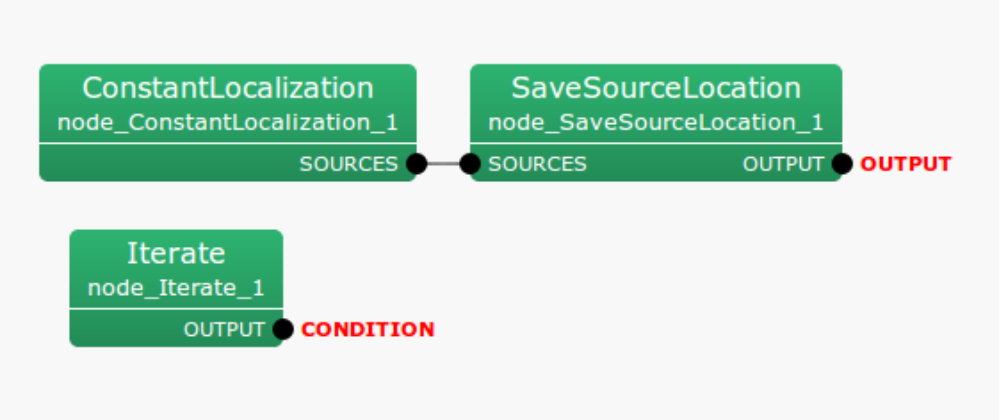


図 6.43: SaveSourceLocation の接続例

ノードの入出力とプロパティ

入力

SOURCES ：`Vector<ObjectRef>` 型．音源定位結果が入力される．`ObjectRef` 型が参照するのは `Source` 型．

出力

OUTPUT ：`Vector<ObjectRef>` 型．入力 (`Source` 型) がそのまま出力される．

パラメータ

FILENAME : `string` 型．デフォルト値はなし．

表 6.32: SaveSourceLocation のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FILENAME	`string`			保存したいファイルの名前

ノードの詳細

エラーメッセージとその原因は以下のとおり

`FILENAME is empty` ノードパラメータ FILENAME にファイル名が指定されていない．

`Can't open file name` ファイルのオープンに失敗した．原因は書き込み権限が無いなど．

6.2.18 SourceIntervalExtender

ノードの概要

音源定位開始時刻を，実際よりも早めたいときに使う．ノードパラメータ PREROLL_LENGTH に与えた分だけ，実際よりも早く定位結果が出力される．

例えば， PREROLL_LENGTH が 5 なら，実際の定位結果が出力される 5 フレーム分前から定位結果が出力される．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源定位の後に音源分離を行うときに，音源定位と音源分離の間に挿入する．なぜなら，音源定位結果は音が発生してから出力するので，実際の音の開始時刻よりもやや遅れてしまう．従って，分離音の先頭が切れてしまう．そこで， SourceIntervalExtender を挿入して強制的に音源定位結果を早く出力させることで, この問題を解決する．

典型的な接続例

図 6.44 に典型的な接続例を示す．図のように，定位結果を元に分離したい場合には，SourceTracker の後に SourceIntervalExtender を挿入する．

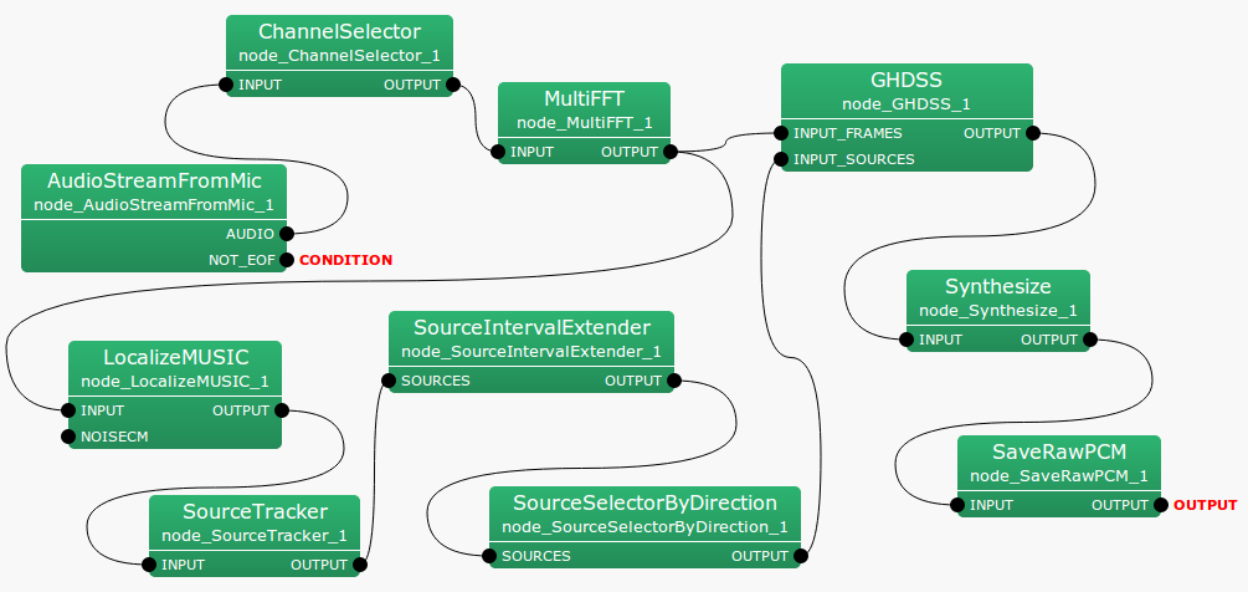


図 6.44: SourceIntervalExtender の接続例: Iterator サブネットワーク

ノードの入出力とプロパティ

入力

SOURCES : Vector<ObjectRef> 型．Source 型で表現される音源定位結果の Vector が入力される．ObjectRef が参照するのは，Source 型のデータである．

出力

OUTPUT : Vector<ObjectRef> 型．早められた出力された音源定位結果が出力される．ObjectRef が参照するのは，Source 型のデータである．

パラメータ

表 6.33: SourceIntervalExtender のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
PREROLL_LENGTH	int	50	[frame]	何フレームだけ早く定位結果を出力し始めるか．

PREROLL_LENGTH : int 型．定位結果を実際よりどれだけ早く出力するかを決定する正の整数．値が小さすぎると分離音の先頭が出力されないので, 前段で用いる音源定位手法の遅延に合わせて設定する必要がある．

ノードの詳細

SourceIntervalExtender 無しで定位結果を元に音源分離を行ったとき，図 6.45 に示すように音源定位の処理時間の分だけ分離音の先頭部分が切れてしまう．特に音声認識の際，音声の先頭部分が切れていると認識性能に悪影響を及ぼすので，本ノードを使って定位結果を早める必要がある．

当然, 音源が定位される前に定位結果を出力するのは不可能である. そこで本ノード以降のネットワークの処理を PREROLL_LENGTH 分だけ遅らせることで “音源定位結果の出力を早める” 機能を実現する. こうして全体を遅らせたあと, 各繰り返しで PREROLL_LENGTH 分だけ入力を先読みし，そこ定位結果があれば，その時点から定位結果の出力を開始する (図 6.46 参照．)

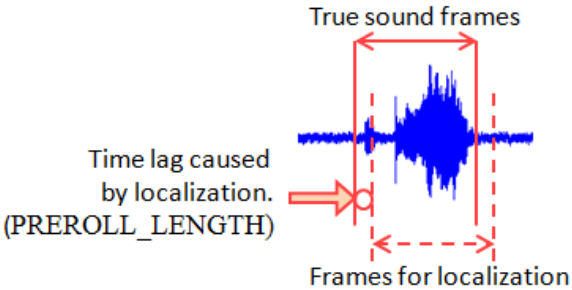


図 6.45: 音源定位結果を実際より早く出力する必要性

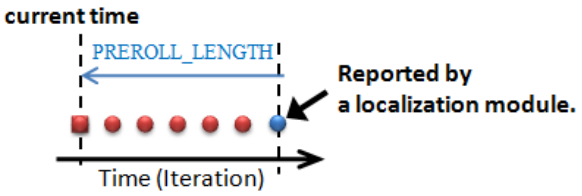


図 6.46: SourceIntervalExtender の動作: 定位結果を見つけると， PREROLL_LENGTH 分だけ事前に出力する．

6.2.19 SourceTracker

ノードの概要

時系列で入力された ID 無しの音源定位結果に対して，到来方向の近さに応じてクラスタリングを行い， ID を与えるノード．本ノードを通過した後の音源定位結果は，同じ音源か否かを ID のみから判定することが可能になる．ただし, 音長による信号の削除は行わない.

必要なファイル

無し.

使用方法

どんなときに使うのか

定位された音源の到来方向は，音源を固定していても（直立した人，固定したスピーカなど）通常同一方向であり続けることはない．従って，異なる時刻での到来方向が，同一音源となるように音源 ID を統一するためには，音源定位結果を追跡する必要がある．SourceTracker では，しきい値より近い到来方向の音源同士に対して同じ ID を与えるというアルゴリズムを用いている．本ノードを用いることで，音源に ID が付与され，ID 毎の処理が行える.

典型的な接続例

通常は，ConstantLocalization ，LocalizeMUSIC などの音源定位ノードの出力を本ノードの入力端子に接続する．すると適切な ID が定位結果に付加されるので，音源定位に基づく音源分離ノード GHDSS などや音源定位結果の表示ノード (DisplayLocalization) に接続する.

図 6.47 に接続例を示す．ここでは，固定の音源定位結果を， SourceTracker を通して表示している．このとき， ConstantLocalization の出力する定位結果が近ければ，それらは一つの音源にまとめて出力される．図中の ConstantLocalization に次のプロパティを与えた場合は，2 つの音源の成す角は MIN_SRC_INTERVAL のデフォルト値 20 [deg] より小さいので，1 つの音源だけが表示される.

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

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

設定値の意味は ConstantLocalization を参照.

ノードの入出力とプロパティ

入力

INPUT : `Vector<ObjectRef>` 型．ID が振られていない音源定位結果.

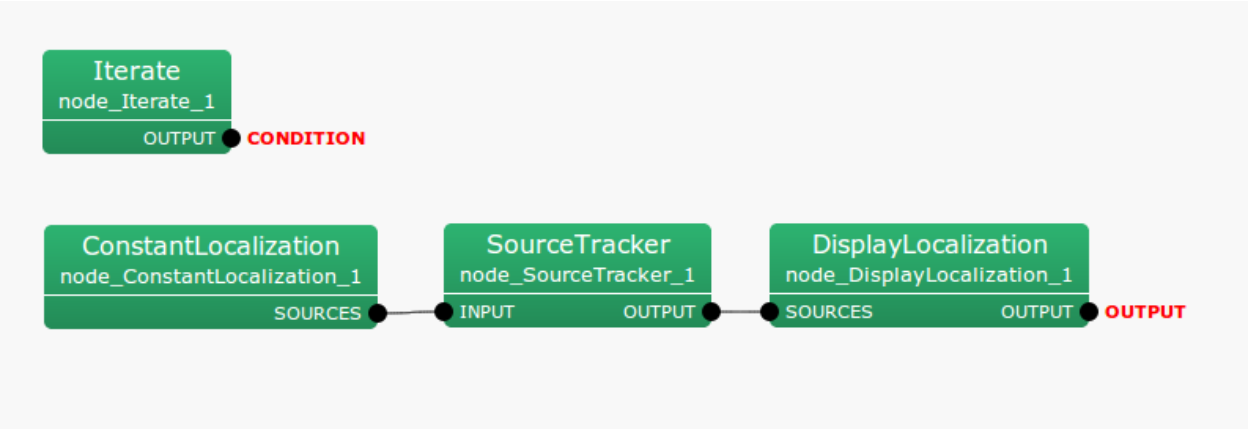


図 6.47: SourceTracker の接続例

出力

OUTPUT : `Vector<ObjectRef>` 型．位置が近い音源に同じ ID を与えた音源定位結果

パラメータ

表 6.34: SourceTracker のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
THRESH	`float`			音源の MUSIC パワーがこれより小さければ無視.
PAUSE_LENGTH	`float`	800	10 [frame]	音源の生存時間.
COMPARE_MODE	`string`	DEG	DEG or TFINDEX	音源間の距離の計算方法. DEG は角度計算, TFINDEX はインデックスの比較.
MIN_SRC_INTERVAL	`float`	20	[deg]	同一の音源とみなす角度差の閾値. (COMPARE_MODE = DEG のとき有効)
MIN_TFINDEX_INTERVAL	`int`	3		同一の音源とみなすインデックス差の閾値. (COMPARE_MODE = TFINDEX のとき有効)
MIN_ID	`int`	0		割り当てられる 音源 ID の最小値
DEBUG	`bool`	`false`		デバッグ出力の有無.

THRESH : `float` 型．音源定位結果を無視すべきノイズか否かを，その MUSIC パワーで判定する．MUSIC パワーが THRESH より小さければノイズであると判断し，その定位結果は出力には反映されなくなる．小さくしすぎるとノイズを拾い，大きくしすぎると目的音の定位が困難になるので，このトレードオフを満たす値を見つける必要がある．

PAUSE_LENGTH : `float` 型．一度定位結果として出力した音源が，どれだけ長く続くと仮定するかを決めるパラメータ．一度定位した方向は，それ以降に音源定位結果が無くても，PAUSE_LENGTH / 10 [frame]

の繰り返しの間だけ同一方向の定位結果を出力し続ける．デフォルト値は 800 なので，1 度定位した方向は，それ以降の 80 [frame] の繰り返しの間は定位結果を出力し続ける．

COMPARE_MODE ：型. DEG を選択すると音源を追跡時の音源方向の比較を三角関数を使った角度計算になり、TFINDEX を選択すると伝達関数中の対応するインデックスの単なる比較になる．インデックスと角度差が等価ならば (順番に録音しているならば、) 計算量が削減可能.

MIN_SRC_INTERVAL ：`float` 型．音源の到来方向の差が MIN_SRC_INTERVAL より小さければ同一の音源とみなして片方の音源定位結果を削除する．こうして音源定位が揺らいでも追跡できる．COMPARE_MODE が DEG のとき有効.

MIN_TFINDEX_INTERVAL ：`int` 型．MIN_SRC_INTERVAL とほぼ同じ．比較するのみ値が異なる．COMPARE_MODE が TFINDEX のとき有効.

MIN_ID ：`int` 型．定位結果ごとに割り振られる ID の開始番号を定める.

DEBUG ：`bool` 型． `true` が与えられると, 音源定位結果が標準エラー出力にも出力される.

ノードの詳細

まず，本節で用いる記号を定義する.

1. ID : 音源の ID
2. MUSIC パワー p: LocalizeMUSIC で計算された、定位された音源方向の MUSIC スペクトルのパワー.
3. 座標 x, y, z: 音源定位方向に対応する，単位球上の直交座標.
4. 継続時間 r: 定位された音源がそれ以降どれだけ続くと仮定するかの指標.

定位された音源の MUSIC パワーを p，音源方向に対応する単位球上の直交座標を x，y，z とする.

現在ノードが保持している音源数を N，新たに入力された音源数を M とする．また，直前の値には ^{last} を，現在の値には ^{cur} の添字をつける．例えば，i 番号めの新たに入力された音源の MUSIC パワーは p_i^{cur} と表示する.

音源の近さを判定する指標の，音源同士の成す角を $\theta \in [0, 180]$ とする.

音源方向の近さの判定方法:

成す角 θ は，二つの音源方向を，$\boldsymbol{q}_1 = (x_1, y_1, z_1)$ と $\boldsymbol{q}_2 = (x_2, y_2, z_2)$ で表現すると次のように求まる.

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

ここで逆余弦関数を用いると，θ が求まる.

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

以下では，表記を簡単にするために，第 i 音源と第 j 音源との成す角を θ_{ij} と表記する.

音源追跡方法:

SourceTracker の音源追跡アルゴリズムを図 6.48 にまとめる．また，青い丸が既にノードが持っている音源位置 (^{last})，緑の丸が新たに入力された音源位置 (^{cur}) を表す．

まず，すべての音源に対して，MUSIC パワー p_i^{cur}，p_j^{last} が THRESH より小さければそれを削除する．次に，既にノードが持つ定位情報と新たに入力された音源位置を比較し，十分近い (=θ_{ij} が MIN_SRC_INTERVAL[deg] 以下) なら統合する．統合された音源には同じ ID が付与され，継続時間 r^{last} が PAUSE_LENGTH でリセットされる．統合は，1 つの音源を残して他のすべての音源位置を削除することで実現される．

θ_{ij} が MIN_SRC_INTERVAL [deg] より大きい音源は，異なる音源とみなされる．ノードが保持しているが，新たに入力されなかった音源位置は，r^{last} を 10 だけ減らす．もし，r^{last} が 0 を下回ったら，音源が消えたとみなして，その音源位置を削除する．新たに入力された音源位置が，既にノードが持つ音源位置のいずれとも異なる場合は，新たな ID を付与され，r^{cur} が PAUSE_LENGTH で初期化される．

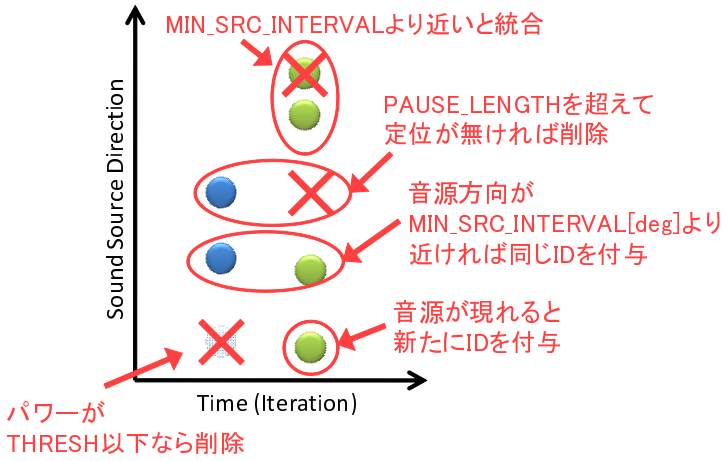


図 6.48: SourceTracker の音源追跡方法．横軸が時間 (=繰り返し回数) で，縦軸が音源方向を表す．

6.2.20 SourceTrackerPF

ノードの概要

SourceTracker の互換ノード. SourceTracker ではしきい値より近い到来方向の音源に対し同一音源 ID を付与するが, 本ノードでは到来方向の角度に基づきパーティクルフィルタによるクラスタリングを行い ID を付与するため, SourceTracker よりロバストな音源追跡が可能である. 一方で，計算コストは SourceTracker に対して増大する．パーティクルの状態遷移モデルにはランダムウォークを用いた非線形のモデル化が行われているが, そのため推定結果は毎回同じにはならないことに注意が必要である. 音源推定位置は毎フレーム更新される.

必要なファイル

無し.

使用方法

どんなときに使うのか

LocalizeMUSIC などの音源定位ノードによって定位された音源の到来方向は，離散値となり，また変動するため音源定位結果の追跡が必要となる．SourceTrackerPF では，しきい値より高いパワーを持つ音源に対して到来方向情報を利用し，パーティクルフィルタによるクラスタリングを行い，各パーティクルグループに音源 ID を付与する.

移動音源の追跡が SourceTracker でうまくいかない場合に用いるとよい．SourceTracker よりロバストな音源追跡が可能となる一方で計算コストは SourceTracker に対して増大する．パラメータの 1 つであるパーティクル数を小さくすることで計算処理時間を削減できる．推定精度とのトレードオフを考慮し調整するとよい.

典型的な接続例

通常は，ConstantLocalization ，LocalizeMUSIC などの音源定位ノードの出力を本ノードの入力端子に接続する．すると適切な 音源 ID が定位結果に付加されるので，音源定位に基づく音源分離ノード GHDSS などや音源定位結果の表示ノード (DisplayLocalization) に接続する.

図 6.49 に接続例を示す.

ノードの入出力とプロパティ

入力

INPUT : `Vector<ObjectRef>` 型．音源 ID が振られていない音源定位結果.

出力

OUTPUT : `Vector<ObjectRef>` 型．音源 ID を与えた音源定位結果.

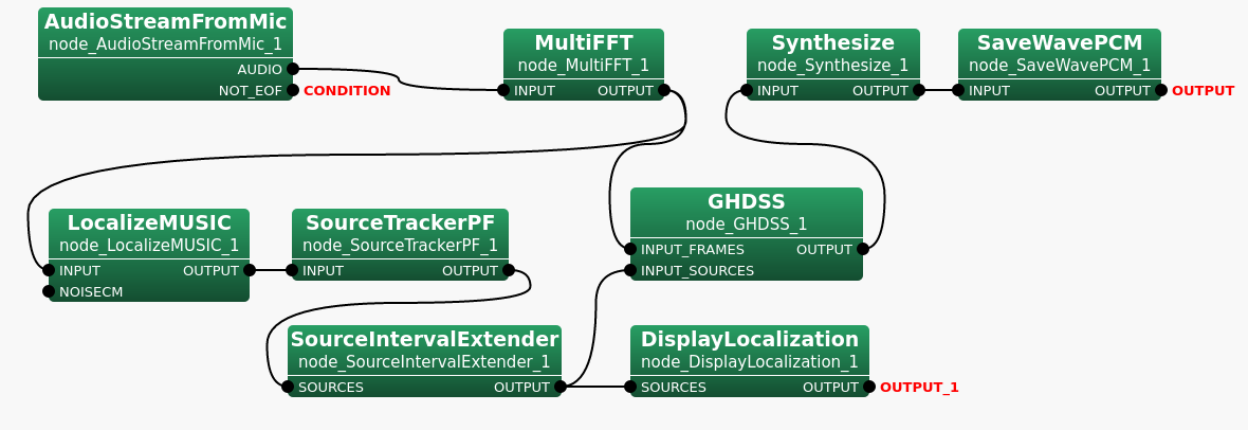


図 6.49: SourceTrackerPF の接続例

パラメータ

表 6.35: SourceTrackerPF のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
THRESH_SOURCE_POWER	float			音源の MUSIC パワーがこれより小さければ無視.
TOTAL_PARTICLE	int	1000		散布されるパーティクルの数.
SOURCE_MAX	int	2		最大音源数.
ANOTHER_SOURCE	float	0.00001		同一音源とみなすパーティクルグループの尤度閾値.
IGNORE_SOURCE	float	0.00001		新音源生成の尤度閾値.
REMOVE_SOURCE	int	150	[frame]	パーティクルグループの生存期間.
OUTPUT_RANGE	float	3.0	[deg]	グループ近傍のパーティクルだとみなす角度範囲.
LIKELIHOOD_SIGMA	float	1.0	[deg]	観測値の尤度を求める際に仮定する確率分布の分散.
STATE_UPDATE_SIGMA	float	1.0	[deg]	ランダムウォークの分散.
SUM_W	float	0.4		発散回避パラメータ.
HISTORY_LOG	bool	false		ログを表示.

THRESH_SOURCE_POWER ：float 型．音源定位結果を無視すべきノイズか否かを，その MUSIC パワーで判定する．MUSIC パワーが THRESH より小さければノイズであると判断し，その定位結果は出力には反映されなくなる．小さくしすぎるとノイズを拾い，大きくしすぎると目的音の定位が困難になるので，このトレードオフを満たす値を見つける必要がある．

TOTAL_PARTICLE ：int 型．計算に使用するパーティクルの総数．数を増やすと計算時間が増えるが少なすぎると確率分布の推定精度が悪くなる．

SOURCE_MAX ：int 型. 最大音源数．最大音源数を増やしてもパーティクルの総数は増加しない．パーティクルを分割して利用する．

ANOTHER_SOURCE : `float` 型．パーティクルグループ内の最大尤度がこの値以下になると観測値との対応付けを終了する．

IGNORE_SOURCE : `float` 型. すべてのパーティクルグループについて，設定値以下の尤度であったら，新音源を生成する準備を行う．

REMOVE_SOURCE : `int` 型. 観測値と対応付けがなされていないパーティクルグループの生存フレーム数.

OUTPUT_RANGE : `float` 型．近傍のパーティクルだと認識するための範囲．パーティクルグループの重心からの角度範囲．この範囲内のパーティクルが当該パーティクルグループに含まれものとして計算される．

LIKELIHOOD_SIGMA : `float` 型．尤度算出パラメータ．仮定される確率分布の分散．分散を大きくとりすぎると観測値によらず尤度が高くなってしまい，小さくしすぎると尤度が低くなってまい発散する．

STATE_UPDATE_SIGMA : `float` 型．状態遷移時のランダムウォークの分散．ランダム値は正規分布に基づく．

SUM_W : `float` 型．出力値に各パーティクルに対して重みづけを行うかどうかの閾値．重要度が小さすぎる場合計算結果が発散するため，重みづけ値の総和が閾値より小さい場合はグループ内のパーティクルの平均値を出力する．

HISTORY_LOG : `bool` 型．パーティクルグループの履歴ログの表示 / 非表示.

ノードの詳細

入力された音源 ID が付与されていない音源定位結果について，初めにパラメータ THRESH_SOURCE_POWER 未満であるかの判定が行われる．MUSIC パワーがしきい値未満である定位結果についてはノイズであるとみなされ破棄される．

THRESH_SOURCE_POWER 以上の音源定位結果については以下のパーティクルフィルタに基づく音源追跡方法により，音源 ID の付与，音源方向の推定が行われる．

パーティクルフィルタによる音源追跡

パーティクルフィルタでは，内部状態 $\boldsymbol{x}(t)$ の遷移モデル $p(\boldsymbol{x}(t)|\boldsymbol{x}(t-1))$ と観測モデル $p(y(t)|\boldsymbol{x}(t))$ を確率的な表現として定義する．なお $y(t)$ は観測ベクトルを表す．i 番目のパーティクルは内部状態 $\boldsymbol{x}_i(t)$ とそのパーティクルが音源追跡結果にどの程度貢献するかを示す重要度 $\boldsymbol{w}_i(t)$ を保持している．重要度は一般に尤度して定義される．

パーティクルフィルタの処理を以下に示す．本ノードの処理は初期化，音源生成・消滅チェック，重要度サンプリング，選択，出力の 5 つのステップで構成される．

Step 1 - 初期化

初期化では，すべてのパーティクルを一様かつランダムに分布させるまた，複数の音源を扱うためにパーティクルグループを導入し，重要度 $\boldsymbol{w}_i$ を下記のように定義している．

$$\sum_{i \in P_k} w_i = 1 \tag{6.19}$$

$$\sum_{k=1}^{S} N_k = N \tag{6.20}$$

ここで，N_k は k 番目のパーティクルグループ P_k のパーティクルの数，S は音源数を，N はパーティクルの総数である．

Step 2 - 音源生成・消滅チェック

この Step は複数音源を扱うための Step である．パーティクルグループ P_k の内部状態は下記のように定義される．

$$\hat{\boldsymbol{x}}_k(t) = \sum_{i \in P_k} \boldsymbol{x}_i \cdot w_i(t) \tag{6.21}$$

時刻 t にける j 番目の観測を $\boldsymbol{y}_j$，$\boldsymbol{y}_j$ と $\hat{\boldsymbol{x}}_k(t)$ がなす角を $\angle\theta$，パラメータ ANOTHER_SOURCE から求められる角度の閾値を $\angle\theta_{th}$ とした場合，以下の処理が行われる．

- $\angle\theta < \angle\theta_{th}$ であれば，$\boldsymbol{y}_j$ を P_k にアソシエートする．
- $\boldsymbol{y}_j$ に対応するパーティクルグループが見つからない場合，新規にパーティクルグループを生成する．
- パーティクルグループ P_k にアソシエートする観測が REMOVE_SOURCE で指定される時間以上得られなかった場合，P_k は消滅する．
- いずれの場合もパーティクルは，式 (6.19, 6.20) が満たされるように再配置される．

Step 3 - 重要度サンプリング

重要度サンプリングは以下の流れで行われる．

1. 遷移モデル $p(\boldsymbol{x}(t)|\boldsymbol{x}(t-1))$ を用いて，$\boldsymbol{x}_i(t-1)$ から，状態 $\boldsymbol{x}_i(t)$ を推定する．
2. 重要度 $w_i(t)$ を式 (6.25) を用いて更新する．
3. 式 (6.19，6.20) に従って，$w_i(t)$ を正規化する．

$\boldsymbol{x}_i(t)$ の要素，音源方向の方位角 $\theta_i(t)$，仰角 $\phi_i(t)$ の遷移モデルは，ランダムウォークの過程に基づき，以下のように定義される．

$$\theta_i(t) = \theta_i(t-1) + r_\theta \tag{6.22}$$

$$\phi_i(t) = \phi_i(t-1) + r_\phi \tag{6.23}$$

r_* は正規分布に基づく乱数である．分散はパラメータ STATE_UPDATE_SIGMA で指定する．
尤度の定義は $\boldsymbol{x}_i(t)$ と $\boldsymbol{y}_j$ のなす角を $\angle\psi$ とすると，

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

となり，最後に w_i を下記の式で更新する．

$$w_i(t) = l(t) \cdot w_i(t-1) \tag{6.25}$$

Step 4 - 選択

重要度 w_i に応じて，パーティクルの更新を行う．
$i \in P_k$ を満たす i に対するパーティクルの数は以下の式で更新される．

$$N_(k_i) \quad = \quad round(N_k \cdot w_i) \tag{6.26}$$

この場合，R_k 個のパーティクルが更新されないままとなっている．

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

これらのパーティクルも，残差重みパラメータ $R_(w_i)$ に従って分配される．

$$R_(w_i) \quad = \quad w_i - N_(k_i) \Big/ \sum_{j \in P_k} N_(k_j) \tag{6.28}$$

Sampling Importance Resampling (SIR) アルゴリズムを用いている．

Step 5 - 出力

更新後のパーティクルの密度から事後確率 $p(\boldsymbol{x}(t) \mid \boldsymbol{x}(t))$ を推定する．
音源 k に対するパーティクルグループの内部状態は式 (6.21) によって推定される．

Step2 から Step2 を音源の追跡が終了するまで繰り返す．
各パーティクルグループの $\hat{\boldsymbol{x}}_k(t)$ を音源位置の推定結果とし，出力する．
音源 ID はパーティクルグループごとに付与され，同一のパーティクルグループでは音源 ID が引き継がれる．

参考文献

(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).

6.3 Separation カテゴリ

6.3.1 BGNEstimator

ノードの概要

マルチチャネル信号のパワースペクトルから，信号に含まれる定常ノイズ (例えば，ファンノイズや背景ノイズ，BackGround Noise) を推定する．推定した定常ノイズは，PostFilter ノードで使用される．

必要なファイル

無し．

使用方法

どんなときに使うのか

信号に含まれるフ定常ノイズ (ファンノイズや背景ノイズ，BackGround Noise) を推定する．この推定値が必要になるノードは，PostFilter である．PostFilter ノードでは，この背景ノイズと，PostFilter 内で推定されるチャネル間リークをもとに，分離処理でとりきれないノイズを抑制する．

典型的な接続例

BGNEstimator ノードの接続例を図 6.50 に示す．入力には，音声波形を周波数領域に変換し求めたパワースペクトルを入力する．出力は，PostFilter ノードで利用される．

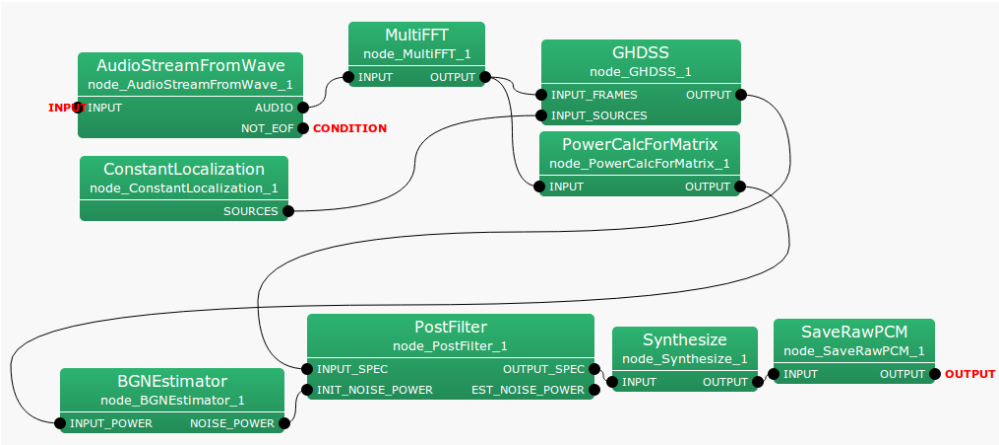


図 6.50: BGNEstimator の接続例

ノードの入出力とプロパティ

入力

INPUT_POWER : `Matrix<float>` 型． マルチチャネルパワースペクトル

表 6.36: BGNEstimator のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
DELTA	float	3.0		パワー比閾値
L	int	150	[frame]	検出時間幅
ALPHA_S	float	0.7		入力信号の平滑化係数
NOISE_COMPENS	float	1.0		定常ノイズの混入率
ALPHA_D_MIN	float	0.05		平滑化係数の最小値
NUM_INIT_FRAME	int	100	[frame]	初期化フレーム数

出力

NOISE_POWER : Matrix<float> 型． 推定された定常ノイズのパワースペクトル．

パラメータ

DELTA : float 型． 3.0 がデフォルト値． パワースペクトルの周波数ビンに音声などの目的音が含まれているかどうかの閾値．大きい値にすると，より多くのパワーを定常ノイズとみなす．

L : int 型． 150 がデフォルト値． 目的音が含まれるかの判断基準となる，過去最もパワーの小さいスペクトル (定常ノイズ成分) を保持する時間．AudioStreamFromWave ノードなどで指定される ADVANCE パラメータ分のシフトが行われる回数として指定する．

ALPHA_S : float 型． 0.7 がデフォルト値． 入力信号を時間方向に平滑化する際の係数． 値が大きいほど，過去のフレームの値の重みを大きく平滑化する．

NOISE_COMPENS : float 型． 1.0 がデフォルト値． 目的音が含まれないと判断されたフレームを，定常ノイズとして重みづけして加算する (定常ノイズの平滑化) ときの重み．

ALPHA_D_MIN : float 型． 0.05 がデフォルト値． 定常ノイズの平滑化処理で，目的音が含まれたと判断されたフレームのパワースペクトルを加える際の最小重み．

NUM_INIT_FRAME : int 型． 100 がデフォルト値． 処理を開始した際，このフレーム数だけ目的音の有無判定を行わず，全て定常ノイズとみなす．

ノードの詳細

以下では，定常ノイズを導出する過程を示す．時間，周波数，チャネルインデックスは 表 6.1 に準拠する．導出のフローは，図 6.51 の通り．

1． 時間方向，周波数方向平滑化: 時間方向の平滑化は，入力パワースペクトル $\boldsymbol{S}(f,k_i)$ と，前フレームの定常ノイズパワースペクトル $\boldsymbol{\lambda}(f-1,k_i)$ の内分により行う．

$$S_m^{smo,t}(f,k_i) = \alpha_s\lambda_m(f-1,k_i) + (1-\alpha_s)S_m(f,k_i) \tag{6.29}$$

周波数方向の平滑化は，時間平滑化された $S_m^{smo,t}(f,k_i)$ に対して行う．

$$S_m^{smo}(f,k_i) = 0.25S_m^{smo}(f,k_{i-1}) + 0.5S_m^{smo}(f,k_i) + 0.25S_m^{smo}(f,k_{i+1}) \tag{6.30}$$

表 6.37: 変数表

変数名	対応パラメータ，または，説明
$S(f,k_i) = [S_1(f,k_i),...,S_M(f,k_i)]^T$	時間フレーム f，周波数ビン k_i の入力パワースペクトル
$\lambda(f,k_i) = [\lambda_1(f,k_i),...,\lambda_M(f,k_i)]^T$	推定されたノイズスペクトル
δ	DELTA，デフォルト 0.3
L	L，デフォルト 150
α_s	ALPHA_S，デフォルト 0.7
θ	NOISE_COMPENS，デフォルト 1.0
α_d^{min}	ALPHA_D_MIN，デフォルト 0.05
N	NUM_INIT_FRAME，デフォルト 100

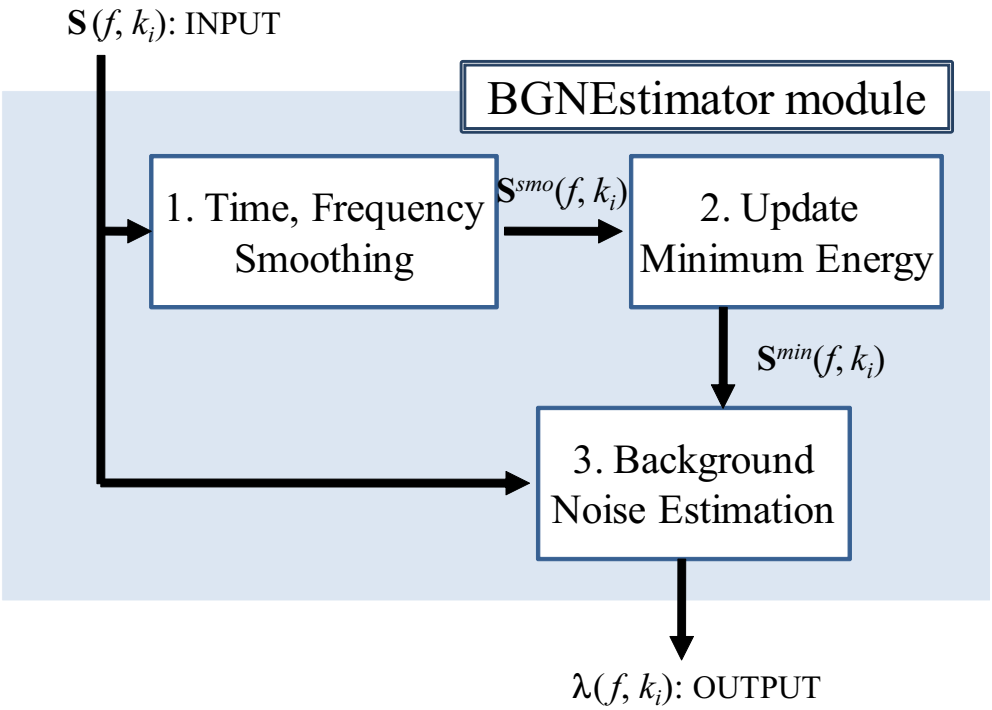


図 6.51: 定常ノイズ推定の流れ

2．最小エネルギーの更新: 目的音の有無を判定するため，処理を開始してからの各チャネル，周波数ビンについての最小のエネルギー $\boldsymbol{S}^{min}$ を計算する．$\boldsymbol{S}^{min}$ は，各チャネル，周波数ビンごとの，処理を開始してからの最小エネルギーで，$\boldsymbol{S}^{tmp}$ は，L フレームごとに更新される暫定最小エネルギーである．

$$S_m^{tmp}(f,k_i) = \begin{cases} S_m^{smo}(f,k_i), & \text{if } f = nL \\ \min\{S_m^{tmp}(f-1,k_i), S_m^{smo}(f,k_i)\}, & \text{if } f \neq nL \end{cases} \tag{6.31}$$

$$S_m^{min}(f,k_i) = \begin{cases} \min\{S_m^{tmp}(f-1,k_i), S_m^{smo}(f,k_i)\}, & \text{if } f = nL \\ \min\{S_m^{min}(f-1,k_i), S_m^{smo}(f,k_i)\}, & \text{if } f \neq nL \end{cases} \tag{6.32}$$

ただし，n は任意の整数である．

3．定常ノイズ推定:

1. 目的音有無の判定
   以下にいずれかが成り立つ場合，該当する時間，周波数に目的音のパワーは存在せず，ノイズのみがあるとみなされる．

$$S_m^{smo}(f,k_i) < \delta S_m^{min}(f,k_i) \text{ または} \tag{6.33}$$

$$f < N \text{ または} \tag{6.34}$$

$$S_m^{smo}(f,k_i) < \lambda_m(f-1,k_i) \tag{6.35}$$

2. 平滑化係数の算出
   定常ノイズのパワーを計算する際に用いられる平滑化係数 α_d は，

$$\alpha_d = \begin{cases} \frac{1}{f+1}, & \text{if } (\frac{1}{f+1} \geq \alpha_d^{min}) \\ \alpha_d^{min}, & \text{if } (\frac{1}{t+1} < \alpha_d^{min}) \\ 0 & (\text{目的音が含まれるとき}) \end{cases} \tag{6.36}$$

   として計算する．定常ノイズは以下の式によって求める．

$$\lambda_m(f,k_i) = (1-\alpha_d)\lambda_m(f-1,k_i) + \alpha_d \theta S_m(f,k_i) \tag{6.37}$$

6.3.2 BeamForming

ノードの概要

以下の手法を用いて音源分離を行う．

- DS : 遅延和ビームフォーミング (Delay-and-Sum beamforming)
- WDS : 重み付き遅延和ビームフォーミング (Weighted Delay-and-Sum beamforming)
- NULL : NULL 制御付きビームフォーミング (NULL beamforming)
- ILSE : 不定項最小二乗誤差ビームフォーミング (Indefinite term and Least Square Estimator based beamforming)
- LCMV : 線形拘束付き最小分散型ビームフォーミング (Linearly Constrained Minimum Variance)
- GJ : 線形拘束付き最小分散, Griffiths-Jim 型 ビームフォーミング
- GICA : 幾何学的独立成分分析 (Geometrically constrained Independent Component Analysis)
- GHDSS：幾何学的高次無相関化音源分離法 (Geometrically constrained Higher-order Decorrelation-based Source Separation)

ノードの入力は,

- 混合音のマルチチャネル複素スペクトル
- 目的音源の方向データ
- 雑音の方向データ

である．また，出力は分離音ごとの複素スペクトルである．

必要なファイル

表 6.38: BeamForming に必要なファイル

対応するパラメータ名	説明
TF_CONJ_FILENAME	マイクロホンアレーの伝達関数

使用方法

どんなときに使うのか

所与の音源方向に対して，マイクロホンアレーを用いて当該方向の音源分離を行う．なお，音源方向として，音源定位部での推定結果，あるいは，定数値を使用することができる．

典型的な接続例

BeamForming ノードの接続例を図 6.52 に示す．入力は以下である．

1. INPUT_FRAMES : MultiFFT 等から得られる混合音の多チャネル複素スペクトル
2. INPUT_SOURCES : LocalizeMUSIC や ConstantLocalization 等から得られる目的音源方向
3. INPUT_NOISE_SOURCES : LocalizeMUSIC や ConstantLocalization 等から得られる雑音方向（オプション入力）

出力は分離音声となる．

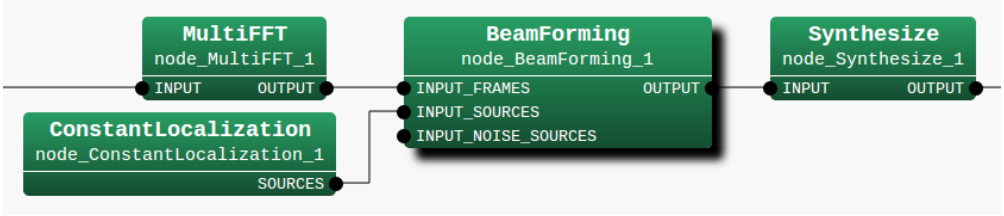


図 6.52: BeamForming の接続例

ノードの入出力とプロパティ

入力

INPUT_FRAMES : `Matrix<complex<float> >` 型．マルチチャネル複素スペクトル．行がチャネル，つまり，各マイクロホンから入力された波形の複素スペクトルに対応し，列が周波数ビンに対応する．

INPUT_SOURCES : `Vector<ObjectRef>` 型．目的音源の方向情報が格納された `Source` 型オブジェクトの `Vector` 配列である．典型的には， SourceTracker ノード， SourceIntervalExtender ノードと繋げ，その出力を用いる．

INPUT_NOISE_SOURCES : `Vector<ObjectRef>` 型．INPUT_SOURCES と同じ `Source` 型オブジェクトの `Vector` 配列である．雑音の方向情報が格納された `Source` 型オブジェクトの `Vector` 配列である．入力はオプションである．NULL，ILSE での音源分離を行う際，雑音によるビームフォーミングの性能低下を避けることができる．

出力

OUTPUT : `Map<int, ObjectRef>` 型．分離音の音源 ID と，分離音の 1 チャネル複素スペクトル (`Vector<complex<float> >` 型) のペア．分離音の数だけ出力される．

パラメータ

LENGTH : `int` 型．分析フレーム長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 512.

ADVANCE : `int` 型．フレームのシフト長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 160.

SAMPLING_RATE : `int` 型．入力音源波形のサンプリング周波数 [Hz]．デフォルト値は 16000.

LOWER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最小周波数値であり，これより下の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．0 以上サンプリング周波数値の半分までの範囲で指定する．

UPPER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最大周波数値であり，これより上の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY である必要がある．

TF_CONJ_FILENAME : `string` 型．伝達関数が記述されたバイナリファイル名を記す．ファイルフォーマットは 5.3.1 節を参照．

INITW_FILENAME : `string` 型．分離行列の初期値が記述されたバイナリファイル名を記す．事前の計算により, 値の収束した分離行列を初期値として与えることで, 音が鳴り始めた最初の部分から精度よく分離することが可能となる. 指定されていると，そのファイルにある分離行列が，指定されていなければ，TF_CONJ_FILENAME ファイルの分離行列データベースから得た分離行列か、幾何学的関係から推定された分離行列が，初期分離行列として使用される．

GICA_SS_METHOD : `string` 型．ブラインド音源分離のための独立成分分析である ICA (Independent Component Analysis) のステップサイズの算出方法を選ぶ．ユーザが指定した値 GICA_SS_MYU に固定する場合は FIX，幾何制約に基づくステップサイズに連動した値 GICA_LC_MYU にする場合は LC_MYU，適応的ステップサイズにする場合は ADAPTIVE を指定．

GICA_SS_MYU : `float` 型．ブラインド音源分離のための分離行列更新時のステップサイズ．デフォルト値は 0.001．GICA_SS_METHOD = FIX の場合は この値が固定ステップサイズの値となる．GICA_SS_METHOD = LC_MYU の場合は無視される．GICA_SS_METHOD = ADAPTIVE の場合は適応的に決定されたステップサイズに GICA_SS_MYU のゲインを乗算してステップサイズとする．この値と GICA_LC_MYU を 0 にし，Delay and Sum 型のビームフォーマの分離行列を INITW_FILENAME として渡すことで，BeamForming は，Delay and Sum 型のビームフォーマと等価な処理が可能となる．

GHDSS_SS_METHOD : `string` 型．高次無相関化に基づく音源分離である HDSS (Higher-order Decorrelation-based Source Separation) のステップサイズの算出方法を選ぶ．ユーザが指定した値 GHDSS_SS_MYU に固定する場合は FIX，幾何制約に基づくステップサイズに連動した値 GHDSS_LC_MYU にする場合は LC_MYU，適応的ステップサイズにする場合は ADAPTIVE を指定．

GHDSS_SS_MYU : `float` 型．高次無相関化に基づく分離行列更新時のステップサイズ．デフォルト値は 0.001．GHDSS_SS_METHOD = FIX の場合は この値が固定ステップサイズの値となる．GHDSS_SS_METHOD = LC_MYU の場合は無視される．GHDSS_SS_METHOD = ADAPTIVE の場合は適応的に決定されたステップサイズに GHDSS_SS_MYU のゲインを乗算してステップサイズとする．この値と GHDSS_LC_MYU を 0 にし，Delay and Sum 型のビームフォーマの分離行列を INITW_FILENAME として渡すことで，BeamForming は，Delay and Sum 型のビームフォーマと等価な処理が可能となる．

LCMV_LC_METHOD : `string` 型．線形拘束付き最小分散に基づくステップサイズの算出方法を選ぶ．ユーザが指定した値 LCMV_LC_MYU に固定する場合は FIX，適応的ステップサイズにする場合は ADAPTIVE を指定．目的音源方向のゲインを一定に保ちながらビームフォーマのパワーが最小になるようにステップサイズを決定する．

LCMV_LC_MYU : `float` 型．線形拘束付き最小分散に基づく分離行列更新時のステップサイズ．デフォルト値は 0.001．LCMV_LC_METHOD = FIX の場合は LCMV_LC_MYU が固定ステップサイズの値となる．

LCMV_LC_METHOD = ADAPTIVE の場合は適応的に決定されたステップサイズに LCMV_LC_MYU のゲインを乗算してステップサイズとする．

GJ_LC_METHOD ：`string` 型．線形拘束付き最小分散に基づくステップサイズの算出方法を選ぶ．ユーザが指定した値 GJ_LC_MYU に固定する場合は FIX，適応的ステップサイズにする場合は ADAPTIVE を指定．目的音源方向のゲインを一定に保ちながら妨害音成分を低減するビームフォーマの係数を更新するステップサイズを決定する．

GJ_LC_MYU ：`float` 型．線形拘束付き最小分散に基づく分離行列更新時のステップサイズ．デフォルト値は 0.001．GJ_LC_METHOD = FIX の場合は GJ_LC_MYU が固定ステップサイズの値となる．GJ_LC_METHOD = ADAPTIVE の場合は適応的に決定されたステップサイズに GJ_LC_MYU のゲインを乗算してステップサイズとする．

GICA_LC_METHOD ：`string` 型．幾何拘束に基づく音源分離である GC (Geometric Constraint) のステップサイズの算出方法を選ぶ．ユーザが指定した値 GICA_LC_MYU に固定する場合は FIX，適応的ステップサイズにする場合は ADAPTIVE を指定．

GICA_LC_MYU ： `float` 型．幾何制約に基づく分離行列更新時のステップサイズ．デフォルト値は 0.001．GICA_LC_METHOD = FIX の場合は GICA_LC_MYU が固定ステップサイズの値となる．GICA_LC_METHOD = ADAPTIVE の場合は適応的に決定されたステップサイズに GICA_LC_MYU のゲインを乗算してステップサイズとする．この値と GICA_SS_MYU を 0 にし，Delay and Sum 型のビームフォーマの分離行列を INITW_FILENAME として渡すことで，BeamForming は，Delay and Sum 型のビームフォーマと等価な処理が可能となる．

GHDSS_LC_METHOD ：`string` 型．幾何拘束に基づく音源分離である GC (Geometric Constraint) のステップサイズの算出方法を選ぶ．ユーザが指定した値 GHDSS_LC_MYU に固定する場合は FIX，適応的ステップサイズにする場合は ADAPTIVE を指定．

GHDSS_LC_MYU ：`float` 型．幾何制約に基づく分離行列更新時のステップサイズ．デフォルト値は 0.001．GHDSS_LC_METHOD = FIX の場合は GHDSS_LC_MYU が固定ステップサイズの値となる．GHDSS_LC_METHOD = ADAPTIVE の場合は適応的に決定されたステップサイズに GHDSS_LC_MYU のゲインを乗算してステップサイズとする．この値と GHDSS_SS_MYU を 0 にし，Delay and Sum 型のビームフォーマの分離行列を INITW_FILENAME として渡すことで，BeamForming は，Delay and Sum 型のビームフォーマと等価な処理が可能となる．

GHDSS_LC_CONST ：`string` 型．幾何制約の手法を選択する．幾何制約に対角成分（直接音成分）のみを使う場合は DIAG，直接音成分に加えて，非対角成分も使用する場合は FULL を指定する．死角は高次無相関化によって自動的に形成されるため， DIAG でも高精度な分離が可能．

GICA_SS_SCAL ：`float` 型．高次相関行列計算における双曲線正接関数 (tanh) のスケールファクタを指定する．デフォルト値は 1.0．0 より大きい正の実数を指定する．値が小さいほど非線形性が少なくなり通常の相関行列計算に近づく．

GHDSS_SS_SCAL ：`float` 型．高次相関行列計算における双曲線正接関数 (tanh) のスケールファクタを指定する．デフォルト値は 1.0．0 より大きい正の実数を指定する．値が小さいほど非線形性が少なくなり通常の相関行列計算に近づく．

GHDSS_NOISE_FLOOR ：`float` 型．入力信号をノイズとみなす振幅の閾値 (上限) を指定する．デフォルト値は 0．入力信号の振幅がこの値以下の場合，ノイズ区間とみなされ，分離行列の更新がされない．ノイズが大きく，分離行列が安定して収束しない場合に，正の実数を指定する．

GHDSS_UPDATE : `string` 型．分離行列の更新方法を決める．高次無相関化に基づく更新を行った後に，幾何制約に基づく更新を行う場合は STEP，高次無相関化に基づく更新と幾何制約に基づく更新を同時に行う場合は TOTAL を指定する．

UPDATE_METHOD_W : `string` 型．音源位置情報が変わった際に，分離行列の再計算が必要となる．この時の音源位置情報が変わったとみなす方法を指定する．分離行列は，内部で対応する音源 ID や音源方向の座標とともに一定時間保存され，一度音が止んでも，同一の方向からの音源と判断される音が検出されると，再び保存された分離行列の値を用いて分離処理が行われる．このとき，分離行列の更新を行うかどうかの基準を設定する．音源 ID によって同方向音源かどうか判断する場合は ID，音源方向を比較して判断する場合は POS，音源 ID を比較し，同一と判断されなかった場合に，さらに音源方向の座標を比較して判断を行う場合は ID_POS を設定する．

UPDATE_ACCEPT_DISTANCE : `float` 型．音源の移動に対して同一音源とみなす距離 [mm]．設定した距離の範囲内であれば更新された分離行列を利用して演算が行われる．デフォルト値は 300.0.

EXPORT_W : `bool` 型．更新された分離行列の結果を出力するかどうかを設定．`true` のとき，EXPORT_W_FILENAME を指定する．

EXPORT_W_FILENAME : `string` 型．分離行列を書きだすファイル名を指定する．フォーマットは 5.3.2 節を参照．

BF_METHOD : `string` 型．音源分離手法を指定する．以下の音源分離手法をサポートしている．

- DS : 遅延和ビームフォーミング (Delay-and-Sum beamforming) [1]
- WDS : 重み付き遅延和ビームフォーミング (Weighted Delay-and-Sum beamforming) [1]
- NULL : NULL 制御付きビームフォーミング (NULL beamforming) [1]
- ILSE : 最小平均二乗誤差制御つきビームフォーミング (Iterative Least Squares with Enumeration) [2]
- LCMV : 線形拘束付き最小分散型 (Linearly Constrained Minimum Variance) ビームフォーミング [3]
- GJ : 線形拘束付き最小分散, Griffiths-Jim 型 ビームフォーミング [4]
- GICA : 幾何学的独立成分分析 (Geometrically constrained Independent Component Analysis) [5]
- GHDSS : 幾何学的高次無相関化音源分離法 (Geometrically constrained Higher-order Decorrelation-based Source Separation) [5]

ENABLE_DEBUG : `bool` 型．`true` が与えられると, 分離状況が標準出力に出力される．デフォルトは `false`.

表 6.39: BF_METHOD = DS,WDS,NULL,ILSE で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	`int`	512	[pt]	分析フレーム長.
ADVANCE	`int`	160	[pt]	フレームのシフト長.
SAMPLING_RATE	`int`	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	`int`	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	`int`	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	`string`			マイクロホンアレーの伝達関数を記したファイル名.
ENABLE_DEBUG	`bool`	`false`		デバッグ出力の可否.

表 6.40: BF_METHOD = LCMV で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	string			マイクロホンアレーの伝達関数を記したファイル名.
INITW_FILENAME	string			分離行列の初期値を記述したファイル名.
LCMV_LC_METHOD	string	ADAPTIVE		線形拘束条件に基づくステップサイズの算出方法.FIX　は固定値 LCMV_LC_MYU，ADAPTIVE　は自動調節.
LCMV_LC_MYU	float	0.001		線形拘束条件に基づく分離行列更新時のステップサイズ.
UPDATE_METHOD_W	string	ID		音源位置情報が変わったとみなす方法.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	音源の移動に対して同一音源とみなす距離.
EXPORT_W	bool	false		分離行列ファイルに書き出すかを指定.
EXPORT_W_FILENAME	string			分離行列を出力するファイル名.
ENABLE_DEBUG	bool	false		デバッグ出力の可否.

表 6.41: BF_METHOD = GJ で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	string			マイクロホンアレーの伝達関数を記したファイル名.
INITW_FILENAME	string			分離行列の初期値を記述したファイル名.
GJ_LC_METHOD	string	ADAPTIVE		線形拘束条件に基づくステップサイズの算出方法.FIX　は固定値 GJ_LC_MYU，ADAPTIVE は自動調節.
GJ_LC_MYU	float	0.001		線形拘束条件に基づく分離行列更新時のステップサイズ.
UPDATE_METHOD_W	string	ID		音源位置情報が変わったとみなす方法.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	音源の移動に対して同一音源とみなす距離.
EXPORT_W	bool	false		分離行列ファイルに書き出すかを指定.
EXPORT_W_FILENAME	string			分離行列を出力するファイル名.
ENABLE_DEBUG	bool	false		デバッグ出力の可否.

ノードの詳細

技術的な詳細: 基本的に詳細は下記の参考文献を参照されたい.

音源分離概要: 音源分離問題で用いる記号を表 6.55 にまとめる．演算はフレーム毎に周波数領域において行われるため，各記号は周波数領域での，一般には複素数の値を表す．音源分離は K 個の周波数ビン $(1 \leq k \leq K)$ それぞれに対して演算が行われるが，本節ではそれを略記する．N, M, f をそれぞれ，音源数，マイク数，フレームインデックスとする．

音のモデルは以下の一般的な線形モデルを扱う．

$$\boldsymbol{X}(f) \quad = \quad \boldsymbol{H}\boldsymbol{S}(f) + \boldsymbol{N}(f). \tag{6.38}$$

表 6.42: BF_METHOD = GICA で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	string			マイクロホンアレーの伝達関数を記したファイル名.
INITW_FILENAME	string			分離行列の初期値を記述したファイル名.
GICA_SS_METHOD	string	ADAPTIVE		ブラインド音源分離のためのステップサイズの算出方法.FIX は固定値 GICA_SS_MYU, LC_MYU は幾何制約に基づくステップサイズ GICA_LC_MYU に連動した値，ADAPTIVE は自動調節.
GICA_SS_MYU	float	0.001		ブラインド音源分離のための分離行列更新時のステップサイズ.
GICA_LC_METHOD	string	ADAPTIVE		幾何制約に基づくステップサイズの算出方法.FIX は固定値 GICA_LC_MYU，ADAPTIVE は自動調節.
GICA_LC_MYU	float	0.001		幾何制約に基づく分離行列更新時のステップサイズ.
GICA_SS_SCAL	float	1.0		高次相関行列計算におけるスケールファクタ.
UPDATE_METHOD_W	string	ID		音源位置情報が変わったとみなす方法.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	音源の移動に対して同一音源とみなす距離.
EXPORT_W	bool	false		分離行列ファイルに書き出すかを指定.
EXPORT_W_FILENAME	string			分離行列を出力するファイル名.
ENABLE_DEBUG	bool	false		デバッグ出力の可否.

分離の目的は,

$$\boldsymbol{Y}(f) \quad = \quad \boldsymbol{W}(f)\boldsymbol{X}(f) \tag{6.39}$$

として，$\boldsymbol{Y}(f)$ が $\boldsymbol{S}(f)$ に近づくように，$\boldsymbol{W}(f)$ を推定することである．最後に推定された $\boldsymbol{W}(f)$ は，EXPORT_W = true にし，EXPORT_W_FILENAME で指定した適当なファイル名で保存することができる．

TF_CONJ_FILENAME で指定する伝達関数ファイルには計測された $\boldsymbol{H}$ を格納する．今後はこれを実際の伝達関数と区別するため，$\hat{\boldsymbol{H}}$ と 表記する．

BF_METHOD = DS,WDS,NULL,ILSE の場合: INPUT_SOURCES 入力端子から入ってくる音源方向と INPUT_NOISE_SOURCES 入力端子から入ってくる雑音方向の情報を用いた $\hat{\boldsymbol{H}}$ を用いて $\boldsymbol{W}(f)$ を決定する．

BF_METHOD = LCMV,GJ の場合: 分離行列更新のための評価関数 $J_{\mathrm{L}}(\boldsymbol{W}(f))$ は，INPUT_SOURCES 入力端子から入ってくる音源方向と INPUT_NOISE_SOURCES 入力端子から入ってくる雑音方向の情報で定義される．分離行列の更新は略記すると以下のようになる．

$$\boldsymbol{W}(f+1) \quad = \quad \boldsymbol{W}(f) + \mu \nabla_{\boldsymbol{W}} J_{\mathrm{L}}(\boldsymbol{W})(f) \tag{6.40}$$

ただし，$\nabla_{\boldsymbol{W}} J_{\mathrm{L}}(\boldsymbol{W}) = \frac{\partial J_{\mathrm{L}}(\boldsymbol{W})}{\partial \boldsymbol{W}}$ である．この μ を LC_MYU で指定できる．LC_METHOD = ADAPTIVE に指定した場合は,

$$\mu \quad = \quad \left. \frac{J_{\mathrm{L}}(\boldsymbol{W})}{\left|\nabla_{\boldsymbol{W}} J_{\mathrm{L}}(\boldsymbol{W})\right|^2} \right|_{\boldsymbol{W}=\boldsymbol{W}(f)} \tag{6.41}$$

と，適応的にステップサイズが計算される．

表 6.43: BF_METHOD = GHDSS で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	string			マイクロホンアレーの伝達関数を記したファイル名.
INITW_FILENAME	string			分離行列の初期値を記述したファイル名.
GHDSS_SS_METHOD	string	ADAPTIVE		高次無相関化に基づくステップサイズの算出方法．FIX は固定値 GHDSS_SS_MYU, LC_MYU は幾何制約に基づくステップサイズ GHDSS_LC_MYU に連動した値, ADAPTIVE は自動調節.
GHDSS_SS_MYU	float	0.001		高次無相関化に基づく分離行列更新時のステップサイズ.
GHDSS_LC_METHOD	string	ADAPTIVE		幾何制約に基づくステップサイズの算出方法. FIX は固定値 GHDSS_LC_MYU, ADAPTIVE は自動調節.
GHDSS_LC_MYU	float	0.001		幾何制約に基づく分離行列更新時のステップサイズ.
GHDSS_LC_CONST	string	FULL		幾何制約の手法．DIAG は対角成分のみを使用, FULL は全成分を使用.
GHDSS_SS_SCAL	float	1.0		高次相関行列計算におけるスケールファクタ.
GHDSS_NOISE_FLOOR	float	0.0		入力信号をノイズとみなす振幅の閾値 (上限).
GHDSS_UPDATE	string	STEP		分離行列の更新方法．STEP は高次無相関化に基づく更新を行った後に幾何制約に基づく更新を行う．TOTAL は高次無相関化に基づく更新と幾何制約に基づく更新を同時に行う.
UPDATE_METHOD_W	string	ID		音源位置情報が変わったとみなす方法.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	音源の移動に対して同一音源とみなす距離.
EXPORT_W	bool	false		分離行列ファイルに書き出すかを指定.
EXPORT_W_FILENAME	string			分離行列を出力するファイル名.
ENABLE_DEBUG	bool	false		デバッグ出力の可否.

表 6.44: 変数の定義

変数	説明
$\boldsymbol{S}(f) = [S_1(f), \ldots, S_N(f)]^T$	f フレーム目の音源の複素スペクトル
$\boldsymbol{X}(f) = [X_1(f), \ldots, X_M(f)]^T$	マイクロホン観測複素スペクトルのベクトル．INPUT_FRAMES 入力に対応.
$\boldsymbol{N}(f) = [N_1(f), \ldots, N_M(f)]^T$	加法性雑音
$\boldsymbol{H} = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_N] \in \mathbb{C}^{M \times N}$	$1 \leq n \leq N$ 番目の音源から $1 \leq m \leq M$ 番目のマイクまでの伝達関数行列
$\boldsymbol{W}(f) = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_M] \in \mathbb{C}^{N \times M}$	分離行列
$\boldsymbol{Y}(f) = [Y_1(f), \ldots, Y_N(f)]^T$	分離音複素スペクトル

BF_METHOD = GICA の場合: 分離行列更新のための評価関数 $J_{\mathrm{G}}(\boldsymbol{W}(f))$ は，INPUT_SOURCES 入力端子から入ってくる音源方向と INPUT_NOISE_SOURCES 入力端子から入ってくる雑音方向の情報とで以下で定義される.

$$J_{\mathrm{G}}(\boldsymbol{W}(f)) \quad = \quad J_{\mathrm{SS}}(\boldsymbol{W}(f)) + J_{\mathrm{LC}}(\boldsymbol{W}(f)) \tag{6.42}$$

ただし，$J_{\mathrm{SS}}(\boldsymbol{W}(f))$ は，ブラインド音源分離に基づく音源分離手法のための評価関数，$J_{\mathrm{LC}}(\boldsymbol{W}(f))$ は幾何制

約に基づく音源分離手法のための評価関数である．分離行列の更新は略記すると以下のようになる．

$$\boldsymbol{W}(f+1) \quad = \quad \boldsymbol{W}(f) + \mu_{\mathrm{SS}} \nabla_{\boldsymbol{W}} \boldsymbol{J}_{\mathrm{SS}}(\boldsymbol{W})(f) + \mu_{\mathrm{LC}} \nabla_{\boldsymbol{W}} \boldsymbol{J}_{\mathrm{LC}}(\boldsymbol{W})(f) \tag{6.43}$$

ただし，$\nabla_{\boldsymbol{W}}$ は，式 (6.40) と同様に $\boldsymbol{W}$ についての偏微分を表す．この μ_{SS} と μ_{LC} をそれぞれ，SS_MYU, LC_MYU で指定できる．SS_METHOD = ADAPTIVE に指定した場合は，

$$\mu_{\mathrm{SS}} \quad = \quad \left. \frac{\boldsymbol{J}_{\mathrm{SS}}(\boldsymbol{W})}{\left|\nabla_{\boldsymbol{W}} \boldsymbol{J}_{\mathrm{SS}}(\boldsymbol{W})\right|^2} \right|_{\boldsymbol{W}=\boldsymbol{W}(f)} \tag{6.44}$$

と，LC_METHOD = ADAPTIVE に指定した場合は，

$$\mu_{\mathrm{LC}} \quad = \quad \left. \frac{\boldsymbol{J}_{\mathrm{LC}}(\boldsymbol{W})}{\left|\nabla_{\boldsymbol{W}} \boldsymbol{J}_{\mathrm{LC}}(\boldsymbol{W})\right|^2} \right|_{\boldsymbol{W}=\boldsymbol{W}(f)} \tag{6.45}$$

と，適応的にステップサイズが計算される．

トラブルシューティング: 基本的には GHDSS ノードのトラブルシューティングと同じ．

参考文献

- [[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

ノードの概要

信号＋ノイズのパワースペクトルからノイズパワースペクトルを除去する時に，推定されたノイズのパワースペクトルをどの程度除去するべきかの最適ゲインを決定するノードである．その他，音声存在確率（6.3.11 節参照）を出力する．ただし，このノードは音声存在確率を常に 1 として出力する．分離音のパワースペクトルと推定ノイズのパワースペクトルの差分を出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

HRLE ノードを用いたノイズ推定時に用いる．

典型的な接続例

CalcSpecSubGain の接続例は図 6.53 の通り．入力は GHDSS で分離後のパワースペクトルおよび HRLE で推定されたノイズのパワースペクトル．出力は VOICE_PROB，GAIN を SpectralGainFilter に接続する．

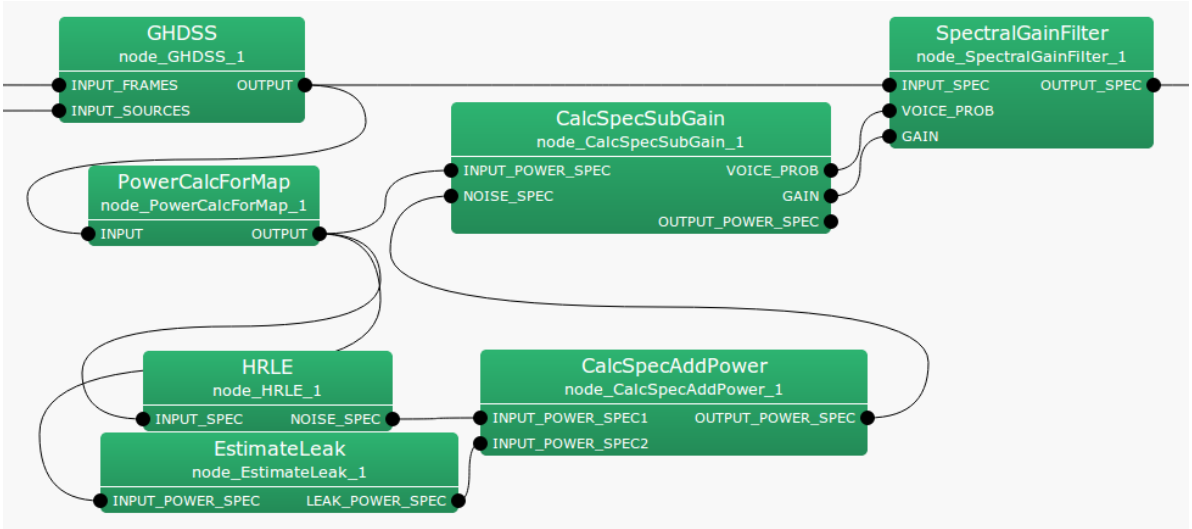


図 6.53: CalcSpecSubGain の接続例

ノードの入出力とプロパティ

入力

表 6.45: CalcSpecSubGain のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
ALPHA	float	1.0		スペクトル減算のゲイン
BETA	float	0.0		最適ゲインの最小値
SS_METHOD	int	2		パワー・振幅スペクトル減算の選択

INPUT_POWER_SPEC ：Map<int, ObjectRef> 型．音源 ID と分離音のパワースペクトルの Vector<float> 型データのペア．

NOISE_SPEC ：Map<int, ObjectRef> 型．音源 ID と推定ノイズのパワースペクトルの Vector<float> 型データのペア．

出力

VOICE_PROB ：Map<int, ObjectRef> 型．音源 ID と音声存在確率の Vector<float> 型データのペア．

GAIN ：Map<int, ObjectRef> 型．音源 ID と最適ゲインの Vector<float> 型データのペア．

OUTPUT_POWER_SPEC ：Map<int, ObjectRef> 型．音源 ID と分離音から推定ノイズを差し引いたパワースペクトル Vector<float> 型データのペア．

パラメータ

ALPHA ：スペクトル減算のゲイン

BETA ：最適ゲインの最小値

SS_METHOD ：パワースペクトル減算か振幅スペクトル減算かの選択

ノードの詳細

信号＋ノイズのパワースペクトルからノイズパワースペクトルを除去する時に，推定されたノイズのパワースペクトルをどの程度除去するべきかの最適ゲインを決定するノードである．音声存在確率（6.3.11 節参照）も出力する．ただし，このノードは音声存在確率を常に 1 として出力する．

分離音からノイズを差し引いたパワースペクトルを $Y_n(k_i)$，分離音のパワースペクトルを $X_n(k_i)$，推定されたノイズのパワースペクトルを $N_n(k_i)$ とすると， OUTPUT_POWER_SPEC からの出力は次のように表される．

$$Y_n(k_i) = X_n(k_i) - N_n(k_i) \tag{6.46}$$

ただし，n は，分析フレーム番号．k_i は，周波数インデクスを表す．最適ゲイン $G_n(k_i)$ は，次のように表される．

$$G_n(k_i) = \begin{cases} \mathrm{ALPHA}\frac{Y_n(k_i)}{X_n(k_i)}, & \text{if } Y_n(k_i) > \mathrm{BETA}, \\ \mathrm{BETA}, & \text{if otherwise.} \end{cases} \tag{6.47}$$

単純に $Y_n(k_i)$ を用いて処理すると，パワーが負になりえる．以後の処理で，パワースペクトルの取り扱いが困難になるので，予め，パワーが負にならないようにノイズのパワースペクトルを除去するためのゲインを計算するのが本ノードの狙いである．

6.3.4 CalcSpecAddPower

ノードの概要

2 つの入力パワースペクトルを加算したスペクトルを出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

HRLE ノードを用いたノイズ推定時に用いる．HRLE ノードで推定されたノイズのパワースペクトルと EstimateLeak で推定されたノイズのパワースペクトルを加算し，トータルのノイズパワースペクトルを求める．

典型的な接続例

CalcSpecAddPower の接続例は図 6.54 の通り．入力は HRLE で推定されたノイズのパワースペクトル及び，EstimateLeak で推定されたノイズのパワースペクトル．出力は CalcSpecSubGain に接続する．

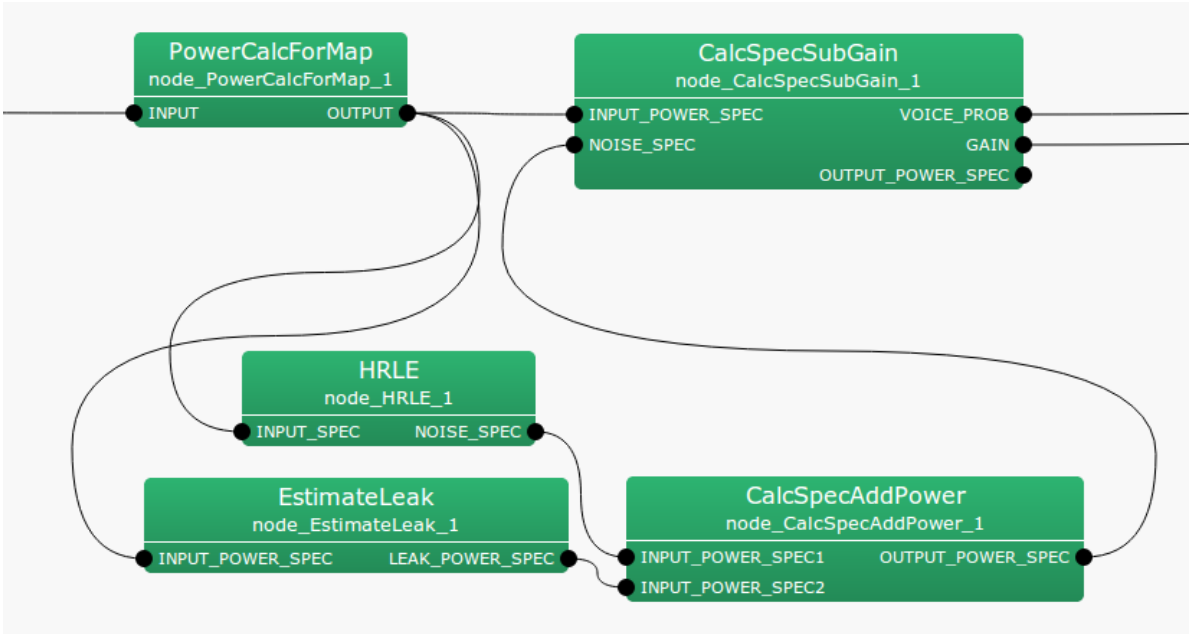


図 6.54: CalcSpecAddPower の接続例

ノードの入出力とプロパティ

入力

INPUT_POWER_SPEC1 : Map<int, ObjectRef> 型．音源 ID とパワースペクトルの Vector<float> 型データのペア．

INPUT_POWER_SPEC2 : Map<int, ObjectRef> 型．音源 ID とパワースペクトルの Vector<float> 型データのペア．

出力

OUTPUT_POWER_SPEC : Map<int, ObjectRef> 型．音源 ID と 2 つの入力を加算したパワースペクトル Vector<float> 型データのペア．

パラメータ

無し．

ノードの詳細

本ノードは，2 つの入力パワースペクトルを加算したスペクトルを出力する．

6.3.5 EstimateLeak

ノードの概要

他チャネルからの漏れ成分の推定を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

GHDSS を使った音源分離後のノイズ除去に用いる．

典型的な接続例

EstimateLeak の接続例は図 6.55 の通り．入力は で音声のパワースペクトルで，GHDSS の出力である．出力は CalcSpecAddPower に接続する．

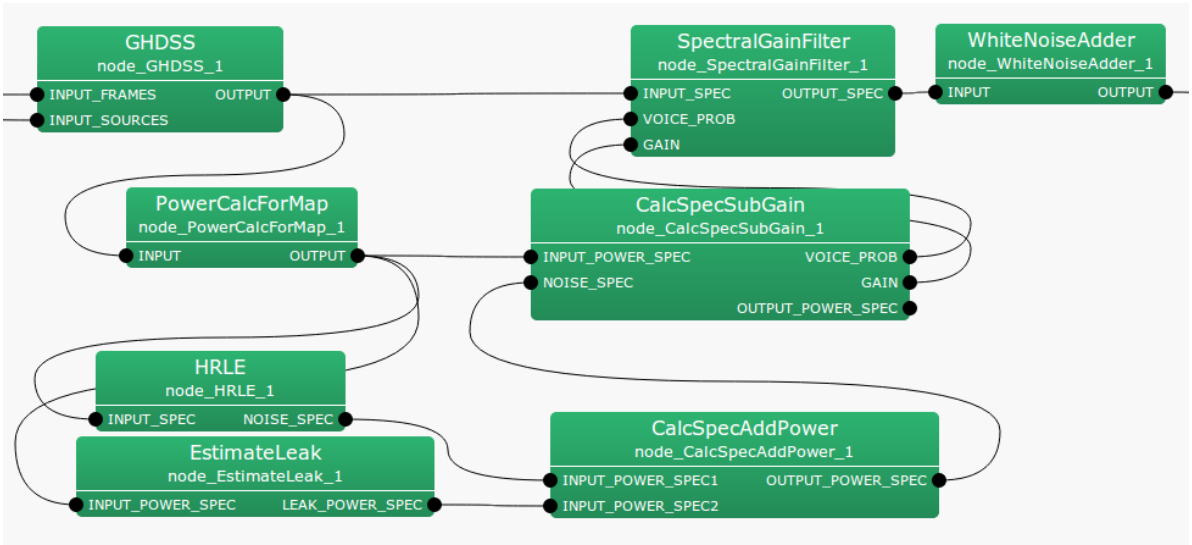


図 6.55: EstimateLeak の接続例

ノードの入出力とプロパティ

入力

INPUT_POWER_SPEC ：`Map<int, ObjectRef>` 型．音源 ID とパワースペクトルの `Vector<float>` 型データのペア．

出力

LEAK_POWER_SPEC : Map<int, ObjectRef> 型．音源 ID と漏れノイズのパワースペクトル Vector<float> 型データのペア．

パラメータ

表 6.46: EstimateLeak のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LEAK_FACTOR	float	0.25		漏れ率．
OVER_CANCEL_FACTOR	float	1		漏れ率重み係数．

ノードの詳細

本ノードは，他チャネルからの漏れ成分の推定を行う．詳細は 6.3.11 節の PostFilter ノード 1-b) 漏れノイズ推定を参照のこと．

6.3.6 GHDSS

ノードの概要

GHDSS (Geometric High-order Dicorrelation-based Source Separation) アルゴリズムに基づいて，音源分離処理を行う．GHDSS アルゴリズムは，マイクロホンアレイを利用した処理で，

1. 音源信号間の高次無相関化
2. 音源方向へ指向性の形成

という 2 つの処理を行う．2. の指向性は，事前に与えられたマイクロホンの位置関係を幾何的制約として処理を行う．また，HARK に実装されている GHDSS アルゴリズムは，マイクロホンの位置関係に相当する情報として，マイクロホンアレイの伝達関数を利用することができる．

ノードの入力は，混合音のマルチチャネル複素スペクトルと，目的音源の方向データである．また，出力は分離音ごとの複素スペクトルである．

HARK 2.1 の変更点:

1. ファイル形式の変更
   TF_CONJ_FILENAME で指定される伝達関数ファイル及び INITW_FILENAME、EXPORT_W_FILENAME で入出力される分離行列ファイルのファイル形式が 5.3 節で示されるファイルフォーマットに変更された．
2. POS 引き継ぎ時の許容誤差の指定方法の変更
   UPDATE_METHOD_TF_CONJ, UPDATE_METHOD_W を POS または ID_POS としたときの、許容誤差の指定方法を距離 [mm] とするよう変更した．

必要なファイル

表 6.47: GHDSS に必要なファイル

対応するパラメータ名	説明
TF_CONJ_FILENAME	マイクロホンアレーの伝達関数
INITW_FILENAME	分離行列初期値

使用方法

どんなときに使うのか

所与の音源方向に対して，マイクロホンアレイを用いて当該方向の音源分離を行う．なお，音源方向として，音源定位部での推定結果，あるいは，定数値を使用することができる．

典型的な接続例

GHDSS ノードの接続例を図 6.56 に示す．入力は以下の 2 つが必要である．

1. INPUT_FRAMES には，混合音の多チャネル複素スペクトル，

2. INPUT_SOURCES には，音源方向．

出力である分離音声に対して音声認識を行うために，MelFilterBank などを利用して，音声特徴量に変換する以外に，以下のような音声認識の性能向上方法もある．

1. PostFilter ノードを利用して，音源分離処理によるチャネル間リークや拡散性雑音を抑圧する（図 6.56 右上参照）．

2. PowerCalcForMap ，HRLE ，SpectralGainFilter を接続して，音源分離処理によるチャネルリークや拡散性雑音を抑圧する（PostFilter と比較して，チューニングが容易）．

3. PowerCalcForMap ，MelFilterBank ，MFMGeneration を接続して，ミッシングフィーチャ理論を用いた音声認識を行うために，ミッシングフィーチャーマスクを生成する（図 6.56 右下参照）．

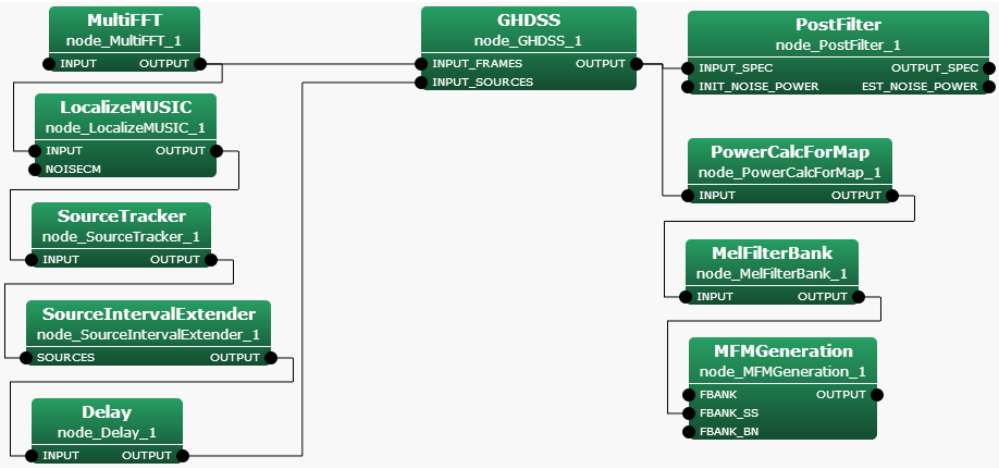


図 6.56: GHDSS の接続例

ノードの入出力とプロパティ

入力

INPUT_FRAMES ：`Matrix<complex<float> >` 型．マルチチャネル複素スペクトル．行がチャネル，つまり，各マイクロホンから入力された波形の複素スペクトルに対応し，列が周波数ビンに対応する．

INPUT_SOURCES ：`Vector<ObjectRef>` 型．音源定位結果等が格納された `Source` 型オブジェクトの `Vector` 配列である．典型的には，SourceTracker ノード，SourceIntervalExtender ノードと繋げ，その出力を用いる．

出力

OUTPUT ：`Map<int, ObjectRef>` 型．分離音の音源 ID と，分離音の 1 チャネル複素スペクトル．(`Vector<complex<float> >` 型) のペア．

パラメータ

LENGTH ：`int` 型．分析フレーム長．前段階における値（AudioStreamFromMic ，MultiFFT ノードなど）と一致している必要がある．

表 6.48: GHDSS のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	string			マイクロホンアレーの伝達関数を記したファイル名.
INITW_FILENAME	string			分離行列の初期値を記述したファイル名.
SS_METHOD	string	ADAPTIVE		高次無相関化に基づくステップサイズの算出方法. FIX, LC_MYU, ADAPTIVE から選択. FIX は固定値, LC_MYU は幾何制約に基づくステップサイズに連動した値, ADAPTIVE は自動調節.
SS_METHOD==FIX				以下 SS_METHOD が FIX の時に有効.
SS_MYU	float	0.001		高次無相関化に基づく分離行列更新時のステップサイズ.
SS_SCAL	float	1.0		高次相関行列計算におけるスケールファクタ.
NOISE_FLOOR	float	0.0		入力信号をノイズとみなす振幅の閾値 (上限).
LC_CONST	string	FULL		幾何制約の手法を決める. DIAG, FULL から選択. DIAG は対角成分を使うのみ. FULL は全成分を使用する.
LC_METHOD	string	ADAPTIVE		幾何制約に基づくステップサイズの算出方法. FIX, ADAPTIVE から選択. FIX は固定値, ADAPTIVE は自動調節.
LC_METHOD==FIX				以下 LC_METHOD が FIX の時に有効.
LC_MYU	float	0.001		高次無相関化に基づく分離行列更新時のステップサイズ.
UPDATE_METHOD_TF_CONJ	string	POS		伝達関数を更新する手法を指定. POS, ID から選択.
UPDATE_METHOD_W	string	ID		分離行列を更新する手法を指定. ID, POS, ID_POS から選択.
UPDATE_ACCEPT_DISTANCE	float	300.0	[mm]	音源移動時に同一音源とみなす距離の閾値.
EXPORT_W	bool	false		分離行列をファイルに書き出すかを指定.
EXPORT_W==true				以下 EXPORT_W が true の時に有効.
EXPORT_W_FILENAME	string			分離行列を書きだすファイル名.
UPDATE	string	STEP		分離行列の更新方法を決める. STEP, TOTAL から選択. STEP は高次無相関化に基づく更新を行った後に, 幾何制約に基づく更新を行う. TOTAL では, 高次無相関化に基づく更新と幾何制約に基づく更新を同時に行う.

ADVANCE : int 型. フレームのシフト長. 前段階における値 (AudioStreamFromMic , MultiFFT ノードなど) と一致している必要がある.

SAMPLING_RATE : int 型. 入力波形のサンプリング周波数.

LOWER_BOUND_FREQUENCY : int 型. GHDSS 処理を行う際に利用する最小周波数値であり, これより下の周波数に対しては処理を行わず, 出力スペクトルの値は 0 となる. 0 以上サンプリング周波数値の半分までの範囲で指定する.

UPPER_BOUND_FREQUENCY : int 型. GHDSS 処理を行う際に利用する最大周波数値であり, これより上の周波数に対しては処理を行わず, 出力スペクトルの値は 0 となる. LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY である必要がある.

TF_CONJ_FILENAME : string 型. 伝達関数の記述されたバイナリファイル名を記す. ファイルフォーマッ

トは 5.3.1 節を参照.

INITW_FILENAME : `string` 型．分離行列の初期値を記したファイル名．事前の計算により，値の収束した分離行列を初期値として与えることで，音が鳴り始めた最初の部分から精度よく分離することが可能となる．ここで与えるファイルは，EXPORT_W を `true` にすることで，予め用意しておく必要がある．ファイルフォーマットは 5.3.2 節を参照.

SS_METHOD : `string` 型．高次無相関化に基づくステップサイズの算出方法を選ぶ．ユーザが指定した値に固定する場合は FIX，幾何制約に基づくステップサイズに連動した値にする場合は LC_MYU，自動調節する場合は ADAPTIVE を指定.

1. FIX のとき: SS_MYU を設定する.
   SS_MYU: `float` 型．0.01 がデフォルト．高次無相関化に基づく分離行列更新時のステップサイズを指定する．この値と LC_MYU を 0 にし，Delay and Sum 型のビームフォーマの分離行列を INITW_FILENAME として渡すことで，GHDSS は，Delay and Sum 型のビームフォーマと等価な処理が可能となる.

SS_SCAL : `float` 型．1.0 がデフォルト．高次相関行列計算における双曲線正接関数 (tanh) のスケールファクタを指定する．0 より大きい正の実数を指定する．値が小さいほど非線形性が少なくなり通常の相関行列計算に近づく.

NOISE_FLOOR : `float` 型．0 がデフォルト．入力信号をノイズとみなす振幅の閾値 (上限) を指定する．入力信号の振幅がこの値以下の場合，ノイズ区間とみなされ，分離行列の更新がされない．ノイズが大きく，分離行列が安定して収束しない場合に，正の実数を指定する.

LC_CONST : `string` 型．幾何制約の手法を選択する．幾何制約に対角成分（直接音成分）のみを使う場合は DIAG 直接音成分に加えて，非対角成分も使用する場合は FULL を指定する．死角は高次無相関化によって自動的に形成されるため， DIAG でも高精度な分離が可能．デフォルトは FULL.

LC_METHOD : `string` 型．幾何制約に基づくステップサイズの算出方法を選ぶ．ユーザが指定した値に固定する場合は FIX，自動調節する場合は ADAPTIVE を指定.

1. FIX のとき: LC_MYU を設定する.
   LC_MYU: `float` 型．0.001 がデフォルト．幾何制約に基づく分離行列更新時のステップサイズを指定する．この値と LC_MYU を 0 にし，Delay and Sum 型のビームフォーマの分離行列を INITW_FILENAME として渡すことで，GHDSS は，Delay and Sum 型のビームフォーマと等価な処理が可能となる.

UPDATE_METHOD_TF_CONJ : `string` 型．ID または POS を指定する．POS がデフォルト．伝達関数の複素共役 TF_CONJ の更新をするかの判断を，各音源に付与された ID に基づいて行う (ID の場合) か，音源位置によって行う (POS の場合) かを指定する.

UPDATE_METHOD_W : `string` 型．ID，POS または ID_POS を指定．ID がデフォルト．音源位置情報が変わった際に，分離行列の再計算が必要となる．この時の音源位置情報が変わったとみなす方法を指定する．分離行列は，内部で対応する音源 ID や音源方向の座標とともに一定時間保存され，一度音が止んでも，同一の方向からの音源と判断される音が検出されると，再び保存された分離行列の値を用いて分離処理が行われる．このとき，分離行列の更新を行うかどうかの基準を設定する．ID を設定した場合，音源 ID によって同方向音源かどうか判断する．POS を設定した場合，音源方向を比較して判断する．ID_POS を設定した場合，音源 ID を比較し，同一と判断されなかった場合は，さらに音源方向の座標を比較して判断を行う.

UPDATE_ACCEPT_DISTANCE : `float` 型．300.0 がデフォルト．音源の移動に対して同一音源とみなす距離 [mm]．設定した距離の範囲内であれば更新された分離行列を利用して演算が行われる．

EXPORT_W : `bool` 型．`false` がデフォルト．GHDSS により更新された分離行列の結果を出力するかどうかを設定．`true` のとき，EXPORT_W_FILENAME を指定．

EXPORT_W_FILENAME : `string` 型．EXPORT_W が `true` のとき有効．分離行列を書きだすファイル名を指定．フォーマットは 5.3.2 節を参照．

UPDATE : `string` 型．分離行列の更新方法を決める．STEP，TOTAL から選択．STEP は高次無相関化に基づく更新を行った後に，幾何制約に基づく更新を行う．TOTAL では，高次無相関化に基づく更新と幾何制約に基づく更新を同時に行う．

ノードの詳細

音源分離の定式化: 音源分離問題の定式化で用いる記号を表 6.49 にまとめる．インデックスの意味は，表 6.1 に準拠する．演算は周波数領域において行われるため，各記号は周波数領域での，一般には複素数の値を表す．伝達関数以外は一般に時間変化するが，同じ時間フレームにおける演算の場合は，時間インデックス f を省略して表記する．また，以下の演算は周波数ビン k_i について述べる．実際には，K 個それぞれの周波数ビン $k_0, \ldots, k_{K-1}$ に対して演算が行われている．

表 6.49: 変数の定義

変数	説明
$S(k_i) = [S_1(k_i), \ldots, S_N(k_i)]^T$	周波数ビン k_i に対応する音源複素スペクトルのベクトル．
$X(k_i) = [X_1(k_i), \ldots, X_M(k_i)]^T$	マイクロホン観測複素スペクトルのベクトル，INPUT_FRAMES に対応．
$N(k_i) = [N_1(k_i), \ldots, N_M(k_i)]^T$	各マイクロホンに作用する加法性ノイズ．
$H(k_i) = [H_{m,n}(k_i)]$	反射，回折などを含む伝達関数行列 $(M \times N)$．
$H_D(k_i) = [H_{Dm,n}(k_i)]$	直接音の伝達関数行列 $(M \times N)$．
$W(k_i) = [W_{n,m}(k_i)]$	分離行列 $(N \times M)$．
$Y(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	分離音複素スペクトル．
μ_{SS}	高次無相関化に基づく分離行列更新時のステップサイズ，SS_MYU に対応．
μ_{LC}	幾何制約に基づく分離行列更新時のステップサイズ．LC_MYU に対応．

混合モデル N 個の音源から発せられた音は，その空間の伝達関数 $H(k_i)$ の影響を受け，M 個のマイクロホンを通じて式 (6.48) のように観測される．

$$X(k_i) = H(k_i)S(k_i) + N(k_i). \tag{6.48}$$

一般に，伝達関数 $H(k_i)$ は，部屋の形や，マイクロホンと音源の位置関係により変化するため，推定は困難である．しかし，音の反射や回折を無視して，直接音のみに限定した伝達関数 $H_D(k_i)$ は，音源とマイクロホンの相対位置が分かっている場合は，次の式 (6.49) のように計算可能である．

$$H_{Dm,n}(k_i) = \exp(-j2\pi l_i r_{m,n}), \tag{6.49}$$

$$l_i = \frac{2\pi\omega_i}{c}, \tag{6.50}$$

ただし，c は音速で，l_i は周波数ビン k_i での周波数 ω_i に対応する波数とする．また，$r_{m,\,n}$ は，マイクロホン m から音源 n までの距離と，座標系の基準点 (たとえば原点) から音源 n までの距離の差である．つまり，音源から各マイクロホンまでの到達時間の差から生じる位相差として，$H_D(k_i)$ は定義される．

分離モデル　分離音の複素スペクトルの行列 $Y(k_i)$ は，

$$Y(k_i) \quad = \quad W(k_i)X(k_i) \tag{6.51}$$

として求める．GHDSS アルゴリズムは，$Y(k_i)$ が $S(k_i)$ に近づくように，分離行列 $W(k_i)$ を推定する．

モデルにおける仮定　このアルゴリズムで既知と仮定する情報は次の通り．

1. 音源数 N
2. 音源位置 (HARK では LocalizeMUSIC ノードが音源位置を推定する)
3. マイクロホン位置
4. 直接音成分の伝達関数 $H_D(k_i)$ (測定する or 式 (6.49) による近似)

未知の情報としては，

1. 観測時の実際の伝達関数 $H(k_i)$
2. 観測ノイズ $N(k_i)$

分離行列の更新式　GHDSS は，下記を満たすように分離行列 $W(k_i)$ の推定を行う．

1. 分離信号を高次無相関化
   すなわち，分離音 $Y(k_i)$ の高次相関行列 $R^{\phi(y)y}(k_i) = E[\phi(Y(k_i))Y^H(k_i)]$ の対角成分以外が 0 になるようにする．ここで ^{H} 作用素はエルミート転置を，$E[]$ は時間平均作用素を，$\phi()$ は非線形関数であり，本ノードでは下記で定義される双曲線正接関数を用いている．

$$\phi(Y) \quad = \quad [\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_N)]^T \tag{6.52}$$

$$\phi(Y_k) \quad = \quad \tanh(\sigma|Y_k|)\exp(j\angle(Y_k)) \tag{6.53}$$

   ここで，σ はスケーリングファクタ（ SS_SCAL に対応）である．

2. 直接音成分は歪みなく分離される (幾何的制約)
   分離行列 $W(k_i)$ と 直接音の伝達関数 $H_D(k_i)$ の積が単位行列になるようにする ($W(k_i)H_D(k_i) = I$).

上の 2 つの要素をあわせた評価関数は以下のようになる．簡単のため，周波数ビン k_i は略す．

$$J(W) \quad = \quad \alpha J_1(W) + \beta J_2(W), \tag{6.54}$$

$$J_1(W) \quad = \quad \sum_{i \neq j} |R^{\phi(y)y}_{i,j}|^2, \tag{6.55}$$

$$J_2(W) \quad = \quad \|WH_D - I\|^2, \tag{6.56}$$

ただし，α および β は重み係数である．また行列のノルムは $\|M\|^2 = tr(MM^H) = \sum_{i,j}|m_{i,j}|^2$ として定義される．式 (6.54) を最小化するための分離行列の更新式は，複素勾配演算 $\frac{\partial}{\partial W^*}$ を利用した勾配法から，

$$W(k_i, f+1) \quad = \quad W(k_i, f) - \mu\frac{\partial J}{\partial W^*}(W(k_i, f)) \tag{6.57}$$

となる．ここで，μ は分離行列の更新量を調節するステップサイズである．通常，式 (6.57) の右辺にある複素勾配を求めると，$R^{xx} = E[XX^H]$ や $R^{yy} = E[YY^H]$ などの期待値計算に複数のフレームの値を要する．GHDSS ノードでの計算は，自己相関行列を求めず，1 つのフレームだけを用いた以下の更新式 (6.58) を用いる．

$$W(k_i, f+1) = W(k_i, f) - \left[\mu_{SS}\frac{\partial J_1}{\partial W^*}(W(k_i, f)) + \mu_{LC}\frac{\partial J_2}{\partial W^*}(W(k_i, f))\right], \tag{6.58}$$

$$\frac{\partial J_1}{\partial W^*}(W) = \left(\phi(Y)Y^H - \mathrm{diag}[\phi(Y)Y^H]\right)\tilde{\phi}(WX)X^H, \tag{6.59}$$

$$\frac{\partial J_2}{\partial W^*}(W) = 2\,(WH_D - I)\,H_D^H, \tag{6.60}$$

ここで，$\tilde{\phi}$ は ϕ の偏微分であり，下記で定義される．

$$\tilde{\phi}(Y) = [\phi(\tilde{Y}_1), \phi(\tilde{Y}_2), \ldots, \phi(\tilde{Y}_N)]^T \tag{6.61}$$

$$\tilde{\phi}(Y_k) = \phi(Y_k) + Y_k\frac{\partial\phi(Y_k)}{\partial Y_k} \tag{6.62}$$

また，$\mu_{SS} = \mu\alpha$，$\mu_{LC} = \mu\beta$ であり，それぞれ高次無相関化および幾何制約に基づくステップサイズをあらわす．ステップサイズの調節を自動にした場合，ステップサイズは次式で計算される．

$$\mu_{SS} = \frac{J_1(W)}{2\|\frac{\partial J_1}{\partial W}(W)\|^2} \tag{6.63}$$

$$\mu_{LC} = \frac{J_2(W)}{2\|\frac{\partial J_2}{\partial W}(W)\|^2} \tag{6.64}$$

式 (6.59，6.60) では，各変数のインデックスを省略したが，いずれも (k_i, f) である．分離行列の初期値は次のようにして求める．

$$W(k_i) = H_D^H(k_i)/M, \tag{6.65}$$

ただし，M はマイクロホン数である．

処理の流れ: GHDSS ノードにおける，時間フレーム f における主な処理を図 6.57 に示す．より詳細には，以下のように固定ノイズに関する処理などが含まれる．

1. (直接音) 伝達関数の取得
2. 分離行列 W 推定
3. 式 (6.51) に従って音源分離処理を実行
4. 分離行列の書き出し (EXPORT_W が `true` のとき)

伝達関数の取得: パラメータ TF_CONJ_FILENAME で指定された伝達関数から，入力された音源定位結果の方向に最も近い位置にあるデータを検索する．
2 フレーム以降については，以下の通りである．

- パラメータ UPDATE_METHOD_TF_CONJ の値によって以下のように，前フレームの伝達関数を引き継ぐか，ファイルから読み込むかを決定する．

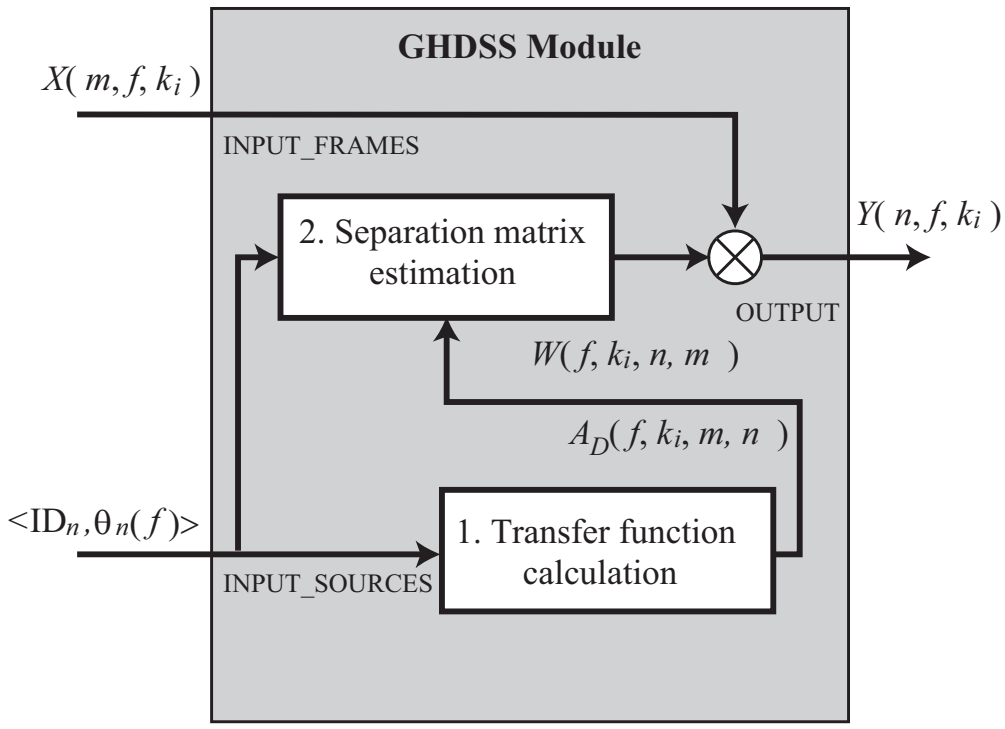


図 6.57: GHDSS の流れ図

UPDATE_METHOD_TF_CONJ が ID

1. 1 フレーム前の ID と取得した ID を比較
   - 同じ: 引き継ぐ
   - 異なる: 読み込む

UPDATE_METHOD_TF_CONJ が POS

1. 1 フレーム前の音源方向と取得した方向を比較
   - 誤差が UPDATE_ACCEPT_DISTANCE 未満: 引き継ぐ
   - 誤差が UPDATE_ACCEPT_DISTANCE 以上: 読み込む

分離行列の推定: 分離行列の初期値は，パラメータ INITW_FILENAME に値を指定するかによって異なる．
パラメータ INITW_FILENAME が指定されていないときは，伝達関数 H_D から分離行列 W を計算する．
パラメータ INITW_FILENAME が指定されているときは，指定された分離行列から，入力された音源定位結果の方向に最も近い位置にあるデータを検索する．
2 フレーム以降については，以下の通りである．分離行列を推定するまでの流れを図 6.58 に示す．ここでは，式 (6.58) に従って，前フレームの分離行列を更新するか，式 (6.65) を用いて，伝達関数を利用して分離行列の初期値の導出が行われる．

- 前フレームの音源定位情報を参照して，消滅した音源がある場合，分離行列は初期化される．
- 音源数に変化がない場合，UPDATE_METHOD_W の値によって分岐する．前フレームにおける，音源 ID,

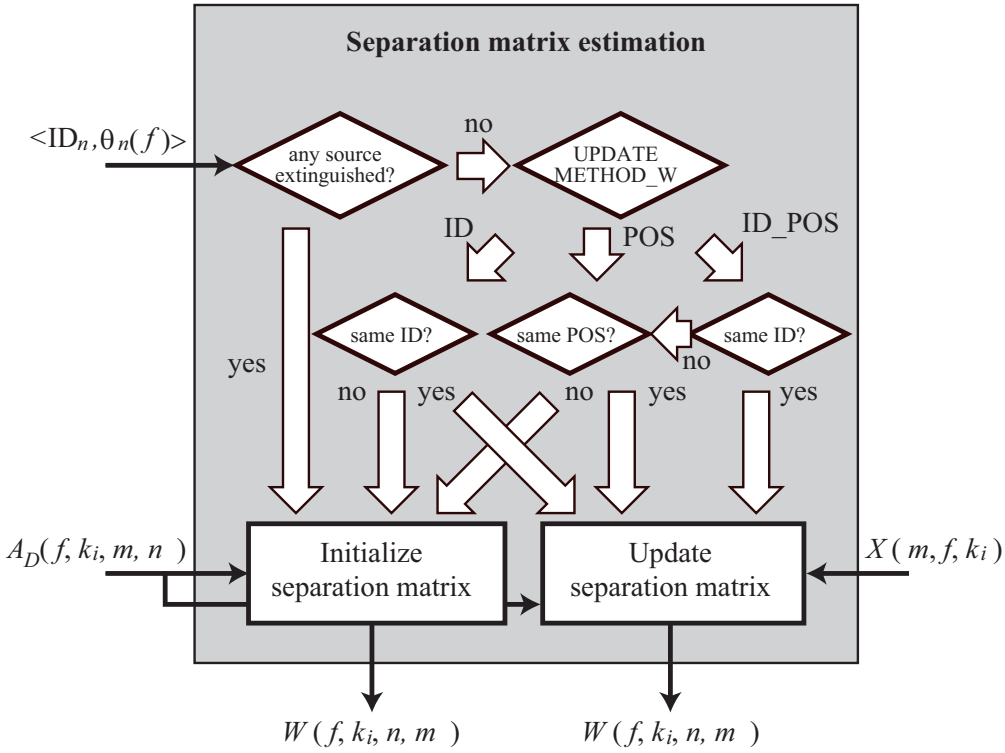


図 6.58: 分離行列推定の流れ図

定位方向を，現在のフレームと比較して，分離行列を継続して使うか，初期化するかを決定する．

UPDATE_METHOD_W が ID

1. 前フレーム ID と比較
   - 同じ: W を更新
   - 異なる: W を初期化

UPDATE_METHOD_W が POS

1. 前フレーム定位方向と比較
   - 誤差が UPDATE_ACCEPT_DISTANCE 未満: W を更新
   - 誤差が UPDATE_ACCEPT_DISTANCE 以上: W を初期化

UPDATE_METHOD_W が ID_POS

1. 前フレーム ID と比較
   - 同じ: W を更新
2. ID が異なった場合，定位方向を比較
   - 誤差が UPDATE_ACCEPT_DISTANCE 未満: W を更新
   - 誤差が UPDATE_ACCEPT_DISTANCE 以上: W を初期化

分離行列の書き出し (EXPORT_W が `true` のとき): EXPORT_W が `true` のとき，収束した分離行列を EXPORT_W_FILENAME で指定したファイルに出力する．
複数の音源が検出された場合，それらの分離行列は全て１つのファイルに出力される．音源が消滅した時点で，その分離行列をファイルに書き出す．
ファイルに書き出す際は，既に保存されている音源の定位方向と比較して，既存音源を上書きするか，新たな音源として追加するかを決定する．

音源が消滅

1. 既に保存されている音源の定位方向と比較
   - 誤差が UPDATE_ACCEPT_DISTANCE 未満: W を上書き保存
   - 誤差が UPDATE_ACCEPT_DISTANCE 以上: W を追加保存

6.3.7 HRLE

ノードの概要

本ノードは，Histogram-based Recursive Level Estimation (HRLE) 法によって定常ノイズレベルを推定する．HRLE は，入力スペクトルのヒストグラム（頻度分布）を計算し，その累積分布とパラメータ Lx により指定した正規化累積頻度からノイズレベルを推定する．ヒストグラムは，指数窓により重み付けされた過去の入力スペクトルから計算され，１フレームごとに指数窓の位置は更新される．

必要なファイル

無し

使用方法

どんなときに使うのか

スペクトル減算によるノイズ抑圧を行うときに用いる．

典型的な接続例

図 6.59 に示すように，入力は GHDSS などの分離ノードの後に接続し，出力は CalcSpecSubGain などの最適ゲインを求めるノードに接続する．図 6.60 は，EstimateLeak を併用した場合の接続例である．

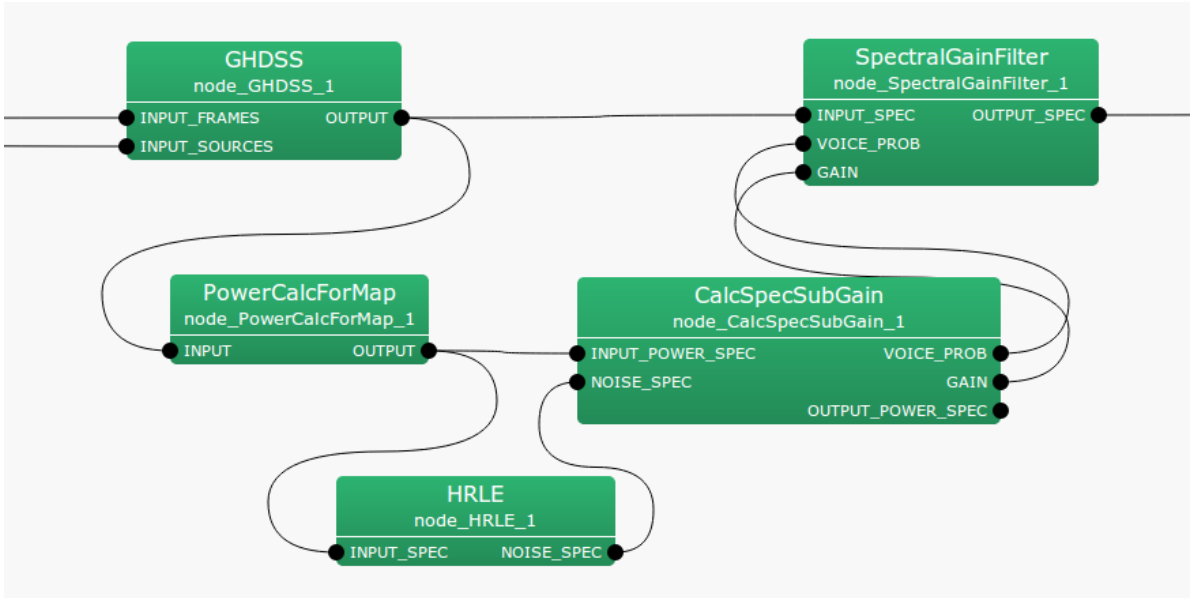


図 6.59: HRLE の接続例 1

ノードの入出力とプロパティ

入力

INPUT_SPEC ：`Map<int, ObjectRef>` 型．入力信号のパワースペクトル

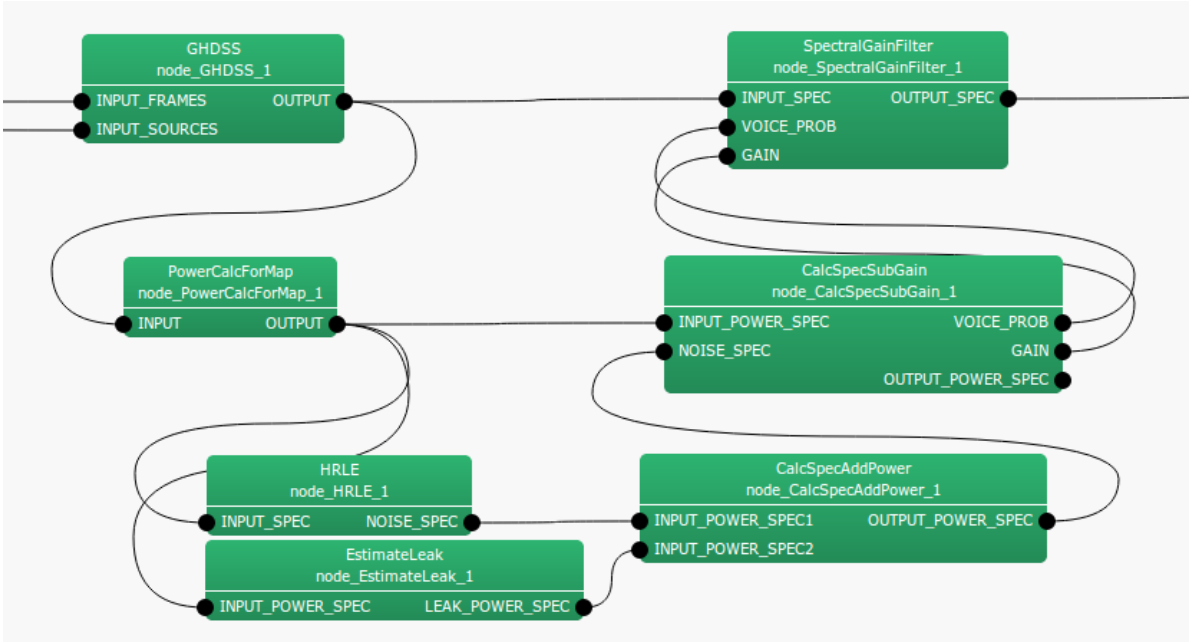


図 6.60: HRLE の接続例 2

表 6.50: HRLE のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LX	float	0.85		正規化累積頻度（Lx 値）.
TIME_CONSTANT_METHOD	string	LEGACY		時定数の定義方法.
TIME_CONSTANT	float		[pt]	時定数.
DECAY_FACTOR	int		[ms]	時定数.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
NUM_BIN	float	1000		ヒストグラムのビン数.
MIN_LEVEL	float	-100	[dB]	ヒストグラムの最小レベル.
STEP_LEVEL	float	0.2	[dB]	ヒストグラムのビンの幅.
DEBUG	bool	false		デバッグモード.

出力

NOISE_SPEC ：`Map<int, ObjectRef>` 型．推定ノイズのパワースペクトル

パラメータ

LX ：`float` 型．累積頻度分布上の正規化累積頻度を 0–1 の範囲で指定する．0 を指定すると最小レベル，1 を指定すると最大レベル，0.5 を指定するとメジアン（中央値）を推定する．デフォルトは 0.85.

TIME_CONST_METHOD ：`string` 型．時定数の定義方法をサンプル数で指定する LEGACY または、ミリ秒で指定する MILLISECOND から選択する．デフォルトは LEGACY．LEGACY の場合，パラメータ TIME_CONSTANT により時定数を指定し，MILLISECOND の場合，パラメータ DECAY_FACTOR により時定数を指定する．

TIME_CONSTANT ：`float` 型．時定数（0 以上）を時間サンプル単位で指定する．

DECAY_FACTOR ：`int` 型．時定数（0 以上）をミリ秒で指定する．

ADVANCE ： `int` 型．フレームのシフト長 [samples] を指定する．デフォルトは 160．前段階のノード (AudioStreamFromMic ， MultiFFT など) における値と一致している必要がある．

SAMPLING_RATE ：`int` 型．入力波形のサンプリング周波数 [Hz] を指定する．デフォルトは 16000.

NUM_BIN ：`float` 型．ヒストグラムのビン数を指定する．デフォルトは 1000.

MIN_LEVEL ：`float` 型．ヒストグラムの最小レベル [dB] を指定する．デフォルトは -100.

STEP_LEVEL ：`float` 型．ヒストグラムのビンの幅 [dB] を指定する．デフォルトは 0.2.

DEBUG ：`bool` 型．デバッグモードを指定する．デフォルトは `false`．デバッグモード (`true`) の場合，累積ヒストグラムの値がコンマ区切りテキストファイル形式で 100 フレーム毎に標準出力に出力される．出力値は，複数の行と列を含む複素行列数値形式であり，行は周波数ビンの位置，列はヒストグラムの位置，各要素は丸括弧で区切られた複素数値（左側が実数，右側が虚数部）を示す.（累積ヒストグラムは，実数値であるため，通常では虚数部は 0 である．しかし今後のバージョンでも 0 であることは保障されない.）1 つのサンプルに対する累積ヒストグラムの加算値は，1 ではなく指数的に増大している（高速化のため）．そのため累積ヒストグラム値は，累積頻度そのものを表してはいない事に注意されたい．各行の累積ヒストグラム値のほとんどが 0 で，最後の列に近い位置のみに値を含む場合，入力値が設定したヒストグラムのレベル範囲を超えて大きい状態（オーバーフロー状態）にあるので，NUM_BIN, MIN_LEVEL,STEP_LEVEL の一部またはすべてを高い値に設定しなおすべきである．また逆に各行の累積ヒストグラム値がほとんど一定値で，最初の列に近い位置のみに異なる低い値が含まれる場合，入力値が設定したヒストグラムのレベル範囲より小さい状態（アンダーフロー状態）にあるので，MIN_LEVEL を低い値に設定しなおすべきである．
出力の例：

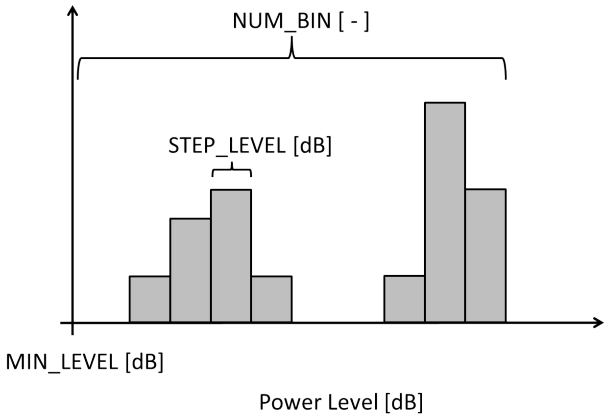


図 6.61: NUM_BIN，MIN_LEVEL，STEP_LEVEL の関係

```
---------- 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
```

ノードの詳細

図 6.62 に HRLE の処理フローを示す．HRLE は，入力パワーからレベルのヒストグラムを求め，その累積分布から Lx レベルを推定する処理となっている．Lx レベルとは，図 6.63 に示すように，累積頻度分布上の正規化累積頻度が x になるレベルである．x は，パラメータであり，例えば，$x = 0$ であれば最小値，$x = 1$ であれば最大値，$x = 0.5$ であれば中央値を推定する処理となる．

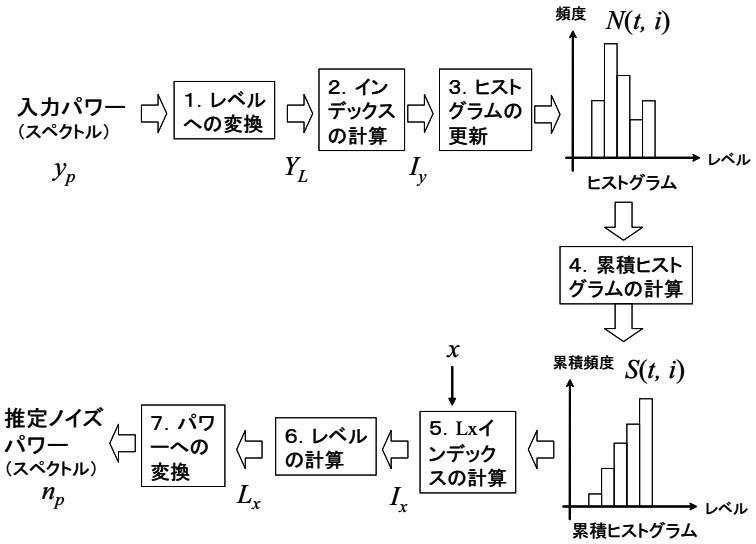


図 6.62: HRLE の処理フロー

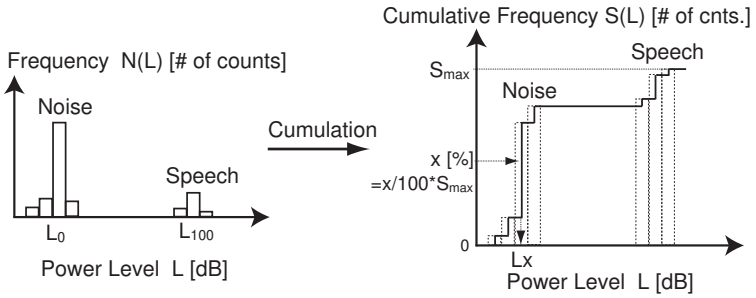


図 6.63: Lx 値の推定

HRLE の具体的な処理手順は，下記の 7 つの数式（図 6.62 の各処理に対応）で示すとおりである．式中で，t は時刻（フレーム単位），y_p は入力パワー (INPUT_SPEC)，n_p は推定ノイズパワー (NOISE_SPEC])，x，α，L_{min}，L_{step} はヒストグラムに関わるパラメータでそれぞれ正規化累積頻度 (LX)，時定数 (TIME_CONSTANT)，ビンの最小レベル (MIN_LEVEL)，ビンのレベル幅 (STEP_LEVEL)，$\lfloor a \rfloor$ は a 以下の a に最も近い整数を示して

いる．また，パラメータを除く全ての変数は，周波数の関数であり，各周波数毎に独立して同じ処理が施される．式中では，簡略化のため周波数を省略した．

$$
\begin{aligned}
Y_L(t) &= 10\log_{10} y_p(t), && (6.66)\\
I_y(t) &= \lfloor (Y_L(t) - L_{min})/L_{step} \rfloor, && (6.67)\\
N(t,l) &= \alpha N(t-1,l) + (1-\alpha)\delta(l - I_y(t)), && (6.68)\\
S(t,l) &= \sum_{k=0}^{l} N(t,k), && (6.69)\\
I_x(t) &= \underset{I}{\operatorname{argmin}} \left[S(t, I_{max}) \frac{x}{100} - S(t,I) \right], && (6.70)\\
L_x(t) &= L_{min} + L_{step} \cdot I_x(t), && (6.71)\\
n_p(t) &= 10^{L_x(t)/10} && (6.72)
\end{aligned}
$$

参考文献

(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

ノードの概要

最尤推定法 (Maximum Likelihood estimation) を用いた音源分離を行う．本アルゴリズムでは入力信号は単一の目的音源とガウス雑音の和であると仮定し，尤度関数最大を条件に分離行列を求める．音源からマイクロホンまでの伝達関数情報，音源の区間情報（発話区間の検出結果），および既知雑音の相関行列が必要となる．

ノードの入力は，

- 混合音のマルチチャネル複素スペクトル
- 目的音源の方向データ
- 既知雑音の相関行列

である．また，出力は分離音ごとの複素スペクトルである．

必要なファイル

表 6.51: ML に必要なファイル

対応するパラメータ名	説明
TF_CONJ_FILENAME	マイクロホンアレーの伝達関数

使用方法

どんなときに使うのか

所与の音源方向に対して，マイクロホンアレーを用いて当該方向の音源分離を行う．なお，音源方向として，音源定位部での推定結果，あるいは，定数値を使用することができる．

典型的な接続例

ML ノードの接続例を図 6.64 に示す．入力は以下である．

1. INPUT_FRAMES : MultiFFT 等から得られる混合音の多チャネル複素スペクトル
2. INPUT_SOURCES : LocalizeMUSIC や ConstantLocalization 等から得られる目的音源方向
3. INPUT_NOISE_CM : CMLoad 等から得られる既知雑音の相関行列

出力は分離音声となる．

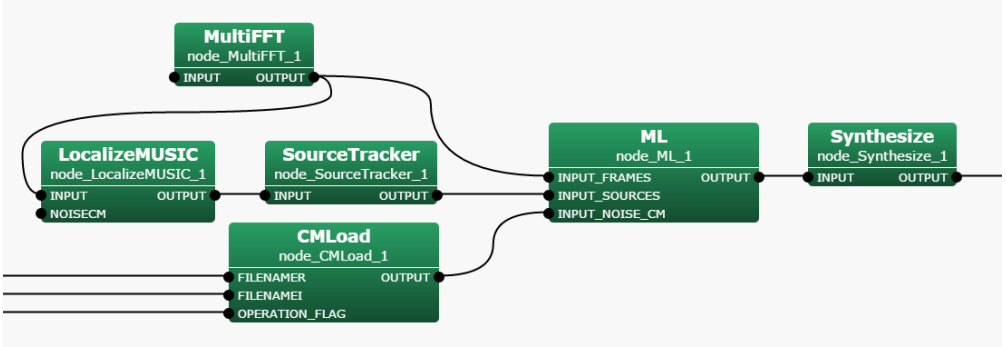


図 6.64: ML の接続例

ノードの入出力とプロパティ

入力

INPUT_FRAMES : `Matrix<complex<float> >` 型．マルチチャネル複素スペクトル．行がチャネル，つまり，各マイクロホンから入力された波形の複素スペクトルに対応し，列が周波数ビンに対応する．

INPUT_SOURCES : `Vector<ObjectRef>` 型．音源定位結果等が格納された `Source` 型オブジェクトの `Vector` 配列である．典型的には， SourceTracker ノード， SourceIntervalExtender ノードと繋げ，その出力を用いる．

INPUT_NOISE_CM : `Matrix<complex<float> >` 型．各周波数ビン毎の相関行列．行は周波数ビン ($NFFT/2+1$ 行)，列は M 次の複素正方行列である相関行列 ($M * M$ 列) に対応する．

出力

OUTPUT : `Map<int, ObjectRef>` 型．分離音の音源 ID と，分離音の 1 チャネル複素スペクトル (`Vector<complex<float> >` 型) のペア．分離音の数だけ出力される．

パラメータ

LENGTH : `int` 型．分析フレーム長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 512.

ADVANCE : `int` 型．フレームのシフト長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 160.

SAMPLING_RATE : `int` 型．入力音源波形のサンプリング周波数 [Hz]．デフォルト値は 16000.

LOWER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最小周波数値であり，これより下の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．0 以上サンプリング周波数値の半分までの範囲で指定する．

UPPER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最大周波数値であり，これより上の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY である必要がある．

TF_CONJ_FILENAME : `string` 型．伝達関数の記述されたバイナリファイル名を記す．ファイルフォーマットは 5.3.1 節を参照．

REG_FACTOR : `float` 型．係数．式 (6.76) 参照．デフォルト値は 0.0001．

ENABLE_DEBUG : `bool` 型．デフォルトは `false`．`true` が与えられると，分離状況が標準出力に出力される．

表 6.52: ML で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	`int`	512	[pt]	分析フレーム長.
ADVANCE	`int`	160	[pt]	フレームのシフト長.
SAMPLING_RATE	`int`	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	`int`	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	`int`	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	`string`			マイクロホンアレーの伝達関数を記したファイル名.
REG_FACTOR	`float`	0.0001		係数．式 (6.76).
ENABLE_DEBUG	`bool`	`false`		デバッグ出力の可否.

ノードの詳細

技術的な詳細: 基本的に詳細は下記の参考文献を参照されたい．

音源分離概要: 音源分離問題で用いる記号を表 6.53 にまとめる．演算はフレーム毎に周波数領域において行われるため，各記号は周波数領域での，一般には複素数の値を表す．音源分離は K 個の周波数ビン ($1 \leq k \leq K$) それぞれに対して演算が行われるが，本節ではそれを略記する．N, M, f をそれぞれ，音源数，マイク数，フレームインデックスとする．

表 6.53: 変数の定義

変数	説明
$\boldsymbol{S}(f) = [S_1(f), \ldots, S_N(f)]^T$	f フレーム目の音源の複素スペクトル
$\boldsymbol{X}(f) = [X_1(f), \ldots, X_M(f)]^T$	マイクロホン観測複素スペクトルのベクトル．INPUT_FRAMES 入力に対応.
$\boldsymbol{N}(f) = [N_1(f), \ldots, N_M(f)]^T$	加法性雑音
$\boldsymbol{H} = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_N] \in \mathbb{C}^{M \times N}$	$1 \leq n \leq N$ 番目の音源から $1 \leq m \leq M$ 番目のマイクまでの伝達関数行列
$\boldsymbol{K}(f) \in \mathbb{C}^{M \times M}$	既知雑音相関行列
$\boldsymbol{W}(f) = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_M] \in \mathbb{C}^{N \times M}$	分離行列
$\boldsymbol{Y}(f) = [Y_1(f), \ldots, Y_N(f)]^T$	分離音複素スペクトル

音のモデルは以下の一般的な線形モデルを扱う．

$$\boldsymbol{X}(f) = \boldsymbol{H}\boldsymbol{S}(f) + \boldsymbol{N}(f) \tag{6.73}$$

分離の目的は，

$$\boldsymbol{Y}(f) = \boldsymbol{W}(f)\boldsymbol{X}(f) \tag{6.74}$$

として，$\boldsymbol{Y}(f)$ が $\boldsymbol{S}(f)$ に近づくように，$\boldsymbol{W}(f)$ を推定することである．

最尤法に基づく分離行列 W_{ML} は次式であらわされる．

$$W_{\mathrm{ML}}(f) = \frac{\tilde{\boldsymbol{K}}^{-1}(f)\boldsymbol{H}}{\boldsymbol{H}^H\tilde{\boldsymbol{K}}^{-1}(f)\boldsymbol{H}} \tag{6.75}$$

ここで，

$$\tilde{\boldsymbol{K}}(f) = \boldsymbol{K}(f) + \|\boldsymbol{K}(f)\|_{\mathrm{F}}\alpha\boldsymbol{I} \tag{6.76}$$

であり，ここで $\|\boldsymbol{K}(f)\|_{\mathrm{F}}$ は既知雑音相関行列 $\boldsymbol{K}(f)$ のフロベニウスノルム，α はパラメータ REG_FACTOR，$\boldsymbol{I}$ は単位行列である．

トラブルシューティング: 基本的には GHDSS ノードのトラブルシューティングと同じ．

参考文献

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

6.3.9 MSNR

ノードの概要

最大 SNR 法 (Maximum Signal-to-Noise Ratio) を用いた音源分離を行う．本アルゴリズムでは目的音源方向のゲインと既知雑音方向のゲインの比が最大となるように，分離行列を更新し音源分離を行う．音源からマイクロホンまでの伝達関数情報を事前に与える必要はないが、音源の区間情報（発話区間の検出結果）が必要となる．

ノードの入力は，

- 混合音のマルチチャネル複素スペクトル
- 目的音源の方向データ

である．また，出力は分離音ごとの複素スペクトルである．

必要なファイル

無し．

使用方法

どんなときに使うのか

所与の音源方向に対して，マイクロホンアレーを用いて当該方向の音源分離を行う．なお，音源方向として，音源定位部での推定結果，あるいは，定数値を使用することができる．目的音源のゲインと既知雑音のゲインの比を利用するため，既知雑音の発生区間情報が必要となる. 本ノードでは音源方向のデータ入力がない時間区間を雑音発生区間として扱うものとする．

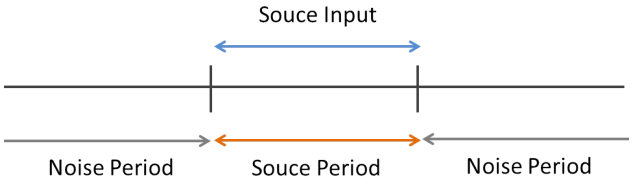


図 6.65: 雑音発生区間の扱い

典型的な接続例

MSNR ノードの接続例を図 6.66 に示す．入力は以下である．

1. INPUT_FRAMES : MultiFFT 等から得られる混合音の多チャネル複素スペクトル
2. INPUT_SOURCES : LocalizeMUSIC や ConstantLocalization 等から得られる目的音源方向

出力は分離音声となる．

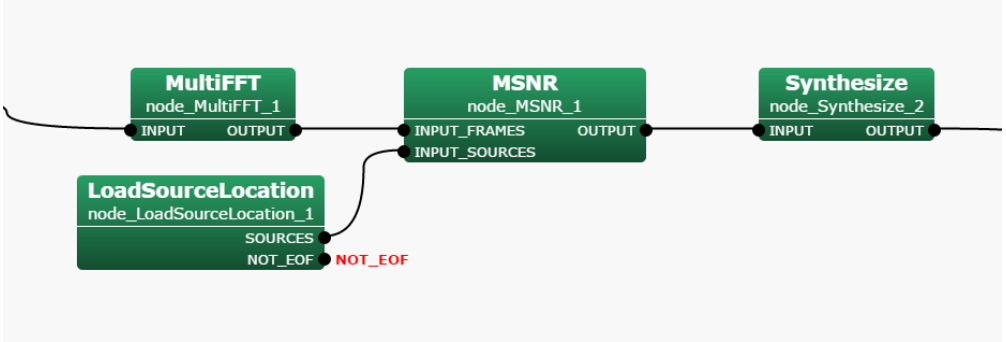


図 6.66: MSNR の接続例

ノードの入出力とプロパティ

入力

INPUT_FRAMES : `Matrix<complex<float> >` 型．マルチチャネル複素スペクトル．行がチャネル，つまり，各マイクロホンから入力された波形の複素スペクトルに対応し，列が周波数ビンに対応する．

INPUT_SOURCES : `Vector<ObjectRef>` 型．目的音源の方向情報が格納された `Source` 型オブジェクトの `Vector` 配列である．典型的には， SourceTracker ノード，SourceIntervalExtender ノードと繋げ，その出力を用いる．

出力

OUTPUT : `Map<int, ObjectRef>` 型．分離音の音源 ID と，分離音の 1 チャネル複素スペクトル (`Vector<complex<float> >` 型) のペア．分離音の数だけ出力される．

パラメータ

LENGTH : `int` 型．分析フレーム長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 512.

ADVANCE : `int` 型．フレームのシフト長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 160.

SAMPLING_RATE : `int` 型．入力音源波形のサンプリング周波数 [Hz]．デフォルト値は 16000.

LOWER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最小周波数値であり，これより下の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．0 以上サンプリング周波数値の半分までの範囲で指定する．

UPPER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最大周波数値であり，これより上の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY である必要がある．

DECOMPOSITION_ALGORITHM : `string` 型．音源分離で用いる演算アルゴリズムの選択．GEVD は一般化固有値分解を，GSVD は一般化特異値分解を表す．GEVD は GSVD に比べて雑音抑制性能が良好だが計算時間がかかる．使用目的や計算機環境に応じて演算アルゴリズムの使い分けができる．

ALPHA ：float 型．フィルタ更新係数．デフォルト値は 0.99.

ENABLE_DEBUG ：bool 型．デフォルトは false．true が与えられると, 分離状況が標準出力に出力される.

表 6.54: MSNR で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長.
ADVANCE	int	160	[pt]	フレームのシフト長.
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	int	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	int	8000	[Hz]	分離処理で用いる周波数の最大値.
DECOMPOSITION_ALGORITHM	string	GEVD		演算アルゴリズム.
ALPHA	float	0.99		フィルタ更新係数.
ENABLE_DEBUG	bool	false		デバッグ出力の可否.

ノードの詳細

技術的な詳細: 基本的に詳細は下記の参考文献を参照されたい.

音源分離概要: 音源分離問題で用いる記号を表 6.55 にまとめる．演算はフレーム毎に周波数領域において行われるため，各記号は周波数領域での，一般には複素数の値を表す．音源分離は K 個の周波数ビン $(1 \leq k \leq K)$ それぞれに対して演算が行われるが，本節ではそれを略記する．N, M, f をそれぞれ，音源数，マイク数，フレームインデックスとする.

表 6.55: 変数の定義

変数	説明
$\boldsymbol{S}(f) = [S_1(f), \ldots, S_N(f)]^T$	f フレーム目の音源の複素スペクトル
$\boldsymbol{X}(f) = [X_1(f), \ldots, X_M(f)]^T$	マイクロホン観測複素スペクトルのベクトル．INPUT_FRAMES 入力に対応
$\boldsymbol{N}(f) = [N_1(f), \ldots, N_M(f)]^T$	加法性雑音
$\boldsymbol{H} = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_N] \in \mathbb{C}^{M \times N}$	$1 \leq n \leq N$ 番目の音源から $1 \leq m \leq M$ 番目のマイクまでの伝達関数行列
$\boldsymbol{K}(f) \in \mathbb{C}^{M \times M}$	既知雑音相関行列
$\boldsymbol{W}(f) = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_M] \in \mathbb{C}^{N \times M}$	分離行列
$\boldsymbol{Y}(f) = [Y_1(f), \ldots, Y_N(f)]^T$	分離音複素スペクトル

音のモデルは以下の一般的な線形モデルを扱う.

$$\boldsymbol{X}(f) = \boldsymbol{H}\boldsymbol{S}(f) + \boldsymbol{N}(f) \tag{6.77}$$

分離の目的は,

$$\boldsymbol{Y}(f) = \boldsymbol{W}(f)\boldsymbol{X}(f) \tag{6.78}$$

として，$\boldsymbol{Y}(f)$ が $\boldsymbol{S}(f)$ に近づくように，$\boldsymbol{W}(f)$ を推定することである.

目的音信号の相関行列を $\boldsymbol{R}_{ss}(f)$，雑音信号の相関行列を $\boldsymbol{R}_{nn}(f)$ とすると，分離行列更新のための評価関数 $J_{\mathrm{MSNR}}(\boldsymbol{W}(f))$ は，以下のように表される．

$$J_{\mathrm{MSNR}}(\boldsymbol{W}(f)) = \frac{\boldsymbol{W}(f))\boldsymbol{R}_{ss}(f)\boldsymbol{W}(f))^H}{\boldsymbol{W}(f))\boldsymbol{R}_{nn}(f)\boldsymbol{W}(f))^H} \tag{6.79}$$

MSNR では，$J_{\mathrm{MSNR}}(\boldsymbol{W}(f))$ を最大とする $\boldsymbol{W}(f)$ を一般化固有値分解または一般化特異値分解を用いて求めている．

ここで，目的音信号の相関行列 $\boldsymbol{R}_{ss}(f)$ は，INPUT_SOURCES 入力端子に目的音源の方向データ入力がある時間区間の信号から得た相関行列 $\boldsymbol{R}_{xx}(f)$ を使って，以下のように更新される．

$$\boldsymbol{R}_{ss}(f+1) = \alpha\boldsymbol{R}_{ss}(f) + (1-\alpha)\boldsymbol{R}_{xx}(f) \tag{6.80}$$

一方，雑音信号の相関行列 $\boldsymbol{R}_{nn}(f)$ は，INPUT_SOURCES 入力端子に目的音源の方向データ入力がない時間区間（雑音発生区間）の信号から得た相関行列 $\boldsymbol{R}_{xx}(f)$ を使って，以下のように更新される．

$$\boldsymbol{R}_{nn}(f+1) = \alpha\boldsymbol{R}_{nn}(f) + (1-\alpha)\boldsymbol{R}_{xx}(f) \tag{6.81}$$

式 (6.80) と式 (6.81) の α が プロパティ ALPHA で指定可能である．

$\boldsymbol{R}_{ss}(f)$ と $\boldsymbol{R}_{nn}(f)$ から $\boldsymbol{W}(f)$ が更新される．

トラブルシューティング: 基本的には GHDSS ノードのトラブルシューティングと同じ．

参考文献

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

6.3.10 MVDR

ノードの概要

最小分散無歪応答法 (Minimum Variance Distortionless Response; MVDR) を用いた音源分離を行う．本アルゴリズムでは目的音源を歪ませない線形拘束条件の下で，出力パワーを最小化するような分離行列を求める．音源からマイクロホンまでの伝達関数情報，音源の区間情報（発話区間の検出結果），および既知雑音の相関行列が必要となる．

ノードの入力は，

- 混合音のマルチチャネル複素スペクトル
- 目的音源の方向データ
- 既知雑音の相関行列

である．また，出力は分離音ごとの複素スペクトルである．

必要なファイル

表 6.56: MVDR に必要なファイル

対応するパラメータ名	説明
TF_CONJ_FILENAME	マイクロホンアレーの伝達関数.

使用方法

どんなときに使うのか

所与の音源方向に対して，マイクロホンアレーを用いて当該方向の音源分離を行う．なお，音源方向として，音源定位部での推定結果，あるいは，定数値を使用することができる．

典型的な接続例

MVDR ノードの接続例を図 6.67 に示す．入力は以下である．

1. INPUT_FRAMES : MultiFFT 等から得られる混合音の多チャネル複素スペクトル
2. INPUT_SOURCES : LocalizeMUSIC や ConstantLocalization 等から得られる目的音源方向
3. INPUT_NOISE_CM : CMLoad 等から得られる既知雑音の相関行列

出力は分離音声となる．

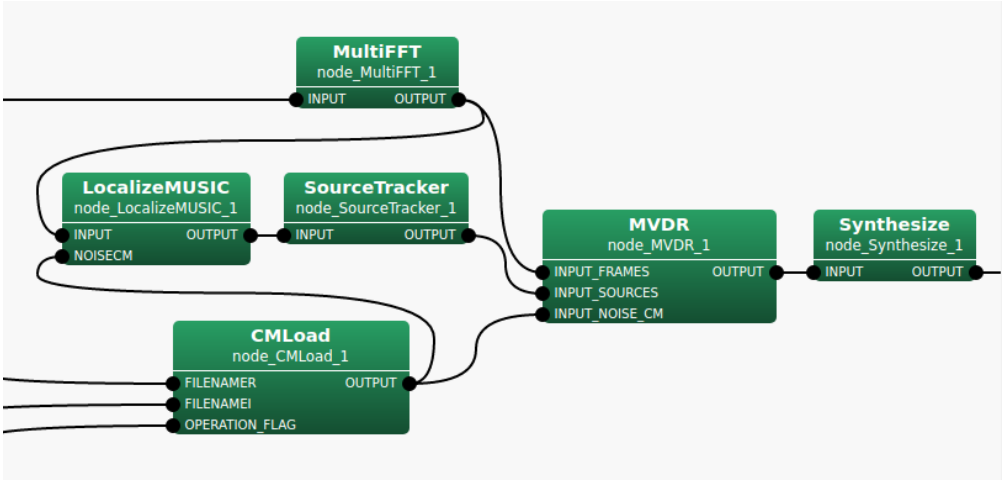


図 6.67: MVDR の接続例

ノードの入出力とプロパティ

入力

INPUT_FRAMES : `Matrix<complex<float> >` 型．マルチチャネル複素スペクトル．行がチャネル，つまり，各マイクロホンから入力された波形の複素スペクトルに対応し，列が周波数ビンに対応する．

INPUT_SOURCES : `Vector<ObjectRef>` 型．音源定位結果等が格納された `Source` 型オブジェクトの `Vector` 配列である．典型的には， SourceTracker ノード， SourceIntervalExtender ノードと繋げ，その出力を用いる．

INPUT_NOISE_CM : `Matrix<complex<float> >` 型．各周波数ビン毎の相関行列．行は周波数ビン ($NFFT/2+1$ 行)，列は M 次の複素正方行列である相関行列 ($M*M$ 列) に対応する．この入力は必須であるためノイズ相関行列を入力しない場合は、 CMIdentityMatrix ノードと繋げ，その出力を用いる．

出力

OUTPUT : `Map<int, ObjectRef>` 型．分離音の音源 ID と，分離音の 1 チャネル複素スペクトル (`Vector<complex<float> >` 型) のペア．分離音の数だけ出力される．

パラメータ

LENGTH : `int` 型．分析フレーム長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 512.

ADVANCE : `int` 型．フレームのシフト長 [samples]．前段階のノード (AudioStreamFromMic， MultiFFT など) における値と一致している必要がある．デフォルト値は 160.

SAMPLING_RATE : `int` 型．入力音源波形のサンプリング周波数 [Hz]．デフォルト値は 16000.

LOWER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最小周波数値であり，これより下の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．0 以上サンプリング周波数値の半分までの範囲で指定する．

UPPER_BOUND_FREQUENCY : `int` 型．分離処理を行う際に利用する最大周波数値であり，これより上の周波数に対しては処理を行わず，出力スペクトルの値は 0 となる．LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY である必要がある．

TF_CONJ_FILENAME : `string` 型．伝達関数の記述されたバイナリファイル名を記す．ファイルフォーマットは 5.3.1 節を参照．

REG_FACTOR : `float` 型．係数．式 (6.85) 参照．デフォルト値は 0.001．

ENABLE_DEBUG : `bool` 型．デフォルトは `false`．`true` が与えられると，分離状況が標準出力に出力される．

表 6.57: MVDR で利用するパラメータ

パラメータ名	型	デフォルト値	単位	説明
LENGTH	`int`	512	[pt]	分析フレーム長.
ADVANCE	`int`	160	[pt]	フレームのシフト長.
SAMPLING_RATE	`int`	16000	[Hz]	サンプリング周波数.
LOWER_BOUND_FREQUENCY	`int`	0	[Hz]	分離処理で用いる周波数の最小値.
UPPER_BOUND_FREQUENCY	`int`	8000	[Hz]	分離処理で用いる周波数の最大値.
TF_CONJ_FILENAME	`string`			マイクロホンアレーの伝達関数を記したファイル名.
REG_FACTOR	`float`	0.001		係数．式 (6.85).
ENABLE_DEBUG	`bool`	`false`		デバッグ出力の可否.

ノードの詳細

技術的な詳細: 基本的に詳細は下記の参考文献を参照されたい．

音源分離概要: 音源分離問題で用いる記号を表 6.58 にまとめる．演算はフレーム毎に周波数領域において行われるため，各記号は周波数領域での，一般には複素数の値を表す．音源分離は K 個の周波数ビン ($1 \leq k \leq K$) それぞれに対して演算が行われるが，本節ではそれを略記する．N, M, f をそれぞれ，音源数，マイク数，フレームインデックスとする．

表 6.58: 変数の定義

変数	説明
$\boldsymbol{S}(f) = [S_1(f), \ldots, S_N(f)]^T$	f フレーム目の音源の複素スペクトル
$\boldsymbol{X}(f) = [X_1(f), \ldots, X_M(f)]^T$	マイクロホン観測複素スペクトルのベクトル．INPUT_FRAMES 入力に対応.
$\boldsymbol{N}(f) = [N_1(f), \ldots, N_M(f)]^T$	加法性雑音
$\boldsymbol{H} = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_N] \in \mathbb{C}^{M \times N}$	$1 \leq n \leq N$ 番目の音源から $1 \leq m \leq M$ 番目のマイクまでの伝達関数行列
$\boldsymbol{K}(f) \in \mathbb{C}^{M \times M}$	既知雑音相関行列
$\boldsymbol{W}(f) = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_M] \in \mathbb{C}^{N \times M}$	分離行列
$\boldsymbol{Y}(f) = [Y_1(f), \ldots, Y_N(f)]^T$	分離音複素スペクトル

音のモデルは以下の一般的な線形モデルを扱う．

$$\boldsymbol{X}(f) = \boldsymbol{H}\boldsymbol{S}(f) + \boldsymbol{N}(f) \tag{6.82}$$

分離の目的は,

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

として，$\boldsymbol{Y}(f)$ が $\boldsymbol{S}(f)$ に近づくように，$\boldsymbol{W}(f)$ を推定することである．

MVDR 法に基づく分離行列 W_{MVDR} は次式であらわされる．

$$W_{\mathrm{MVDR}}(f) \quad = \quad \frac{\tilde{\boldsymbol{K}}^{-1}(f)\boldsymbol{H}}{\boldsymbol{H}^{H}\tilde{\boldsymbol{K}}^{-1}(f)\boldsymbol{H}} \tag{6.84}$$

ここで,

$$\tilde{\boldsymbol{K}}(f) \quad = \quad \boldsymbol{K}(f) + \alpha\boldsymbol{I} \tag{6.85}$$

であり，ここで α はパラメータ REG_FACTOR, $\boldsymbol{I}$ は相関行列の正則化を行うための対角行列である．

トラブルシューティング: 基本的には GHDSS ノードのトラブルシューティングと同じ．

参考文献

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

6.3.11 PostFilter

ノードの概要

このノードは，音源分離ノード GHDSS によって分離された複素スペクトルに対し，音声認識精度を向上するための後処理を行う．同時に，ミッシングフィーチャーマスクを生成するための，ノイズパワースペクトルの生成も行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

このノードは，GHDSS ノードによって分離されたスペクトルの整形と，ミッシングフィーチャーマスクを生成するために必要なノイズスペクトルを生成する時に用いる．

典型的な接続例

PostFilter ノードの接続例は図 6.68 の通り．入力の接続として，INPUT_SPEC は GHDSS ノードの出力，INIT_NOISE_POWER は BGNEstimator ノードの出力と接続する．

出力について，図 6.68 では

1. 分離音 (OUTPUT_SPEC) の音声特徴抽出 (MSLSExtraction ノード),
2. 分離音と分離音に含まれるノイズのパワー (EST_NOISE_POWER) から音声認識時のミッシングフィーチャーマスク生成 (MFMGeneration ノード)

の接続例を示している．

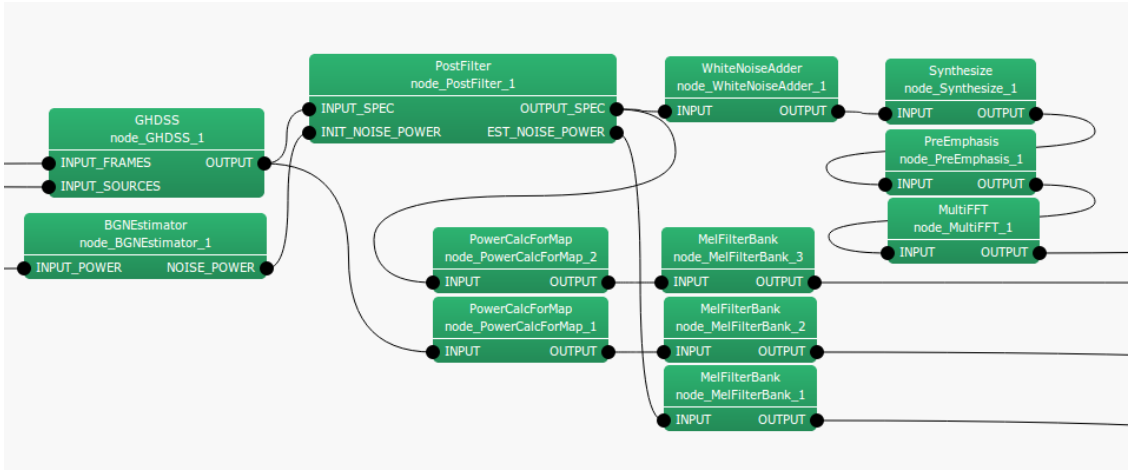


図 6.68: PostFilter の接続例

ノードの入出力とプロパティ

入力

INPUT_SPEC : `Map<int, ObjectRef>` 型．GHDSS ノードからの出力と同じ型．音源 ID と，分離音の複素スペクトルである `Vector<complex<float> >` 型データのペア．

INPUT_NOISE_POWER : `Matrix<float>` 型．BGNEstimator ノードによって推定された定常ノイズのパワースペクトル．

出力

OUTPUT_SPEC : `Map<int, ObjectRef>` 型．入力 INPUT_SPEC から，ノイズ除去がされた分離音の複素スペクトル．`Object` 部分は `Vector<complex<float> >` 型．

EST_NOISE_POWER : `Map<int, ObjectRef>` 型．OUTPUT_SPEC の各分離音に対して，含まれていると推定されたノイズのパワーが，`Vector<float>` 型データとして ID とペアになっている．

パラメータ

ノードの詳細

式で用いられる添字は，表 6.1 で定義されているものに準拠する．また，以降の式では，特に必要のない場合は，時間フレームインデックス f を省略して表記する．

図 6.69 は，PostFilter ノードの流れ図である．入力としては，GHDSS ノードからの分離音スペクトルと，BGNEstimator ノードの定常ノイズパワースペクトルが得られる．出力には，音声が強調された分離音スペクトルと，分離音に混入しているノイズのパワースペクトルである．

処理の流れは

1. ノイズ推定
2. SNR 推定
3. 音声存在確率推定
4. ノイズ除去

となっている．

1) ノイズ推定:

ノイズ推定処理の流れを図 6.70 に示す．PostFilter ノードが対処するノイズは，
a) マイクロホンの接点などが要因となる定常ノイズ ,
b) 除去しきれなかった別の音源の音 (漏れノイズ),
c) 前フレームの残響,
の 3 つである．

最終的な分離音に含まれるノイズ $\lambda(f, k_i)$ は,

$$\lambda(f,k_i) \quad = \quad \lambda^{sta}(f,k_i) + \lambda^{leak}(f,k_i) + \lambda^{rev}(f-1,k_i) \tag{6.86}$$

として求められる．ただし，$\lambda^{sta}(f,k_i), \lambda^{leak}(f,k_i), \lambda^{rev}(f-1,k_i)$ はそれぞれ，定常ノイズ，漏れノイズ，前フレームの残響を表す．

表 6.59: PostFilter のパラメータ表 (前半)

パラメータ名	型	デフォルト値	単位	説明
MCRA_SETTING	bool	false		ノイズ除去手法である，MCRA 推定に関するパラメータ設定項目を表示する時，true にする．
MCRA_SETTING				以下，MCRA_SETTING が true の時に表示される
STATIONARY_NOISE_FACTOR	float	1.2		定常ノイズ推定時の係数．
SPEC_SMOOTH_FACTOR	float	0.5		入力パワースペクトルの平滑化係数．
AMP_LEAK_FACTOR	float	1.5		漏れ係数．
STATIONARY_NOISE_MIXTURE_FACTOR	float	0.98		定常ノイズの混合比．
LEAK_FLOOR	float	0.1		漏れノイズの最小値．
BLOCK_LENGTH	int	80		検出時間幅．
VOICEP_THRESHOLD	int	3		音声存在判定の閾値．
EST_LEAK_SETTING	bool	false		漏れ率推定に関するパラメータ設定項目を表示する時，true にする．
EST_LEAK_SETTING				以下，EST_LEAK_SETTING が true の時に表示される．
LEAK_FACTOR	float	0.25		漏れ率．
OVER_CANCEL_FACTOR	float	1		漏れ率重み係数．
EST_REV_SETTING	bool	false		残響成分推定に関するパラメータ設定項目を表示する時，true にする．
EST_REV_SETTING				以下，EST_REV_SETTING が true の時に表示される．
REVERB_DECAY_FACTOR	float	0.5		残響パワーの減衰係数．
DIRECT_DECAY_FACTOR	float	0.2		分離スペクトルの減衰係数．
EST_SN_SETTING	bool	false		SN 比推定に関するパラメータ設定項目を表示する時，true にする．
EST_SN_SETTING				以下，EST_SN_SETTING が true の時に表示される．
PRIOR_SNR_FACTOR	float	0.8		事前 SNR と事後 SNR の比率．
VOICEP_PROB_FACTOR	float	0.9		音声存在確率の振幅係数．
MIN_VOICEP_PROB	float	0.05		最小音声存在確率．
MAX_PRIOR_SNR	float	100		事前 SNR の最大値．
MAX_OPT_GAIN	float	20		最適ゲイン中間変数 v の最大値．
MIN_OPT_GAIN	float	6		最適ゲイン中間変数 v の最小値．

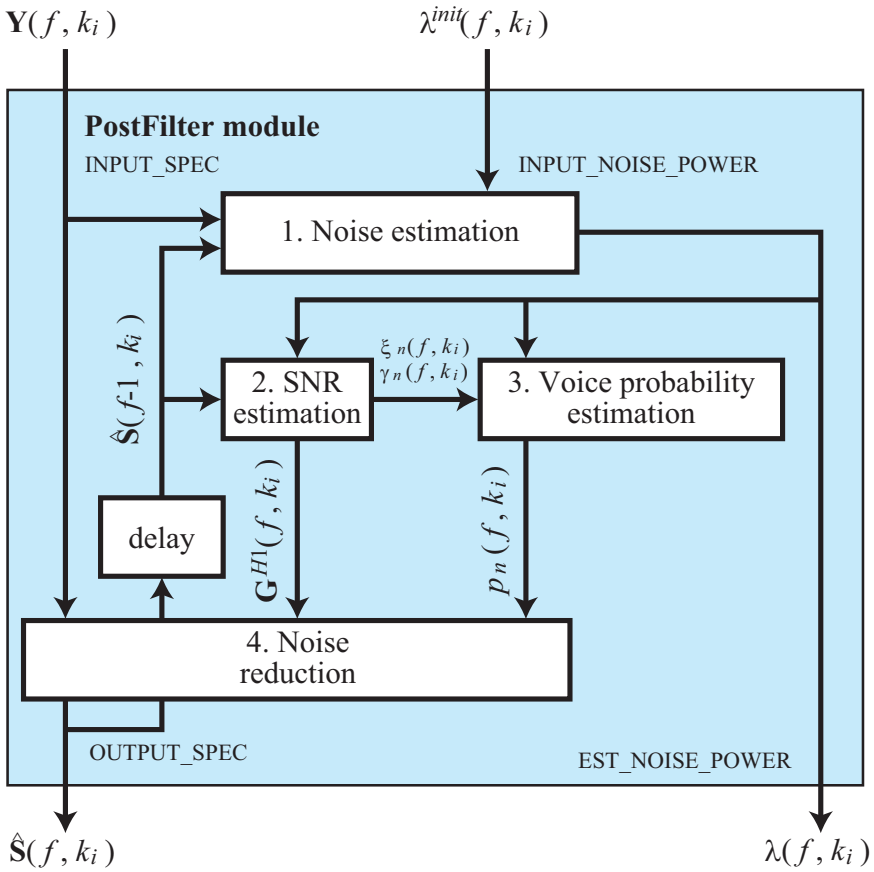


図 6.69: PostFilter の流れ図

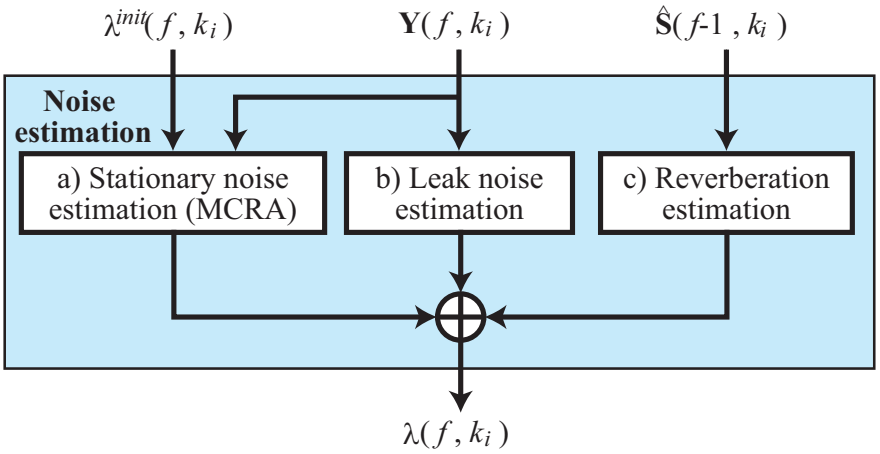


図 6.70: ノイズ推定の手順

1-a) MCRA 法による定常ノイズ推定　1-a) で用いる変数は 表 6.61 に基づく．

まず，入力スペクトルを 1 フレーム前のパワーと平滑化したパワースペクトル $S(f,k_i) = [S_1(f,k_i),\ldots,S_N(f,k_i)]$ を求める．

$$S_n(f,k_i) = \alpha_s S_n(f-1,k_i) + (1-\alpha_s)|Y_n(k_i)|^2 \tag{6.87}$$

次に，S^{tmp}，S^{min} を更新する．

$$S_n^{min}(f,k_i) = \begin{cases} \min\{S_n^{min}(f-1,k_i), S_n(f,k_i)\} & \text{if } f \neq nL \\ \min\{S_n^{tmp}(f-1,k_i), S_n(f,k_i)\} & \text{if } f = nL \end{cases}, \tag{6.88}$$

$$S_n^{min}(f,k_i) = \begin{cases} \min\{S_n^{tmp}(f-1,k_i), S_n(f,k_i)\} & \text{if } f \neq nL \\ S_n(f,k_i) & \text{if } f = nL \end{cases}, \tag{6.89}$$

ただし，n は任意の整数である．S^{min} はノイズ推定を始めてからの最小パワーを保持し，S^{tmp} は最近の L フレームの極小パワーを保持している．L フレームごとに S^{tmp} は更新される．

続いて，最小パワーと入力分離音のパワーの比から，音声が含まれるかどうかを判定する．

$$S_n^r(k_i) = \frac{S_n(k_i)}{S^{min}(k_i)}, \tag{6.90}$$

$$I_n(k_i) = \begin{cases} 1 & \text{if } S_n^r(k_i) > \delta \\ 0 & \text{if } S_n^r(k_i) \leq \delta \end{cases} \tag{6.91}$$

$I_n(k_i)$ に音声が含まれる場合 1，含まれない場合 0 となる．この判定結果をもとに，前フレーム定常ノイズと，現在のフレームのパワーとの混合比 $\alpha_{d,n}^C(k_i)$ を決める．

$$\alpha_{d,n}^C(k_i) = (\alpha_d - 1)I_n(k_i) + 1. \tag{6.92}$$

次に，分離音のパワースペクトルに含まれる漏れノイズを除去する．

$$S_n^{leak}(k_i) = \sum_{p=1}^{N} |Y_p(k_i)|^2 - |Y_n(k_i)|^2, \tag{6.93}$$

$$S_n^0(k_i) = |Y_n(k_i)|^2 - qS_n^{leak}(k_i), \tag{6.94}$$

ただし，$S_n^0(k_i) < S_{floor}$ のとき，

$$S_n^0(k_i) = S_{floor} \tag{6.95}$$

に値が変更される．

漏れノイズを除いたパワースペクトル $S_n^0(f,k_i)$ と，前フレームの推定定常ノイズ $\lambda^{sta}(f-1,k_i)$ または，BGNEstimator からの出力である $bf\lambda^{init}(f,k_i)$ を混合することで，現在のフレームの定常ノイズを求める．

$$\lambda_n^{sta}(f,k_i) = \begin{cases} \alpha_{d,n}^C(k_i)\lambda_n^{sta}(f-1,k_i) + (1-\alpha_{d,n}^C(k_i)rS_n^0(f,k_i) & \text{if音源位置に変更なし} \\ \alpha_{d,n}^C(k_i)\lambda_n^{init}(f,k_i) + (1-\alpha_{d,n}^C(k_i)rS_n^0(f,k_i) & \text{if音源位置に変更あり} \end{cases} \tag{6.96}$$

1-b) 漏れノイズ推定　1-b) で用いる変数は 表 6.62 に基づく．

いくつかのパラメータを次のように計算する．

$$\beta = -\frac{\alpha^{leak}}{1-(\alpha^{leak})^2 + \alpha^{leak}(1-\alpha^{leak})(N-2)} \tag{6.97}$$

$$\alpha = 1-(N-1)\alpha^{leak}\beta \tag{6.98}$$

このパラメータを用いて，平滑化されたスペクトル $S(k_i)$ と，式 (6.93) で求められた，他の分離音のパワーから自分の分離音のパワーを除いたパワースペクトル $S_n^{leak}(k_i)$ を混合する．

$$Z_n(k_i) \quad = \quad \alpha S_n(k_i) + \beta S_n^{leak}(k_i), \tag{6.99}$$

ただし，$Z_n(k_i) < 1$ になる場合は，$Z_n(k_i) = 1$ とする．

最終的な漏れノイズのパワースペクトル $\lambda^{leak}(k_i)$ は，

$$\lambda_n^{leak} \quad = \quad \alpha^{leak}\left(\sum_{n' \neq n} Z_{n'}(k_i)\right) \tag{6.100}$$

として求める．

1-c) 残響推定　1-c) で用いる変数は 表 6.63 に基づく．

残響のパワーは，前フレームの推定残響パワー $\lambda^{rev}(f-1,k_i) = [\lambda_1^{rev}(f-1,k_i),\ldots,\lambda_N^{rev}(f-1,k_i)]^T$ と，前フレームの分離スペクトル $\hat{S}(f-1,k_i) = [\hat{S}_1(f-1,k_i),\ldots,\hat{S}_N(f-1,k_i)]^T$ から次のように計算される．$\hat{S}_n(f-1,k_i)$ は複素数であることに注意．

$$\lambda_n^{rev}(f,k_i) \quad = \quad \gamma\left(\lambda_n^{rev}(f-1,k_i) + \Delta|\hat{S}_n(f-1,k_i)|^2\right) \tag{6.101}$$

2) SNR 推定:

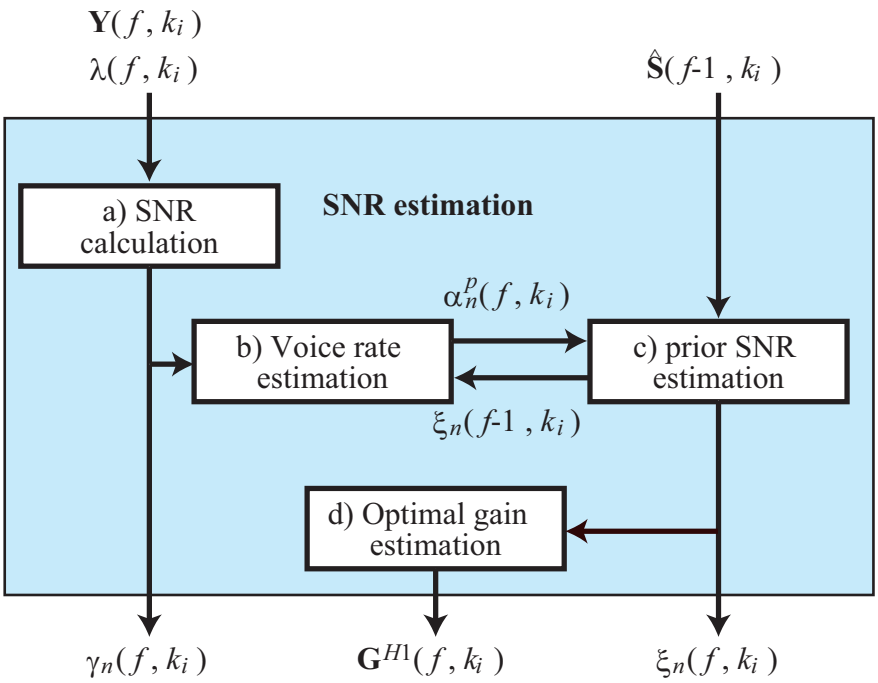


図 6.71: SNR 推定の手順

SNR 推定の流れを図 6.71 に示す．SNR 推定は，

a) SNR の計算

b) ノイズ混入前の事前 SNR 推定

c) 音声含有率の推定

d) 最適ゲインの推定
から成る．
表 6.64 のベクトルの要素は，各分離音の値に対応する．

2-a) SNR の計算　2-a) で用いる変数は，表 6.64 に従う．ここでは，入力の複素スペクトル $Y(k_i)$ と，前段で推定されたノイズのパワースペクトル $\lambda(k_i)$ を元に，SNR $\gamma_n(k_i)$ が計算される．

$$\gamma_n(k_i) = \frac{|Y_n(k_i)|^2}{\lambda_n(k_i)} \tag{6.102}$$

$$\gamma_n^C(k_i) = \begin{cases} \gamma_n(k_i) & \text{if } \gamma_n(k_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.103}$$

2-b) 音声含有率の推定　2-b) で用いる変数は，表 6.65 に従う．
音声含有率 $\alpha_n^p(f, k_i)$ は，前フレームの事前 SNR $\xi_n(f-1, k_i)$ を用いて次のように計算される．

$$\alpha_n^p(f, k_i) = \alpha_{mag}^p \left(\frac{\xi_n(f-1, k_i)}{\xi_n(f-1, k_i) + 1} \right)^2 + \alpha_{min}^p \tag{6.104}$$

2-c) ノイズ混入前の事前 SNR 推定　2-c) で用いる変数は，表 6.66 に従う．
事前 SNR $\xi_n(k_i)$ は，次のようにして計算する．

$$\xi_n(k_i) = \left(1 - \alpha_n^p(k_i)\right)\xi_{tmp} + \alpha_n^p(k_i)\gamma_n^C(k_i) \tag{6.105}$$

$$\xi_{tmp} = a\frac{|\hat{S}_n(f-1, k_i)|^2}{\lambda_n(f-1, k_i)} + (1-a)\xi_n(f-1, k_i) \tag{6.106}$$

ただし，ξ_{tmp} は計算上の一時的な変数で，前フレームの推定 SNR$\gamma_n(k_i)$ と，事前 SNR $\xi_n(k_i)$ の内分値である．また，$\xi_n(k_i) > \xi^{max}$ となる場合，$\xi_n(k_i) = \xi^{max}$ と値を変更する．

2-d) 最適ゲインの推定　2-d) で用いる変数は，表 6.67 に従う．
最適ゲイン計算の前に，上で求めた事前 SNR$\xi_n(k_i)$ と，推定 SNR$\gamma_n(k_i)$ を用いて，以下の中間変数 $v_n(k_i)$ を計算する．

$$v_n(k_i) = \frac{\xi_n(k_i)}{1 + \xi_n(k_i)}\gamma_n(k_i) \tag{6.107}$$

$v_n(k_i) > \theta^{max}$ の場合，$v_n(k_i) = \theta^{max}$ とする．
音声がある場合の最適ゲイン $G^{H1}(k_i) = [G_1^{H1}(k_i), \ldots, G_N^{H1}(k_i)]$ は，

$$G_n^{H1}(k_i) = \frac{\xi_n(k_i)}{1 + \xi_n(k_i)} \exp\left\{ \frac{1}{2} \int_{v_n(k_i)}^{\inf} \frac{e^{-t}}{t} \mathrm{d}t \right\} \tag{6.108}$$

として求める．ただし，

$$\begin{aligned} G_n^{H1}(k_i) &= 1 \quad \text{if} v_n(k_i) < \theta^{min} \\ G_n^{H1}(k_i) &= 1 \quad \text{if} G_n^{H1}(k_i) > 1. \end{aligned} \tag{6.109}$$

3) 音声存在確率推定:
音声存在確率推定の流れを図 6.72 に示す．音声存在確率推定は，
a) 3 種類の帯域ごとに事前 SNR の平滑化

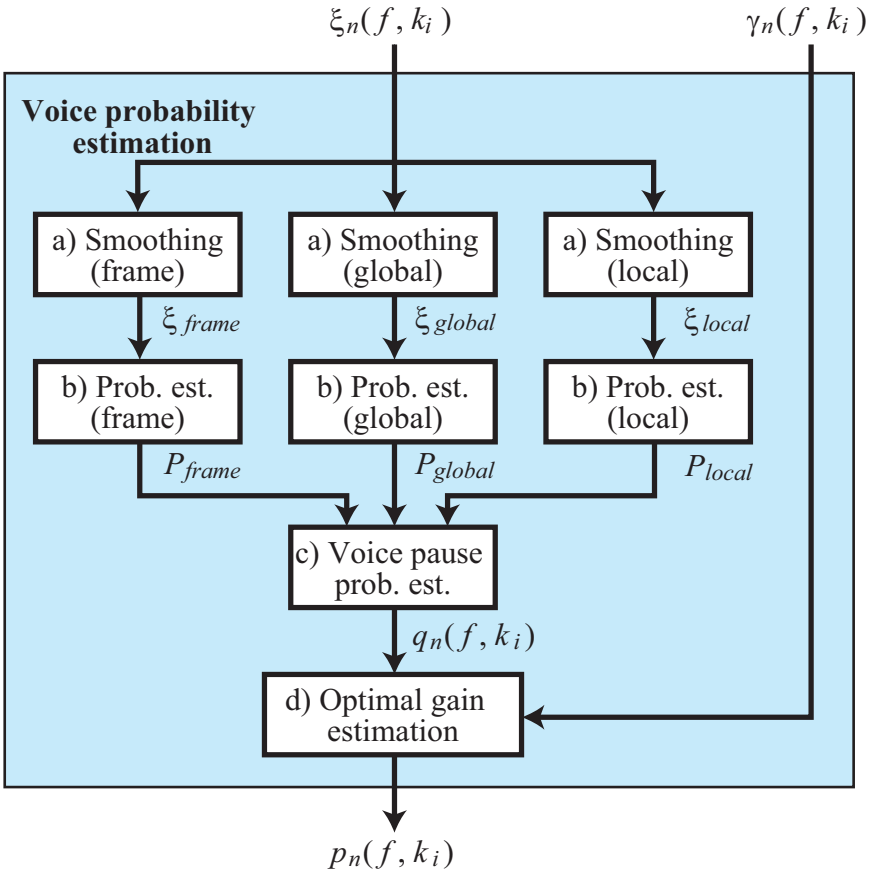


図 6.72: 音声存在確率推定の手順

b) 各帯域で，平滑化した SNR を元に，暫定的な音声確率を推定
c) 3 つの暫定確率をもとに音声休止確率を推定
d) 最終的な音声存在確率を推定
から成る．

3-a) 事前 SNR の平滑化　3-a) で用いる変数を表 6.68 にまとめる．

まず，式 (6.105) で計算された事前 SNR $\xi_n(f, k_i)$ と，前フレームの時間平滑化事前 SNR $\zeta_n(f-1, k_i)$ で，時間平滑化を行う．

$$\zeta_n(f, k_i) \quad = \quad b\zeta_n(f-1, k_i) + (1-b)\xi_n(f, k_i) \tag{6.110}$$

周波数方向の平滑化は，その窓の大きさによって，frame，global，local の順に小さくなっていく．

- frame での周波数平滑化
  周波数ビン $F_{st} \sim F_{en}$ の範囲で加算平均による平滑化が行われる．

$$\zeta_n^f(k_i) \quad = \quad \frac{1}{F_{en} - F_{st} + 1} \sum_{k_j = F_{st}}^{F_{en}} \zeta_n(k_j) \tag{6.111}$$

- global での周波数平滑化

global では，幅 G での hanning 窓を用いた周波数平滑化が行われる．

$$\zeta_n^g(k_i) = \sum_{j=-(G-1)/2}^{(G-1)/2} w_{han}(j+(G-1)/2)\zeta_n(k_{i+j}), \tag{6.112}$$

$$w_{han}(j) = \frac{1}{C}\left(0.5 - 0.5\cos\left(\frac{2\pi j}{G}\right)\right), \tag{6.113}$$

ただし，C は $\sum_{j=0}^{G-1} w_{han}(j) = 1$ にするための正規化係数.

- local での周波数平滑化
  local では，幅 F での三角窓を用いた周波数平滑化が行われる．

$$\zeta_n^l(k_i) = 0.25\zeta_n(k_i-1) + 0.5\zeta_n(k_i) + 0.25(k_i+1) \tag{6.114}$$

3-b) 暫定音声確率を推定　3-b) で用いる変数を表 6.69 に示す.

- $P_n^f(k_i)$ と $\zeta_n^{peak}(k_i)$ の計算
  まず，$\zeta_n^{peak}(f,k_i)$ を以下のように求める．

$$\zeta_n^{peak}(f,k_i) = \begin{cases} \zeta_n^f(f,k_i), & \text{if } \zeta_n^f(f,k_i) > Z_{thres}\zeta_n^f(f-1,k_i) \\ \zeta_n^{peak}(f-1,k_i), & \text{if otherwise.} \end{cases} \tag{6.115}$$

ただし，$\zeta_n^{peak}(k_i)$ の値はパラメータ $Z_{min}^{peak}, Z_{max}^{peak}$ の範囲に入るようにする．すなわち，

$$\zeta_n^{peak}(k_i) = \begin{cases} Z_{min}^{peak}, & \text{if } \zeta_n^{peak}(k_i) < Z_{min}^{peak} \\ Z_{max}^{peak}, & \text{if } \zeta_n^{peak}(k_i) > Z_{max}^{peak} \end{cases} \tag{6.116}$$

次に，$P_n^f(k_i)$ を次のように求める．

$$P_n^f(k_i) = \begin{cases} 0, & \text{if } \zeta_n^f(k_i) < \zeta_n^{peak}(k_i)Z_{min}^f \\ 1, & \text{if } \zeta_n^f(k_i) > \zeta_n^{peak}(k_i)Z_{max}^f \\ \frac{\log\left(\zeta_n^f(k_i)/\zeta_n^{peak}(k_i)Z_{min}^f\right)}{\log\left(Z_{max}^f/Z_{min}^f\right)}, & \text{otherwise} \end{cases} \tag{6.117}$$

- $P_n^g(k_i)$ の計算
  次の通りに計算する．

$$P_n^g(k_i) = \begin{cases} 0, & \text{if } \zeta_n^g(k_i) < Z_{min}^g \\ 1, & \text{if } \zeta_n^g(k_i) > Z_{max}^g \\ \frac{\log\left(\zeta_n^g(k_i)/Z_{min}^g\right)}{\log\left(Z_{max}^g/Z_{min}^g\right)}, & \text{otherwise} \end{cases} \tag{6.118}$$

- $P_n^l(k_i)$ の計算
  次の通りに計算する．

$$P_n^l(k_i) = \begin{cases} 0, & \text{if } \zeta_n^l(k_i) < Z_{min}^l \\ 1, & \text{if } \zeta_n^l(k_i) > Z_{max}^l \\ \frac{\log\left(\zeta_n^l(k_i)/Z_{min}^l\right)}{\log\left(Z_{max}^l/Z_{min}^l\right)}, & \text{otherwise} \end{cases} \tag{6.119}$$

3-c) 音声休止確率推定　3-c) で用いる変数を表 6.70 に示す．

音声休止確率 $q_n(k_i)$ は，3 つの周波数帯域の平滑化結果を元にして計算した暫定の音声確率 $P_n^{f,g,l}(k_i)$ を次のように統合して得られる．

$$q_n(k_i) \quad = \quad 1 - \left(1 - a^l + a^l P_n^l(k_i)\right)\left(1 - a^g + a^g P_n^g(k_i)\right)\left(1 - a^f + a^f P_n^f(k_i)\right), \tag{6.120}$$

ただし，$q_n(k_i) < q_{min}$ のとき，$q_n(k_i) = q_{min}$ とし，$q_n(k_i) > q_{max}$ のとき，$q_n(k_i) = q_{max}$ とする．

3-d) 音声存在確率推定　音声存在確率 $p_n(k_i)$ は，音声休止確率 $q_n(k_i)$，事前 SNR $\zeta_n(k_i)$，式 (6.107) により導出された中間変数 $v_n(k_i)$ を用いて次のように導出する．

$$p_n(k_i) \quad = \quad \left\{1 + \frac{q_n(k_i)}{1 - q_n(k_i)}\,(1 + \zeta_n(k_i))\exp\left(-v_n(k_i)\right)\right\}^{-1} \tag{6.121}$$

4) ノイズ除去: 出力である音声強調された分離音 $\hat{S}_n(k_i)$ は，入力である分離音スペクトル $Y_n(k_i)$ に対して，最適ゲイン $G_n^{H1}(k_i)$，音声存在確率 $p_n(k_i)$ を次のように作用させることで導出する．

$$\hat{S}_n(k_i) \quad = \quad Y_n(k_i) G_n^{H1}(k_i) p_n(k_i) \tag{6.122}$$

表 6.60: PostFilter のパラメータ表 (後半)

パラメータ名	型	デフォルト値	単位	説明
EST_VOICEP_SETTING	bool	false		音声確率推定に関するパラメータを設定する時，true にする．
EST_VOICEP_SETTING				以下，EST_VOICEP_SETTING が true の時に有効．
PRIOR_SNR_SMOOTH_FACTOR	float	0.7		時間平滑化係数．
MIN_FRAME_SMOOTH_SNR	float	0.1		周波数平滑化 SNR の最小値 (frame)．
MAX_FRAME_SMOOTH_SNR	float	0.316		周波数平滑化 SNR の最大値 (frame)．
MIN_GLOBAL_SMOOTH_SNR	float	0.1		周波数平滑化 SNR の最小値 (global)．
MAX_GLOBAL_SMOOTH_SNR	float	0.316		周波数平滑化 SNR の最大値 (global)．
MIN_LOCAL_SMOOTH_SNR	float	0.1		周波数平滑化 SNR の最小値 (local)．
MAX_LOCAL_SMOOTH_SNR	float	0.316		周波数平滑化 SNR の最大値 (local)．
UPPER_SMOOTH_FREQ_INDEX	int	99		周波数平滑化上限ビンインデックス．
LOWER_SMOOTH_FREQ_INDEX	int	8		周波数平滑化下限ビンインデックス．
GLOBAL_SMOOTH_BANDWIDTH	int	29		周波数平滑化バンド幅 (global)．
LOCAL_SMOOTH_BANDWIDTH	int	5		周波数平滑化バンド幅 (local)．
FRAME_SMOOTH_SNR_THRESH	float	1.5		周波数平滑化 SNR の閾値．
MIN_SMOOTH_PEAK_SNR	float	1.0		周波数平滑化 SNR ピークの最小値．
MAX_SMOOTH_PEAK_SNR	float	10.0		周波数平滑化 SNR ピークの最大値．
FRAME_VOICEP_PROB_FACTOR	float	0.7		音声確率平滑化係数 (frame)．
GLOBAL_VOICEP_PROB_FACTOR	float	0.9		音声確率平滑化係数 (global)．
LOCAL_VOICEP_PROB_FACTOR	float	0.9		音声確率平滑化係数 (local)．
MIN_VOICE_PAUSE_PROB	float	0.02		音声休止確率の最小値．
MAX_VOICE_PAUSE_PROB	float	0.98		音声休止確率の最大値．

表 6.61: 変数の定義

変数	説明，対応するパラメータ
$Y(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	周波数ビン k_i に対応する分離音複素スペクトル
$\lambda^{init}(k_i) = \left[\lambda_1^{init}(k_i), \ldots, \lambda_N^{init}(k_i)\right]^T$	定常ノイズ推定に用いる初期値パワースペクトル
$\lambda^{sta}(k_i) = \left[\lambda_1^{sta}(k_i), \ldots, \lambda_N^{sta}(k_i)\right]^T$	推定された定常ノイズパワースペクトル．
α_s	入力パワースペクトルの平滑化係数．パラメータ SPEC_SMOOTH_FACTOR，デフォルト 0.5
$S^{tmp}(k_i) = \left[S_1^{tmp}(k_i), \ldots, S_N^{tmp}(k_i)\right]$	最小パワー計算用のテンポラリ変数．
$S^{min}(k_i) = \left[S_1^{min}(k_i), \ldots, S_N^{min}(k_i)\right]$	最小パワーを保持する変数．
L	S_{tmp} の保持フレーム数．パラメータ BLOCK_LENGTH，デフォルト 80
δ	音声存在判定の閾値．パラメータ VOICEP_THRESHOLD，デフォルト 3.0
α_d	推定定常ノイズの混合比．パラメータ STATIONARY_NOISE_MIXTURE_FACTOR，デフォルト 0.98
$Y^{leak}(k_i)$	分離音に含まれると推定される漏れノイズのパワースペクトル
q	入力分離音パワーから漏れノイズを除くときの係数．パラメータ AMP_LEAK_FACTOR, デフォルト 1.5
S_{floor}	漏れノイズ最小値．パラメータ LEAK_FLOOR, デフォルト 0.1
r	定常ノイズ推定時の係数．パラメータ STATIONARY_NOISE_FACTOR, デフォルト 1.2

表 6.62: 変数の定義

変数	説明，対応するパラメータ
$\lambda^{leak}(k_i)$	漏れノイズのパワースペクトル，各分離音の要素から成るベクトル．
α^{leak}	全分離音パワーの合計に対する漏れ率． LEAK_FACTOR × OVER_CANCEL_FACTOR
$S_n(f, k_i)$	式 (6.87) で求める平滑化パワースペクトル

表 6.63: 変数の定義

変数	説明，対応するパラメータ
$\lambda^{rev}(f, k_i)$	時間フレーム f での残響のパワースペクトル
$\hat{S}(f-1, k_i)$	前フレームの PostFilter の出力したノイズ除去後分離音スペクトル
γ	前フレーム残響パワーの減衰係数．パラメータ REVERB_DECAY_FACTOR，デフォルト 0.5
Δ	前フレーム分離音の減衰係数．パラメータ DIRECT_DECAY_FACTOR，デフォルト 0.2

表 6.64: 主な変数の定義

変数	説明，対応するパラメータ
$Y(k_i)$	PostFilter ノードの入力である分離音の複素スペクトル
$\hat{S}(k_i)$	PostFilter ノードの出力となる，整形された分離音複素スペクトル
$\lambda(k_i)$	前段で推定されたノイズのパワースペクトル
$\gamma_n(k_i)$	分離音 n の SNR
$\alpha_n^p(k_i)$	音声含有率
$\xi_n(k_i)$	事前 SNR
$G^{H1}(k_i)$	分離音の SNR を向上させるための最適ゲイン

表 6.65: 変数の定義

変数	説明，対応するパラメータ
α_{mag}^p	事前 SNR 係数．パラメータ VOICEP_PROB_FACTOR，デフォルト 0.9
α_{min}^p	最小音声含有率．パラメータ MIN_VOICEP_PROB，デフォルト 0.05

表 6.66: 変数の定義

変数	説明，対応するパラメータ
a	前フレーム SNR の内分比．パラメータ PRIOR_SNR_FACTOR，デフォルト 0.8
ξ^{max}	事前 SNR の上限．パラメータ MAX_PRIOR_SNR，デフォルト 100

表 6.67: 変数の定義

変数	説明，対応するパラメータ
θ^{max}	中間変数 $v_n(k_i)$ 最大値．パラメータ MAX_OPT_GAIN，デフォルト 20
θ^{min}	中間変数 $v_n(k_i)$ 最小値．パラメータ MIN_OPT_GAIN，デフォルト 6

表 6.68: 変数の定義

変数	説明，対応するパラメータ
$\zeta_n(k_i)$	時間平滑化した事前 SNR
$\xi_n(k_i)$	事前 SNR
$\zeta_n^f(k_i)$	周波数平滑化 SNR (frame)
$\zeta_n^g(k_i)$	周波数平滑化 SNR (global)
$\zeta_n^l(k_i)$	周波数平滑化 SNR (local)
b	パラメータ PRIOR_SNR_SMOOTH_FACTOR，デフォルト 0.7
F_{st}	パラメータ LOWER_SMOOTH_FREQ_INDEX，デフォルト 8
F_{en}	パラメータ UPPER_SMOOTH_FREQ_INDEX，デフォルト 99
G	パラメータ GLOBAL_SMOOTH_BANDWIDTH，デフォルト 29
L	パラメータ LOCAL_SMOOTH_BANDWIDTH，デフォルト 5

表 6.69: 変数の定義

変数	説明，対応するパラメータ
$\zeta_n^{f,g,l}(k_i)$	各帯域で平滑化された SNR
$P_n^{f,g,l}(k_i)$	各帯域での暫定音声確率
$\zeta_n^{peak}(k_i)$	平滑化 SNR のピーク
Z_{min}^{peak}	パラメータ MIN_SMOOTH_PEAK_SNR，デフォルト値 1
Z_{max}^{peak}	パラメータ MAX_SMOOTH_PEAK_SNR，デフォルト値 10
Z_{thres}	FRAME_SMOOTH_SNR_THRESH，デフォルト値 1.5
$Z_{min}^{f,g,l}$	パラメータ MIN_FRAME_SMOOTH_SNR, MIN_GLOBAL_SMOOTH_SNR, MIN_LOCAL_SMOOTH_SNR，デフォルト値 0.1
$Z_{max}^{f,g,l}$	パラメータ MAX_FRAME_SMOOTH_SNRF, MAX_GLOBAL_SMOOTH_SNR, MAX_LOCAL_SMOOTH_SNR，デフォルト値 0.316

表 6.70: 変数の定義

変数	説明，対応するパラメータ
$q_n(k_i)$	音声休止確率
a^f	FRAME_VOICEP_PROB_FACTOR，デフォルト，0.7
a^g	GLOBAL_VOICEP_PROB_FACTOR，デフォルト，0.9
a^l	LOCAL_VOICEP_PROB_FACTOR，デフォルト，0.9
q_{min}	MIN_VOICE_PAUSE_PROB，デフォルト，0.02
q_{max}	MAX_VOICE_PAUSE_PROB，デフォルト，0.98

6.3.12 SemiBlindICA

ノードの概要

多チャンネル観測信号に含まれる既知信号 (システム発話の音声信号など) を除去する．参考文献 [1] を実装したモジュールである．

必要なファイル

無し．

使用方法

どんなときに使うのか

音声対話システムでの使用例を示す．近接マイクを用いない音声対話システムは，ユーザの口元とマイクの間に距離があるため，システム発話もマイクに混入することがある．その場合，システムのマイクから入力される信号には，ユーザの発話とシステムの発話が混ざっているため，ユーザ発話の音声認識精度が劣化することがある．

より一般的には，マイクロフォンアレイで観測した多チャンネル信号に，波形既知の信号が含まれている場合，既知信号を除去することが出来る．上記例では，システム発話が既知信号である．ここで，既知とする信号については，再生時の波形がわかっていれば良い (例: スピーカーで再生する wav ファイルがある など)．一般に，スピーカーで再生の波形とマイクで観測した時の波形は，スピーカーからマイクへの伝達の過程で変化し，また，伝達時間に応じた多少の時間ずれが生じる．SemiBlindICA モジュールは，それらの伝達過程や時間ずれも考慮して観測信号から既知信号を除去するため，再生時の波形が与えられれば良い．

典型的な接続例

図 6.73 and 6.74 に SemiBlindICA の使用例を示す．図 6.73 では，未知信号と既知信号が混ざって観測された多チャンネル音響信号を INPUT に，既知信号を REFERENCE に，それぞれ MultiFFT モジュールで時間周波数領域に変換し，入力としている．OUTPUT は，INPUT から REFERENCE の成分を抑圧した未知信号が出力されており，LocalizeMUSIC モジュールを用いて定位をするなど，未知信号に対する処理が行われていく．

図 6.74 は，1 チャンネル目は既知信号，2 チャンネル目は既知信号と未知信号が混合された信号を含むステレオ wav ファイルに対する SemiBlindICA モジュールの使用例を示す．ChannelSelector モジュールを利用することで未知観測信号チャネルと既知信号チャネルを切り分け，それぞれ INPUT， REFERENCE に入力している．その出力は，図 6.74 のようにネットワークを構成することで，SaveWavePCM モジュールを用いて，分離抽出された未知信号成分を wav ファイルとして保存することが出来る．

ノードの入出力とプロパティ

入力

INPUT : `Matrix<complex<float> >` 型．マイクロホンアレイで観測したマルチチャネル複素スペクトル．MultiFFT モジュールで時間周波数領域に変換したあとの信号を入力とする．

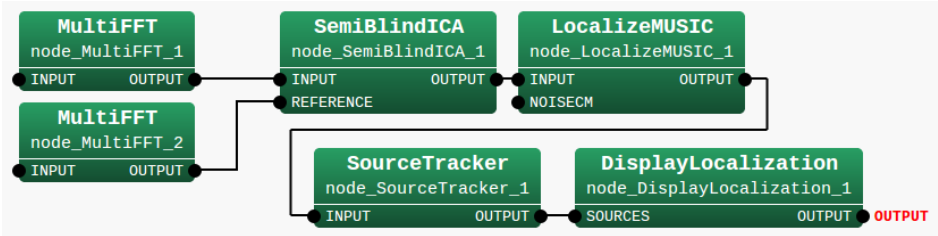


図 6.73: SemiBlindICA の基本的な利用例

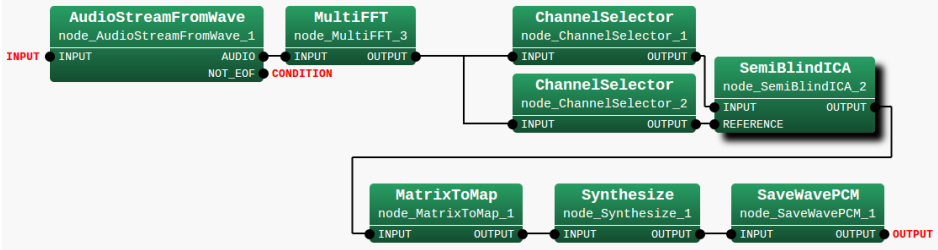


図 6.74: SemiBlindICA で左右チャンネルを使って未知信号を抽出する例

REFERENCE : `Matrix<complex<float> >` 型．既知信号の複素スペクトル．MultiFFT モジュールで時間周波数領域に変換したあとの信号を用いる．

出力

OUTPUT : `Matrix<complex<float> >` 型．入力の INPUT から既知信号 REFERENCE を除去した信号が INPUT と同様，マルチチャネル複素スペクトル型として出力される．

パラメータ

CHANNEL 入力多チャネル信号 INPUT のチャネル数．

LENGTH 短時間フーリエ変換のフレーム長．HARK のデフォルト設定では，512 [pt] である．

INTERVAL 既知信号を除去するフィルタ長を短時間フーリエ変換のシフト幅に応じた補正をするための係数．この補正を multirate repeating と呼び，フィルタ学習の収束性能向上が期待できる [(2)]．数式中は K で示す．

TAP_LOWERFREQ 周波数ビン 0 [Hz] におけるフィルタ長．既知信号と観測信号の時間ずれ，観測環境の残響時間を考慮する．残響時間が長い環境では大きめに，また，低周波領域は一般に大きめの値が必要．数式中は M_L で示す．

TAP_UPPERFREQ ナイキスト周波数ビンにおけるフィルタ長．各周波数ビンにおけるフィルタ長は，TAP_LOWERFREQ と線形補間によって決定する．数式中は M_U で示す．

DECAY 各時間フレームに対応する既知信号除去フィルタ係数更新に用いる学習係数について，過去のフレームに対応する既知信号除去フィルタ学習係数の減衰度合．屋内など，残響が存在する環境では，過去のフレームに対応するフィルタ係数は指数的に減衰する．フィルタ係数値のスケールが指数的な広がりを持っているため，学習係数も同様に指数的に減衰させることで，学習の効率化を図ることが出来る．1 のとき，全フィルタ係数が同様の学習係数で更新される． 0.6–0.8 程度が経験的によく用いられる．数式中は λ で示す．

表 6.71: SemiBlindICA のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
CHANNEL	int	1		入力 INPUT のチャネル数.
LENGTH	int	512	[pt]	短時間フーリエ変換のフレーム長.
INTERVAL	int	1		短時間フーリエ変換のシフト幅によるフィルタ長補正パラメータ．シフトのオーバラップが大きくなる場合 (シフト幅が小さい場合) に大きな値にする.
TAP_LOWERFREQ	int	8	[frame]	周波数ビン 0 [Hz] におけるフィルタ長.
TAP_UPPERFREQ	int	4	[frame]	ナイキスト周波数ビンにおけるフィルタ長.
DECAY	float	0.8		既知信号除去フィルタ係数更新に用いる学習係数の時間フレームごとの減衰度合い.
MU_FILTER	float	0.01		既知信号除去フィルタの学習係数．非負値を用いる.
MU_REFERENCE	float	0.01		既知信号の正規化パラメータの学習係数. 非負値を用いる.
MU_UNKNOWNSIGNAL	float	0.01		観測信号中の既知でない，未知信号の正規化パラメータの学習係数．非負値を用いる.
IS_ZERO	float	0.0001		入力信号が 0 とみなす閾値．ただし，時間周波数領域でのパワーであることに注意.
FILE_FILTER_IN	string	-null		既知信号除去フィルタの初期値を格納したファイル名．デフォルト値の “-null” のときは，ファイル入力を用いない.
FILE_FILTER_OUT	string	-null		既知信号除去フィルタを保存するときのファイル名．デフォルト値の “-null” のときは，ファイル出力しない.
OUTPUT_FREQ	int	150	[frame]	上記フィルタを保存する時間フレームの間隔.

MU_FILTER 既知信号除去フィルタの学習は勾配法によって行うが，フィルタ係数更新時に評価関数の勾配にかけられる学習係数である．非負値を用いる．大きい値に設定すると，フィルタ係数の 1 度ずつの更新も大きく変化させることが出来るが，フィルタ係数が (局所) 最適解の前後を揺れ動き，収束しないというリスクがある．一方，小さい値に設定すると，いつかは収束することが期待できるが，(局所) 最適解に到達する更新回数が増えるというリスクが生じる．数式中は μ_w で示す.

MU_REFERENCE 既知信号の正規化パラメータの学習係数．既知信号の正規化処理は，既知信号除去フィルタの収束を加速させるために行う．数式中は μ_α で示す.

MU_UNKNOWNSIGNAL 観測信号中の未知信号の正規化パラメータの学習係数．この正規化処理も，MU_REFERENCE のときと同様に，既知信号除去フィルタの収束性向上のために行う．数式中は μ_β で示す.

IS_ZERO 計算資源節約のため，入力信号が 0 に近いときは処理を省略するが，入力信号が 0 とみなす閾値．ただし，時間周波数領域でのパワーであることに注意．

FILE_FILTER_IN 既知信号除去フィルタの初期値を格納したファイル名．“-null” のとき，ファイル入力を用いない．

FILE_FILTER_OUT 既知信号除去フィルタを保存する場合のファイル名．“-null” のときは，ファイル出力なし．

OUTPUT_FREQ 既知信号除去フィルタを保存する時間フレームの間隔．

ノードの詳細

SemiBlindICA では，短時間フーリエ変換 (STFT) 領域における音の混合モデルに基づいて，未知信号と既知音との独立性条件を用いた独立成分分析 (ICA) を適応し，観測信号から既知音を分離する．本モジュールでは入力の多チャンネル音響信号に対し，各チャンネル・周波数ビン個別にこの処理を適応し，入力に含まれる既知信号を分離した多チャンネル音響信号を出力する．

混合モデルと分離過程: SemiBlindICA では既知音の再生空間における残響を考慮した混合モデルを使用する．このモデルは，STFT 領域での線形混合モデルで表現され，ω を周波数インデックス, f をフレームインデックスとして観測信号 $X(\omega, f)$ は以下のように定式化される．

$$X(\omega, f) = N(\omega, f) + \sum_{m=0}^{M} H(\omega, m) S(\omega, f - m)$$

ここで，$N(\omega, f)$ は未知信号，$S(\omega, f)$ は既知信号を表し，$H(\omega, m)$ は m 番目の遅延フレームの伝達係数を表す．

混合過程が瞬時混合として扱えるため，ICA を適用することで既知信号を分離する．分離過程を以下に示す．

$$\begin{aligned}
\begin{pmatrix} \hat{N}(\omega, f) \\ \boldsymbol{S}(\omega, f) \end{pmatrix} &= \begin{pmatrix} a(\omega) & -\boldsymbol{w}^T(\omega) \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} X(\omega, f) \\ \boldsymbol{S}(\omega, f) \end{pmatrix} \qquad (6.123) \\
\boldsymbol{S}(\omega, f) &= [S(\omega, f), S(\omega, f - K), \cdots, S(\omega, f - M(\omega)K)]^T \\
\boldsymbol{w}(\omega) &= [w_0(\omega), w_1(\omega), \cdots, w_{M(\omega)}(\omega)]^T \\
M(\omega) &= \mathrm{floor}\left(\omega/\omega_{nyq}(M_U - M_L)\right) + M_L
\end{aligned}$$

ここで，ω_{nyq} はナイキスト周波数に相当する周波数ビン番号を，M_L，M_U は 0Hz に対応する周波数ビンと ω_{nyq} におけるフィルタ長を表し，$\boldsymbol{w}(\omega)^T$ は $M + 1$ 次の分離フィルタ $\boldsymbol{w}(\omega)$ の転置である．また，K は既知信号を除去するフィルタ長を短時間フーリエ変換のシフト幅に応じて補正するための係数である．この補正を multirate repeating[(2)] と呼び, フィルタ学習の収束性能向上が期待できる．

分離フィルタの推定: 分離フィルタは，$\hat{N}$，$\boldsymbol{S}$ の結合確率密度と周辺確率密度の積との距離である Kullback-Leibler Divergence (KLD) を最小化することで推定する．非ホロノミック拘束 [(3)] と自然勾配法により，後述する正規化された未知信号 $\hat{N}_n$ および既知信号 $\boldsymbol{S}_n$ から，以下の学習則が得られる．

$$\begin{aligned}
\boldsymbol{w}(\omega, f + 1) &= \boldsymbol{w}(\omega, f) + \boldsymbol{\mu}_w \Phi_{\hat{N}_n(\omega)}\left(\hat{N}_n(\omega, f)\right) \bar{\boldsymbol{S}}_n(\omega, f) \\
a(\omega) &= 1
\end{aligned}$$

ここで，$\bar{x}$ は x の複素共役を表す．また，$\Phi_x(x)$ には $\tanh(|x|)e^{j\theta(x)}$ を，$\boldsymbol{\mu}_w$ には次式を用いる．

$$\boldsymbol{\mu}_w = \mathrm{diag}\left(\mu_w, \mu_w \lambda^{-1}, \cdots, \mu_w \lambda^{-M(\omega)}\right) \qquad (6.124)$$

これは室内でのインパルス応答が時間方向に指数的に減衰する知見からと，収束の高速化のために用いられる．得られた分離フィルタから出力である未知信号の推定値 $\hat{N}$ を計算するには式 (6.123) より，

$$\hat{N}(\omega, f) = X(\omega, f) - \boldsymbol{w}(\omega, f)^T \boldsymbol{S}_n(\omega, f) \tag{6.125}$$

を計算すればよい．

非ホロノミック拘束により，$\hat{N}$ を正規化する必要がある．なぜなら，同拘束により $E[1 - \Phi_x(x\alpha_x)\bar{x}\bar{\alpha}_x] = 1$ が満たされなければならないからである．一般に，自然勾配法による KLD 最小化における，x の正規化係数 ν_x は以下の式で更新される．

$$\nu_x(f+1) = \nu_x(f) + \mu_x[1 - \Phi_x(x(f)\nu_x(f))\bar{x}(f)\bar{\nu}_x(f)]\nu_x(f)$$

同様に，$\hat{N}$ の正規化は正規化係数 α を用いて次式で導出される．

$$\hat{N}_n(f) = \alpha(f)\hat{N}(f) \tag{6.126}$$

$$\alpha(f+1) = \alpha(f) + \mu_\alpha[1 - \Phi_{\hat{N}_n}(\hat{N}_n(f))\bar{\hat{N}}_n(f)]\alpha(f) \tag{6.127}$$

SemiBlindICA では，収束の高速化のために観測信号の正規化も行う．これは，$\hat{N}$ の時と同様に正規化係数 β を用いて次式で導出される．

$$S_n(f) = \beta(f)S(f) \tag{6.128}$$

$$\beta(f+1) = \beta(f) + \mu_\beta[1 - \Phi_{S_n}(S_n(f))\bar{S}_n(f)]\beta(f) \tag{6.129}$$

$$\boldsymbol{S}_n(f) = [S_n(f), S_n(f-K), \cdots, S_n(f-MK)]$$

処理の流れ: SemiBlindICA のメインアルゴリズムは式 (6.124) ~ (6.129) についてある周波数ビン，フレームに対して処理を行い，対応する出力を得る．Algorithm 1 にメインアルゴリズムの概略を示す．

Algorithm 1 SemiBlindICA のメインアルゴリズム

*** ある周波数ビン ω，フレーム f に対し以下を実行 ***
$\hat{N}(\omega, f)$ を計算 (式 (6.125))
$\hat{N}(\omega, f)$ と $\boldsymbol{S}(\omega, f)$ を正規化 (式 (6.126，6.128))
フィルタ係数 $\boldsymbol{w}(\omega, f)$ を更新 (式 (6.124))
正規化係数 $\alpha(\omega, f)$，$\beta(\omega, f)$ を更新 (式 6.127，6.129)
計算した $\hat{N}(\omega, f)$ を出力

全体の処理を Algorithm 2 に示す．新しい入力フレームを得るたびに，チャンネル ch，周波数ビン ω ごとに出力を計算するオンライン処理となっている．

Algorithm 2 全体の処理

```
*** 新しいフレームが入力される度に以下を実行 ***
f ← f + 1
for ch in 0, ···, C do
  for ω in 0, ···, ω_nqt do
    メインアルゴリズムを現在の f，ω について実行
  end for
end for
```

参考文献

(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

(4) 武田 龍 et al., “独立成分分析を応用したロボット聴覚による残響下におけるバージイン発話認識,” 日本ロボット学会第 26 回学術講演会, 1A2-02, Sep. 2008.

6.3.13 SpectralGainFilter

ノードの概要

本ノードは，入力された分離音スペクトルに最適ゲイン，音声存在確率（PostFilter を参照）を乗じ，出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

HRLE，CalcSpecSubGain を用いて分離音スペクトルからノイズを抑制した音声スペクトを得る際に用いる．

典型的な接続例

SpectralGainFilter の接続例は図 6.75 の通り．入力は，GHDSS から出力された分離スペクトルおよび CalcSpecSubGain などから出力される，最適ゲイン，音声存在確率である．図では出力の例として，Synthesize ,SaveRawPCM に接続し，音声ファイルを作成している．

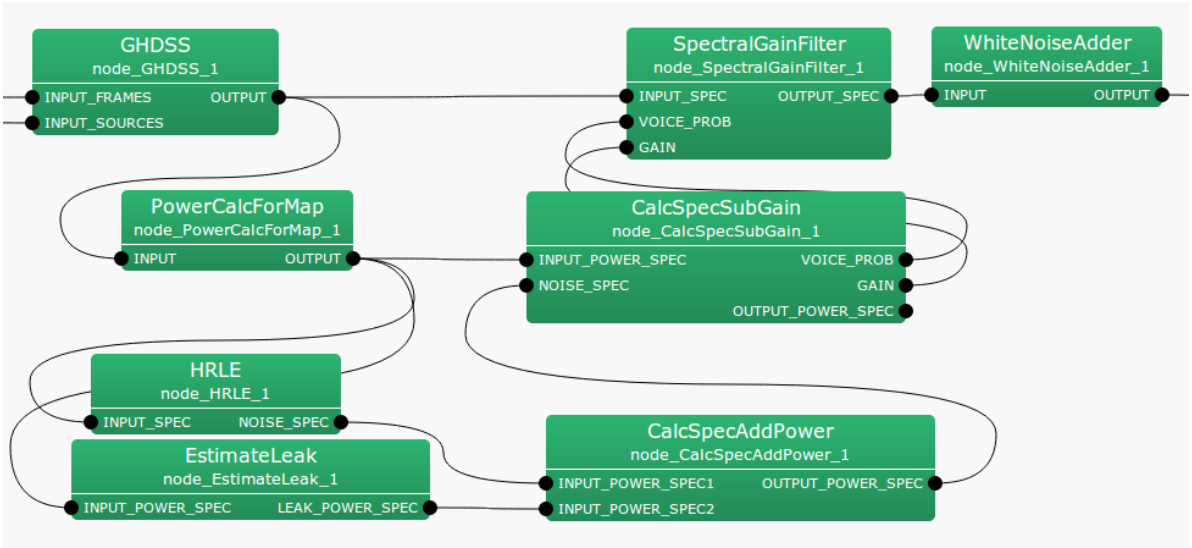


図 6.75: SpectralGainFilter の接続例

ノードの入出力とプロパティ

入力

INPUT_SPEC ：Map<int, ObjectRef> 型．GHDSS の出力と同じ型．音源 ID と分離音の複素スペクトルである，Vector<complex<float> > 型データのペア．

VOICE_PROB ：Map<int, ObjectRef> 型．音源 ID と音声存在確率の Vector<float> 型データのペア．

GAIN ：Map<int, ObjectRef> 型．音源 ID と最適ゲインの Vector<float> 型データのペア．

出力

OUTPUT_SPEC ：Map<int, ObjectRef> 型．GHDSS の出力と同じ型．音源 ID と分離音の複素スペクトルである，Vector<complex<float> > 型データのペア．

パラメータ

なし

ノードの詳細

本ノードは，入力された音声スペクトルに最適ゲイン，音声存在確率を乗じ，音声を強調した分離音スペクトルを出力する．

出力である音声が強調された分離音 $Y_n(k_i)$ は，入力である分離音スペクトルを $X_n(k_i)$ ，最適ゲインを $G_n(k_i)$, 音声存在確率を $p_n(k_i)$ とすると次のように表される．

$$Y_n(k_i) = X_n(k_i)G_n(k_i)p_n(k_i) \tag{6.130}$$

6.4 FeatureExtraction カテゴリ

6.4.1 Delta

ノードの概要

本ノードは，静的特徴ベクトルから動的特徴量ベクトルを求める．特徴抽出ノードである MSLSExtraction や MFCCExtraction の後段に接続するのが一般的な使い方である．これらの特徴量抽出ノードは，静的特徴ベクトルを求めると共に，動的特徴量を保存する領域を確保している．この時の動的特徴量は，0 に設定されている．Delta ノードでは，静的特徴ベクトル値を使って動的特徴量ベクトル値を計算し，値を設定する．従って，入出力でベクトルの次元数は変らない．

必要なファイル

無し．

使用方法

どんなときに使うのか

静的特徴から動的特徴量を求める場合に本ノードを使う．通常，MFCCExtraction や MSLSExtraction の後に用いる．

典型的な接続例

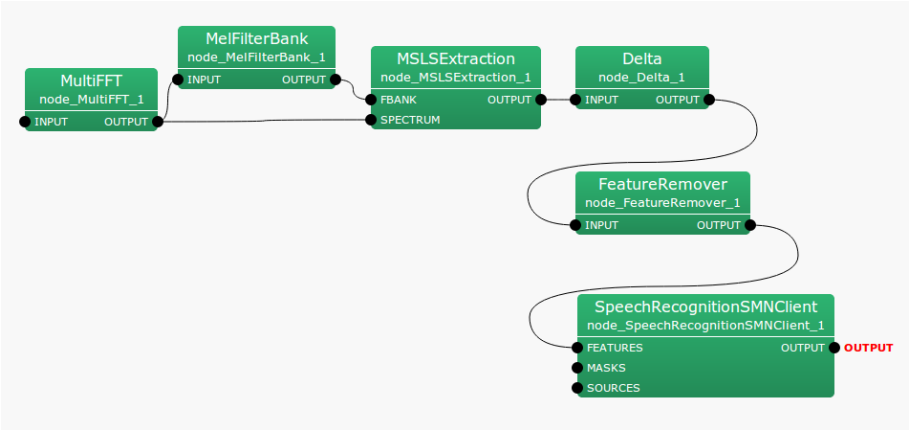


図 6.76: Delta の典型的な接続例

ノードの入出力とプロパティ

入力

INPUT : `Map<int, ObjectRef>` 型．音源 ID と特徴量ベクトルの `Vector<float>` 型のデータのペア．

表 6.72: Delta パラメータ表

パラメータ名	型	デフォルト値	単位	説明
FBANK_COUNT	int	13		静的特徴の次元数

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID と特徴量ベクトルの Vector<float> 型のデータのペア．

パラメータ

FBANK_COUNT : int 型である．処理する特徴量の次元数．値域は，正の整数．特徴量抽出ノードの直後に接続する場合は，特徴量抽出で指定した FBANK_COUNT を指定．ただし，特徴量抽出でパワー項を使用するオプションを true にしている場合は， FBANK_COUNT + 1 を指定する．

ノードの詳細

本ノードは，静的特徴ベクトルから動的特徴ベクトルを求める．入力の次元数は，静的特徴と動的特徴の次元数を合せた次元数である．FBANK_COUNT 以下の次元要素を静的特徴とみなし，動的特徴量を計算する．FBANK_COUNT より高次の次元要素に動的特徴量を入れる．

フレーム時刻 f における，入力特徴ベクトルを，

$$x(f) = [x(f,0), x(f,1), \ldots, x(f,P-1)]^T \tag{6.131}$$

と表す．ただし，P は，FBANK_COUNT である．

$$y(f) = [x(f,0), x(f,1), \ldots, x(f,2P-1)]^T \tag{6.132}$$

出力ベクトルの各要素は，

$$y(f,p) = \begin{cases} x(f,p), & \text{if } p = 0, \ldots, P-1, \\ w \sum_{\tau=-2}^{2} \tau \cdot x(f+\tau, p), & \text{if } p = P, \ldots, 2P-1, \end{cases} \tag{6.133}$$

である．ただし，$w = \frac{1}{\sum_{\tau=-2}^{\tau=2} \tau^2}$ である．図 6.77 に Delta の入出力フローを示す．

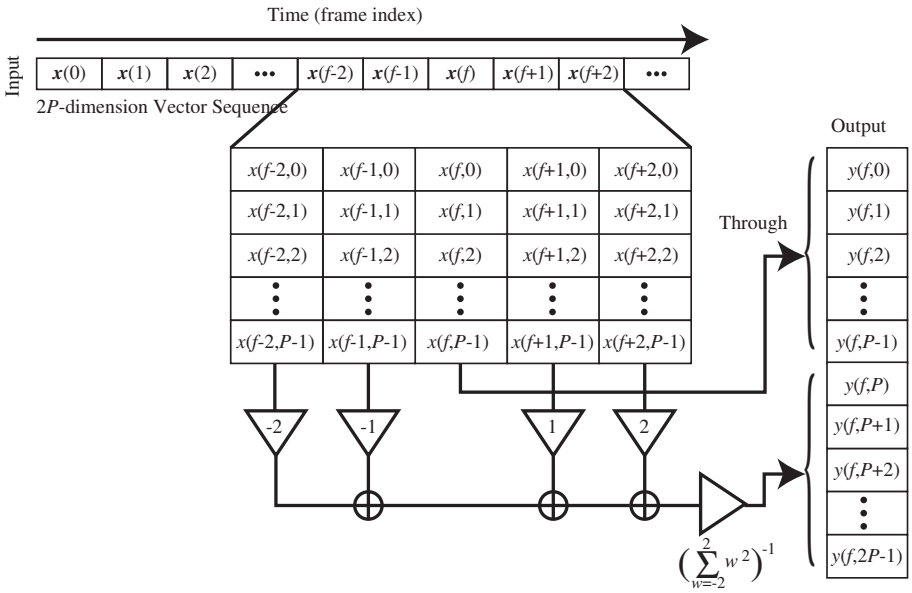


図 6.77: Delta の入出力フロー.

6.4.2 FeatureRemover

ノードの概要

本ノードは，入力ベクトル中から指定した次元要素を削除し，ベクトルの次元を減らしたベクトルを出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

音響特徴量やミッシングフィーチャーマスクベクトルなどのベクトル型の要素の中から不要な要素を削除し，次元数を減らすときに使う．

通常，特徴量の抽出処理は，静的特徴に続いて，動的特徴量を抽出する．その際に，静的特徴が不要となる場合がある．不要な特徴量を削除するには，本ノードを使用する．特に対数パワー項を削除することが多い．

典型的な接続例

MSLSExtraction や MFCCExtraction で対数パワー項を計算し，その後，Delta を用いると，デルタ対数パワー項を計算できる．対数パワー項を計算しなければデルタ対数パワーを計算できないため，一度対数パワー項を含めて音響特徴量を計算してから，対数パワー項を除去する．本ノードは，通常 Delta の後段に接続し，対数パワー項を削除するために用いる．

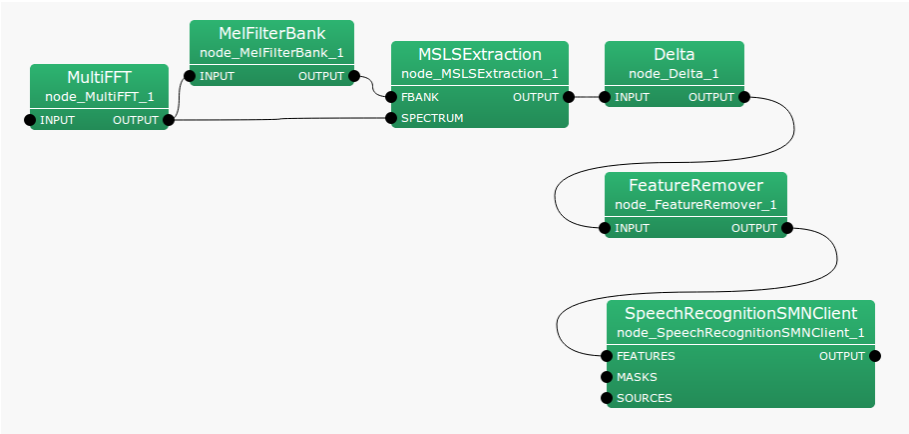


図 6.78: FeatureRemover の典型的な接続例

ノードの入出力とプロパティ

入力

表 6.73: FeatureRemover パラメータ表

パラメータ名	型	デフォルト値	単位	説明
SELECTOR	Object	<Vector<int> >		次元インデックスからなるベクトル（複数指定可）

INPUT : Map<int, ObjectRef> 型．音源 ID と特徴量ベクトルの Vector<float> 型のデータのペア．

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID と特徴量ベクトルの Vector<float> 型のデータのペア．

パラメータ

SELECTOR : Vector<int> 型．値域は，0 以上入力特徴量の次数未満．いくつ指定してもよい．1 次元目と 3 次元目の要素を削除し，入力ベクトルを 2 次元減らす場合は， <Vector<int> 0 2> とする．次元指定のインデクスが 0 から始まっていることに注意．

ノードの詳細

本ノードは，入力ベクトル中から不要な次元要素を削除し，ベクトルの次元を減らす．

音声信号を分析すると，分析フレームの対数パワーは，発話区間で大きい傾向がある．特に有声部分で大きい．従って，音声認識において対数パワー項を音響特徴量に取り入れることで認識精度の向上が見込める．しかしながら，対数パワー項を直接特徴量として用いると，収音レベルの違いが音響特徴に直接反映される．音響モデルの作成に用いた対数パワーレベルと収音レベルに差が生じると音声認識精度が低下する．機器の設定を固定しても，発話者が常に同一レベルで発話するとは限らない．そこで，対数パワー項の動的特徴量であるデルタ対数パワーを用いる．これにより，収音レベルの違いに頑健で，かつ発話区間や有声部分を表す特徴を捉えることが可能となる．

6.4.3 MelFilterBank

ノードの概要

入力スペクトルにメルフィルタバンク処理を行ない，各フィルタチャネルのエネルギーを出力する．入力スペクトルは，2 種類あり，入力によって出力結果が異なる点に留意．

必要なファイル

無し．

使用方法

どんなときに使うのか

音響特徴量を求める前処理として使用する．MultiFFT，PowerCalcForMap，PreEmphasis の直後に使用する．MFCCExtraction，MSLSExtraction の前段で使用する．

典型的な接続例

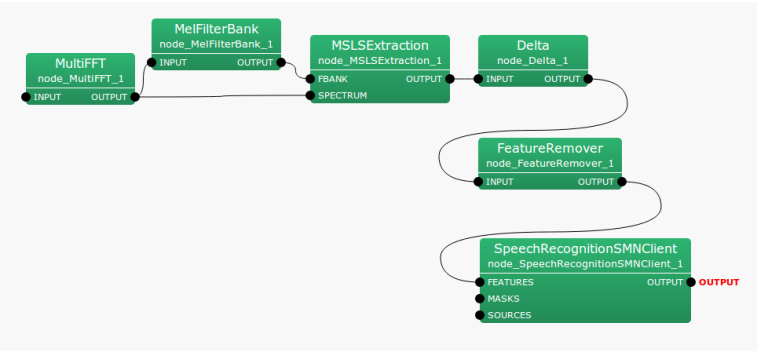


図 6.79: MelFilterBank の接続例

ノードの入出力とプロパティ

表 6.74: MelFilterBank のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	分析フレーム長
SAMPLING_RATE	int	16000	[Hz]	サンプリング周波数
CUTOFF	int	8000	[Hz]	ローパスフィルタのカットオフ周波数
MIN_FREQUENCY	int	63	[Hz]	フィルタバンクの下限周波数
MAX_FREQUENCY	int	8000	[Hz]	フィルタバンクの上限周波数
FBANK_COUNT	int	13		フィルタバンク数

入力

INPUT : Map<int, ObjectRef> 型．音源 ID とパワースペクトル Vector<float> 型または，複素スペクトル Vector<complex<float> > 型のデータのペア．ただし，パワースペクトルを選択した場合，複素スペクトルを選択した場合と比較して出力エネルギーが 2 倍になる．

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID とフィルタバンクの出力エネルギーから構成されるベクトルの Vector<float> 型のデータのペア．出力ベクトルの次元数は，FBANK_COUNT の 2 倍である．0 から FBANK_COUNT-1 までに，フィルタバンクの出力エネルギーが入り，FBANK_COUNT から 2 *FBANK_COUNT-1 までには，0 が入る．0 が入れられる部分は，動的特徴量用のプレースホルダーである．動的特徴量が不要な場合は FeatureRemover を用いて削除する必要がある．

パラメータ

LENGTH : int 型．分析フレーム長である．入力スペクトルの周波数ビン数に等しい．値域は正の整数である．

SAMPLING_RATE : int 型．サンプリング周波数である．値域は正の整数である．

CUTOFF : int 型．離散フーリエ変換時のアンチエーリアシングフィルタのカットオフ周波数．SAMPLING_RATE の 1/2 以下である．

MIN_FREQUENCY : int 型．フィルタバンクの下限周波数．値域は正の整数でかつ CUTOFF 以下．

MAX_FREQUENCY : int 型．フィルタバンクの上限周波数．値域は正の整数でかつ CUTOFF 以下．

FBANK_COUNT : int 型．フィルタバンク数である．値域は正の整数である．

ノードの詳細

メルフィルタバンク処理を行ない，各チャネルのエネルギーを出力する．各バンクの中心周波数は，メルスケール [1] 上で等間隔に配置する．チャネル毎の中心周波数は，最小周波数ビン SAMPLING_RATE/LENGTH から SAMPLING_RATECUTOFF/LENGTH までを FBANK_COUNT 分割し決定する．

リニアスケールとメルスケールの変換式は，

$$m = 1127.01048 \log(1.0 + \frac{\lambda}{700.0}) \tag{6.134}$$

である．ただし，リニアスケール上での表現を λ (Hz)，メルスケール上での表現を m とする．図 6.80 に 8000 Hz までの変換例を示す．赤点は，SAMPLING_RATE が 16000 Hz，CUTOFF が 8000 Hz，かつ FBANK_COUNT が 13 の場合の，各バンクの中心周波数を表す．各バンクの中心周波数が，メルスケール上で等間隔なことを確認できる．

図 6.81 にメルスケール上の各フィルタバンクの窓関数を示す．中心周波数部分で 1.0 となり，隣接チャネルの中心周波数部分で 0.0 となる三角窓である．中心周波数がチャネル毎にメルスケール上で等間隔で，対象な形状である．これらの窓関数は，リニアスケール上では図 6.82 のように表現される．高域のチャネルでは，広い帯域をカバーしている．

入力するリニアスケール上で表現されたパワースペクトルに図 6.82 に示す窓関数で重み付けし，各チャネル毎にエネルギーを求め，出力する．

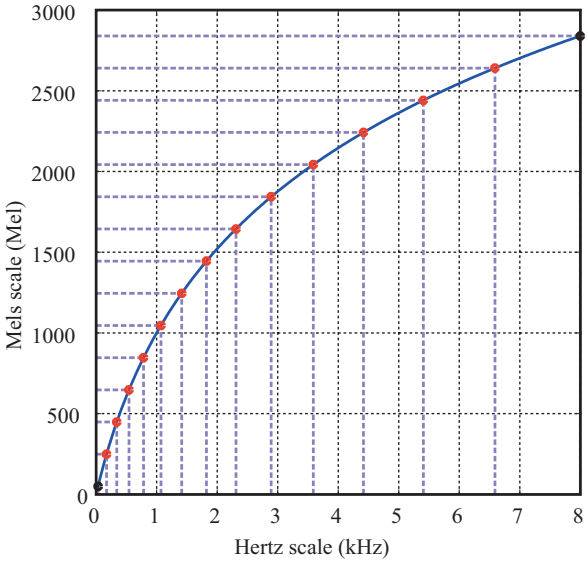


図 6.80: リニアスケールとメルスケールの対応

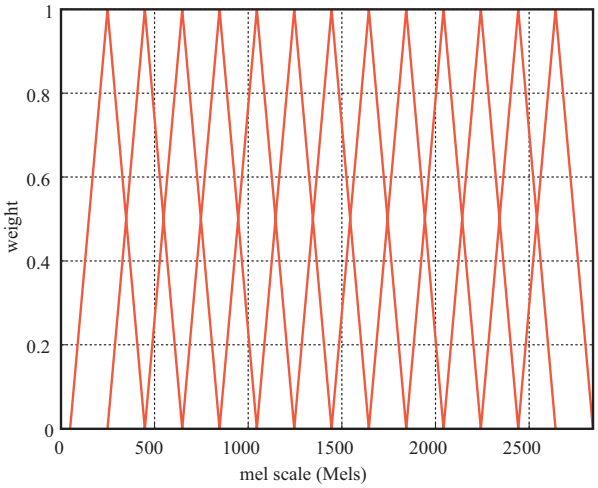


図 6.81: メルスケール上での窓関数

参考文献:

(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.

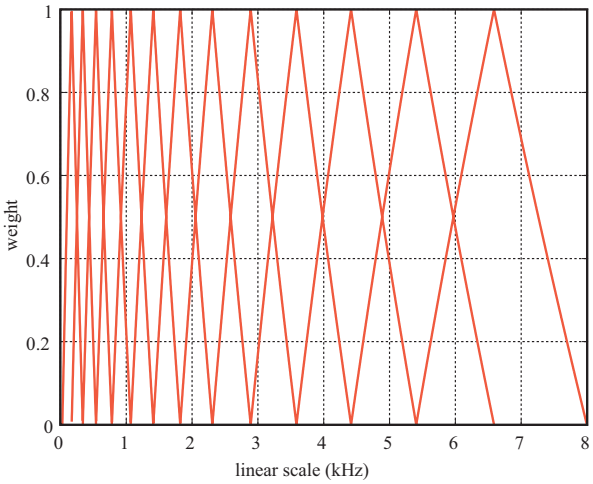


図 6.82: リニアスケール上での窓関数

6.4.4 MFCCExtraction

ノードの概要

本ノードは，音響特徴量の１つであるメルケプストラム係数 (MFCC : Mel-Frequency Cepstrum Coefficients) を求める．メルケプストラム係数と対数スペクトルパワーを要素とする音響特徴量ベクトルを生成する．

必要なファイル

無し．

使用方法

どんなときに使うのか

メルケプストラム係数を要素とする音響特徴量を生成するために用いる．音響特徴量ベクトルを生成するために用いる．例えば，音響特徴量ベクトルを音声認識ノードに入力し，音韻や話者を識別する．

典型的な接続例

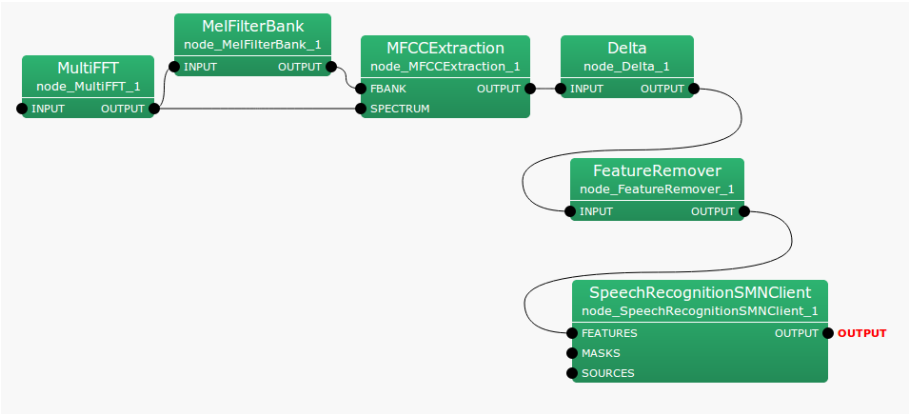


図 6.83: MFCCExtraction の典型的な接続例

ノードの入出力とプロパティ

表 6.75: MFCCExtraction のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FBANK_COUNT	int	24		入力スペクトルにかけるフィルタバンク数
NUM_CEPS	int	12		リフタリングで残すケプストラム係数の数
USE_POWER	bool	false		対数パワーを特徴量に含めるか含めないかの選択

入力

FBANK : Map<int, ObjectRef> 型．音源 ID とフィルタバンクの出力エネルギーから構成されるベクトルの Vector<float> 型のデータのペア．

SPECTRUM : Map<int, ObjectRef> 型．音源 ID と複素スペクトルから構成されるベクトルの Vector<complex<float> > 型のデータのペア．

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID と MFCC と対数パワー項から構成されるベクトルの Vector<float> 型のデータのペア．

パラメータ

FBANK_COUNT : int 型．入力スペクトルにかけるフィルタバンク数．デフォルト値は 24 である．値域は，正の整数．値を大きくすると 1 バンク当りの担当周波数帯域が狭くなり，周波数分解能の高い音響特徴量が求まる．より大きな FBANK_COUNT を設定すると，音響特徴をより精細に表現する．音声認識には，必ずしも精細な表現が最適ではなく，発声する音響環境に依存する．

NUM_CEP : int 型．リフタリングで残すケプストラム係数の数．デフォルト値は 12．値域は，正の整数．値を大きくすると音響特徴量の次元数が増える．より細かなスペクトル変化を表現する音響特徴量が求まる．

USE_POWER : bool 型．対数パワーを特徴量に含めて出力する場合は true 指定．

ノードの詳細

音響特徴量の 1 つであるメルケプストラム係数 (MFCC : Mel-Frequency Cepstrum Coefficients) と対数パワーを求める．MFCC と対数スペクトルパワーを次元要素とする音響特徴量を生成する．

対数スペクトルに，三角窓のフィルタバンクをかける．三角窓の中心周波数は，メルスケール上で等間隔になるように配置する．各フィルタバンクの出力対数エネルギーをとり，離散コサイン変換 (Discrate Cosine Transform) する．得られた係数をリフタリングした係数が MFCC である．

本ノードの入力部の FBANK には，各フィルタバンクの出力対数エネルギーが入力されることが前提である．

フレーム時刻 f における FBANK への入力ベクトルを，

$$x(f) = [x(f,0), x(f,1), \cdots, x(f,P-1)]^T \tag{6.135}$$

と表す．ただし，P は，入力特徴ベクトルの次元数で，FBANK_COUNT である．出力されるベクトルは，$P+1$ 次元ベクトルで，メルケプストラム係数とパワー項から構成される．1 次元目から P 次元目までは，メルケプストラム係数で，$P+1$ 次元目は，パワー項である．本ノードの出力ベクトルは，

$$y(f) = [y(f,0), y(f,1), \ldots, y(f,P-1), E]^T \tag{6.136}$$

$$y(f,p) = L(p) \cdot \sqrt{\frac{2}{P}} \cdot \sum_{q=0}^{P-1} \left\{ \log(x(q)) \cos\left(\frac{\pi(p+1)(q+0.5)}{P} \right) \right\} \tag{6.137}$$

ただし，E は，パワー項（後述）で，リフタリング係数は，

$$L(p) = 1.0 + \frac{Q}{2} \sin\left(\frac{\pi(p+1)}{Q} \right), \tag{6.138}$$

である．ただし，$Q = 22$ である．

パワー項は，SPECTRUM 部の入力ベクトルから求める．入力ベクトルを

$$s = [s(0), \ldots, s(K-1)]^T, \tag{6.139}$$

と表す．ただし， K は，FFT 長である．K は，SPECTRUM に接続された `Map` の次元数によって決る．対数パワー項は，

$$E = \log\left(\frac{1}{K}\sum_{k=0}^{K-1} s(k)\right) \tag{6.140}$$

6.4.5 MSLSExtraction

ノードの概要

本ノードは，音響特徴量の１つであるメルスケール対数スペクトル (MSLS : Mel-Scale Log-Spectrum) と対数パワーを求める．メルスケール対数スペクトル係数と対数スペクトルパワーを要素とする音響特徴量ベクトルを生成する．

必要なファイル

無し．

使用方法

どんなときに使うのか

メルスケール対数スペクトル係数と対数パワーを次元要素とする．音響特徴量ベクトルを生成するために用いる．例えば，音響特徴量ベクトルを音声認識ノードに入力し，音韻や話者を識別する．

典型的な接続例

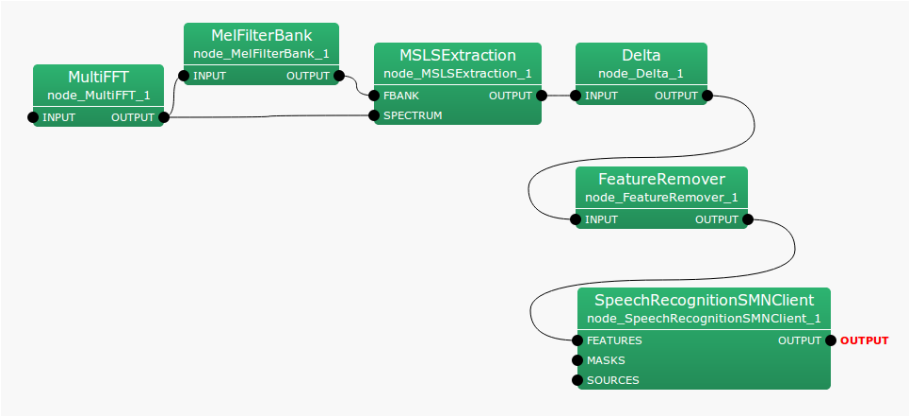


図 6.84: MSLSExtraction の典型的な接続例

ノードの入出力とプロパティ

入力

FBANK : `Map<int, ObjectRef>` 型．音源 ID とフィルタバンクの出力エネルギーから構成されるベクトルの `Vector<float>` 型のデータのペア．

SPECTRUM : `Map<int, ObjectRef>` 型．音源 ID と複素スペクトルから構成されるベクトルの `Vector<complex<float> >` 型のデータのペア．

表 6.76: MSLSExtraction のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FBANK_COUNT	int	13		入力スペクトルにかけるフィルタバンク数．実装は 13 に最適化されている．
NORMALIZATION_MODE	string	CEPSTRAL		特徴量の正規化手法
USE_POWER	bool	false		対数パワーを特徴量に含めるか含めないかの選択

出力

OUTPUT ：Map<int, ObjectRef> 型．である．音源 ID と MSLS と対数パワー項から構成されるベクトルの Vector<float> 型のデータのペア．本ノードは， MSLS の静的特徴を求めるノードであるが，出力には，動的特徴量部を含んだベクトルを出力する．動的特徴量部分には， 0 が設定される．その様子を図 6.85 に示す．

パラメータ

FBANK_COUNT ：int 型．入力スペクトルにかけるフィルタバンク数．値域は，正の整数．値を大きくすると 1 バンク当りの担当周波数帯域が狭くなり，周波数分解能の高い音響特徴量が求まる．典型的な設定値は，13 から 24 である．ただし，現在の実装では，この値が 13 に固定されるよう最適化されているので，デフォルト値の利用を強く推奨する．より大きな FBANK_COUNT を設定すると，音響特徴をより精細に表現する．音声認識には，必ずしも精細な表現が最適ではなく，発声する音響環境に依存する．

NORMALIZATION_MODE ：string 型．CEPSTRAL または SPECTRAL を指定可能．正規化をケプストラムドメイン／スペクトラムドメインで行うかを選択．

USE_POWER ：true にすると音響特徴量に対数パワー項を追加．false にすると省略．音響特徴量にパワー項を利用することは稀であるが，音声認識には，デルタ対数パワーが有効であるとされる．true にし，後段でデルタ対数パワーを計算し，それを音響特徴量として用いる．

ノードの詳細

本ノードは，音響特徴量の１つであるメルスケール対数スペクトル (MSLS : Mel-Scale Log-Spectrum) と対数パワーを求める．メルスケール対数スペクトル係数と対数スペクトルパワーを次元要素とする音響特徴量を生成する．

本ノードの FBANK 入力端子には， 各フィルタバンクの出力対数エネルギーを入力する．指定する正規化手法によって，出力する MSLS の計算方法が異なる．

以下で，正規化手法ごとに本ノードの出力ベクトルの計算方法を示す．

CEPSTRAL ：FBANK 端子への入力を，

$$x = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.141}$$

と表す．ただし，P は，入力特徴ベクトルの次元数で，FBANK_COUNT である．出力されるベクトルは，$P+1$ 次元ベクトルで， MSLS 係数とパワー項から構成される．1 次元目から P 次元目までは，MSLS で，

$P+1$ 次元目は，パワー項である．本ノードの出力ベクトルは，

$$y = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.142}$$

$$y(p) = \frac{1}{P}\sum_{q=0}^{P-1}\left\{L(q)\cdot\sum_{r=0}^{P-1}\left\{\log(x(r))\cos\left(\frac{\pi q(r+0.5)}{P}\right)\right\}\cos\left(\frac{\pi q(p+0.5)}{P}\right)\right\} \tag{6.143}$$

である．ただし，リフタリング係数は，

$$L(p) = \begin{cases} 1.0, & (p = 0, \ldots, P-1), \\ 0.0, & (p = P, \ldots, 2P-1), \end{cases} \tag{6.144}$$

とする．ただし，$Q = 22$ である．

SPECTRAL ：FBANK 部の入力を

$$x = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.145}$$

と表す．ただし，P は，入力特徴ベクトルの次元数で，FBANK_COUNT である．出力されるベクトルは，$P+1$ 次元ベクトルで， MSLS 係数とパワー項から構成される．1 次元目から P 次元目までは，MSLS で，$P+1$ 次元目は，パワー項である．本ノードの出力ベクトルは，

$$y = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.146}$$

$$y(p) = \begin{cases} (\log(x(p)) - \mu) - 0.9(\log(x(p-1)) - \mu), & if \;\; p = 1, \ldots, P-1 \\ \log(x(p), & if \;\; p = 0, \end{cases} \tag{6.147}$$

$$\mu = \frac{1}{P}\sum_{q=0}^{P-1}\log(x(q)), \tag{6.148}$$

である．周波数方向の平均除去と，ピーク強調処理を適用ている．

対数パワー項は，SPECTRUM 端子の入力を

$$s = [s(0), s(1), \ldots, s(N-1)]^T \tag{6.149}$$

と表す．ただし，N は，SPECTRUM 端子に接続された `Map` のサイズによって決る．`Map` は，0 から π までのスペクトル表現を B 個のビンに格納しているとすると，$N = 2(B-1)$ である．この時，パワー項は，

$$p = \log\left(\frac{1}{N}\sum_{n=0}^{N-1}s(n)\right) \tag{6.150}$$

である．

2*P*-dimension vector sequence

Static feature (*P*-dimensional vector)	$y(f\text{-}2,0)$	$y(f\text{-}1,0)$	• • •	$y(f\text{-}2,0)$	$y(f\text{-}1,0)$	$y(f,0)$	$y(f\text{+}1,0)$	$y(f\text{+}2,0)$
	$y(f\text{-}2,1)$	$y(f\text{-}1,1)$	• • •	$y(f\text{-}2,1)$	$y(f\text{-}1,1)$	$y(f,1)$	$y(f\text{+}1,1)$	$y(f\text{+}2,1)$
	$y(f\text{-}2,2)$	$y(f\text{-}1,2)$	• • •	$y(f\text{-}2,2)$	$y(f\text{-}1,2)$	$y(f,2)$	$y(f\text{+}1,2)$	$y(f\text{+}2,2)$
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	$y(f\text{-}2,P\text{-}1)$	$y(f\text{-}1,P\text{-}1)$	• • •	$y(f\text{-}2,P\text{-}1)$	$y(f\text{-}1,P\text{-}1)$	$y(f,P\text{-}1)$	$y(f\text{+}1,P\text{-}1)$	$y(f\text{+}2,P\text{-}1)$
Dynamic feature (*P*-dimensional vector)	$y(f\text{-}2,P)$	$y(f\text{-}1,P)$	• • •	$y(f\text{-}2,P)$	$y(f\text{-}1,P)$	$y(f,P)$	$y(f\text{+}1,P)$	$y(f\text{+}2,P)$
	$y(f\text{-}2,P\text{+}1)$	$y(f\text{-}1,P\text{+}1)$	• • •	$y(f\text{-}2,P\text{+}1)$	$y(f\text{-}1,P\text{+}1)$	$y(f,P\text{+}1)$	$y(f\text{+}1,P\text{+}1)$	$y(f\text{+}2,P\text{+}1)$
	$y(f\text{-}2,P\text{+}2)$	$y(f\text{-}1,P\text{+}2)$	• • •	$y(f\text{-}2,P\text{+}2)$	$y(f\text{-}1,P\text{+}2)$	$y(f,P\text{+}2)$	$y(f\text{+}1,P\text{+}2)$	$y(f\text{+}2,P\text{+}2)$
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	$y(f\text{-}2,2P\text{-}1)$	$y(f\text{-}1,2P\text{-}1)$	• • •	$y(f\text{-}2,2P\text{-}1)$	$y(f\text{-}1,2P\text{-}1)$	$y(f,2P\text{-}1)$	$y(f\text{+}1,2P\text{-}1)$	$y(f\text{+}2,2P\text{-}1)$

Time (frame index)

**Shadowed elements are filled with ZERO.*

図 6.85: MSLSExtraction の出力パラメータ

6.4.6 PreEmphasis

ノードの概要

音声認識用の音響特徴量抽出の際に高域の周波数を強調する処理（プリエンファシス）を行い，ノイズへの頑健性を高める．

必要なファイル

無し．

使用方法

一般的に，MFCC 特徴量抽出の前に用いる．また，HARK で一般的に用いている MSLS 特徴量抽出の際にも前処理として用いることができる．

典型的な接続例

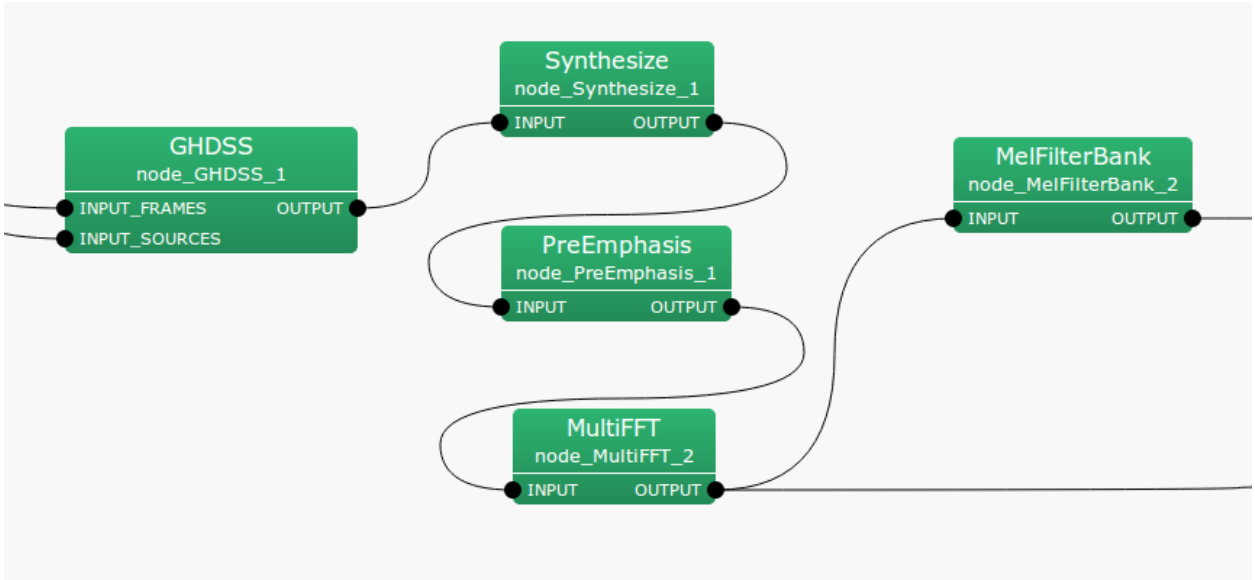


図 6.86: PreEmphasis の接続例

ノードの入出力とプロパティ

入力

INPUT ：`Map<int, ObjectRef>`，入力信号が時間領域波形の場合は，`ObjectRef` は，`Vector<float>` として扱われる．また，周波数領域の信号の場合は，`Vector<complex<float> >` として扱われる．

出力

表 6.77: PreEmphasis のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	信号長もしくは FFT の窓長
SAMPLING_RATE	int	16000	[Hz]	サンプリングレート
PREEMCOEF	float	0.97		プリエンファシス係数
INPUT_TYPE	string	WAV		入力信号タイプ

OUTPUT : Map<int, ObjectRef>，高域強調された信号が出力される．ObjectRef は，入力の種類に対応して，時間領域波形では Vector<float>，周波数領域信号では Vector<complex<float> > となる．

パラメータ

LENGTH INPUT_TYPE が SPECTRUM の場合は FFT 長であり，前段のモジュールと値を合わせる必要がある．INPUT_TYPE が WAV の場合は，１フレームに含まれる信号の信号長を表し，同様に前段のノードと値を合わせる必要がある．通常の構成では，FFT 長と信号長は一致する．

SAMPLING_RATE LENGTH と同様，他のノードと値を合わせる必要がある．

PREEMCOEF 以下で c_p として表わされるプリエンファシス係数．音声認識では，0.97 が一般的に用いられる．

INPUT_TYPE 入力のタイプは WAV, SPECTRUM の２種類が用意されている．WAV は時間領域波形入力の際に用いる．また，SPECTRUM は周波数領域信号入力の際に用いる．

ノードの詳細

プリエンファシスの必要性や一般的な音声認識における効果に関しては，様々な書籍や論文で述べられているので，それらを参考にしてほしい．一般的には，この処理を行った方がノイズに頑健になると言われているが，HARK では，マイクロホンマイクロホンアレイ処理を行っているためか，この処理の有無による性能差はそれほど大きくない．ただし，音声認識で用いる音響モデルを学習する際に用いた音声データとパラメータを合わせる必要性がある．つまり，音響モデル学習で用いたデータにプリエンファシスを行っていれば，入力データに対してもプリエンファシスを行った方が性能が高くなる．

具体的には PreEmphasis は，入力信号の種類に対応して，２種類の処理からなっている．

時間領域での高域強調:

時間領域の場合は，t をフレーム内でのサンプルを表すインデックスとし，入力信号を $s[t]$，高域強調した信号を $p[t]$，プリエンファシス係数を c_p とすれば，以下の式によって表すことができる．

$$p[t] = \begin{cases} s[t] - c_p \cdot s[t-1] & t > 0 \\ (1 - c_p) \cdot s[0] & t = 0 \end{cases} \tag{6.151}$$

周波数領域での高域強調:

時間領域のフィルタと等価なフィルタを周波数領域で実現するため，上記のフィルタのインパルス応答に対して，周波数解析を行ったスペクトルフィルタを用いている．また，低域（下から 4 バンド分）と高域（$fs/2$ - 100 Hz 以上）は，誤差を考慮して，強制的に 0 としている．ただし，fs は，サンプリング周波数を表す．

6.4.7 SaveFeatures

ノードの概要

特徴量ベクトルをファイルに保存する．

必要なファイル

無し．

使用方法

どんなときに使うのか

MFCC，MSLS などの音響特徴量を保存する時に使用する．

典型的な接続例

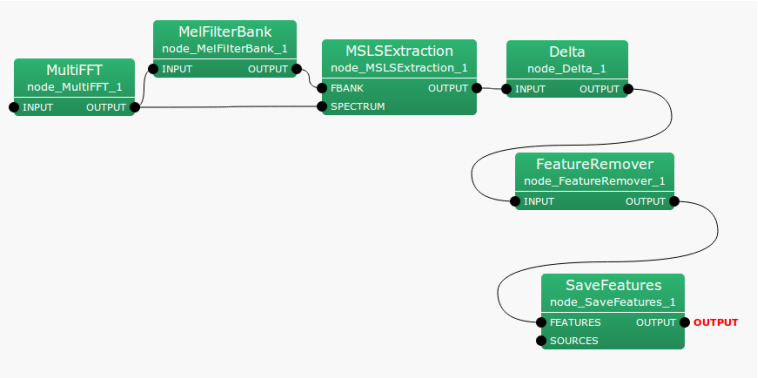


図 6.87: SaveFeatures の接続例

ノードの入出力とプロパティ

表 6.78: SaveFeatures のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
BASENAME	`string`			保存する時のファイル名の Prefix

入力

FEATURES ：`Map<int, ObjectRef>` 型．特徴量ベクトルは `Vector<float>` で示される．

SOURCES ：`Vector<ObjectRef>` 型である．この入力は，オプションである．

出力

OUTPUT : `Map<int, ObjectRef>` 型である．

パラメータ

BASENAME : `string` 型である．保存する時のファイル名の Prefix で，保存時は，Prifix の後に SOURCES の ID が付与されて特徴量が保存される．

ノードの詳細

特徴量ベクトルを保存する．保存するファイルの形式は，ベクトル要素を IEEE 754 の 32 ビット浮動小数点数形式，リトルエンディアンで保存する．名前付ルールは， BASENAME プロパティで与えた Prifix の後に ID 番号が付与される．

6.4.8 SaveHTKFeatures

ノードの概要

特徴量ベクトルを HTK (The Hidden Markov Model Toolkit) で扱えるファイル形式で保存する．

必要なファイル

無し．

使用方法

どんなときに使うのか

MFCC，MSLS などの音響特徴量を保存する時に使用する．SaveFeatures と異なり，HTK で扱えるように専用のヘッダが追加されて保存できる．

典型的な接続例

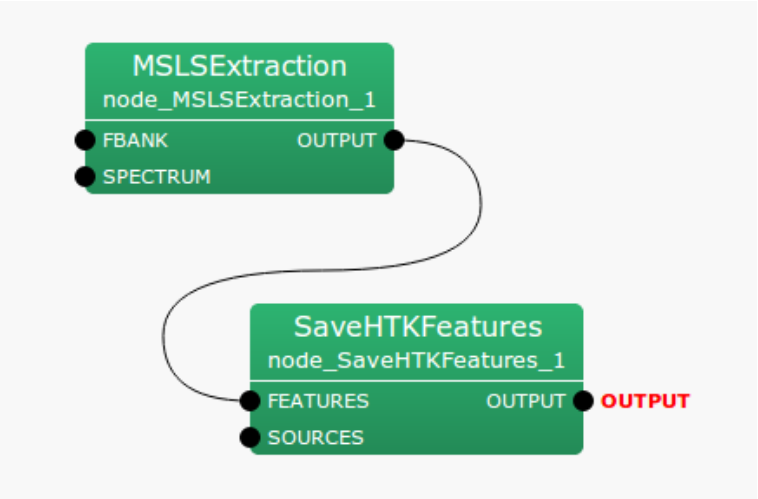


図 6.88: SaveHTKFeatures の接続例

ノードの入出力とプロパティ

表 6.79: SaveHTKFeatures のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
BASENAME	string			保存する時のファイル名の Prefix
HTK_PERIOD	int	100000	[100 nsec]	フレーム周期
FEATURE_TYPE	string	USER		特徴量の型

入力

FEATURES : `Map<int, ObjectRef>` 型である．

SOURCES : `Vector<ObjectRef>` 型である．この入力は，オプションである．

出力

OUTPUT : `Map<int, ObjectRef>` 型である．

パラメータ

BASENAME : `string` 型である．保存する時のファイル名の Prefix で，保存時は，Prifix の後に SOURCES の ID が付与されて特徴量が保存される．

HTK_PERIOD : フレーム周期の設定で単位は [100 nsec] である．サンプリング周波数 16000[Hz] でシフト長 160 の場合，フレーム周期は 10[msec] となるので 10[ms] = 100000 * 100[nsec] で，100000 が適当な設定値となる．

FEATURE_TYPE : HTK で扱う特徴量の形式の指定．HTK 独自の型に従う．例えば，`MFCC_E_D` の場合は「(MFCC+パワー) + デルタ (MFCC+パワー)」となる．HARK で計算した特徴量の中身と合わせるように設定する．詳細は HTKbook を参照されたい．

ノードの詳細

特徴量ベクトルを HTK で扱われる形式で保存する．保存するファイルの形式は，HTK 固有のヘッダを付与した後，ベクトル要素を IEEE 754 の 32 ビット浮動小数点数形式，ビッグエンディアンで保存する．名前付ルールは， BASENAME プロパティで与えた Prifix の後に ID 番号が付与される．HTK のヘッダの設定はプロパティで変更可能．

6.4.9 SpectralMeanNormalization

ノードの概要

入力音響特徴量から特徴量の平均を除去することを意図したノードである．ただし，実時間処理を実現するためには，当該発話の平均を除去することができない問題がある．当該発話の平均値をなんらかの値を用いてい推定あるいは，近似する必要がある．

必要なファイル

無し．

使用方法

どんなときに使うのか

音響特徴量の平均を除去したい時に使用する．音響モデル学習用音声データと認識用音声データの収録環境の平均値のミスマッチを除去できる．

音声収録環境においてマイクロホンの特性は，統一できないことが多い．特に，音響モデル学習時と認識時の音声収録環境は，必ずしも等しくない．通常，学習用の音声コーパス作成者と，認識用音声データの収録者が異なるから環境を揃えることは困難である．従って，音声の収録環境に依存しない特徴量を用いる必要がある．

例えば，学習データ収録に使用するマイクロホンと認識時に使用するマイクロフォンは通常異なる．マイクロホンの特性の違いが，収録音の音響特徴のミスマッチとして現れ，認識性能の低下を招く．マイクロホンの特性の違いは，時不変であり，平均スペクトルの差となって現れる．従って，平均スペクトルを除去することで，簡易的に収録環境に依存した成分を特徴量から除去できる．

典型的な接続例

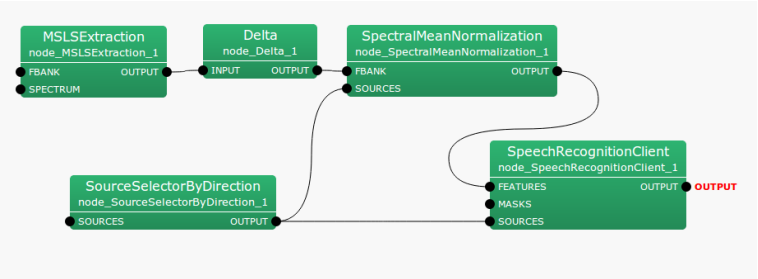


図 6.89: SpectralMeanNormalization の接続例

ノードの入出力とプロパティ

入力

FBANK ：`Map<int, ObjectRef>` 型．音源 ID と特徴量ベクトルの `Vector<float>` 型のデータのペア．

表 6.80: SpectralMeanNormalization のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FBANK_COUNT	int	13		入力特徴パラメータの次元数

SOURCES : Vector<ObjectRef> 型である．音源位置．

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID と特徴量ベクトルの Vector<float> 型のデータのペア．

パラメータ

FBANK_COUNT : int 型である．値域は 0 または正の整数である．

ノードの詳細

入力音響特徴量から特徴量の平均を除去することを意図したノードである．ただし，実時間処理を実現するためには，当該発話の平均を除去することができない問題がある．当該発話の平均値をなんらかの値を用いてい推定あるいは，近似する必要がある．

当該発話の平均を除去する替りに，前発話の平均を近似値とし，除去することで実時間平均除去を実現する．この方法では，更に音源方向を考慮しなければならない．音源方向によって伝達関数が異なるため，当該発話と前発話が異なる方向から受音された場合には，前発話の平均は，当該発話の平均の近似として不適当である．この場合，当該発話の平均の近似として，当該発話よりも前に発話されかつ，同方向からの発話の平均を用いる．

最後に以後の平均除去に備え，当該発話の平均を計算し，当該発話方向の平均値としてメモリ内に保持する．発話中に音源が 10 [deg] 以上移動する場合は，別音源として，平均を計算する．

6.5 MFM カテゴリ

6.5.1 DeltaMask

ノードの概要

本ノードは，静的特徴のミッシングフィーチャーマスクベクトルから動的特徴量のミッシングフィーチャーマスクベクトルを求め，静的特徴と動的特徴のミッシングフィーチャーマスクから構成されるマスクベクトルを生成する．

必要なファイル

無し．

使用方法

どんなときに使うのか

ミッシングフィーチャー理論に基づき，特徴量を信頼度に応じてマスクして音声認識を行うために用いる．通常，MFMGeneration の後段に用いる．

典型的な接続例

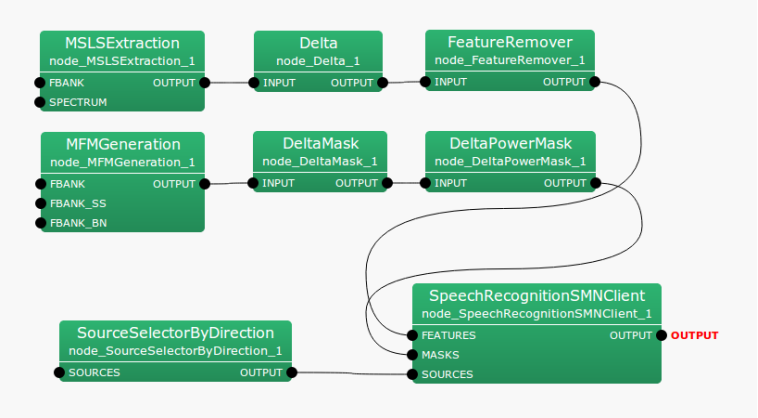


図 6.90: DeltaMask の典型的な接続例

ノードの入出力とプロパティ

表 6.81: DeltaMask パラメータ表

パラメータ名	型	デフォルト値	単位	説明
FBANK_COUNT	int			静的特徴の次元数

入力

INPUT : `Map<int, ObjectRef>` 型．音源 ID と特徴量のマスクベクトルの `Vector<float>` 型のデータのペア．マスク値は，0.0 から 1.0 の実数で，0.0 が特徴量を信頼しない， 1.0 が信頼する状態を表す．

出力

OUTPUT : `Map<int, ObjectRef>` 型．音源 ID と特徴量のマスクベクトルの `Vector<float>` 型のデータのペア．マスク値は，0.0 から 1.0 の実数で，0.0 が特徴量を信頼しない状態， 1.0 が信頼する状態を表す．

パラメータ

FBANK_COUNT : `int` 型である．処理する特徴量の次元数．値域は，正の整数．

ノードの詳細

本ノードは，静的特徴のマスクベクトルから動的特徴量のマスクベクトルを求め，静的特徴と動的特徴のミッシングフィーチャーマスクから構成されるマスクベクトルを生成する．

フレーム時刻 f における，入力マスクベクトルを，

$$m(f) = [m(f,0), m(f,1), \ldots, m(f,2P-1)]^T \tag{6.152}$$

と表す．ただし，P は，入力マスクベクトルのうち，静的特徴を表わす次元数を表わし，FBANK_COUNT で与える．静的特徴のマスク値を用い，動的特徴のマスク値を求め， P から $2P-1$ 次元の要素に入れて出力ベクトルを生成する．出力ベクトル $m'(f)$ は，

$$y'(f) = [m'(f,0), m'(f,1), \ldots, m'(f,2P-1)]^T \tag{6.153}$$

$$m'(f,p) = \begin{cases} m(f,p), & \text{if } p = 0, \ldots, P-1, \\ \prod_{\tau=-2}^{2} m(f+\tau, p), & \text{if } p = P, \ldots, 2P-1, \end{cases} \tag{6.154}$$

である．図 6.91 に DeltaMask の入出力フローを示す．

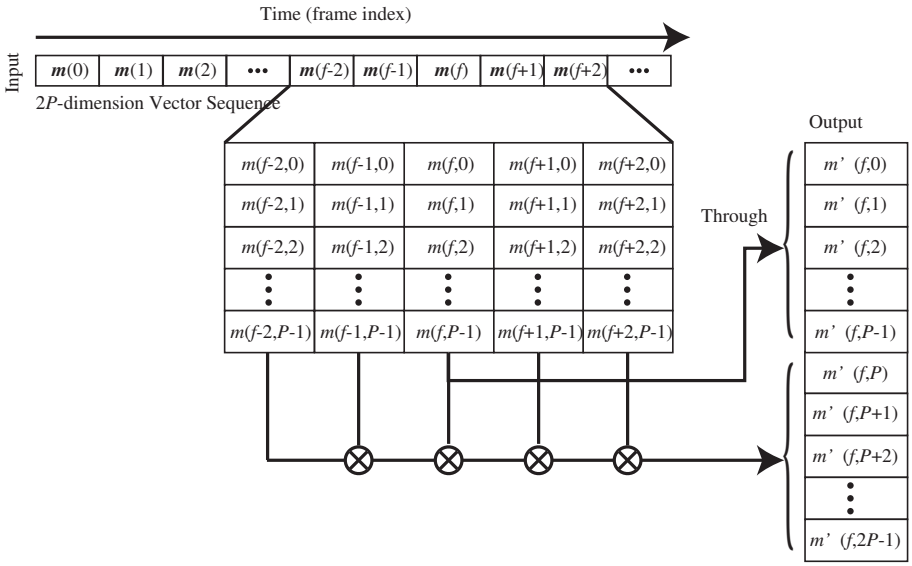


図 6.91: DeltaMask の入出力フロー.

6.5.2 DeltaPowerMask

ノードの概要

本ノードは，音響特徴量の１つである動的対数パワーのマスク値を生成する．生成したマスク値を入力のマスクベクトルの要素に追加する．

必要なファイル

無し．

使用方法

どんなときに使うのか

ミッシングフィーチャー理論に基づき，特徴量を信頼度に応じてマスクして音声認識を行う．通常，DeltaMask の後段に用いる．

典型的な接続例

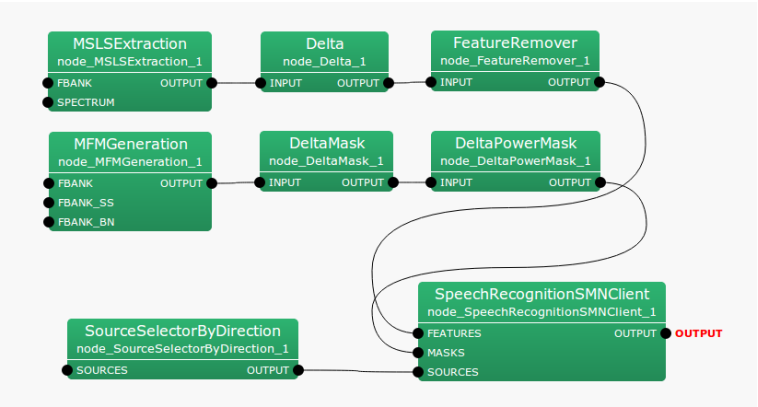


図 6.92: DeltaPowerMask の典型的な接続例

ノードの入出力とプロパティ

INPUT ：`Map<int, ObjectRef>` 型．音源 ID と特徴量のマスクベクトルの `Vector<float>` 型のデータのペア．マスク値は，0.0 から 1.0 の実数で，0.0 が特徴量を信頼しない状態， 1.0 が信頼する状態を表す．

出力

OUTPUT ：`Map<int, ObjectRef>` 型．音源 ID と特徴量のマスクベクトルの `Vector<float>` 型のデータのペア．マスク値は，0.0 から 1.0 の実数で，0.0 が特徴量を信頼しない状態， 1.0 が信頼する状態を表す．

パラメータ

ノードの詳細

本ノードは，音響特徴量の１つである動的対数パワーのマスク値を生成する．生成するマスク値は，常に 1.0 である．出力マスクの次元数は，入力マスクの次元数 + 1 次元である．

6.5.3 MFMGeneration

ノードの概要

本ノードは，ミッシングフィーチャー理論に基く音声認識のためのミッシングフィーチャーマスク (Missing-Feature-Mask:MFM) を生成する．

必要なファイル

無し．

使用方法

どんなときに使うのか

ミッシングフィーチャー理論に基く音声認識するために使用する．MFMGeneration は，PostFilter と GHDSS の出力からミッシングフィーチャーマスクを生成する．そのため PostFilter と GHDSS の利用が前提条件である．

典型的な接続例

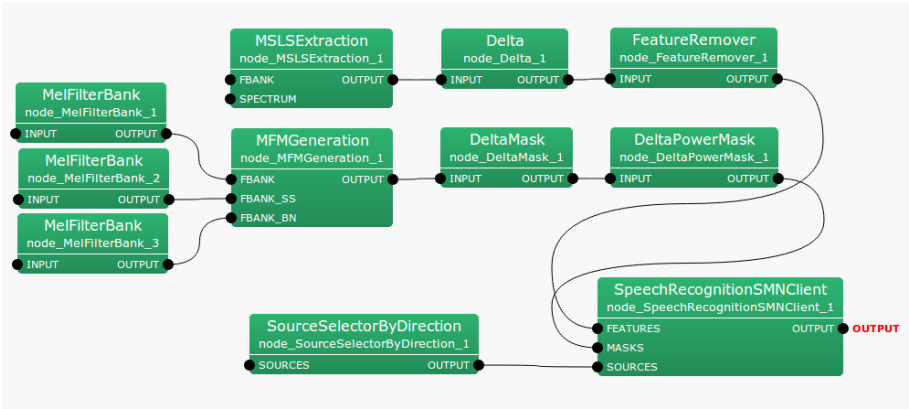


図 6.93: MFMGeneration の接続例

ノードの入出力とプロパティ

表 6.82: MFMGeneration のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FBANK_COUNT	int	13		音響特徴量の次元数
THRESHOLD	float	0.2		0.0 から 1.0 の間の連続値を 0.0 (信頼しない) または 1.0 (信頼する) に量子化するためのしきい値

入力

FBANK : Map<int, ObjectRef> 型．音源 ID と PostFilter の出力から求めたメルフィルタバンク出力エネルギーから構成されるベクトルの Vector<float> 型のデータのペア．

FBANK_SS : Map<int, ObjectRef> 型．音源 ID と GHDSS の出力から求めたメルフィルタバンク出力エネルギーから構成されるベクトルの Vector<float> 型のデータのペア．

FBANK_BN : Map<int, ObjectRef> 型．音源 ID と BGNEstimator の出力から求めたメルフィルタバンク出力エネルギーから構成されるベクトルの Vector<float> 型のデータのペア．

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID と ミッシングフィーチャーマスクベクトルから構成されるベクトルの Vector<float> 型のデータのペア．ベクトルの要素は 0.0 (信頼しない) または 1.0 (信頼する) である．出力ベクトルは，2*FBANK_COUNT 次元ベクトルで， FBANK_COUNT 以上の次元要素は，全て 0 である．動的特徴量用のミッシングフィーチャーマスクのプレースホルダである．

パラメータ

FBANK_COUNT : int 型である．音響特徴量の次元数である．

THRESHOLD : float 型である．ノード内部で計算する 0.0(信頼しない) から 1.0(信頼する) までの信頼度を量子化するためのしきい値である．しきい値に 0.0 を設定すると，すべての信頼度がしきい値以上になり，すべてのマスク値が 1.0 になる．このときの処理は，通常の音声認識処理と等価になる．

ノードの詳細

ミッシングフィーチャー理論に基く音声認識のためのミッシングフィーチャーマスクを生成する．

信頼度 $r(p)$ をしきい値 THRESHOLD でしきい値処理し，マスク値を 0.0 (信頼しない) また 1.0 (信頼する) に量子化する．信頼度は，PostFilter , GHDSS , BGNEstimator の出力から求めたメルフィルタバンクの出力エネルギー $f(p), b(p), g(p),$ から求める．このときフレーム番号 f のマスクベクトルは，

$$m(f) = [m(f,0), m(f,1), \ldots, m(f,P-1)]^T \tag{6.155}$$

$$m(f,p) = \begin{cases} 1.0, & r(p) > \mathrm{THRESHOLD} \\ 0.0, & r(p) \leq \mathrm{THRESHOLD} \end{cases}, \tag{6.156}$$

$$r(p) = \min(1.0, (f(p) + 1.4 * b(p))/(fg(p) + 1.0)), \tag{6.157}$$

$$\tag{6.158}$$

である．ただし，P は，入力特徴ベクトルの次元数で，FBANK_COUNT で指定する正の整数である．実際に出力するベクトルの次元数は， 2*FBANK_COUNT 次のベクトルである．FBANK_COUNT 以上の次元要素は，0 で埋められる．これは，動的特徴量マスク値を入れるためのプレースホルダである．図 6.94 に出力ベクトル列の模式図を示す．

2*P*-dimension vector sequence

Static feature (*P*-dimensional vector)	$m(f0,0)$	$m(1,0)$	• • •	$m(f\text{-}2,0)$	$m(f\text{-}1,0)$	$m(f,0)$	$m(f+1,0)$	$m(f+2,0)$
	$m(f0,1)$	$m(1,1)$	• • •	$m(f\text{-}2,1)$	$m(f\text{-}1,1)$	$m(f,1)$	$m(f+1,1)$	$m(f+2,1)$
	$m(0,2)$	$m(1,2)$	• • •	$m(f\text{-}2,2)$	$m(f\text{-}1,2)$	$m(f,2)$	$m(f+1,2)$	$m(f+2,2)$
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	$m(0,P\text{-}1)$	$m(1,P\text{-}1)$	• • •	$m(f\text{-}2,P\text{-}1)$	$m(f\text{-}1,P\text{-}1)$	$m(f,P\text{-}1)$	$m(f+1,P\text{-}1)$	$m(f+2,P\text{-}1)$
Dynamic feature (*P*-dimensional vector)	$m(0,P)$	$m(1,P)$	• • •	$m(f\text{-}2,P)$	$m(f\text{-}1,P)$	$m(f,P)$	$m(f+1,P)$	$m(f+2,P)$
	$m(0,P+1)$	$m(1,P+1)$	• • •	$m(f\text{-}2,P+1)$	$m(f\text{-}1,P+1)$	$m(f,P+1)$	$m(f+1,P+1)$	$m(f+2,P+1)$
	$m(0,P+2)$	$m(1,P+2)$	• • •	$m(f\text{-}2,P+2)$	$m(f\text{-}1,P+2)$	$m(f,P+2)$	$m(f+1,P+2)$	$m(f+2,P+2)$
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	$m(0,2P\text{-}1)$	$m(1,2P\text{-}1)$	• • •	$m(f\text{-}2,2P\text{-}1)$	$m(f\text{-}1,2P\text{-}1)$	$m(f,2P\text{-}1)$	$m(f+1,2P\text{-}1)$	$m(f+2,2P\text{-}1)$

Time (frame index)

**Shadowed elements are filled with ZERO.*

図 6.94: MFMGeneration の出力ベクトル列

6.6 ASRIF カテゴリ

6.6.1 SpeechRecognitionClient

ノードの概要

音響特徴量をネットワーク経由で音声認識ノードに送信するノードである．

必要なファイル

無し．

使用方法

どんなときに使うのか

音響特徴量を HARK 外のソフトウェアに送信するために用いる．例えば，大語彙連続音声認識ソフトウェア Julius [1] に送信し，音声認識を行う．

典型的な接続例

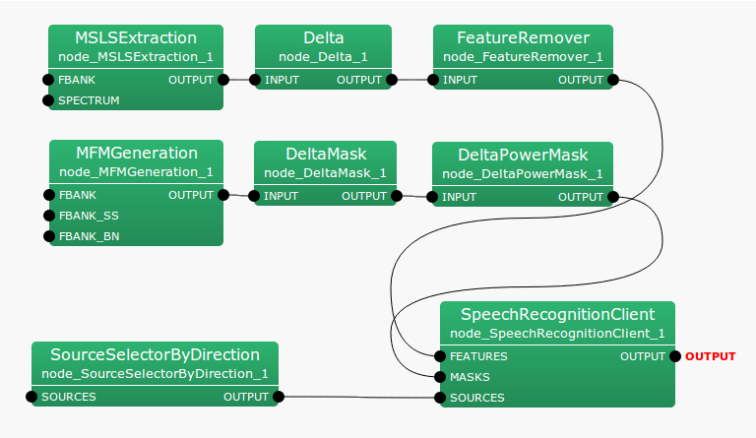


図 6.95: SpeechRecognitionClient の接続例

ノードの入出力とプロパティ

入力

FEATURES ：`Map<int, ObjectRef>` 型．音源 ID と特徴量ベクトルの `Vector<float>` 型のデータのペア．

MASKS ：`Map<int, ObjectRef>` 型．音源 ID とマスクベクトルの `Vector<float>` 型のデータのペア．

SOURCES ：`Vector<ObjectRef>` 型．

表 6.83: SpeechRecognitionClient のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
MFM_ENABLED	bool	true		ミッシングフィーチャーマスクを送出するかしないかの選択
HOST	string	127.0.0.1		Julius/Julian が動いているサーバのホスト名/IP アドレス
PORT	int	5530		ネットワーク送出用ポート番号
SOCKET_ENABLED	bool	true		ソケット出力をするかどうかを決めるフラグ

出力

OUTPUT : Vector<ObjectRef> 型.

パラメータ

MFM_ENABLED : bool 型. true の場合, MASKS を転送する. false の場合は, 入力の MASKS を無視し, すべて 1 のマスクを転送する.

HOST : string 型. 音響パラメータを転送するホストの IP アドレスでる. SOCKET_ENABLED が false の場合は, 無効である.

PORT : int 型. 音響パラメータ転送するソケット番号である. SOCKET_ENABLED が false の場合は, 無効である.

SOCKET_ENABLED : bool 型. true で音響パラメータをソケットに転送し, false で転送しない.

ノードの詳細

MFM_ENABLED が true かつ SOCKET_ENABLED のとき, 音響特徴量ベクトルとマスクベクトルをネットワークポートを経由で音声認識ノードに送信するノードである. MFM_ENABLED が false のとき, ミッシングフィーチャー理論を使わない音声認識になる. 実際には, マスクベクトルの値をすべて 1 , つまりすべての音響特徴量を信頼する状態にしてマスクベクトルを送り出す. SOCKET_ENABLED が false のときは, 特徴量を音声認識ノードに送信しない. これは, 音声認識エンジンが外部プログラムに依存しているため, 外部プログラムを動かさずに HARK のネットワーク動作チェックを行うために使用する. HOST は, ベクトルを送信する外部プログラムが動作する HOST の IP アドレスを指定する. PORT は, ベクトルを送信するネットワークポート番号を指定する.

参考文献:

(1) http://julius.sourceforge.jp/en_index.php

6.6.2 SpeechRecognitionSMNClient

ノードの概要

音響特徴量をネットワーク経由で音声認識ノードに送信するノードである．SpeechRecognitionClient との違いは，入力特徴ベクトルの平均除去 (Spectral Mean Normalization: SMN) を行う点である．ただし，本ノードでは当該発話区間全体の平均を除去する手法を用いている．したがって，オンラインで使用する場合でも当該発話が終わるまで送信が行われず実時間処理とならない問題がある．実時間処理を実現するためには，当該発話区間全体の特徴量を得ずに当該発話の平均値をなんらかの値を用いてい推定あるいは，近似する必要がある．近似処理の詳細は，ノードの詳細部分を参照のこと．

必要なファイル

無し．

使用方法

どんなときに使うのか

音響特徴量を HARK 外のソフトウェアに送信するために用いる．例えば，大語彙連続音声認識ソフトウェア Julius [(1)] に送信し，音声認識を行う．

典型的な接続例

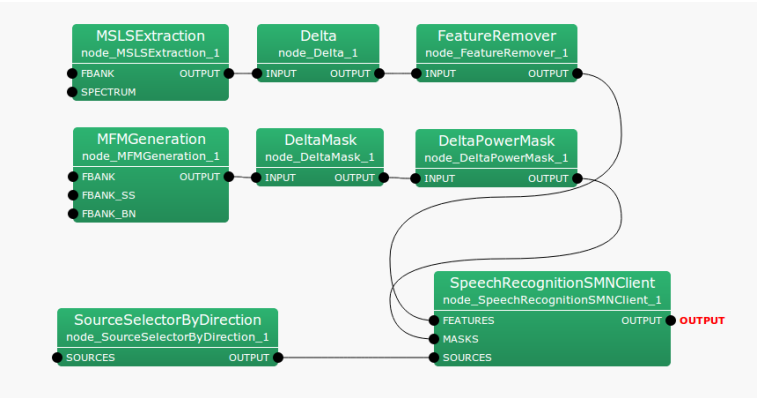


図 6.96: SpeechRecognitionSMNClient の接続例

ノードの入出力とプロパティ

入力

FEATURES ：`Map<int, ObjectRef>` 型．音源 ID と特徴量ベクトルの `Vector<float>` 型のデータのペア．

MASKS ：`Map<int, ObjectRef>` 型．音源 ID とマスクベクトルの `Vector<float>` 型のデータのペア．

表 6.84: SpeechRecognitionSMNClient のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
MFM_ENABLED	bool	true		ミッシングフィーチャーマスクを送出するかしないかの選択
HOST	string	127.0.0.1		Julius/Julian が動いているサーバのホスト名/IP アドレス
PORT	int	5530		ネットワーク送出用ポート番号
SOCKET_ENABLED	bool	true		ソケット出力をするかどうかを決めるフラグ

SOURCES : Vector<ObjectRef> 型.

出力

OUTPUT : Vector<ObjectRef> 型.

パラメータ

MFM_ENABLED : bool 型. true の場合, MASKS を転送する. false の場合は, 入力の MASKS を無視し, すべて 1 のマスクを転送する.

HOST : string 型. 音響パラメータを転送するホストの IP アドレスでる. SOCKET_ENABLED が false の場合は, 無効である.

PORT : int 型. 音響パラメータを転送するソケット番号である. SOCKET_ENABLED が false の場合は, 無効である.

SOCKET_ENABLED : bool 型. true で音響パラメータをソケットに転送し, false で転送しない.

ノードの詳細

MFM_ENABLED が true かつ SOCKET_ENABLED のとき, 音響特徴量ベクトルとマスクベクトルをネットワークポートを経由で音声認識ノードに送信するノードである. MFM_ENABLED が false のとき, ミッシングフィーチャー理論を使わない音声認識になる. 実際には, マスクベクトルの値をすべて 1 , つまりすべての音響特徴量を信頼する状態にしてマスクベクトルを送り出す. SOCKET_ENABLED が false のときは, 特徴量を音声認識ノードに送信しない. これは, 音声認識エンジンが外部プログラムに依存しているため, 外部プログラムを動かさずに HARK のネットワーク動作チェックを行うために使用する. HOST は, ベクトルを送信する外部プログラムが動作する HOST の IP アドレスを指定する. PORT は, ベクトルを送信するネットワークポート番号を指定する.

参考文献:

(1) http://julius.sourceforge.jp/en_index.php

6.7 MISC カテゴリ

6.7.1 ChannelSelector

ノードの概要

マルチチャネルの音声波形や複素スペクトルのデータから，指定したチャネルのデータだけを指定した順番に取り出す．

必要なファイル

無し．

使用方法

どんなときに使うのか

入力されたマルチチャネルの音声波形や複素スペクトルのデータの中から，必要のないチャネルを削除したいとき，あるいは，チャネルの並びを入れ替えたいとき．あるいは，チャネルを複製したいとき．

典型的な接続例

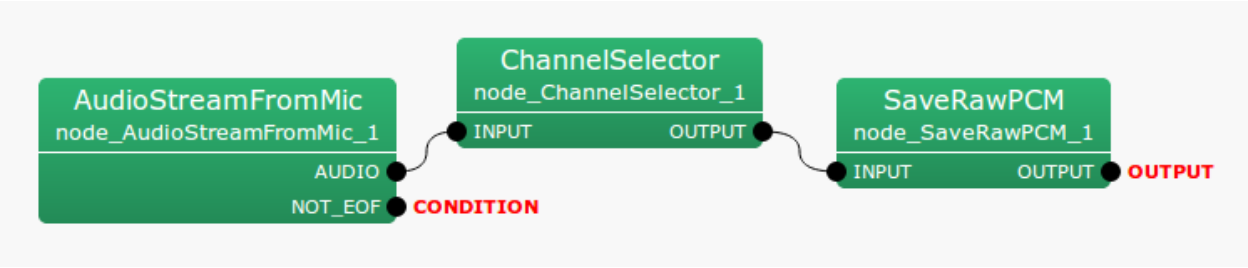


図 6.97: ChannelSelector の典型的な接続例

図 6.97 に典型的な接続例を示す．このネットワークファイルによって，マルチチャネルの音声ファイルのいくつかのチャネルだけを抽出できる．主な入力元は AudioStreamFromMic，AudioStreamFromWave，MultiFFT，主な出力先は SaveRawPCM，MultiFFT などである．

ノードの入出力とパラメータ

入力

INPUT : `Matrix<float>` もしくは `Matrix<complex<float> >` 型．マルチチャネルの音声波形または複素スペクトルのデータ．

出力

OUTPUT : Matrix<float> もしくは Matrix<complex<float> > 型．マルチチャネルの音声波形または複素スペクトルのデータ.

パラメータ

表 6.85: ChannelSelector パラメータ表

パラメータ名	型	デフォルト値	単位	説明
SELECTOR	Object	Vector< int >		出力するチャネルの番号を指定

SELECTOR : 型，デフォルト値は無し．使用するチャネルの，チャネル番号を指定する．チャネル番号は 0 からはじまる．

例: 5 チャネル (0-4) のうち 2，3，4 チャネルだけを使うときは (1) のように，さらに 3 チャネルと 4 チャネルを入れ替えたい時は (2) のように指定する．

(1) <Vector<int> 2 3 4>

(2) <Vector<int> 2 4 3>

ノードの詳細

入力の $N \times M$ 型行列 (Matrix) から指定したチャネルの音声波形（もしくは複素スペクトル）データだけを抽出し，新たな $N' \times M$ 型行列のデータを出力する．ただし，N は入力チャネル数，N' は出力チャネル数.

6.7.2 CombineSource

ノードの概要

LocalizeMUSIC や ConstantLocalization 等から出力された 2 つの音源定位結果を結合し，一つにまとめて出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

複数の音源定位結果を GHDSS 等の後段処理で使用したい時に使える．以下の使用事例が考えられる．

- LocalizeMUSIC を複数使用する場合
- ConstantLocalization と LocalizeMUSIC の両方を組み合わせる場合

典型的な接続例

主に，ConstantLocalization，LoadSourceLocation，LocalizeMUSIC などの音源定位結果を入力として接続し，GHDSS や SpeechRecognitionClient などの音源定位結果が必要なモジュールを出力側に接続する．
図 6.98 は，LocalizeMUSIC と ConstantLocalization を組み合わせる例である．

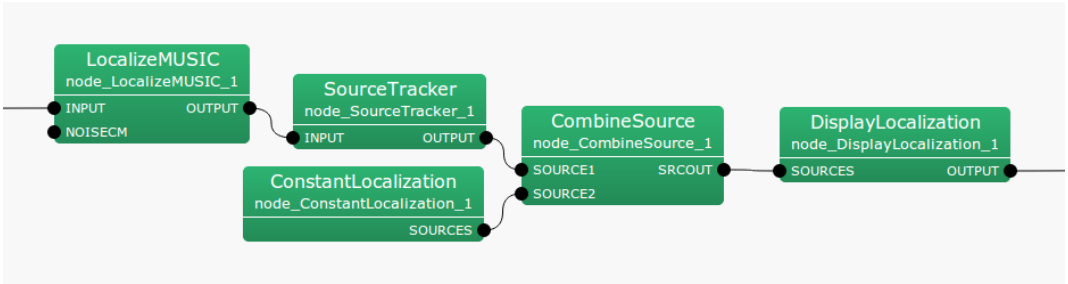


図 6.98: CombineSource の接続例

ノードの入出力とプロパティ

入力

SOURCES1 ： `Vector<ObjectRef>` 型．結合したい音源定位結果を接続する．`ObjectRef` が参照するのは，`Source` 型のデータである．

SOURCES2 : Vector<ObjectRef> 型．結合したい音源定位結果を接続する．ObjectRef が参照するのは，Source 型のデータである．

出力

SRCOUT : Vector<ObjectRef> 型．結合後の音源定位結果を出力する．ObjectRef が参照するのは，Source 型のデータである．

パラメータ

なし

ノードの詳細

本ノードは二つの音源定位結果を一つにまとめて出力する．音源の方位角，仰角，パワー，ID は引き継がれる．

6.7.3 DataLogger

ノードの概要

入力されたデータにパラメータで指定したラベルを付与して標準出力またはファイルに出力する．

必要なファイル

無し．

使用方法

どんなときに使うのか

ノードのデバッグの際や，ノードの出力を保存して実験や解析に利用したい場合に使う．

典型的な接続例

例えば，各音源の特徴量をテキストで出力して解析したい場合には，以下のような接続すればよい．

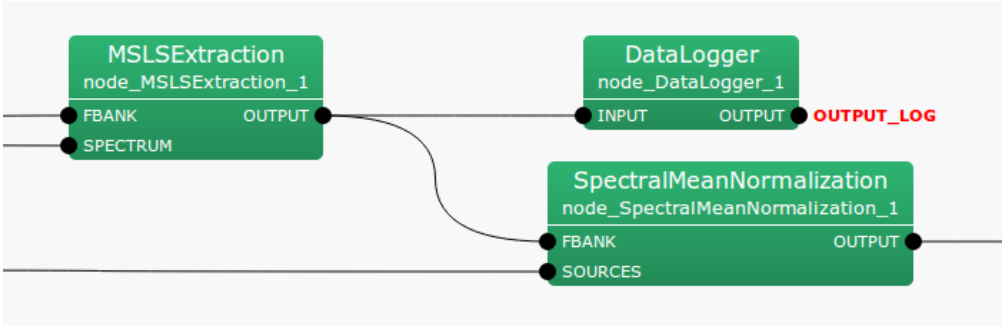


図 6.99: DataLogger の接続例

ノードの入出力とプロパティ

表 6.86: ModuleName のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LABEL	string			出力するデータに付与するラベル

入力

INPUT : any．ただし，サポートする型は，Map<int, float>，Map<int, int> または Map<int, ObjectRef> である．Map<int, ObjectRef> の ObjectRef は， Vector<float> または Vector<complex<float> > のみをサポートしている．

出力

OUTPUT ：入力と同じ．

パラメータ

LABEL ：複数の DataLogger を利用したときにどの DataLogger が出力した結果か分かるように，出力するデータに付与する文字列を指定する．

ノードの詳細

入力されたデータにパラメータで指定したラベルを付与して標準出力またはファイルに出力する．サポートしている型は HARK でよく利用する音源 ID を キーとした `Map` 型のみである．出力される形式は以下の通りである．

ラベル フレームカウント キー 1 値 1 キー 2 値 2 . . .

本ノードの 1 フレームカウントの出力は 1 行で，上記のように最初にパラメータで指定したラベル，次にフレームカウント，その後に `Map` 型のキーと値を全てスペース区切りで出力する．値が `Vector` の時は，すべての要素がスペース区切りで出力される．

6.7.4 HarkParamsDynReconf

ノードの概要

LocalizeMUSIC , SourceTracker , HRLE のパラメータをネットワークの実行中に変更できるようにネットワーク通信を介してパラメータを受信し，それらのノードに渡す．

必要なファイル

無し．

使用方法

どんなときに使うのか

LocalizeMUSIC , SourceTracker , HRLE のパラメータをネットワークを実行しながら変更したい時に使う．

典型的な接続例

図 6.100 に典型的な接続例を示す．

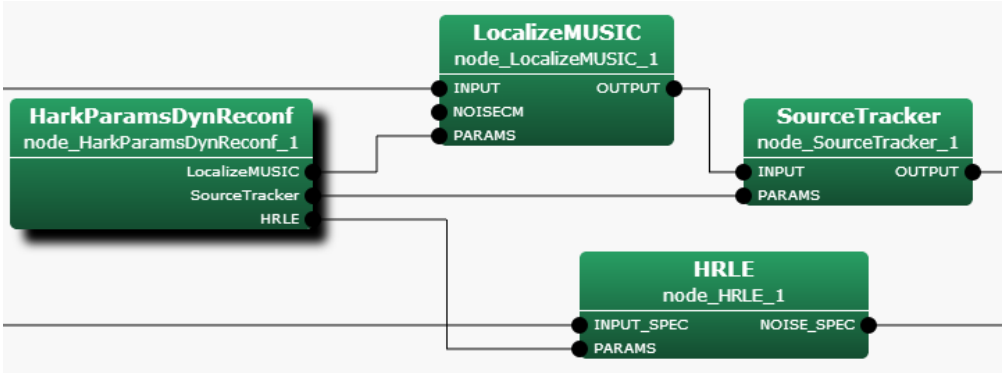


図 6.100: HarkParamsDynReconf の接続例

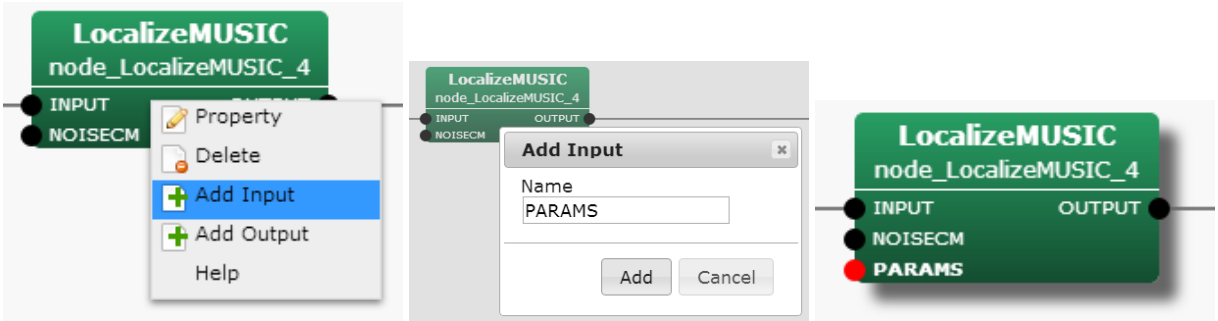
図 6.100 の LocalizeMUSIC , SourceTracker , HRLE には，デフォルトでは非表示の PARAMS 入力端子が追加されている．非表示入力の追加方法を図 6.101 に示す．

ノードの入出力とプロパティ

入力

無し．

出力



Step 1: ノードを右クリックし，Add Input をクリック　Step 2: Name の入力フォームに PARAMS を記入し，Add をクリック　Step 3: ノードに PARAMS 入力端子が追加される

図 6.101: 非表示入力の追加 : PARAMS 入力端子の表示

LocalizeMUSIC : `Vector<ObjectRef>` 型．LocalizeMUSIC のパラメータを出力する．LocalizeMUSIC の PARAMS 入力端子に接続する．

SourceTracker : `Vector<ObjectRef>` 型．SourceTracker のパラメータを出力する．SourceTracker の PARAMS 入力端子に接続する．

HRLE : `Vector<ObjectRef>` 型．HRLE のパラメータを出力する．HRLE の PARAMS 入力端子に接続する．

パラメータ

表 6.87: HarkParamsDynReconf のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
PORT	`int`	9999		ソケット通信のポート番号
ENABLE_DEBUG	`bool`	`false`		デバッグ出力の ON/OFF

PORT : `int` 型．ソケット通信のポート番号を指定する．

ENABLE_DEBUG : `bool` 型．デバッグ出力の ON/OFF を指定する．

ノードの詳細

本ノードがソケット通信のサーバーとなり，クライアントプログラムから LocalizeMUSIC , SourceTracker , HRLE のパラメータをノンブロッキングで受信して，それらのノードに渡す．

受信データは `float` 型で長さ 12 の配列（以下，buff[12] とする）である必要があり，受信したフレームではパラメータを更新し，受信しなかったフレームでは前回のパラメータを保持する．

buff[12] は以下のようにデコードされて次段ノードに送信される．

- **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]

本ノードはクライアントプログラムの再接続に対応している.
以下，クライアントプログラムの例を示す (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

ノードの概要

SaveMapFrames で作成されたファイルのデータを Map <int , Vector<float> > 型または Map <int , Vector<complex<float> > > 型のコンテナに読み込む．

必要なファイル

“ 内部ファイル ”と呼ばれるテキスト形式（TEXT）またはバイナリ形式（RAW）のデータファイルと，フレーム毎のデータファイルのリストを含む“ メインファイル ”と呼ばれるテキストファイルが必要．これらすべては SaveMapFrames によって作成されたファイルである．

使用方法

どんなときに使うのか

このノードは，あらかじめ SaveMapFrames によってファイルに保存されたフレームのデータを使用し，後にデータを wav ファイルに変換するなどの処理に用いる．

典型的な接続例

図 6.102 に，ネットワークで LoadMapFrames を使用した接続例を示す．LoadMapFrames が読み込むファイルは，FILENAME パラメータの値によって指定される．このネットワークでは，Synthesize は LoadMapFrames の出力である Map <int , Vector<complex<float> > > 型のデータを時間領域の波形に変換する．次に，SaveWavePCM は，Synthesize によって出力されたデータをサウンドファイルとして保存する．このネットワークでの LoadSourceLocation はオプションで，SaveWavePCM がオプションで Vector<ObjectRef> 型の音源定位結果を取ることができる例を示している．ObjectRef はソースタイプのデータを参照する必要があることに注意．LoadSourceLocation は SaveSourceLocation によって生成されたファイルからデータを読み込むが，LoadMapFrames は SaveMapFrames によって生成されたファイルからデータを読み込む．

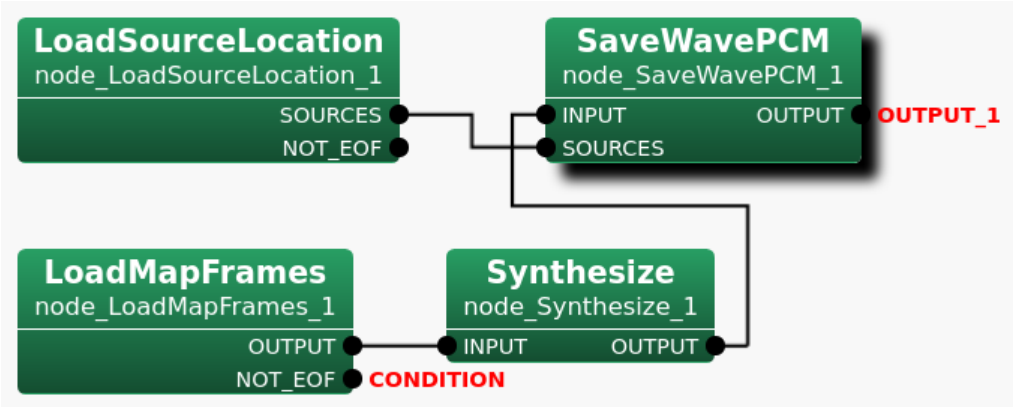


図 6.102: LoadMapFrames の接続例

ノードの入出力とプロパティ

入力

無し.

出力

OUTPUT : Map<int, ObjectRef> 型の Map <int , Vector<float> > または Map <int , Vector<complex<float> > > データがマップのキーにマップされるフレームのデータ.

NOT_EOF : bool 型. プログラムが各フレームの“ 内部ファイル ”のリストを含む“ メインファイル ”を読み込んでいるときにファイルの終わりに達したかどうかを示すフラグ. プログラムが“ メインファイル ”のファイルの最後に到達すると false に設定され, その後プログラムは終了する. それ以外は true に設定される.

パラメータ

表 6.88: LoadMapFrames のパラメータリスト

パラメータ名	型	デフォルト値	単位	説明
FILENAME	string			各フレームのデータファイルのリストを含むテキストファイルの名前.
INPUT_TYPE	string	TEXT_float		入力データファイルのファイル形式とデータ型. TEXT_floa, RAW_float, TEXT_complex_float, または RAW_complex_float を選択.
COL_SIZE	int	512		データファイル内のデータを復元する Map <int , Vector<ObjectRef> > 型の Vector<ObjectRef> の列サイズ.

FILENAME : string 型. SaveMapFrames によって作成された各フレームのデータファイルのリストを含む“ メインファイル ”の名前. SaveMapFrames によって生成されるファイルの詳細については, ノードのリファレンスを参照.

INPUT_TYPE : string 型. 入力データファイルのファイル形式とデータ型. ドロップダウンメニューから適切なものを選択. 次の 4 つのオプションがある.

- TEXT_float
- RAW_float
- TEXT_complex_float
- RAW_complex_float

デフォルト値は TEXT_float. 間違った値を選択すると, ネットワークは正常に動作しない. 図 6.103 は,“ メインファイル ”と“ 内部ファイル ”の両方のサンプルファイル名を示している.“ 内部ファイル ”の各ファイル名は, データ型, 例では“ Vector_float ”を示している. ファイル拡張子は, ファイル形式を示す. 各ファイルの拡張子は, テキスト（TEXT）は「txt」, バイナリ（RAW）は「raw」である.

COL_SIZE ：int 型．入力データファイルに保存された Map <int , Vector<ObjectRef> > 型のデータの Vector サイズ．図 6.103 に示すように，読み込むファイルの名前で見つけることができる．間違った値を指定すると，ネットワークは正常に動作しない．デフォルト値は 512.

詳細については，SaveMapFrames のノードの詳細を参照.

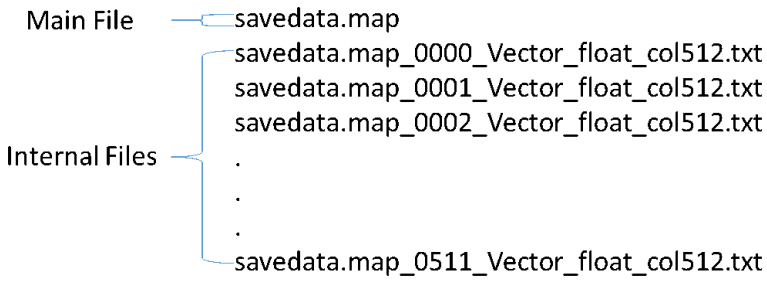


図 6.103: SaveMapFrames で生成されたファイル

SaveMapFrames によって生成されるファイルの詳細については，SaveMapFrames のノードリファレンスを参照.

ノードの詳細

LoadMapFrames は，Map <int , Vector<float> > 型または Map <int , Vector<complex<float> > > 型のいずれかで，SaveMapFrames で生成されたファイルを読み込んでコンテナ内のデータを復元する．LoadMapFrames は 2 種類のファイルを読み込む．1 つは，各キーに分離されたフレームのデータを含む“ 内部ファイル ”と呼ばれるデータファイル．もう 1 つは，“ メインファイル ”と呼ばれるテキストファイルで，各フレーム番号に続く“ 内部ファイル ”のリストを含む．LoadMapFrames は，“ 内部ファイル ”に保存されたデータを対応するデータ型のコンテナに適切に復元するために，メインファイルのフレーム番号とファイル名の両方から情報を取得する．表 6.89 に，使用可能なすべての INPUT_TYPE パラメータ値と対応する出力データ型を示す.

表 6.89: LoadMapFrames の出力データ型

INPUT_TYPE	出力データ型
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> > >

LoadMapFrames は，フレーム番号の後にファイル名がない場合，空の Map を出力することに注意．これは，フレームに関連するデータがないことを意味する．“ メインファイル ”ファイルの終わりに達することによって，すべてのデータファイルが読み込まれたことを検出すると，LoadMapFrames は NOT_EOF を false に設定し，ファイルからコンテナへデータを読み込む反復処理を終了できる.

6.7.6 LoadMatrixFrames

ノードの概要

SaveMatrixFrames で作成されたデータファイルをフレーム毎に Matrix<float> 型または Matrix<complex<float> > 型に読み込む．

必要なファイル

SaveMatrixFrames によって作成されたテキスト形式（TEXT）またはバイナリ形式（RAW）のデータファイル．

使用方法

どんなときに使うのか

このノードは，SaveMatrixFrames によってファイルに保存されたフレームのデータを Matrix<float> 型または Matrix<complex<float> > 型に復元し，ファイル内のデータを処理できるようにする．

典型的な接続例

図 6.104 に，ネットワークで LoadMatrixFrames を使用した接続例を示す．このサンプルネットワークは，データファイルに保存されたマトリックスタイプのフレームのデータから最終的に wav ファイルを作成するためのものである．Synthesize は Map <int , Vector<complex<float> > > 型でのみデータを受け入れるので，データ型変換のためには LoadMatrixFrames と Synthesize の間に MatrixToMap が必要である．

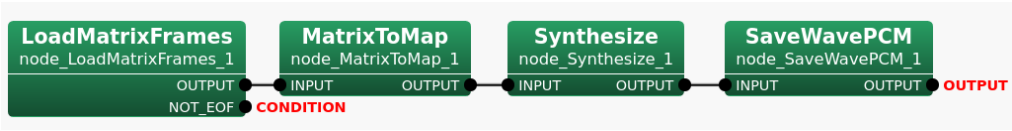


図 6.104: LoadMatrixFrames の接続例

ノードの入出力とプロパティ

入力

無し．

出力

OUTPUT ：Matrix<float> または Matrix<complex<float> > の Matrix<ObjectRef> 型．フレームのデータ．

NOT_EOF ：bool 型．プログラムがデータファイルを読み込んでいるときにファイルの終わりに達したかどうかを示すフラグ．プログラムがデータファイルの最後に到達すると false に設定され，その後プログラムは終了する．それ以外は true に設定される．

パラメータ

FILENAME ：`string` 型．読み込むデータファイルの名前．

INPUT_TYPE ：`string` 型．読み込むデータファイルのファイル形式とデータ型．ドロップダウンメニューから適切なものを選択．次の 4 つのオプションがある．

- TEXT_float
- RAW_float
- TEXT_complex_float
- RAW_complex_float

デフォルト値は TEXT_float．間違った値を選択すると，ネットワークは正常に動作しない．

ROW_SIZE ：`int` 型．データファイル内のデータを復元する `Matrix<ObjectRef>` 型の行サイズ．間違った値を指定すると，ネットワークは正常に動作しない．デフォルト値は 8.

COL_SIZE ：`int` 型．データファイル内のデータを復元する `Matrix<ObjectRef>` 型の列サイズ．間違った値を指定すると，ネットワークは正常に動作しない．デフォルト値は 512.

表 6.90: LoadMatrixFrames のパラメータリスト

パラメータ名	型	デフォルト値	単位	説明
FILENAME	`string`			読み込むデータファイルの名前.
INPUT_TYPE	`string`	TEXT_float		TEXT_float データファイルのファイル形式とデータ型．TEXT_float, RAW_float, TEXT_complex_float, または RAW_complex_float を選択.
ROW_SIZE	`int`	8		データファイル内のデータを復元する `Matrix<ObjectRef>` 型の行サイズ.
COL_SIZE	`int`	512		データファイル内のデータを復元する `Matrix<ObjectRef>` 型の列サイズ.

LoadMatrixFrames のパラメータ値設定を簡単にするには，SaveMatrixFrames によってファイルにデータを保存するときに特別なパターン {tag：format}，つまりフォーマット文字列を使用することを強く推奨する．ファイル名には，データファイル内のデータを復元する `Matrix<ObjectRef>` 型のデータ型，行サイズ，および列サイズが含まれる．

表 6.91 に，FILENAME を除く各パラメータのフォーマット文字列と出力の例を示す．{tag：format} の詳細については，SaveMatrixFrames ノードのリファレンス，特に SaveMatrixFrames のタグリストを参照．

適切なファイル拡張子を付けることは，ファイル形式を見つける時間を節約するのにも役立つ．そうでない場合，TEXT タイプはテキストファイルであり，RAW タイプはバイナリファイルである．Ubuntu プラットフォームでは，RAW ファイルはテキストエディタではアクセスできないが，Windows では開くことができる．いずれにしても，RAW ファイルは TEXT ファイルとは異なり人間が読める形式ではない．

表 6.91: 行サイズが 8 で，列サイズが 512 である Matrix<complex<float> > の書式文字列と出力の例

書式文字列（FILENAME パラメータ値）	出力（書式付きファイル名）
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

ノードの詳細

LoadMatrixFrames は SaveMatrixFrames で生成されたファイルを読み込み，ファイル内のデータをフレーム毎に Matrix<float> 型または Matrix<complex<float> > 型に復元する．フレーム毎に Matrix<ObjectRef> 型にデータを読み込む反復処理は，INPUT_TYPE，ROW_SIZE，および COL_SIZE で指定されたパラメータ値に従って実行される．表 6.92 に，使用可能なすべての INPUT_TYPE パラメータ値と，対応する出力データ型を LoadMatrixFrames に示す．

表 6.92: LoadMatrixFrames の出力データ型

INPUT_TYPE	出力データ型
Text Float	Matrix<float>
Raw Float	Matrix<float>
Text Complex Float	Matrix<complex<float> >
Raw Complex Float	Matrix<complex<float> >

LoadMatrixFrames は，データファイルのファイルの最後に到達すると NOT_EOF を false に設定し，ファイルから Matrix<ObjectRef> にデータを読み込む反復処理を終了できる．

6.7.7 LoadVectorFrames

ノードの概要

SaveVectorFrames で作成されたデータファイルをフレーム毎に Vector<float> 型または Vector<complex<float>> 型に読み込む.

必要なファイル

SaveVectorFrames によって作成されたテキスト形式（TEXT）またはバイナリ形式（RAW）のデータファイル.

使用方法

どんなときに使うのか

このノードは，SaveVectorFrames によってファイルに保存されたフレームのデータを Vector<float> 型または Vector<complex<float> > 型に復元し，ファイル内のデータを処理できるようにする.

典型的な接続例

図 6.105 に，ネットワークで LoadVectorFrames を使用した接続例を示す．このサンプルネットワークは，最後に入力音源の位置を表示するためのものである．LocalizeMUSIC の隠れた出力端子の代わりに，NormalizeMUSIC の MUSIC_SPEC 入力端子に対して，LoadVectorFrames は Vector<float> 型のあらゆる方向の MUSIC スペクトルのパワーを出力する．NormalizeMUSIC は，LocalizeMUSIC で指定されたソースポジションと LoadVectorFrames で提供された MUSIC スペクトラムの両方を使用し，SourceTracker による音源検出を安定させることができる.

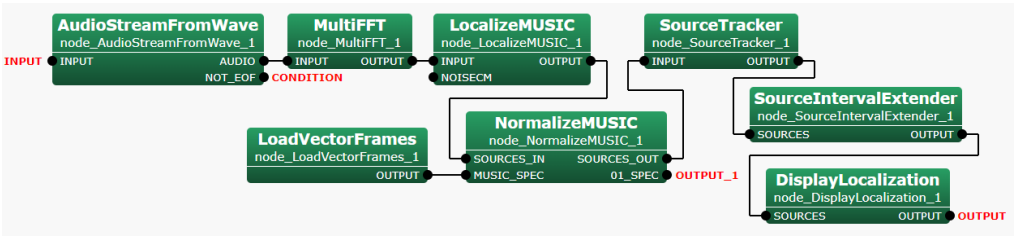


図 6.105: LoadVectorFrames の接続例

ノードの入出力とプロパティ

入力

無し.

出力

OUTPUT : Vector<float> または Vector<ObjectRef> 型の Vector<complex<float> > 型．フレームのデータ．

NOT_EOF : bool 型．プログラムがデータファイルを読み込んでいるときにファイルの終わりに達したかどうかを示すフラグ．プログラムがデータファイルの最後に到達すると false に設定され，その後プログラムは終了する．それ以外は true に設定される．

パラメータ

表 6.93: LoadVectorFrames のパラメータリスト

パラメータ名	型	デフォルト値	単位	説明
FILENAME	string			読み込むデータファイルの名前．
INPUT_TYPE	string	TEXT_float		TEXT_float データファイルのファイル形式とデータ型．TEXT_float, RAW_float, TEXT_complex_float, または RAW_complex_float を選択．
COL_SIZE	int	512		データファイル内のデータを復元する Vector<ObjectRef> 型の列サイズ．

FILENAME : string 型．読み込むデータファイルの名前．

INPUT_TYPE : string 型．読み込むデータファイルのファイル形式とデータ型．ドロップダウンメニューから適切なものを選択．次の 4 つのオプションがある．

- TEXT_float
- RAW_float
- TEXT_complex_float
- RAW_complex_float

デフォルト値は TEXT_float．間違った値を選択すると，ネットワークは正常に動作しない．

COL_SIZE : int 型．データファイル内のデータを復元する Vector<ObjectRef> 型の列サイズ．間違った値を指定すると，ネットワークは正常に動作しない．デフォルト値は 512．

LoadVectorFrames のパラメータ値設定を簡単にするには，SaveVectorFrames によってファイルにデータを保存するときに特別なパターン {tag：format}，つまりフォーマット文字列を使用することを強く推奨する．ファイル名には，データタイプと，データファイル内のデータを復元する Vector<ObjectRef> タイプの列サイズが含まれる．

表 6.94 に，FILENAME を除く各パラメータのフォーマット文字列と出力の例を示す．{tag：format} の詳細については，SaveVectorFrames ノードのリファレンス，特に SaveVectorFrames のタグリストを参照．

適切なファイル拡張子を付けることは，ファイル形式を見つける時間を節約するのにも役立つ．そうでない場合，TEXT タイプはテキストファイルであり，RAW タイプはバイナリファイルである．Ubuntu プラットフォームでは，RAW ファイルはテキストエディタではアクセスできないが，Windows では開くことができる．いずれにしても，RAW ファイルは TEXT ファイルとは異なり人間が読める形式ではない．

表 6.94: 列サイズが 20 の Vector<complex<float> > の書式文字列と出力の例

書式文字列（FILENAME パラメータ値）	出力（書式付きファイル名）
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

ノードの詳細

LoadVectorFrames は，SaveVectorFrames で生成されたファイルを読み取り，ファイル内のデータをフレーム毎に Vector<float> 型または Vector<complex<float> > 型に復元する．フレーム毎に Vector<ObjectRef> 型にデータを読み込む反復処理は，INPUT_TYPE および COL_SIZE で指定されたパラメータ値に従って実行される．表 6.95 に，使用可能なすべての INPUT_TYPE パラメータ値と，LoadVectorFrames の対応する出力データ型を示す．

表 6.95: LoadVectorFrames の出力データ型

INPUT_TYPE	出力データ型
Text Float	Vector<float>
Raw Float	Vector<float>
Text Complex Float	Vector<complex<float> >
Raw Complex Float	Vector<complex<float> >

LoadVectorFrames は，データファイルのファイルの最後に到達すると NOT_EOF を false に設定し，ファイルから Vector<ObjectRef> にデータを読み込む反復処理を終了できる．

6.7.8 MapIDOffset

ノードの概要

Map<int, ObjectRef> の キーを整数分だけシフトする.

必要なファイル

無し.

使用方法

どんなときに使うのか

Map<int, ObjectRef> の キーを整数分だけシフトする. 入力の ObjectRef の型は, Vector<float> , Vector<complex<float> > , Matrix<float> , Matrix<complex<float> > .

ノードの入出力とプロパティ

入力

INPUT : Map<int, ObjectRef> 型の, Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型 または , Map< int , Matrix<float> > または , Map< int , Matrix<complex<float> > > 型.

出力

OUTPUT : Map<int, ObjectRef> 型の, Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型 または , Map< int , Matrix<float> > または , Map< int , Matrix<complex<float> > > 型.

パラメータ

表 6.96: MapIDOffset パラメータ表

パラメータ名	型	デフォルト値	単位	説明
ID_OFFSET	int	0		キーのシフト値.
DEBUG	bool	false		変換状況を出力するかどうかの選択.

ID_OFFSET : int 型. キーのシフト値を指定する. マイナスの値を指定することも可能. デフォルトは 0.

DEBUG : bool 型. true が与えられると, 変換状況が標準出力に出力される. デフォルトは false.

6.7.9 MapMatrixValueOverwrite

ノードの概要

Map<int, ObjectRef> 型の ObjectRef が Matrix<ObjectRef> である時，その一部分の要素を指定した値で置き換える．

必要なファイル

無し．

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Matrix<ObjectRef> である時，その一部分の要素を指定した値で置き換える．値は， INPUT の Map<int , Matrix<ObjectRef> > の ObjectRef に応じて型変換される．

ノードの入出力とプロパティ

入力

INPUT ：Map<int, ObjectRef> 型の Map< int , Matrix < int > > または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型.

出力

OUTPUT ：Map<int, ObjectRef> 型の Map< int , Matrix < int > > または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型.

パラメータ

OVERWRITTEN_ROW_MIN ：int 型．INPUT の Map<int, ObjectRef> の Matrix<ObjectRef> において置き換える要素の開始行インデクス．デフォルトは 0.

OVERWRITTEN_ROW_MAX ：int 型．INPUT の Map<int, ObjectRef> の Matrix<ObjectRef> において置き換える要素の終了行インデクス．デフォルトは 0.

OVERWRITTEN_COL_MIN ：int 型．INPUT の Map<int, ObjectRef> の Matrix<ObjectRef> において置き換える要素の開始列インデクス．デフォルトは 0.

OVERWRITTEN_COL_MAX ：int 型．INPUT の Map<int, ObjectRef> の Matrix<ObjectRef> において置き換える要素の終了列インデクス．デフォルトは 0.

OVERWRITE_VALUE_REAL ：float 型．置き換える値を指定する．INPUT が， Map< int , Matrix < int > > の場合は int に型変換され， Map< int , Matrix<complex<float> > > の場合は置き換える複素数の実部となる．デフォルトは 0.

OVERWRITE_VALUE_IMAG ：float 型．置き換える複素数の虚部の値を指定する．INPUT が Map< int , Matrix<complex<float> > > の場合のみ有効．デフォルトは 0.

DEBUG ：bool 型．true が与えられると, 置換状況が標準出力に出力される．デフォルトは false.

表 6.97: MapMatrixValueOverwrite パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OVERWRITTEN_ROW_MIN	int	0		置き換えられる Matrix 要素の開始行インデクス
OVERWRITTEN_ROW_MAX	int	0		置き換えられる Matrix 要素の終了行インデクス
OVERWRITTEN_COL_MIN	int	0		置き換えられる Matrix 要素の開始列インデクス
OVERWRITTEN_COL_MAX	int	0		置き換えられる Matrix 要素の終了列インデクス
OVERWRITE_VALUE_REAL	float	0		置き換える値． INPUT が， Map< int , Matrix < int > > の場合は int に型変換され， Map< int , Matrix<complex<float> > > の場合は複素数の実部となる
OVERWRITE_VALUE_IMAG	float	0		置き換える複素数の虚部の値． INPUT が Map< int , Matrix<complex<float> > > の場合のみ有効
DEBUG	bool	false		置換状況を出力するかどうかの選択

ノードの詳細

＜例＞

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

ノードの概要

二つの Map<int, ObjectRef> で同じキーを持つ ObjectRef の演算を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

二つの Map<int, ObjectRef> で同じキーを持つ ObjectRef の演算を行う．ObjectRef は Float , Complex , Vector<float> , Vector<complex<float> > , Matrix<float> , Matrix<complex<float> > 型．演算は，加算，減算，乗算，除算，一致しているか，最大値を選択，最小値を選択，連結のいずれかである．二つの入力の Map<int, ObjectRef> の ObjectRef は同じ型，同じサイズでなくてはならない．

ノードの入出力とプロパティ

入力

INPUT1 ：Map<int, ObjectRef> 型の，Map< int , Float > または Map< int , Complex > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > を持つ 型 または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型．

INPUT2 ：Map<int, ObjectRef> 型の，Map< int , Float > または Map< int , Complex > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型 または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型．

出力

OUTPUT ：Map<int, ObjectRef> 型の，Map< int , Float > または Map< int , Complex > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型 または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型 または Map< int , bool > 型．

パラメータ

OPERATION_TYPE ：string 型．二つの Map<int, ObjectRef> の ObjectRef の演算方法．Add, Sub, Mul, Div, Equal, Max, Min, Concate から選択．Add が指定されると加算，Sub が指定されると減算，Mul が指定されると乗算，Div が指定されると除算，Equal が指定されると一致してるか，Max が指定されるとより大きい値を選択，Min が指定されるとより小さい値を選択，Concate が指定されると二つの ObjectRef の連結をする．デフォルトは Add.

DEBUG ：bool 型．true が与えられると，操作状況が標準出力に出力される．デフォルトは false.

ノードの詳細

表 6.98: MapOperator パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OPERATION_TYPE	string	Add		ObjectRef の操作方法． Add, Sub, Mul, Div, Equal, Max, Min, Concate から選択する． 順に，加算，減算，乗算，除算，一致しているか，最大値を選択，最小値を選択，連結を示す．
DEBUG	bool	false		操作状況を出力するかどうかの選択．

表 6.99: 演算と入出力の型

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> > >	Concate	same as input type, but size is different

6.7.11 MapSelectorBySource

ノードの概要

複数の音源分離結果のうち，Source に含まれる情報（ID，パワー，方向）に基づきパラメータで指定した条件を満たすものだけを出力させたいときに用いる．複数音源入力時に音源の ID，パワーに対してその最小，最大にもとづいて出力したり，値の範囲を指定して出力することができる．また音源の方向に関しては，複数音源のうち指定した角度方向に最も近い音源を出力したり，方向の範囲を角度で指定し，その範囲内にある音源のみを出力することができる．

必要なファイル

無し．

使用方法

どんなときに使うのか

たとえば，音源分離ノード GHDSS などの出力結果を PlayAudio で再生する場合．このとき， GHDSS ノードの後に Synthesize ， MapSelectorBySource を接続し，Lch，Rch で各々別の方向の音源を試聴したい場合に用いる．このように，Map 型の入力の一部を出力したい場合に用いる．

典型的な接続例

図 6.106 に接続例を示す．図に示すように，このノードは， GHDSS などの音源分離モジュールの後段に接続される．

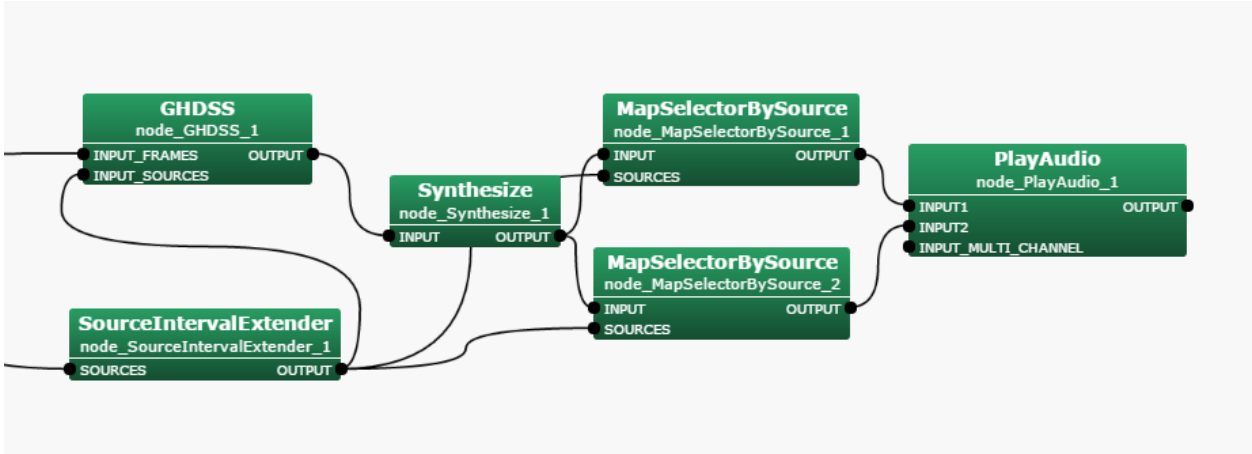


図 6.106: MapSelectorBySource の接続例: これは Iterator サブネットワーク

ノードの入出力とプロパティ

入力

INPUT : `Map<int, ObjectRef>` 型．通常は音源分離ノードの後段に接続されるので，`Map` のキーになる `int` には音源 ID が対応する．`ObjectRef` は分離を表す `Vector<float>` 型 (パワースペクトル) か `Vector<complex<float>>` 型 (複素スペクトル) である．

SOURCES : `Vector<ObjectRef>` 型．ID 付きの音源方向．各 Vector の中身は，ID 付きの音源情報を示す Source 型になっている．このパラメータの指定は必須である．

出力

OUTPUT : `Map<int, ObjectRef>` 型．パラメータで指定した条件を満たしたデータが出力される．

パラメータ

SELECTION_TYPE : `string` 型．音源の選択条件の種類を ALL，ID，POWER，DIRECTION(∗),DIRECTION_AZIMUTH,DIREC から選択する．ALL の場合はすべてのすべての結果を出力する．ID の場合は音源の ID， POWER の場合は音源のパワー， DIRECTION(∗),DIRECTION_AZIMUTH,DIRECTION_ELEVATION の場合は音源の方向，についてそれぞれのパラメータで指定した条件を満たす結果を出力する．

ID_SELECTION_TYPE : `string` 型．SELECTION_TYPE で ID を選択した場合に，その選択条件を LATEST，OLDEST，BETWEEN から選択する．LATEST では最も新しい音源が， OLDEST では最も古い音源が，BETWEEN ではパラメータ ID_RANGE_MIN と ID_RANGE_MAX で指定された範囲の ID の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

ID_RANGE_MIN : `int` 型．ID_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の ID の下限値を指定する

ID_RANGE_MAX : `int` 型．ID_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の ID の上限値を指定する

POWER_SELECTION_TYPE : `string` 型．SELECTION_TYPE で POWER を選択した場合に，その選択条件を HIGHEST，LOWEST，BETWEEN から選択する．HIGHEST では 最もパワーの大きな音源が，LOWEST では最もパワーの小さな音源が，BETWEEN ではパラメータ POWER_RANGE_MIN と POWER_RANGE_MAX で指定された範囲のパワーの音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

POWER_RANGE_MIN : `float` 型．POWER_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源のパワーの下限値を指定する．

POWER_RANGE_MAX : `float` 型．POWER_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源のパワーの上限値を指定する．

表 6.100: MapSelectorBySource のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
SELECTION_TYPE	string	ID		音源の選択条件の種類．ALL,ID,POWER,DIRECTION(∗),DIRECTION_AZI から選択．
ID_SELECTION_TYPE	string	LATEST		音源を ID により選択する場合の選択条件．LATEST，OLDEST，BETWEEN から選択．
ID_RANGE_MIN	int	0		選択する音源の ID の下限値．
ID_RANGE_MAX	int	0		選択する音源の ID の上限値．
POWER_SELECTION_TYPE	string	HIGHEST		音源をパワーにより選択する場合の選択条件．HIGHEST，LOWEST，BETWEEN から選択．
POWER_RANGE_MIN	float	0		選択する音源のパワーの下限値．
POWER_RANGE_MAX	float	40.0		選択する音源のパワーの上限値．
DIRECTION_SELECTION_TYPE(∗)	string	BETWEEN		音源を角度により選択する場合の選択条件．NEAREST，BETWEEN から選択．
DIRECTION(∗)	float	0	[deg]	選択する音源の近傍の角度．
DIRECTION_RANGE_MIN(∗)	float	0	[deg]	選択する音源の角度の下限値．
DIRECTION_RANGE_MAX(∗)	float	360.0	[deg]	選択する音源の角度の上限値．
AZIMUTH_SELECTION_TYPE	string	BETWEEN		音源を方位角により選択する場合の選択条件．NEAREST，BETWEEN から選択．
AZIMUTH	float	0	[deg]	選択する音源の近傍の方位角．
AZIMUTH_RANGE_MIN	float	0	[deg]	選択する音源の方位角の下限値．
AZIMUTH_RANGE_MAX	float	360.0	[deg]	選択する音源の方位角の上限値．
ELEVATION_SELECTION_TYPE	string	BETWEEN		音源を仰角により選択する場合の選択条件．NEAREST，BETWEEN から選択．
ELEVATION	float	0	[deg]	選択する音源の近傍の仰角．
ELEVATION_RANGE_MIN	float	0	[deg]	選択する音源の仰角の下限値．
ELEVATION_RANGE_MAX	float	360.0	[deg]	選択する音源の仰角の上限値．
DEBUG_PRINT	bool	false		エラーメッセージ出力．

DIRECTION_SELECTION_TYPE ：string 型．SELECTION_TYPE で DIRECTION を選択した場合に，その選択条件を NEAREST，BETWEEN から選択する．NEAREST ではパラメータ DIRECTION に指定された角度に最も近い音源が，BETWEEN ではパラメータ DIRECTION_RANGE_MIN と DIRECTION_RANGE_MAX で指定された角度の範囲の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．(∗)

DIRECTION ：float 型．DIRECTION_SELECTION_TYPE で NEAREST を選択した場合に，ここで指定した角度に最も近い音源が出力される．(∗)

DIRECTION_RANGE_MIN ：float 型．DIRECTION_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の角度の下限値を指定する．(∗)

DIRECTION_RANGE_MAX ：float 型．DIRECTION_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の角度の上限値を指定する．(∗)

AZIMUTH_SELECTION_TYPE : `string` 型．SELECTION_TYPE で DIRECTION_AZIMUTH を選択した場合に，その選択条件を NEAREST，BETWEEN から選択する．NEAREST ではパラメータ AZIMUTH に指定された方位角に最も近い音源が，BETWEEN ではパラメータ AZIMUTH_RANGE_MIN と AZIMUTH_RANGE_MAX で指定された方位角の範囲の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

AZIMUTH : `float` 型．AZIMUTH_SELECTION_TYPE で NEAREST を選択した場合に，ここで指定した方位角にもっとも近い音源が出力される．

AZIMUTH_RANGE_MIN : `float` 型．AZIMUTH_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の方位角の下限値を指定する．

AZIMUTH_RANGE_MAX : `float` 型．AZIMUTH_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の方位角の上限値を指定する．

ELEVATION_SELECTION_TYPE : `string` 型．SELECTION_TYPE で DIRECTION_ELEVATION を選択した場合に，その選択条件を NEAREST，BETWEEN から選択する．NEAREST ではパラメータ ELEVATION に指定された仰角に最も近い音源が，BETWEEN ではパラメータ ELEVATION_RANGE_MIN と ELEVATION_RANGE_MAX で指定された仰角の範囲の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

ELEVATION : `float` 型．ELEVATION_SELECTION_TYPE で NEAREST を選択した場合に，ここで指定した仰角にもっとも近い音源が出力される．

ELEVATION_RANGE_MIN : `float` 型．ELEVATION_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の仰角の下限値を指定する．

ELEVATION_RANGE_MAX : `float` 型．ELEVATION_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の仰角の上限値を指定する．

DEBUG_PRINT : `bool` 型．モジュールのエラーメッセージを出力する．

(∗) 互換性のためにあるパラメータ

SELECTION_TYPE の DIRECTION は DIRECTION_AZIMUTH と，DIRECTION は AZIMUTH と，DIRECTION_RANGE_MIN は AZIMUTH_RANGE_MIN と，DIRECTION_RANGE_MAX は AZIMUTH_RANGE_MAX と同じ．

6.7.12 MapToMap

ノードの概要

Map<int, ObjectRef> の ObjectRef の Vector<float> 型と Vector<complex<float> > 型の変換，あるいは，Matrix<float> 型と Matrix<complex<float> > 型の変換を行う．Map<int, ObjectRef> の ObjectRef の型変換を行う．Map< int , Vector<float> > 型と Map< int , Vector<complex<float> > > 型の変換，または，Map< int , Matrix<float> > 型と Map< int , Matrix<complex<float> > > 型の変換.

必要なファイル

無し.

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Vector<ObjectRef> または Matrix<ObjectRef> である時に，Vector<float> 型から Vector<complex<float> > 型へ，Vector<complex<float> > 型から Vector<float> 型へ，または，Matrix<float> 型から Matrix<complex<float> > 型へ，Matrix<complex<float> > 型から Matrix<float> 型へ変換する際に用いる.

ノードの入出力とプロパティ

入力

INPUT : Map<int, ObjectRef> 型の，Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型 または，Map< int , Matrix<float> > または，Map< int , Matrix<complex<float> > > 型.

出力

OUTPUT : Map<int, ObjectRef> 型の，Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型 または，Map< int , Matrix<float> > または，Map< int , Matrix<complex<float> > > 型.

パラメータ

METHOD_COMPLEX_TO_FLOAT : string 型．Vector<complex<float> > 型から Vector<float> 型へ，または，Matrix<complex<float> > 型から Matrix<float> 型への変換方法を指定する．INPUT の複素数の，絶対値が出力される「magnitude」，実部が出力される「real」，虚部が出力される「imaginary」から選択する．デフォルトは magnitude.

METHOD_FLOAT_TO_COMPLEX : string 型．Vector<float> 型から Vector<complex<float> > 型へ，または，Matrix<float> 型から Matrix<complex<float> > 型への変換方法を指定する．複素数の虚部に，0 が出力される「zero」，実部の絶対値が出力される「helbert」から選択する．デフォルトは zero.

DEBUG : bool 型．true が与えられると，変換状況が標準出力に出力される．デフォルトは false.

ノードの詳細

表 6.101: MapToMap パラメータ表

パラメータ名	型	デフォルト値	単位	説明
METHOD_COMPLEX_TO_FLOAT	string	magnitude		Vector<complex<float> > 型から Vector<float> 型へ，または，Matrix<complex<float> > 型から Matrix<float> 型への変換方法．INPUT の複素数の，絶対値が出力される「magnitude」，実部が出力される「real」，虚部が出力される「imaginary」から選択する．
METHOD_FLOAT_TO_COMPLEX	string	zero		Vector<float> 型から Vector<complex<float> > 型へ，または，Matrix<float> 型から Matrix<complex<float> > 型への変換方法．複素数の虚部に，0 が出力される「zero」，実部の絶対値が出力される「helbert」から選択する．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

表 6.102: MapToMap 変換表

INPUT	OUTPUT	使用するパラメータ
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

ノードの概要

Map< int , Matrix<float> > 型から Matrix<float> 型へ， Map< int , Matrix<complex<float> > > 型から Matrix<complex<float> > 型への変換を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Matrix である時に， Map< int , Matrix<float> > 型から Matrix<float> 型へ，または， Map< int , Matrix<complex<float> > > 型から Matrix<complex<float> > 型へ変換する際に用いる．

ノードの入出力とプロパティ

入力

INPUT : Map<int, ObjectRef> 型の，Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型．

出力

OUTPUT : any ．ただし，サポートする型は Matrix<float> または Matrix<complex<float> > 型．

パラメータ

表 6.103: MapToMatrix パラメータ表

パラメータ名	型	デフォルト値	単位	説明
METHOD	string	min_key		Map<int, ObjectRef> から Matrix<ObjectRef> への変換方法．キーが最小または最大の Matrix が出力される min_key, max_key, 合計または平均を算出した Matrix が出力される average, summation から選択する．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

METHOD : string 型．Map<int, ObjectRef> から Matrix<ObjectRef> への変換方法を指定する．入力の Map<int, ObjectRef> の中でキーが最小または最大の ObjectRef の Matrix<float> あるいは Matrix<complex<float> > が出力される「min_key」「max_key」, 入力の Map<int, ObjectRef> の ObjectRef の合計または平均を算出した Matrix<float> または Matrix<complex<float> > が出力される「summation」「average」から選択する．デフォルトは min_key.

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

ノードの詳細

表 6.104: MapToMatrix 変換表

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		

＜例＞

INPUT: キーと 2X2 のマトリクスの値が 3 つ

$$\left\{ 0, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix} \right\}$$

OUTPUT(1): キー 0 の値の 2X2 のマトリクス

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

OUTPUT(2): キー 2 の値の 2X2 のマトリクス

$$\begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix}$$

OUTPUT(3): キー 0 から 2 の値の平均の 2X2 のマトリクス

$$\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

OUTPUT(4): キー 0 から 2 の値の合計の 2X2 のマトリクス

$$\begin{bmatrix} 15 & 18 \\ 21 & 24 \end{bmatrix}$$

6.7.14 MapToVector

ノードの概要

Map< int , Vector<float> > 型から Vector<float> 型へ， Map< int , Vector<complex<float> > > 型から Vector<complex<float> > 型への変換を行う

必要なファイル

無し.

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Vector である時に, Map< int , Vector<float> > 型から Vector<float> 型へ，または， Map< int , Vector<complex<float> > > 型から Vector<complex<float> > 型へ変換する際に用いる.

ノードの入出力とプロパティ

入力

INPUT : Map<int, ObjectRef> 型の，Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型.

出力

OUTPUT : any . ただし，サポートする型は Vector<float> または Vector<complex<float> > 型.

パラメータ

表 6.105: MapToVector パラメータ表

パラメータ名	型	デフォルト値	単位	説明
METHOD	string	min_key		Map<int, ObjectRef> から Vector<ObjectRef> への変換方法．キーが最小または最大の Vector が出力される min_key, max_key, 合計または平均を算出した Vector が出力される summation, average から選択.
DEBUG	bool	false		変換状況を出力するかどうかの選択.

METHOD : string 型．Map<int, ObjectRef> から Vector<ObjectRef> への変換方法を指定する．入力の Map<int, ObjectRef> の中でキーが最小または最大の ObjectRef の Vector<float> あるいは Vector<complex<float> > が出力される「min_key」「max_key」, 入力の Map<int, ObjectRef> の ObjectRef の合計または平均を算出した Vector<float> または Vector<complex<float> > が出力される「summation」「average」から選択する．デフォルトは min_key.

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

ノードの詳細

表 6.106: MapToVector 変換表

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		

＜例＞

INPUT: キーと要素 3 のベクタの値が 3 つ

$$\{0,\ <1\ \ 2\ \ 3>\},\quad \{1,\ <4\ \ 5\ \ 6>\},\quad \{2,\ <7\ \ 8\ \ 9>\}$$

OUTPUT(1): キー 0 の値の要素 3 のベクタ

$$<1\ \ 2\ \ 3>$$

OUTPUT(2): キー 2 の値の要素 3 のベクタ

$$<7\ \ 8\ \ 9>$$

OUTPUT(3): キー 0 から 2 の値の各要素の平均

$$<4\ \ 5\ \ 6>$$

OUTPUT(4): キー 0 から 2 の値の各要素の合計

$$<12\ \ 15\ \ 18>$$

6.7.15 MapVectorValueOverwrite

ノードの概要

Map<int, ObjectRef> 型の ObjectRef が Vector<ObjectRef> である時，その一部分の要素を指定した値で置き換える．

必要なファイル

無し．

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Vector<ObjectRef> である時，その一部分の要素を指定した値で置き換える． 値は， INPUT の Map<int , Vector<ObjectRef> > の ObjectRef に応じて型変換される．

ノードの入出力とプロパティ

入力

INPUT ：Map<int, ObjectRef> 型の Map< int , Vector<int> > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型．

出力

OUTPUT ：Map<int, ObjectRef> 型の Map< int , Vector<int> > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型．

パラメータ

OVERWRITTEN_MIN ：int 型．INPUT の Map<int , Vector<ObjectRef> > の Vector<ObjectRef> において置き換える要素の開始インデクス．デフォルトは 0.

OVERWRITTEN_MAX ：int 型．INPUT の Map<int , Vector<ObjectRef> > の Vector<ObjectRef> において置き換える要素の終了インデクス．デフォルトは 0.

OVERWRITE_VALUE_REAL ：float 型．置き換える値を指定する．INPUT が， Map< int , Vector<int> > の場合は int に型変換され， Map< int , Vector<complex<float> > > の場合は置き換える複素数の実部となる．デフォルトは 0.

OVERWRITE_VALUE_IMAG ：float 型．置き換える複素数の虚部の値を指定する．INPUT が Map< int , Vector<complex<float> > > の場合のみ有効．デフォルトは 0.

DEBUG ：bool 型．true が与えられると, 置換状況が標準出力に出力される．デフォルトは false.

表 6.107: MapVectorValueOverwrite パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OVERWRITTEN_MIN	int	0		置き換えられる Vector 要素の開始インデクス.
OVERWRITTEN_MAX	int	0		置き換えられる Vector 要素の終了インデクス.
OVERWRITE_VALUE_REAL	float	0		置き換える値. INPUT が, Map< int , Vector<int> > の場合は int に型変換され, Map< int , Vector<complex<float> > > の場合は複素数の実部となる.
OVERWRITE_VALUE_IMAG	float	0		置き換える複素数の虚部の値. INPUT が Map< int , Vector<complex<float> > > の場合のみ有効.
DEBUG	bool	false		置換状況を出力するかどうかの選択.

ノードの詳細

＜例＞

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

ノードの概要

Matrix<float> または Matrix<complex<float> > 型と Map<int, ObjectRef> 型の変換を行う

必要なファイル

無し.

使用方法

どんなときに使うのか

Matrix<float> または Matrix<complex<float> > 型 を Map<int, ObjectRef> 型に変換する際に用いる．Matrix<float> 型から Map<int , Matrix<float> > 型へ, または, Matrix<complex<float> > 型から Map<int , Matrix<complex<float> > > 型へ変換される．入力が Map<int, ObjectRef> 型しか受け付けないノード，例えば PreEmphasis ， MelFilterBank や SaveRawPCM などに接続する際に使用する．

典型的な接続例

MatrixToMap ノードの接続例を図 6.107，6.108 に示す.

図 6.107 は，AudioStreamFromMic ノードでマイクロホンから音声波形データを取り込み，ChannelSelector ノードにて必要なチャネルを選別し，MatrixToMap ノードによって Matrix<float> 型データを Map<int, ObjectRef> 型に変換する．その出力を SaveRawPCM ノードに接続し，音声波形データをファイルとして保存する.

図 6.108 は，波形のスペクトルを Map<int, ObjectRef> 型で得たいときの MatrixToMap ノードの使い方である．図のように，MultiFFT ノードを接続するのは，MatrixToMap の前でも後でも良い.

ノードの入出力とプロパティ

入力

INPUT ：any ．ただし，サポートする型は Matrix<float> または Matrix<complex<float> > 型.

出力

OUTPUT ：Map<int, ObjectRef> 型の，Map<int , Matrix<float> > または Map<int , Matrix<complex<float> > > 型 または Map<int , Vector<float> > または Map<int , Vector<complex<float> > > 型.

パラメータ

OUTPUT_TYPE ：string 型．出力される型を指定する．キーに 0， 値 ObjectRef に入力の Matrix が出力される「map_of_matrix」，キーに入力の Matrix の行インデクス，値に入力の Matrix の行成分が抽出される「map_of_row_vectors」，キーに入力の Matrix の列インデクス，値に入力の Matrix の列成分が抽出される「map_of_column_vectors」から選択する．デフォルトは map_of_row_vectors .

DEBUG ：bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

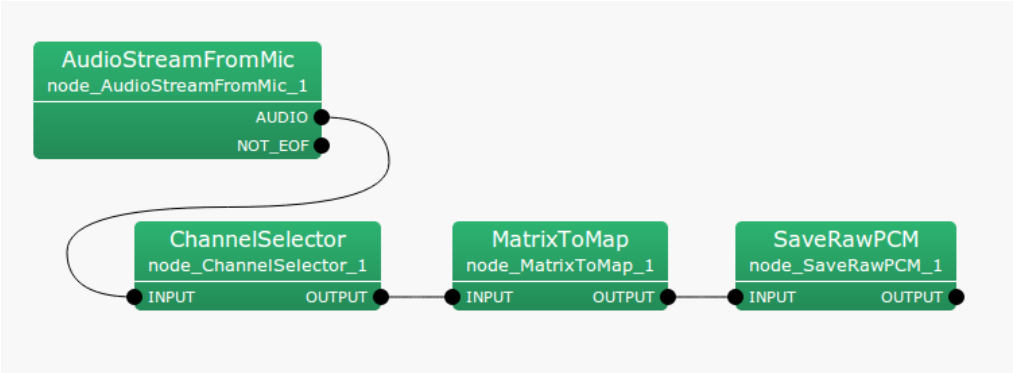


図 6.107: MatrixToMap の接続例 – SaveRawPCM への接続

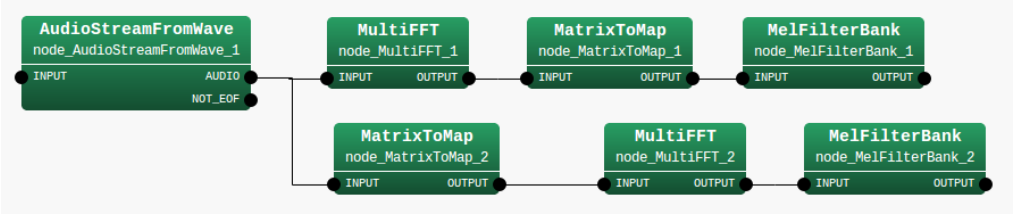


図 6.108: MatrixToMap の接続例 – MultiFFT との接続

表 6.108: MatrixToMap パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OUTPUT_TYPE	string	map_of_row_vectors		出力される型を指定．キーに 0, 値 ObjectRef に入力の Matrix が出力される「map_of_matrix」，キーに入力の Matrix の行インデクス，値に入力の Matrix の行成分が抽出される「map_of_row_vectors」，キーに入力の Matrix の列インデクス，値に入力の Matrix の列成分が抽出される「map_of_column_vectors」から選択．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

ノードの詳細

表 6.109: MatrixToMap 変換表

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}	

＜例＞

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

ノードの概要

Matrix<float> 型と Matrix<complex<float> > 型の変換を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

Matrix<float> 型から Matrix<complex<float> > 型へ，Matrix<complex<float> > 型から Matrix<float> 型へ変換する際に用いる．

ノードの入出力とプロパティ

入力

INPUT ：any ．ただし，サポートする型は Matrix<float> または Matrix<complex<float> > 型．

出力

OUTPUT ：any ．ただし，サポートする型は Matrix<float> または Matrix<complex<float> > 型．

パラメータ

表 6.110: MatrixToMatrix パラメータ表

パラメータ名	型	デフォルト値	単位	説明
METHOD_COMPLEX_TO_FLOAT	string	magnitude		Matrix<complex<float> > から Matrix<float> への変換方法．INPUT の複素数の，絶対値が出力される「magnitude」，実部が出力される「real」，虚部が出力される「imaginary」から選択する．
METHOD_FLOAT_TO_COMPLEX	string	zero		Matrix<float> から Matrix<complex<float> > への変換方法．複素数の虚部に，0 が出力される「zero」，実部の絶対値が出力される「helbert」から選択する．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

METHOD_COMPLEX_TO_FLOAT ：string 型．Matrix<complex<float> > 型から Matrix<float> 型への変換方法を指定する．INPUT の複素数の，絶対値が出力される「magnitude」，実部が出力される「real」，虚部が出力される「imaginary」から選択する．デフォルトは magnitude．

METHOD_FLOAT_TO_COMPLEX : string 型．Matrix<float> 型から Matrix<complex<float> > 型への変換方法を指定する．複素数の虚部に， 0 が出力される「zero」，実部の絶対値が出力される「helbert」から選択する．デフォルトは zero.

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

ノードの詳細

表 6.111: MatrixToMatrix 変換表

INPUT	OUTPUT	使用するパラメータ
Matrix<float>	Matrix<complex<float> >	METHOD_FLOAT_TO_COMPLEX
Matrix<complex<float> >	Matrix<float>	METHOD_COMPLEX_TO_FLOAT

6.7.18 MatrixToVector

ノードの概要

Matrix<float> 型から Vector<float> 型へ，または，Matrix<complex<float> > 型から Vector<complex<float> > 型へ変換を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

Matrix<float> 型から Vector<float> 型へ，または，Matrix<complex<float> > 型から Vector<complex<float> > 型へ変換する際に用いる．

ノードの入出力とプロパティ

入力

INPUT ：any ．ただし，サポートする型は Matrix<float> または Matrix<complex<float> > 型.

出力

OUTPUT ：any ．ただし，サポートする型は Vector<float> または Vector<complex<float> > 型.

パラメータ

METHOD ：string 型．Matrix<float> 型から Vector<float> 型へ，または，Matrix<complex<float> > 型から Vector<complex<float> > 型への変換方法を指定する．Matrix 要素の行順または列順で Vector 要素を形成する「reshape」，Matrix 要素の各行または各列の合計値あるいは平均値で Vector 要素を形成する「accumulate」から選択する．デフォルトは reshape.

RESHAPE_ORDER ：string 型．パラメータ METHOD で reshape が指定されている場合に, その方法を指定する．Matrix 要素の行順で Vector 要素を形成する「row」, Matrix 要素の列順で Vector 要素を形成する「column」から選択する．デフォルトは row.

ACCUMULATE_METHOD ：string 型．パラメータ METHOD で accumulate が指定されている場合に, その方法を指定する．Matrix 要素の, 各行の要素を合計する「row_sum」, 各行の要素を平均する「row_avg」, 各列の要素を合計する「column_sum」, 各列の要素を平均する「column_avg」から選択する．デフォルトは row_sum.

DEBUG ：bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

表 6.112: MatrixToVector パラメータ表

パラメータ名	型	デフォルト値	単位	説明
METHOD	string	reshape		Matrix から Vector への変換方法. Matrix 要素の行順または列順で Vector 要素を形成する「reshape」, Matrix 要素の各行または各列の合計値あるいは平均値で Vector 要素を形成する「accumulate」から選択.
RESHAPE_ORDER	string	row		reshape の方法. Matrix 要素の行順で Vector 要素を形成する「row」, Matrix 要素の列順で Vector 要素を形成する「column」から選択.
ACCUMULATE_METHOD	string	row_sum		accumulate の方法. Matrix 要素の, 各行の要素を合計する「row_sum」, 各行の要素を平均する「row_avg」, 各列の要素を合計する「column_sum」, 各列の要素を平均する「column_avg」から選択.
DEBUG	bool	false		変換状況を出力するかどうかの選択.

表 6.113: MatrixToVector 変換表

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)
				col_avg			(6)
Matrix<complex<float> >		reshape	row	-	Vector<complex<float> >	(NxM)	
			column				
		accumulate	-	row_sum		N	
				row_avg			
				col_sum		M	
				col_avg			

ノードの詳細

＜例＞

INPUT:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

OUTPUT(1):

$$< 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 >$$

OUTPUT(2):

$$< 1 \quad 3 \quad 5 \quad 2 \quad 4 \quad 6 >$$

OUTPUT(3):

$$< 3 \quad 7 \quad 11 > \longleftarrow < 1+2 \quad 3+4 \quad 5+6 >$$

OUTPUT(4):

$$< 1.5 \quad 3.5 \quad 5.5 > \longleftarrow < (1+2)/2 \quad (3+4)/2 \quad (5+6)/2 >$$

OUTPUT(5):

$$< 9 \quad 12 > \longleftarrow < 1+3+5 \quad 2+4+6 >$$

OUTPUT(6):

$$< 3 \quad 4 > \longleftarrow < (1+3+5)/3 \quad (2+4+6)/3 >$$

6.7.19 MatrixValueOverwrite

ノードの概要

Matrix<ObjectRef> の一部分の要素を指定した値で置き換える．

必要なファイル

無し．

使用方法

どんなときに使うのか

Matrix<ObjectRef> 型の Matrix < int > または Matrix<float> または Matrix<complex<float> > の一部分の要素の値を，指定した値で置き換える．指定した値は Matrix<ObjectRef> の ObjectRef の型に応じて変換される．

ノードの入出力とプロパティ

入力

INPUT : any．ただし，サポートする型は Matrix < int > または Matrix<float> または Matrix<complex<float> > 型．

出力

OUTPUT : any．ただし，サポートする型は Matrix < int > または Matrix<float> または Matrix<complex<float> > 型．

パラメータ

OVERWRITTEN_ROW_MIN : int 型．置き換える Matrix<ObjectRef> 要素の開始行インデクスを指定する．デフォルトは 0.

OVERWRITTEN_ROW_MAX : int 型．置き換える Matrix<ObjectRef> 要素の終了行インデクスを指定する．デフォルトは 0.

OVERWRITTEN_COL_MIN : int 型．置き換える Matrix<ObjectRef> 要素の開始列インデクスを指定する．デフォルトは 0.

OVERWRITTEN_COL_MAX : int 型．置き換える Matrix<ObjectRef> 要素の終了列インデクスを指定する．デフォルトは 0.

OVERWRITE_VALUE_REAL : float 型．置き換える値を指定する．INPUT が， Matrix < int > の場合は int に型変換され， Matrix<complex<float> > の場合は置き換える複素数の実部となる．デフォルトは 0.

OVERWRITE_VALUE_IMAG : float 型．置き換える複素数の虚部の値を指定する．INPUT が Matrix<complex<float> > の場合のみ有効．デフォルトは 0.

DEBUG : bool 型．true が与えられると，置換状況が標準出力に出力される．デフォルトは false.

表 6.114: MatrixValueOverwrite パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OVERWRITTEN_ROW_MIN	int	0		置き換えられる Matrix 要素の開始行インデクス.
OVERWRITTEN_ROW_MAX	int	0		置き換えられる Matrix 要素の終了行インデクス.
OVERWRITTEN_COL_MIN	int	0		置き換えられる Matrix 要素の開始列インデクス.
OVERWRITTEN_COL_MAX	int	0		置き換えられる Matrix 要素の終了列インデクス.
OVERWRITE_VALUE_REAL	float	0		置き換える値. INPUT が, Matrix < int > の場合は int に型変換され, Matrix<complex<float> > の場合は置き換える複素数の実部となる.
OVERWRITE_VALUE_IMAG	float	0		置き換える複素数の虚部の値. INPUT が Matrix<complex<float> > の場合のみ有効.
e DEBUG	bool	false		置換状況を出力するかどうかの選択.

ノードの詳細

＜例＞

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

ノードの概要

入力信号をダウンサンプリングして出力する．ローパスフィルタは窓関数法を用いており，窓関数はカイザー窓である．

必要なファイル

無い．

使用方法

どんなときに使うのか 入力信号のサンプリング周波数が 16kHz でない場合等に用いる．HARK ノードは基本的にサンプリング周波数を 16kHz と仮定している．そのため，入力信号が 48kHz であったりする場合には，ダウンサンプリングを行ない，サンプリング周波数を 16kHz まで下げる必要がある．

注意 1（ADVANCE の値域）： 処理の都合上，前段の入力ノード，例えば，AudioStreamFromMic や AudioStreamFromWave のパラメータ設定に制限を設ける．それらのパラメータ，LENGTH と ADVANCE の差: OVERLAP = LENGTH - ADVANCE，は十分大きな値でなければならない．より具体的には，このノードのローパスフィルタ長 N より大きな値でなけらばならない．このノードのデフォルトの設定では，おおよそ 120 以上あれば十分であるので，ADVANCE が LENGTH の 4 分の 1 以上なら問題は起きないであろう．また，次の**注意 2** の要求も満す必要がある．

注意 2（ADVANCE 値の設定）： このノードの ADVANCE は，後段ノード（GHDSS など）における ADVANCE 値の SAMPLING_RATE_IN / SAMPLING_RATE_OUT 倍に設定する必要がある．これは仕様であり，これ以外の値に設定したときの動作は保証しない．例えば，後段ノードで ADVANCE = 160 に設定されている場合かつ SAMPLING_RATE_IN / SAMPLING_RATE_OUT = 3 である場合，このノードや前段ノードの ADVANCE は 480 に設定する必要がある．

注意 3（前段ノードでの LENGTH 値の要求）： このノード以前（AudioStreamFromMic など）における LENGTH 値も，後段ノード（GHDSS など）での値の SAMPLING_RATE_IN / SAMPLING_RATE_OUT 倍に設定しておくことを要求する．例えば，SAMPLING_RATE_IN / SAMPLING_RATE_OUT = 3 なら，GHDSS で LENGTH = 512，ADVANCE = 160 に設定されているなら，AudioStreamFromMic では LENGTH = 1536，ADVANCE = 480 に設定するのが望ましい．

典型的な接続例 下記に典型的な接続例を示す．このネットワークファイルは，Wave ファイル入力を読み込み，ダウンサンプルを行ない，Raw ファイルで保存を行なう．Wave ファイル入力は Constant，InputStream，AudioStreamFromMic，を繋ぐことで実現される．その後，MultiDownSampler によって，ダウンサンプリングを実行し，SaveRawPCM で出力波形を保存する．

ノードの入出力とプロパティ

入力

INPUT ：`Matrix<float>` 型. マルチチャネル音声波形データ（時間領域波形）.

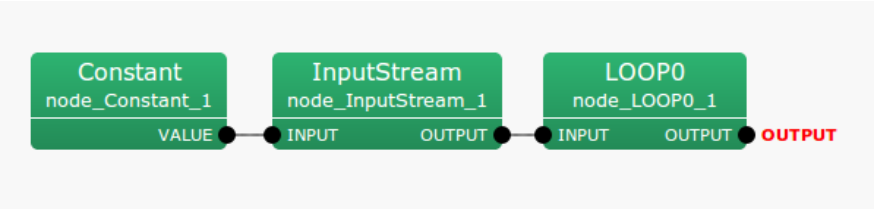


図 6.109: MultiDownSampler の接続例: Main ネットワーク

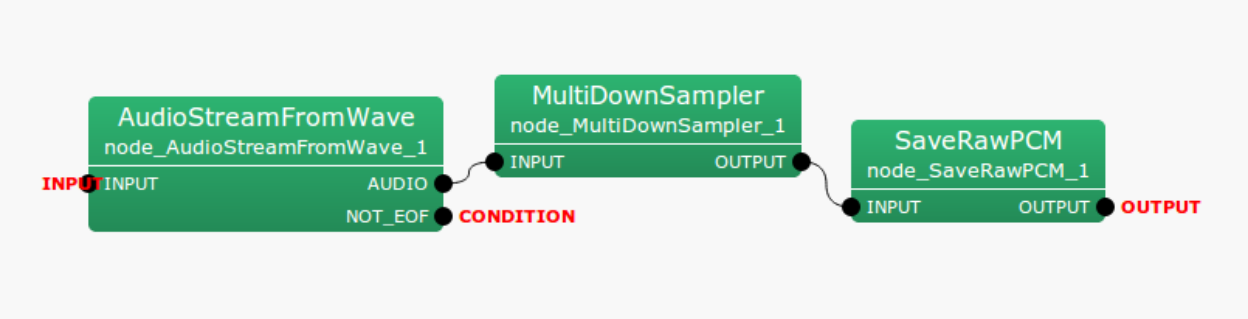


図 6.110: MultiDownSampler の接続例: Iteration(LOOP0) ネットワーク

出力

OUTPUT : `Matrix<float>` 型．ダウンサンプルされたマルチチャネル音声波形データ（時間領域波形）．

表 6.115: MultiDownSampler のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
ADVANCE	`int`	480	[pt]	INPUT 信号でのイタレーション毎にフレームをシフトさせる長さ．特殊な設定が必要であるので，パラメータ説明を参考にすること．
SAMPLING_RATE_IN	`int`	48000	[Hz]	INPUT 信号のサンプリング周波数．
SAMPLING_RATE_OUT	`int`	16000	[Hz]	OUTPUT 信号のサンプリング周波数．
Wp	`float`	0.28	$[\frac{\omega}{2\pi}]$	ローパスフィルタ通過域端．INPUT を基準とした正規化周波数 [0.0 – 1.0] の値で指定．
Ws	`float`	0.34	$[\frac{\omega}{2\pi}]$	ローパスフィルタ阻止域端．INPUT を基準とした正規化周波数 [0.0 – 1.0] の値で指定．
As	`float`	50	[dB]	阻止域最小減衰量．

パラメータ 各パラメータはローパスフィルタ，ここではカイザー窓の周波数特性を定めるものが多い．図 6.111 に記号とフィルタ特性の関係を示すので，対応関係に注意して読み進めること．

ADVANCE : `int` 型．480 がデフォルト値．音声波形に対する処理のフレームを，波形の上でシフトする幅をサンプル数で指定する．ただし，INPUT 以前のノードで設定されている値を用いる．**注意:** OUTPUT 以降で設定されている値の SAMPLING_RATE_IN / SAMPLING_RATE_OUT 倍の値に設定する必要がある．

SAMPLING_RATE_IN : `int` 型．48000 がデフォルト値．入力波形のサンプリング周波数を指定する．

SAMPLING_RATE_OUT : `int` 型．16000 がデフォルト値．出力波形のサンプリング周波数を指定する．この時，SAMPLING_RATE_IN の整数分の一の値しか対応できないことに注意が必要．

Wp : `float` 型．デフォルト値は 0.28．ローパスフィルタ通過域端周波数を INPUT を基準とした正規化周波数 [0.0 – 1.0] の値によって指定する．入力のサンプリング周波数が 48000 [Hz] で，0.28 の値に設定した場合，約 48000 ∗ 0.28 = 13440 [Hz] から，ローパスフィルタのゲインが減少しはじめる．

Ws : `float` 型．デフォルト値は 0.34．ローパスフィルタ阻止域端周波数を INPUT を基準とした正規化周波数 [0.0 – 1.0] の値によって指定する．入力のサンプリング周波数が 48000 [Hz] で，0.34 の値に設定した場合，約 48000 ∗ 0.34 = 16320 [Hz] から，ローパスフィルタのゲインが安定しはじめる．

As : `float` 型．デフォルト値は 50．阻止域最小減衰量を [dB] で表現した値．デフォルト値を用いた場合，阻止帯域のゲインは通過帯域を 0 とした場合，約 -50 [dB] となる．

ここで，Wp，Ws，As の値をシビアに設定，例えば，Wp，Ws を遮断周波数 Ws 近くに設定するとカイザー窓の周波数特性精度が向上する．しかし，ローパスフィルタの次元が増大し，処理時間の増大を招く．この関係はトレードオフである．

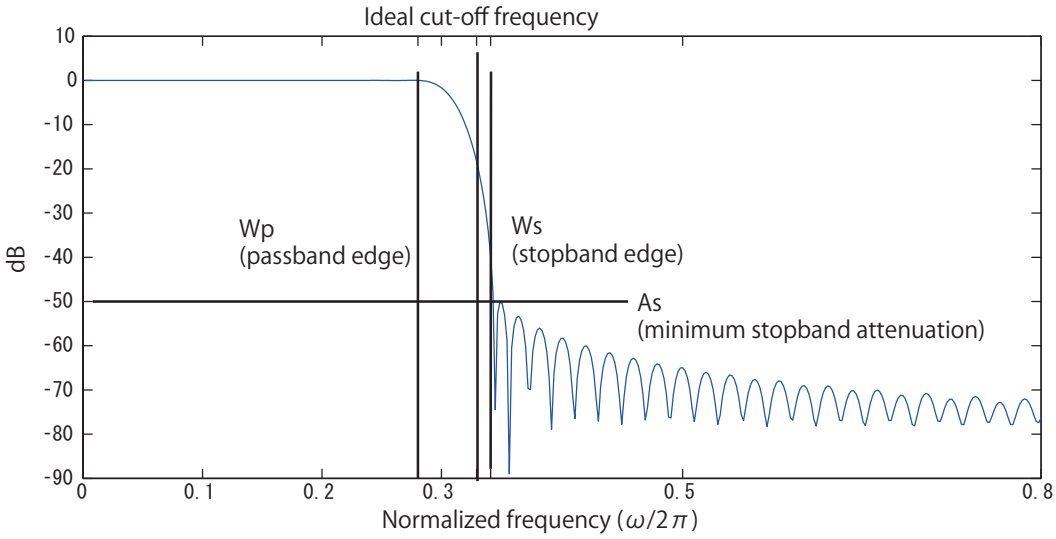


図 6.111: デフォルト設定のフィルタ特性: 横軸 正規化周波数 [$\omega/2\pi$]，縦軸 ゲイン [dB]

ノードの詳細

MultiDownSampler は，マルチチャネル信号をカイザー窓を用いたローパスフィルタによって帯域制限を行い，ダウンサンプリングを行なうノードである．具体的には 1) カイザー窓，2) 理想低域応答，の合成によって FIR ローパスフィルタを作成・実行した後，SAMPLING_RATE_OUT/SAMPLING_RATE_IN のダウンサンプルを行なう．

FIR フィルタ: 有限インパルス応答 $h(n)$ を用いたフィルタリングは次式によって行なわれる．

$$s_{\mathrm{out}}(t) = \sum_{i=0}^{N} h(n) s_{\mathrm{in}}(t-n) \tag{6.159}$$

ここで，$s_{\mathrm{out}}(t)$ は出力信号，$s_{\mathrm{in}}(t)$ は入力信号である．

マルチチャネル信号の場合は，各チャネルの信号に対して独立にフィルタリングが行なわれる．この時，用いる有限インパルス応答 $h(n)$ は同一のものである．

理想低域応答: 遮断周波数が ω_c である理想低域応答は以下の式によって定められる．

$$H_i(e^{j\omega}) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \text{otherwise} \end{cases} \tag{6.160}$$

このインパルス応答は,

$$h_i(n) = \frac{\omega_c}{\pi}\left(\frac{sin(\omega n)}{\omega n}\right), \quad -\infty \leq n \leq \infty \tag{6.161}$$

となる．このインパルス応答は非因果的かつ有界入力有界出力（BIBO: bounded input bounded output）安定条件を満さない．

この理想フィルタから FIR フィルタを得るには，インパルス応答を途中で打ち切る．

$$h(n) = \begin{cases} h_i(n), & |n| \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.162}$$

ここで N はフィルタの次数である．このフィルタはインパルス応答の打ち切りによって，通過域と阻止域にはリプルが発生する．また，阻止域最小減衰量 As も約 21 dB に止まり，十分な減衰量を得ることができない．

カイザー窓を用いた窓関数法によるローパスフィルタ: 上述の打ち切り法による特性を改善するため，理想インパルス応答 $h_i(n)$ に窓関数 $v(n)$ を掛けた，次式のインパルス応答を代りに用いる．

$$h(n) = h_i(n)v(n) \tag{6.163}$$

ここではカイザー窓を用いて，ローパスフィルタを設計する．カイザー窓は次式によって定義される．

$$v(n) = \begin{cases} \frac{I_0\left(\beta\sqrt{1-(nN/2)^2}\right)}{I_0(\beta)}, & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.164}$$

ここで，β は窓の形状を定めるパラメータ，$I_0(x)$ は 0 次の変形ベッセル関数であり，

$$I_0(x) = 1 + \sum_{k=1}^{\infty}\left(\frac{(0.5x)^k}{k!}\right) \tag{6.165}$$

から得られる．

パラメータ β は低域通過フィルタで求められる減衰量に応じて決まる．ここでは下記の指標によって定める．

$$\beta = \begin{cases} 0.1102(As - 8.7) & As > 50, \\ 0,5842(As - 21)^{0.4} + 0.07886(As - 21) & 21 < As < 50, \\ 0 & As < 21 \end{cases} \tag{6.166}$$

残りはフィルタ次数と遮断周波数 ω_c を定めれば，窓関数法によってローパスフィルタを実現できる．フィルタ次数 N は，阻止域最小減衰量 As と遷移域 $\Delta f = (Ws - Wp)/(2\pi)$ を用いて，

$$N \approx \frac{As - 7.95}{14.36\Delta f} \tag{6.167}$$

と見積もる．また，遮断周波数 ω_c を $0.5(Wp + Ws)$ と設定する．

ダウンサンプリング: ダウンサンプリングは，ローパスフィルタを通過させた信号から SAMPLING_RATE_IN / SAMPLING_RATE_OUT のサンプル点を間引くことによって実現される．例えば，デフォルトの設定では 48000/16000 = 3 であるから，入力サンプルを 3 回に 1 回 取り出し，出力サンプルとすれば良い．

参考文献:

(1) 著: P. Vaidyanathan, 訳: 西原 明法, 渡部 英二, 吉田 俊之, 杉野 暢彦: “マルチレート信号処理とフィルタバンク”, 科学技術出版, 2001.

6.7.21 MultiFFT

ノードの概要

マルチチャネル音声波形データに対し，高速フーリエ変換 (Fast Fourier Transformation: FFT) を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

このノードは，マルチチャネル音声波形データを，スペクトルに変換して時間周波数領域で解析を行いたいときに用いる．音声認識に用いる特徴抽出の前処理として用いられることが多い．

典型的な接続例

図 6.112 で，MultiFFT ノードに `Matrix<float>`，`Map<int, ObjectRef>` 型の入力を与える例を示す．

図 6.112 の上のパスは AudioStreamFromWave ノードから `Matrix<float>` 型の多チャネル音響信号を受け取り，MultiFFT ノードで `Matrix<complex<float> >` 型の複素スペクトルに変換したのち，LocalizeMUSIC ノードに入力される．

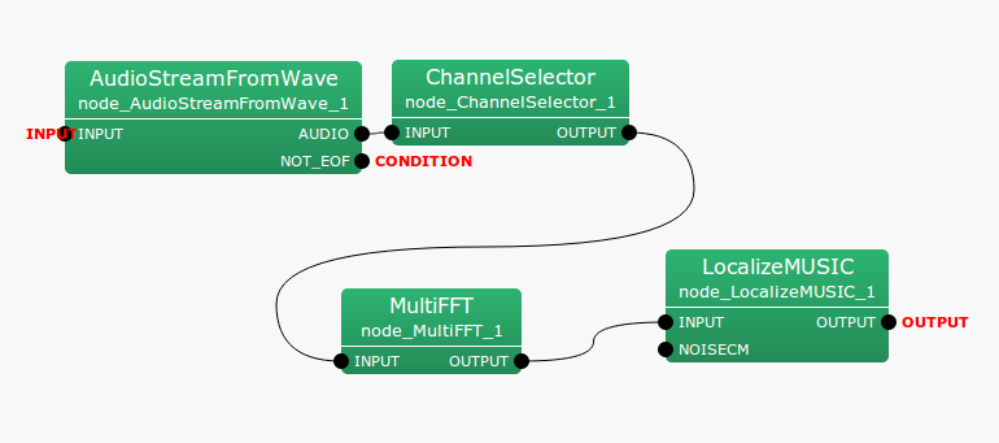


図 6.112: MultiFFT の接続例

ノードの入出力とプロパティ

入力

INPUT ：型は `Matrix<float>` または `Map<int, ObjectRef>`．マルチチャネル音声波形データ．`Map<int, ObjectRef>` の部分は `Vector<float>` 型．行列のサイズが $M \times L$ のとき，M がチャネル数，L が波形のサンプル数を表す．L は，パラメータ LENGTH と値が等しい必要がある．

表 6.116: MultiFFT のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	フーリエ変換を適用する信号の長さ
WINDOW	string	CONJ		フーリエ変換を行う際の窓関数の種類．CONJ，HAMMING，RECTANGLE,HANNING から選択する．それぞれ，複素窓，ハミング窓，矩形窓, ハニング窓を示す．
WINDOW_LENGTH	int	512	[pt]	フーリエ変換を行う際の窓関数の長さ．

出力

OUTPUT ：型は `Matrix<complex<float> >` または `Map<int, ObjectRef>`．入力に対応したマルチチャネル複素スペクトル．入力が `Matrix<float>` 型のとき，出力は `Matrix<complex<float> >` 型となり，入力が `Map<int, ObjectRef>` 型のとき，出力は `Map<int, ObjectRef>` 型となる．部分は `Vector<complex<float> >` 型となる．入力行列のサイズが $M \times L$ のとき，出力行列のサイズは，$M \times L/2 + 1$ となる．

パラメータ

LENGTH ：型は `int`．デフォルト値は 512．フーリエ変換を適用する信号の長さを指定する．アルゴリズムの性質上，2 のべき乗の値をとる．また，WINDOW_LENGTH より大きい値にする必要がある．

WINDOW ：型は `string`．デフォルト値は CONJ．CONJ，HAMMING，RECTANGLE,HANNING から選択する．それぞれ，複素窓，ハミング窓，矩形窓, ハニング窓を意味する．音声信号の解析には，HAMMING 窓がよく用いられる．

WINDOW_LENGTH ：型は `int`．デフォルト値は 512．窓関数の長さを指定する．値を大きくすると，周波数解像度は増す半面，時間解像度は減る．直感的には，この値を増やすと，音の高さの違いに敏感になるが，音の高さの変化に鈍感になる．

ノードの詳細

LENGTH，WINDOW_LENGTH の目安: 音声信号の解析には，20 ~ 40 [ms] に相当する長さのフレームで分析するのが適当である．サンプリング周波数を f_s [Hz]，窓の時間長を x [ms] とすると，フレーム長 L [pt] は，

$$L = \frac{f_s x}{1000}$$

で求められる．
例えば，サンプリング周波数が 16 [kHz] のとき，デフォルト値の 512 [pt] は，32 [ms] に相当する．パラメータ LENGTH は，高速フーリエ変換の性質上，2 の累乗の値が適しているため，512 を選ぶ．
　より音声の解析に適したフレームの長さを指定するため，窓関数の長さ WINDOW_LENGTH は，400 [pt] (サンプリング周波数が 16 [kHz] のとき，25 [ms] に相当) に設定することもある．

各窓関数の形: 各窓関数 $w(k)$ の形は次の通り．k はサンプルのインデックス，L は窓関数の長さ，FFT 長を $NFFT$ とし，k は $0 \leq k < L$ の範囲を動く．FFT 長が窓の長さよりも大きいとき，$NFFT \leq k < L$ における窓関数の値には，0 が埋められる．

CONJ，複素窓:

$$w(k) = \begin{cases} 0.5 - 0.5\cos\left(\frac{4k}{L}C\right), & \text{if} 0 \le k < L/4 \\ \sqrt{1 - \left\{0.5 - 0.5\cos\left(\frac{2L-4k}{L}C\right)\right\}^2}, & \text{if} L/4 \le k < 2L/4 \\ \sqrt{1 - \left\{0.5 - 0.5\cos\left(\frac{4k-2L}{L}C\right)\right\}^2}, & \text{if} 2L/4 \le k < 3L/4 \\ 0.5 - 0.5\cos\left(\frac{4L-4k}{L}C\right), & \text{if} 3L/4 \le k < L \\ 0, & \text{if} NFFT \le k < L \end{cases},$$

ただし，$C = 1.9979$ である．

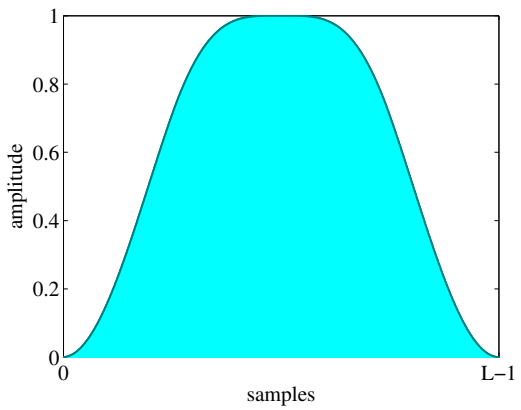


図 6.113: 複素窓関数の形状

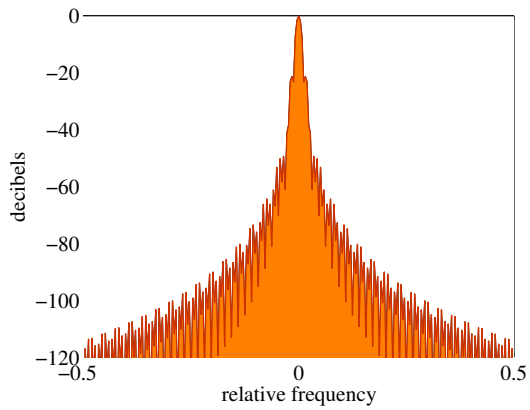


図 6.114: 複素窓関数の周波数応答

図 6.113 と図 6.114 はそれぞれ，複素窓関数の形状および周波数応答である．図 6.114 における横軸はサンプリング周波数に対して，相対的な周波数の値を意味する．一般に，窓関数の周波数応答は，横軸が 0 におけるピークが鋭い方が性能が良いとされる．縦軸の値は，フーリエ変換などの周波数解析を行ったとき，ある周波数ビンに他の周波数成分のパワーが漏れてくる量を表す．

HAMMING，ハミング窓:

$$w(k) = \begin{cases} 0.54 - 0.46\cos\frac{2\pi k}{L-1}, & \text{if } 0 \le k < L, \\ 0, & \text{if } L \le k < NFFT \end{cases}$$

ただし，π は円周率を表す．

図 6.115，6.115 はそれぞれ，ハミング窓関数の形状と周波数応答である．

RECTANGLE，矩形窓:

$$w(k) = \begin{cases} 1, & \text{if } 0 \le k < L \\ 0, & \text{if } L \le k < NFFT \end{cases}$$

図 6.117，6.117 はそれぞれ，矩形窓関数の形状と周波数応答である．

HANNING，ハニング窓:

$$w(k) = \begin{cases} 0.5 - 0.5\cos\frac{2\pi k}{L-1}, & \text{if } 0 \le k < L, \\ 0, & \text{if } L \le k < NFFT \end{cases}$$

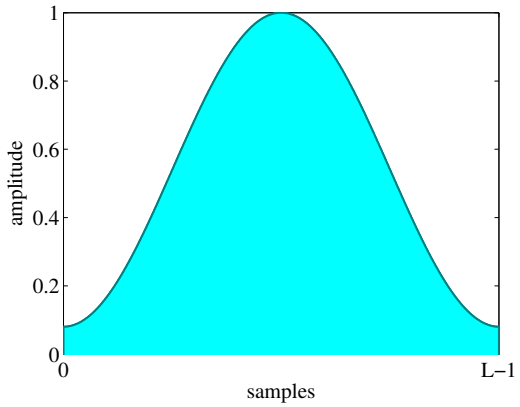


図 6.115: ハミング窓関数の形状

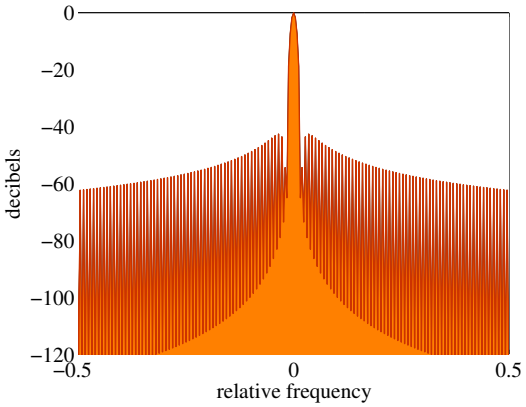


図 6.116: ハミング窓関数の周波数応答

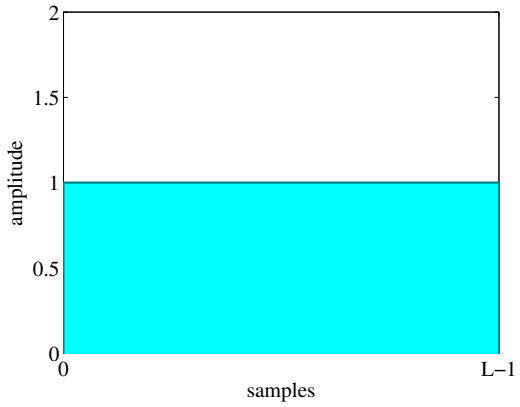


図 6.117: 矩形窓関数の形状

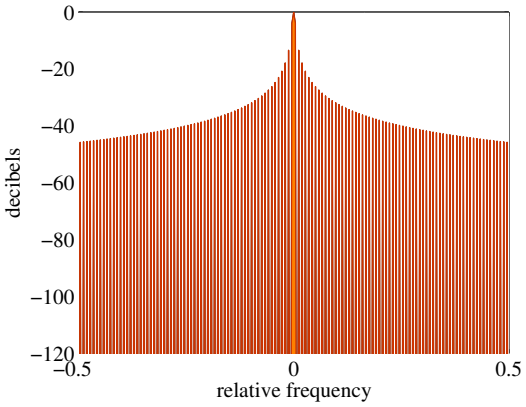


図 6.118: 矩形窓関数の周波数応答

6.7.22 MultiGain

ノードの概要

入力信号のゲインを調節する．

必要なファイル

無し．

使用方法

どんなときに使うのか

主に入力信号をクリップしないよう，もしくは，増幅する場合に使用する．例えば，入力に 24 [bit] で量子化された音声波形データを用いる場合，16 [bit] を仮定して構築したシステムを用いる場合には，このノードを利用して，8 [bit] 分，ゲインを落とすなどといった用途に用いる．

典型的な接続例

AudioStreamFromMic や AudioStreamFromWave の直後に直接配置するか，ChannelSelector を間に挟んで配置することが多い．

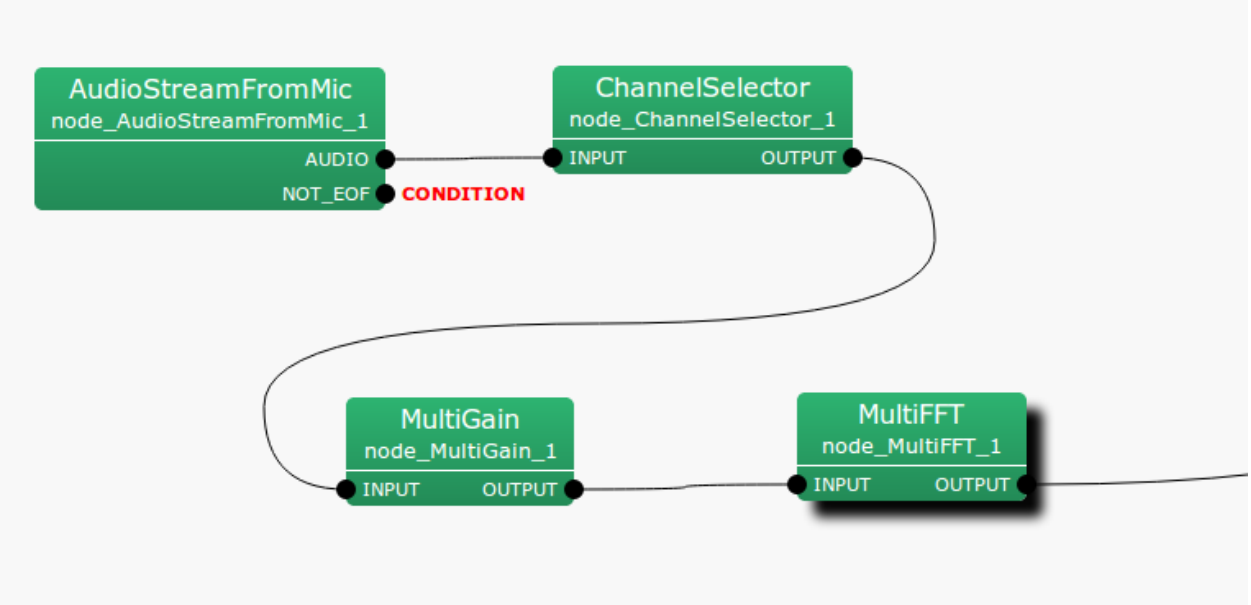


図 6.119: MultiGain の接続例

ノードの入出力とプロパティ

入力

表 6.117: MultiGain のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
GAIN	float	1.0		ゲイン値

INPUT : Matrix<float> 型．マルチチャネル音声波形データ（時間領域波形）．

出力

OUTPUT : Matrix<float> 型．ゲイン調節されたマルチチャネル音声波形データ（時間領域波形）．

パラメータ

GAIN : float 型．ゲインパラメータ．1.0 で，入力をそのまま出力することに相当する．

ノードの詳細

入力の各チャネルが GAIN パラメータで指定した値を乗じた値となって出力される．使用時は時間領域波形の入力を仮定していることに注意．

例えば，40 dB ゲインを落としたい場合には，下記のような計算を行い，0.01 を指定すればよい．

$$20 \log x = -40 \tag{6.168}$$

$$x = 0.01 \tag{6.169}$$

6.7.23 PowerCalcForMap

ノードの概要

Map<int, ObjectRef> 型の ID 付きマルチチャネル複素スペクトルを，実パワー（または振幅）スペクトルに変換する．

必要なファイル

無し．

使用方法

どんなときに使うのか

複素スペクトルを実パワー（または振幅）スペクトルに変換したいときに用いる．入力が Map<int, ObjectRef> 型のときはこのノードを用いる．入力が Matrix<complex<float> > 型の時は，PowerCalcForMatrix ノードを用いる．

典型的な接続例

図 6.120 に PowerCalcForMap ノードの使用例を示す．MultiFFT ノードから得られた Map<int, ObjectRef> 型複素スペクトルを，Map<int, ObjectRef> 型のパワースペクトルに変換したのち，MelFilterBank ノードに入力している．

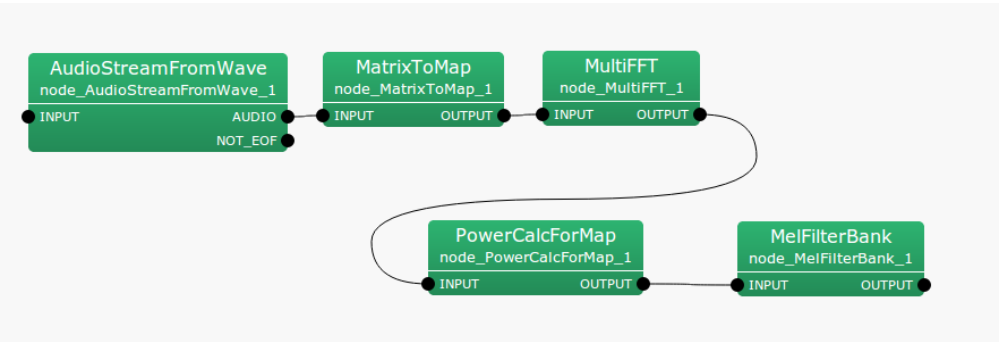


図 6.120: PowerCalcForMap の接続例

ノードの入出力とプロパティ

表 6.118: PowerCalcForMap のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
POWER_TYPE	string	POW		パワーか振幅かの選択

入力

INPUT : `Map<int, ObjectRef>` 型．`ObjectRef` 部分に，`Matrix<complex<float> >` 型の複素行列が格納されている．

出力

OUTPUT : `Map<int, ObjectRef>` 型．`ObjectRef` 部分に，入力の複素行列の各要素について，パワー（または絶対値）を取った実行列が格納されている．

パラメータ

POWER_TYPE : パワースペクトル (POW) か振幅スペクトル (MAG) かの選択．

ノードの詳細

入力の複素行列 $M_{i,j}$ (i, j はそれぞれ，行，列のインデックス) に対して，出力の実行列 $N_{i,j}$ は次のように求める．

$$
\begin{aligned}
N_{i,j} &= M_{i,j}M^*_{i,j} \ \text{(if POWER\_TYPE=POW)}, \\
N_{i,j} &= abs(M_{i,j}) \ \text{(if POWER\_TYPE=MAG)},
\end{aligned}
$$

ただし，$M^*_{i,j}$ は，$M_{i,j}$ の複素共役を表す．

6.7.24 PowerCalcForMatrix

ノードの概要

Matrix<complex<float> > 型のマルチチャネル複素スペクトルを，実パワー（または振幅）スペクトルに変換する．

必要なファイル

無し．

使用方法

どんなときに使うのか

複素スペクトルを実パワー（または振幅）スペクトルに変換したいときに用いる．入力が Matrix<complex<float> > 型のときはこのノードを用いる．入力が Map<int, ObjectRef> 型のときは PowerCalcForMap ノードを用いる．

典型的な接続例

図 6.121 に PowerCalcForMatrix ノードの使用例を示す．MultiFFT ノードから得られた Matrix<complex<float> > 型複素スペクトルを，Matrix<float> 型のパワースペクトルに変換したのち，BGNEstimator ノードに入力している．

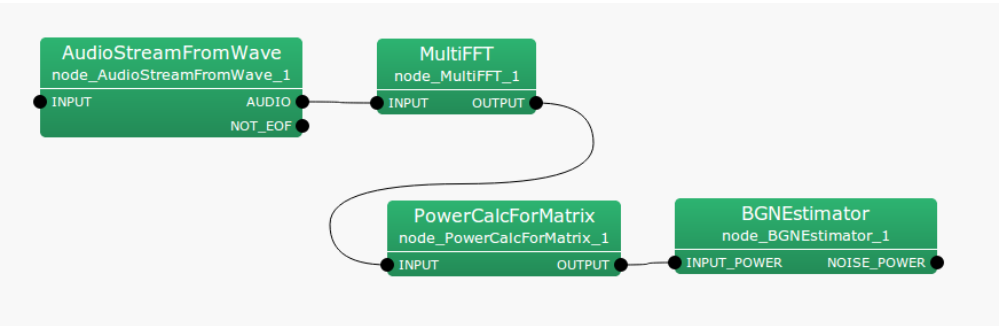


図 6.121: PowerCalcForMatrix の接続例

ノードの入出力とプロパティ

表 6.119: PowerCalcForMatrix のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
POWER_TYPE	string	POW		パワーか振幅かの選択

入力

INPUT : `Matrix<complex<float> >` 型．各要素が複素数の行列．

出力

OUTPUT : `Matrix<float>` 型．入力の各要素のパワー（または絶対値）を取った実行列．

パラメータ

POWER_TYPE : パワースペクトル (POW) か振幅スペクトル (MAG) かの選択

ノードの詳細

入力の複素行列 $M_{i,j}$ (i, j はそれぞれ行，列のインデックス) に対して，出力の実行列 $N_{i,j}$ は次のように求める．

$$
\begin{aligned}
N_{i,j} &= M_{i,j}M^*_{i,j} \ \text{(if POWER\_TYPE=POW)}, \\
N_{i,j} &= abs(M_{i,j}) \ \text{(if POWER\_TYPE=MAG)},
\end{aligned}
$$

ただし，$M^*_{i,j}$ は，$M_{i,j}$ の複素共役を表す．

6.7.25 ResizeMapMatrixValues

ノードの概要

Map<int, ObjectRef> 型の ObjectRef が Matrix<ObjectRef> である時，その要素のサイズを変える．

必要なファイル

無し．

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Matrix<ObjectRef> である時，その要素のサイズを変える．元のサイズより小さくする場合は必要な数だけ切り詰められ，元のサイズより大きくする場合は必要な数だけ 0 が埋められる．

ノードの入出力とプロパティ

入力

INPUT : Map<int, ObjectRef> 型の Map< int , Matrix < int > > または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型.

出力

INPUT : Map<int, ObjectRef> 型の Map< int , Matrix < int > > または Map< int , Matrix<float> > または Map< int , Matrix<complex<float> > > 型.

パラメータ

表 6.120: ResizeMapMatrixValues パラメータ表

パラメータ名	型	デフォルト値	単位	説明
RESIZE_TYPE	string	RELATIVE		パラメータ SIZE_ROW と SIZE_COLUMN の扱い方．RELATIVE は相対値で，ABSOLUTE は絶対値で指定することを示す．
SIZE_ROW	int	0		元の行数に追加する行数, または, 置き換える行数．パラメータ RESIZE_TYPE による．
SIZE_COLUMN	int	0		元の列数に追加する列数, または, 置き換える列数．パラメータ RESIZE_TYPE による．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

RESIZE_TYPE : `string` 型．パラメータ SIZE_ROW と SIZE_COLUMN の扱い方を指定する．元の行数に SIZE_ROW を, 元の列数に SIZE_COLUMN を加算して行列サイズを変更する「RELATIVE」, 行数を SIZE_ROW に, 列数を SIZE_COLUMN にして行列サイズを変更する「ABSOLUTE」から選択する．デフォルトは RELATIVE.

SIZE_ROW : `int` 型．元の行数に追加する行数, または, 置き換える行数．どちらかであるかは, パラメータ RESIZE_TYPE による．(∗) デフォルトは 0.

SIZE_COLUMN : `int` 型．元の列数に追加する列数, または, 置き換える列数．どちらかであるかは, パラメータ RESIZE_TYPE による．(∗) デフォルトは 0.

DEBUG : `bool` 型．`true` が与えられると, 変換状況が標準出力に出力される．デフォルトは `false`.

(∗)

パラメータ RESIZE_TYPE が RELATIVE の場合，元の行列サイズを (A,B) とすると，変更後のサイズは (A+SIZE_ROW, B+SIZE_COLUMN) になり、ABSOLUTE の場合，変更後のサイズは (SIZE_ROW, SIZE_COLUMN) になる．変更後のサイズが変更前のサイズよりも小さくなる場合は，行列の最後から必要な数だけ切り詰められ，変更後のサイズが変更前のサイズより大きくなる場合は，行列の最後に必要な数だけ 0 が追加される．変更後のサイズが，負になる場合はエラーとなり，0 になる場合は空の行列が出力される．

ノードの詳細

＜例＞

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:

$$\left\{ 0, \left[1 \right] \right\}, \left\{ 1, \left[5 \right] \right\}, \left\{ 2, \left[9 \right] \right\}$$

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:

$$\left\{ 0, \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \end{bmatrix} \right\}, \left\{ 1, \begin{bmatrix} 5 & 6 & 0 & 0 & 0 \end{bmatrix} \right\}, \left\{ 2, \begin{bmatrix} 9 & 10 & 0 & 0 & 0 \end{bmatrix} \right\}$$

6.7.26 ResizeMapVectorValues

ノードの概要

Map<int, ObjectRef> 型の ObjectRef が Vector<ObjectRef> である時，その要素のサイズを変える．

必要なファイル

無し．

使用方法

どんなときに使うのか

Map<int, ObjectRef> 型の ObjectRef が Vector<ObjectRef> である時，その要素のサイズを変える．元のサイズより小さくする場合は必要な数だけ切り詰められ，元のサイズより大きくする場合は必要な数だけ 0 が埋められる．

ノードの入出力とプロパティ

入力

INPUT ：Map<int, ObjectRef> 型の Map< int , Vector<int> > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型.

出力

OUPUT ：Map<int, ObjectRef> 型の Map< int , Vector<int> > または Map< int , Vector<float> > または Map< int , Vector<complex<float> > > 型.

パラメータ

表 6.121: ResizeMapVectorValues パラメータ表

パラメータ名	型	デフォルト値	単位	説明
RESIZE_TYPE	string	RELATIVE		パラメータ SIZE の扱い方．RELATIVE は相対値で, ABSOLUTE は絶対値で指定することを示す.
SIZE	int	0		元の Vector に追加する要素の数, または, 置き換える Vector のサイズ要素サイズ．パラメータ RESIZE_TYPE による．
DEBUG	bool	false		変換状況を出力するかどうかの選択.

RESIZE_TYPE ：string 型．パラメータ SIZE の扱い方を指定する．元のサイズに SIZE を加算してサイズ変更する「RELATIVE」,SIZE にサイズ変更する「ABSOLUTE」から選択する．デフォルトは RELATIVE.

SIZE : int 型．元の Vector に追加する要素の数，または置き換える Vector のサイズ．どちらかであるかは，パラメータ RESIZE_TYPE による。パラメータ RESIZE_TYPE が,RELATIVE の場合, 元のサイズを (A) とすると変更後のサイズは (A+SIZE) になり,ABSOLUTE の場合, 変更後のサイズは (SIZE) になる．変更後のサイズが変更前のサイズよりも小さくなる場合は, 要素の最後から必要な数だけ切り詰められ, 変更後のサイズが変更前のサイズより大きくなる場合は, 要素の最後に必要な数だけ 0 が追加される．変更後のサイズが, 負になる場合はエラーとなり,0 になる場合は空の Vector が出力される．デフォルトは 0.

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

ノードの詳細

＜例＞

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 > }

6.7.27 SaveMapFrames

ノードの概要

Map<int, ObjectRef> 型のコンテナに格納されていたフレームのデータを，Map のキーごとに別々にファイルに保存する． これらのデータファイルに加えて，各フレームのデータファイルのリストを記載したファイルを提供する． ObjectRef は Vector<float> 型または Vector<complex<float> > 型のいずれかでなければならないことに注意．

必要なファイル

無し．

使用方法

どんなときに使うのか

このノードは，元々フレームのデータが格納されていた Map<int, ObjectRef> 型のキーごとに，フレームのデータを別々にファイルに保存するために用いる． 各キーに対して生成された各ファイルは， 同じキーにマップされたデータで構成される．さらに，フレーム毎に，これらのデータファイルのリストを記載したファイルを生成する． 下記の ”典型的な接続例”で示すネットワーク例では， SaveMapFrames は， GHDSS によって出力された Map<int , Vector<complex<float> > > 型のデータを入力として受け取る． データは，分離音の音源 ID と複素スペクトルのペアになる． ここで，キーは音源 ID を表す． 各キー用に生成された各ファイルは，同じ音源 ID にマッピングされたデータで構成される．

典型的な接続例

図 6.122 は，ネットワークに SaveMapFrames を使用した接続例を示す． このネットワークでは，音源分離は GHDSS アルゴリズムを使用して実行され，出力は SaveMapFrames によってファイルに保存される． GHDSS アルゴリズムに基づくサウンド分離の詳細については，GHDSS のノードリファレンスを参照のこと．

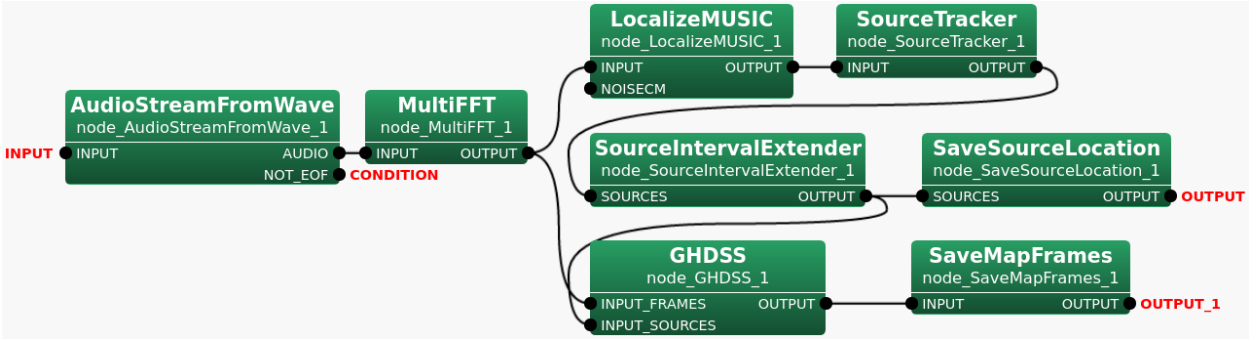


図 6.122: SaveMapFrames の接続例

ノードの入出力とプロパティ

入力

INPUT ：データが Map のキーにマップされる Map<int, ObjectRef> 型のフレームのデータ．フレームのデータはキーにマップされる．サポートされている型は， Map<int , Vector<float> > 型および Map<int , Vector<complex<float> > > 型．

出力

OUTPUT ：2 種類のファイル． 一つは ”内部ファイル”と呼ばれるデータファイルで，もう一つは ”メインファイル”と呼ばれる ”内部ファイル”のリスト． ”内部ファイル”は，入力 Map<int, ObjectRef> のキーの数だけ生成されるので，出力ファイルの総数はキーの数に 1 を加えたものになる．

パラメータ

表 6.122: SaveMapFrames のパラメータ表

Parameter name	Type	Default value	Unit	Description
パラメータ名	型	デフォルト値	単位	説明
FILENAME	string			出力するテキストファイルのファイル名．フレームごとのデータファイルのリストが含まれる． このファイル名は，出力するデータファイルのファイル名のベース名になる．
OUTPUTTYPE	string	TEXT		出力するデータファイルのファイル形式オプション． テキストファイルの場合は TEXT，バイナリファイルの場合は RAW を選択する．
TEXTFORMAT	string	FIXED		OUTPUTTYPE を TEXT に設定した場合に出力するデータファイルのファイル形式オプション． FIXED か SCIENTIFIC を選択する．

FILENAME ：string type. 各フレームの「内部ファイル」のリストが書き込まれている「メインファイル」のファイル名．

OUTPUTTYPE ：string type. 「内部ファイル」のファイル形式オプション． テキストファイルの場合は TEXT，バイナリファイルの場合は RAW を選択する． デフォルト値は TEXT． 「メインファイル」は，このパラメータで指定された値に関係なく，テキストファイルとなる．

TEXTFORMAT ：string type. OUTPUTTYPE を TEXT に設定した場合に出力するデータファイルのファイル形式オプション． FIXED か SCIENTIFIC を選択する．

ノードの詳細

SaveMapFrames は，フレームごとに Map<int , Vector<float> > 型または Map<int , Vector<complex<float> > > 型コンテナのいずれかの型に格納されたフレームのデータを入力として受け取り，そのデータをキーごとに別々のファイルに保存する． 出力は次の 2 種類のファイル．

2 種類の出力ファイル:

1. メインファイル -すべてのフレームの「内部ファイル（下記参照）」リストが書き込まれたテキストファイル． 各リストはフレームごとに区切られる．

2. 内部ファイル - `Map<int, ObjectRef>` 型のフレームのデータがキーごとに個別に保存されたデータファイル． これらのファイルは，キーの数だけ作成される． 各ファイルは，同じキーにマップされたフレームデータで構成される． ファイル形式は，OUTPUTTYPE パラメーターで指定． OUTPUTTYPE パラメータを RAW に設定すると，フレームのデータは，リトルエンディアン順に IEEE 754 32 ビット単精度浮動小数点数形式で書き込まれる． このパラメータのデフォルト値 TEXT は，RAW とは異なり人間が読めるデータを出力する．

Map の型についての詳細は， `Map` を参照のこと．

出力ファイルのファイル名:

1. メインファイル - FILENAME パラメータで指定する．

2. 内部ファイル - 以下のパターンに従って自動的に生成． ”TEXT”と ”RAW”は OUTPUTTYPE パラメータの選択であることに注意． このノードでは， ”containertype”は常に ”Vector”で， ”datatype”は ”float”または ”complex_float”， ”colsize”は 1 つのキーにマップされるデータのサイズとなる．

   TEXT: FILENAME + Map key + {contianertype} + {datatype} + ”_col”{colsize} + .txt
   RAW: FILENAME + Map key + {contianertype} + {datatype} + ”_col”{colsize} +. raw

   上記のように，OUTPUTTYPE パラメータで TEXT または RAW を選択した場合のファイル名の違いは，ファイル拡張子だけである． ファイル名の例を，以下の Main ファイルのサンプルとして示す．

出力ファイルの内容:

1. メインファイル
   フレームごとの「内部ファイル」のリスト． 1 つのフレームが複数の異なるキーにマッピングされたデータで構成されている場合，それぞれのデータは対応するキーのファイルに保存されるため，1 つのフレームに対して複数のファイル名が表示される． 1 つのフレームのデータがファイルに保存されるたびに，ファイル名がフレーム番号に続くリストに追加される． フレーム番号は 0 から始まる．

   関連するパラメータ値を以下のように設定した場合，6397 フレーム分のデータを処理した後の「メインファイル」の内容例を以下に示す．

Parameter name	Value
FILENAME	Savedata.map
OUTPUTTYPE	TEXT

   左側の灰色の数字は，実際にファイル名が「メインファイル」にどのように書き込まれているかを読者

が容易に理解できるようにするための行番号であり，その行番号は実際のファイルには書かれていないことに注意.

メインファイルのサンプル:

```
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

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

⋮

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
```

`Map<int, ObjectRef>` 型の空のコンテナが受信されると，フレーム番号のみがそのフレームのファイルに書き込まれる． 以下は，このような空のコンテナがいくつかのフレームで受信されたことを示す Main ファイルの例． FILENAME パラメータの値は ”file”.

```
0
1
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
⋮
```

左の灰色の数字は，実際にファイル名が「メインファイル」にどのように書かれているかを読者が容易に理解できるようにするための行番号を示しているため，これらの行番号は実際のファイルには書き込まれないことに注意.

2. 内部ファイル
表 6.123 に，OUTPUTTYPE パラメータで TEXT を選択した場合のファイル形式を示す． 各データはスペースで区切られ，各フレームのデータは改行で区切られる.

表 6.123: OUTPUTTYPE パラメータが TEXT の場合のファイル形式

データ型	ファイルフォーマット
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()

ここで，n は，フレームのデータが格納されている入力 *Map<int, ObjectRef>* の *ObjectRef* である *Vector* のサイズ.

ベクターサイズ

Vector Size は，入力として受け取った Map<int , Vector<ObjectRef> > の Vector<ObjectRef> のサイズ． このノードの ObjectRef は， float または complex<float> ． 上記のサンプルネットワーク（図 6.122）では， SaveMapFrames の前に MultiFFT が接続されているため，257 になる． MultiFFT では，Vector のサイズは，長さ/ 2 + 1 の式を使用して計算される． MultiFFT のウィンドウの長さはデフォルトで 512． 数式を適用した結果，257.

保存されるパターン

SaveMapFrames は，フレームごとに，各データがマップされているキーによってフレームのデータを別々のファイルに保存する． 図 6.123 に，6397 フレームのデータが 0000 から 0007 のキーにマッピングされ，257 サイズのベクターデータが各ファイルに書き込まれている例を示す.

サンプルネットワーク（図 6.122）では， SaveMapFrames の入力は Map<int, ObjectRef> 型のコンテナで， GHDSS の出力． 各 Map<int, ObjectRef> は，分離された音源 ID と複素スペクトルのペア． 内部ファイルは，音源の数だけ生成される． 入力用のフレームのデータが音源分離の結果である場合には，

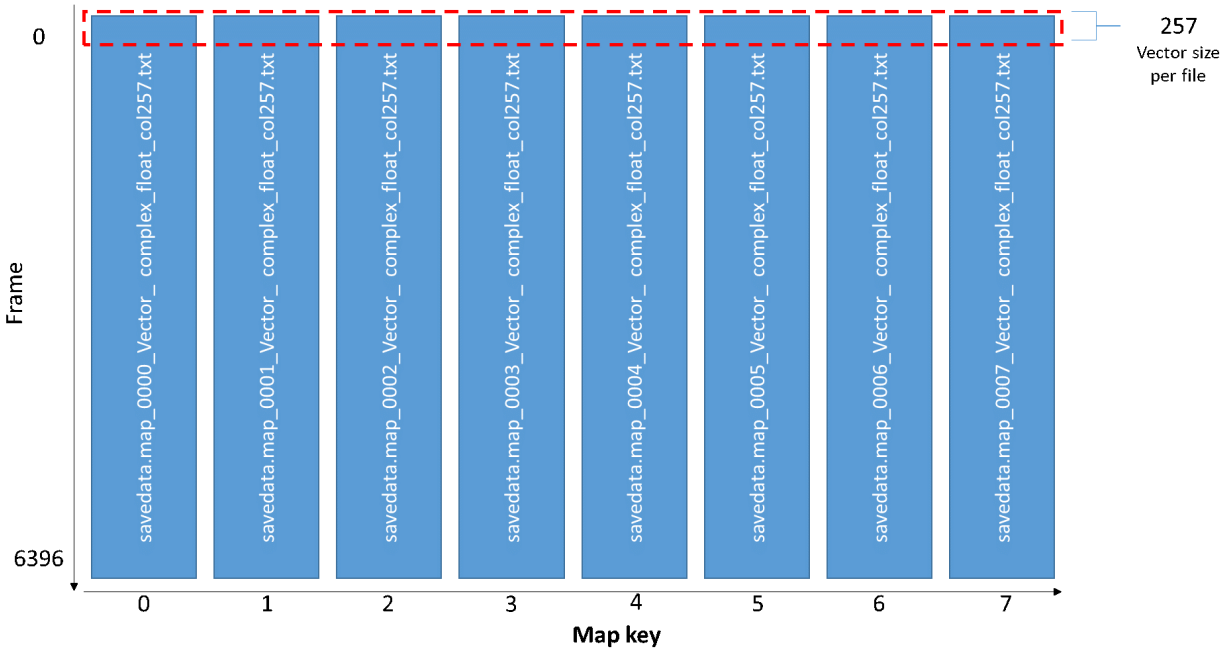


図 6.123: 全フレームの全データを含む Map パターンの保存

全ての音源が同時に音を出さないか，ある持続時間を有する場合には，フレームのデータが部分的に見つからないことが起こりうる． このような場合の例を図 6.124 に示す．フレーム [0] では，257 サイズのベクターデータが 0000，0001，0002 のファイルに順次書き込まれる． フレーム [100] では，257 サイズのベクターデータが，0000，0002，0003，0004 のファイルに順次書き込まれる．

ここで，n は音源の数．

内部ファイルのサンプル:

図 6.125 に，表 6.123 で説明した TEXT ファイル形式のデータの書き方を示す． この例では，入力データが 6397 フレーム，ベクターサイズが 512，入力データ型が Map<`int` , `Vector<float>` > であると仮定する． これらの情報は自動的に内部で検出され，ユーザーの入力は必要ない． 図 6.124 および図 6.125 の左側のフレーム番号はファイルにはないことに注意．

図 6.126 は，入力データが Map<`int` , `Vector<complex<float>` > > 型であり，入力データが 6397 フレームで，ベクターサイズがサンプルネットワーク（図 6.122）とまったく同じ場合の例を示す．

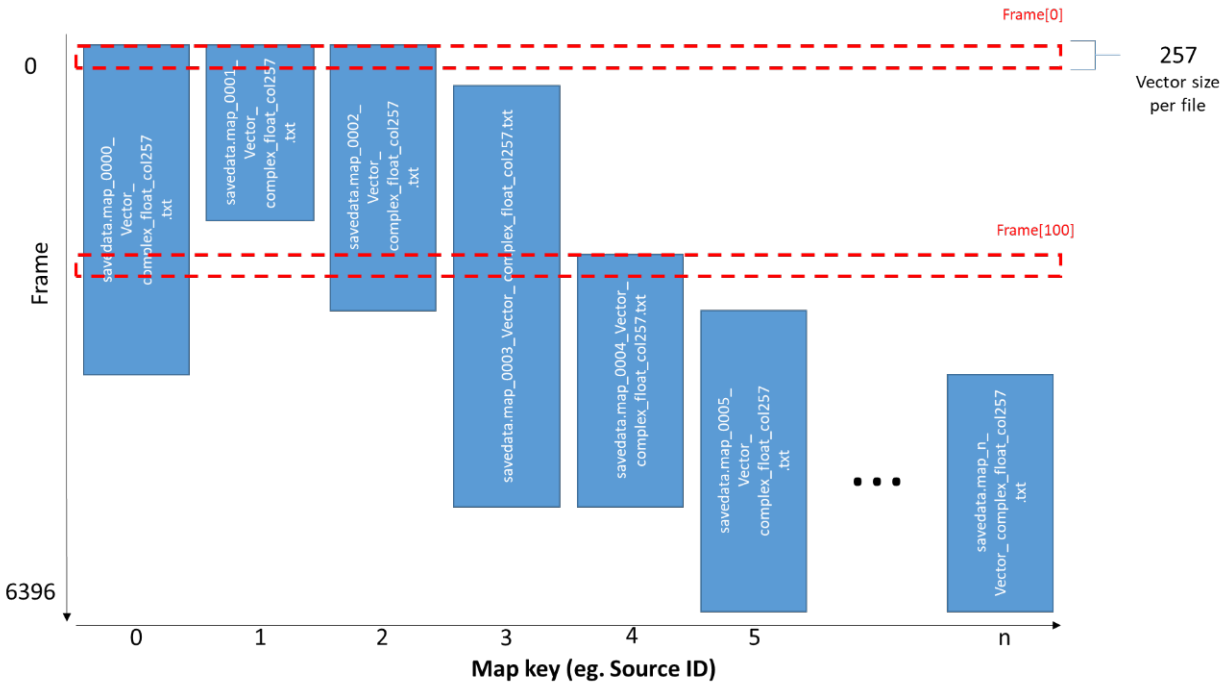


図 6.124: 複数の時刻と期間を音源とした Map パターンの保存

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[511]

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[511]

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[511]

.
.
.
.
.
.

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[511]

図 6.125: TEXT を float 型とした SaveMapFrames の内部ファイルのサンプル

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()

図 6.126: TEXT を Complex float 型とした SaveMapFrames の内部ファイルのサンプル

6.7.28 SaveMatrixFrames

ノードの概要

Matrix<ObjectRef> 型のフレームのデータをファイルに保存する．サポートされている ObjectRef は float 型と complex<float> 型のみである．

必要なファイル

無し．

使用方法

どんなときに使うのか

このノードは，複数のフレームの Matrix<ObjectRef> 型データを 1 つのファイルに保存するために用いる．

典型的な接続例

下の図 6.127 は，MultiFFT が Matrix<complex<float> > 型のフレームのデータを出力する SaveMatrixFrames を使用したネットワークの例を示す．

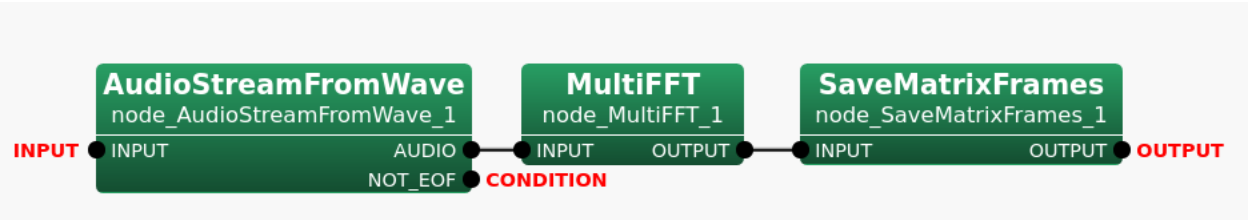


図 6.127: SaveMatrixFrames の接続例

ノードの入出力とプロパティ

入力

INPUT ：Matrix<ObjectRef> 型のフレームのデータ．サポートされている型は Matrix<float> 型と Matrix<complex<float> > 型．

出力

OUTPUT ：複数のフレームの Matrix<ObjectRef> 型データを保存したファイル．

パラメータ

FILENAME ：string type. 出力ファイルのファイル名．

表 6.124: SaveMatrixFrames のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FILENAME	string			出力ファイルのファイル名.
OUTPUTTYPE	string	TEXT		出力ファイルのファイル形式オプション． テキストファイルの場合は TEXT，バイナリファイルの場合は RAW を選択する．
TEXTFORMAT	string	FIXED		OUTPUTTYPE を TEXT に設定した場合に出力するデータファイルのファイル形式オプション． FIXED か SCIENTIFIC を選択する．

OUTPUTTYPE ：string type. 出力ファイルのファイル形式オプション． テキストファイルの場合は TEXT，バイナリファイルの場合は RAW を選択する． デフォルト値は TEXT.

TEXTFORMAT ：string type. OUTPUTTYPE を TEXT に設定した場合に出力するデータファイルのファイル形式オプション． FIXED か SCIENTIFIC を選択する．

ノードの詳細

SaveMatrixFrames は，複数のフレームの Matrix<float> 型または Matrix<complex<float> > 型のデータを入力として受け取り，それらのすべてのデータを 1 つのファイルに保存する． ファイル形式は，OUTPUTTYPE パラメータで指定する． OUTPUTTYPE パラメータを RAW に設定すると，フレームのデータは，リトルエンディアン順に IEEE 754 32 ビット単精度浮動小数点数形式で書き込まれる． このパラメータのデフォルト値 TEXT は，RAW とは異なり人間が読めるデータを出力する．

表 6.125 に，OUTPUTTYPE パラメータで TEXT が選択されたときに，ファイル内でデータがどのようにフォーマットされるかの詳細を示す． 各データはスペースで区切られ，フレームのデータはラインフィードによって Matrix のサイズでフレームごとに分離される．

表 6.125: TEXT OUTPUTTYP のファイル形式

データ型	ファイルフォーマット
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()

ここで，n と m は，フレームのデータが格納されている入力の行列の列サイズと行サイズである．

SaveMatrixFrames は，各フレームのデータを受け取ると，マトリックスの行と列のサイズをチェックする． 上のサンプルネットワーク（図 6.127）では， MultiFFT は Matrix<complex<float> > を出力する． 入力行列サイズが M × L の場合，出力行列のサイズは MultiFFT では M × L / 2 + 1 であり，M はチャネル数を表し，L はサンプル数を表す． MultiFFT のウィンドウの長さはデフォルトで 512 である． L の式を適用すると，列のサイズが 257 となる． チャネル数が 8 の場合，行サイズは 8 になる． MultiFFT 出力の詳細については， MultiFFT ノードリファレンスを参照されたい．

入力データが Matrix<float> 型に格納されていて，列サイズが 512 で行サイズが 8 で，合計 6397 のフレームがあるとすると，出力ファイルの内容は図 6.128 のようになる． 表 6.125 に記述されている TEXT ファイル形式でデータが保存されているものとする．

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]

図 6.128: データがフロート型である TEXT ファイル形式の出力データファイルのサンプル内容

図 6.128 および図 6.129 の左側に書かれたフレーム番号は，ファイルに書き込まれないことに注意．

図 6.129 は，入力データが図 6.127 のサンプルネットワークのような Matrix<complex<float> > 型の場合の出力ファイルのサンプル内容を示す．行サイズと列サイズはそれぞれ 8 と 257． 合計で 6397 のフレームがある． また，データは表 6.125 に記述されている TEXT ファイル形式で保存されているものとする．

Filename:

FILENAME パラメータに値を指定するときは，フォーマット文字列と呼ばれるデータプロパティを持つ特別なパターン {tag：format} を使用するオプションを使用できる． フォーマット文字列を使用すると，入力データの情報を含んだファイル名を設定することができ，後の使用に便利である．使用可能なタグを，表 6.126 に示す．

表 6.126: Tag list of SaveMatrixFrames のタグ表

タグ	説明	単位
datatype	データ型 (float または complex float)	String
rowsize	行列の行サイズ	Integer
colsize	行列の列サイズ	Integer
dim	データの次元 (固定値． 行列は 2)	Integer

```
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()
```

図 6.129: SaveMatrixFrames の TEXT Complex Float 型出力ファイルの例

表 6.127 にフォーマット文字列とその出力の例を示す.

表 6.127: `Matrix<complex<float> >` 型のフォーマット文字列とその出力の例．行サイズは 8，列サイズは 512 とする

フォーマット文字列 (FILENAME パラメータ値)	出力 (フォーマット指定されたファイル名)
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

LoadMatrixFrames を使用して SaveMatrixFrames によって出力されたファイルを `Matrix<ObjectRef>` 型にロードするときは，列サイズ，行サイズ，およびデータ型をパラメータで指定する必要がある． したがって，ファイル名にデータ情報を持つことは，後の使用に役立つ.

表 6.127 に示されているように，ファイル名の拡張子に関する規則はない． ユーザーが指定する任意の値にすることができる． ファイル名の拡張子は無くても良い.

{tag：format} の ”format”の値はオプション． ”format”は，03d のような桁数を指定するために使用できる．フォーマット指定子は，C の printf と同じである.

6.7.29 SaveVectorFrames

ノードの概要

`Vector<ObjectRef>` 型のフレームのデータをファイルに保存する． サポートされている `ObjectRef` は `float` 型と `complex<float>` 型のみである．

必要なファイル

無し．

使用方法

どんなときに使うのか

このノードは，複数のフレームの `Vector<ObjectRef>` 型データを 1 つのファイルに保存するために用いる．

典型的な接続例

図 6.130 は，ネットワーク内で SaveVectorFrames を使用した接続例を示す．このネットワークでは，SaveVectorFrames は通常は非表示である LocalizeMUSIC の SPECTRUM 端末で出力されたデータを入力として受け取る． それらのデータは方向毎の MUSIC スペクトルのパワーである． 将来の使用または文書化のために，SaveVectorFrames を使用してデータをファイルに保存する． LocalizeMUSIC の非表示出力 SPECTRUM の詳細については， LocalizeMUSIC ノードリファレンス，特に LocalizeMUSIC の 6.2.14.4 ノードの入出力とプロパティを参照されたい．

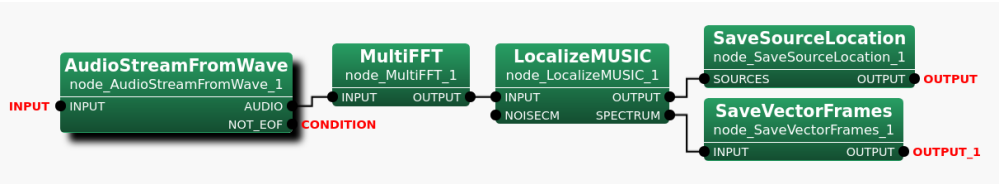


図 6.130: SaveVectorFrames の接続例

ノードの入出力とプロパティ

入力

INPUT ：`Vector<ObjectRef>` 型のフレームのデータ．サポートされている型は `Vector<float>` 型と `Vector<complex<float>>` 型．

出力

OUTPUT ：複数のフレームの `Vector<ObjectRef>` 型データを保存したファイル．

表 6.128: SaveVectorFrames のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
FILENAME	string			出力ファイルのファイル名.
OUTPUTTYPE	string	TEXT		出力ファイルのファイル形式オプション． テキストファイルの場合は TEXT，バイナリファイルの場合は RAW を選択する．

パラメータ

FILENAME ：string type. 出力ファイルのファイル名.

OUTPUTTYPE ：string type. 出力ファイルのファイル形式オプション． テキストファイルの場合は TEXT，バイナリファイルの場合は RAW を選択する． デフォルト値は TEXT.

ノードの詳細

SaveVectorFrames は，複数のフレームの Vector< float > 型または Vector<complex<float> > 型のデータを入力として受け取り，それらのすべてのデータを 1 つのファイルに保存する．ファイル形式は，OUTPUTTYPE パラメータで指定する． OUTPUTTYPE パラメータを RAW に設定すると，フレームのデータは，リトルエンディアン順に IEEE 754 32 ビット単精度浮動小数点数形式で書き込まれる． このパラメータのデフォルト値 TEXT は，RAW とは異なり人間が読めるデータを出力する．

表 6.129 に，OUTPUTTYPE パラメータで TEXT が選択されたときに，ファイル内でデータがどのようにフォーマットされるかの詳細を示す． 各データはスペースで区切られ，フレームのデータはラインフィードによって Vector のサイズでフレームごとに分離される．

表 6.129: TEXT OUTPUTTYPE のファイル形式

データ型	ファイルフォーマット
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()

ここで，ｎは，フレームのデータが格納されている入力ベクターのサイズである．

SaveVectorFrames は，データを受け取ったときに Vector のサイズをチェックする． 上のサンプルネットワーク（図 6.130）では， LocalizeMUSIC は Vector<float> を出力する． LocalizeMUSIC で指定された伝達関数が 72 個の音源から生成された場合，予想されるベクターサイズは 72 になる．伝達関数は音伝播のモデルである．伝達関数の音源数は，ベクターサイズに影響を与える要因の 1 つであり，伝達関数作成中に決定される． 伝達関数の詳細については，https://www.hark.jp/document/harktv/の「HARK Transfer Function Tutorial」のビデオを参照して頂きたい． 伝達関数を生成するための詳細なドキュメントは，https://www.hark.jp/document/transfer-function-generation-manuals/を参照されたい．

入力データが Vector<float> 型に格納されていて，列サイズが 72 で，合計 6397 のフレームがあるとすると，出力ファイルの内容は図 6.131 のようになる． 表 6.129 に記述されている TEXT ファイル形式でデータが保存されているものとする．

```
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]
```

図 6.131: SaveVectorFrames の TEXT Float 型出力ファイルの例

図 6.131 および図 6.132 の左側に書かれたフレーム番号は，ファイルに書き込まれないことに注意．

図 6.132 は，入力データが図 6.130 のサンプルネットワークのような Vector<complex<float> > 型の場合の出力ファイルのサンプル内容を示す．列サイズは 71． 合計で 6397 のフレームがある． また，データは表 6.129 に記述されている TEXT ファイル形式で保存されているものとする．

```
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()
```

図 6.132: SaveVectorFrames の TEXT Complex Float 型出力ファイルの例

Filename:

FILENAME パラメータに値を指定するときは，フォーマット文字列と呼ばれるデータプロパティを持つ特別なパターン {tag：format} を使用するオプションを使用できる． フォーマット文字列を使用すると，入力データの情報を含んだファイル名を設定することができ，後の使用に便利である．
使用可能なタグを，表 6.130 に示す．

表 6.130: SaveVectorFrames のタグ表

タグ	説明	単位
datatype	Data Type (float or complex float)	String
colsize	Column size of Vector	Integer
dim	Data Dimension (Fixed value. 1 for Vector)	Integer

表 6.130 にフォーマット文字列とその出力の例を示す..

表 6.131: `Vector<complex<float> >` 型のフォーマット文字列とその出力の例． 配列のサイズは 20 とする．

フォーマット文字列 (FILENAME パラメータ値)	**出力 (フォーマット指定されたファイル名)**
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

LoadVectorFrames を使用して SaveVectorFrames によって出力されたファイルを `Vector<ObjectRef>` 型にロードするときは，列サイズおよびデータ型をパラメータで指定する必要がある． したがって，ファイル名にデータ情報を持つことは，後の使用に役立つ．

表 6.131 に示されているように，ファイル名の拡張子に関する規則はない． ユーザーが指定する任意の値にすることができる． ファイル名の拡張子は無くても良い．

{tag：format} の ”format”の値はオプション． ”format”は，03d のような桁数を指定するために使用できる．フォーマット指定子は，C の printf と同じである．

6.7.30 SegmentAudioStreamByID

ノードの概要

ID 情報を利用した音響ストリームを切り出し，ID 情報を付加した出力を行う．

必要なファイル

なし

使用方法

どんなときに使うのか

音響信号全体を一つのストリームとして処理するのではなく，音声部分だけなど，ある部分のみ切り出して処理を際に有用なノードである．ID をキーとして切り出しを行うので，入力には ID 情報が必須である．同じ ID が続く区間を信号の区間として切り出し，ID 情報を付加して，一チャネルの `Map` データとして出力する．

典型的な接続例

入力ストリームが 2 つの音声信号の混合音であると仮定する．ユーザが GHDSS などを用いて分離した信号と時間的に同じ区間のオリジナルの混合音を比較したい場合，1 ch の音響ストリームと音源定位によって検出した音源をこのノードに入力する．この際に，出力は， GHDSS や PostFilter 分離音と完全に同じフォーマット (`Map`) で出力される．

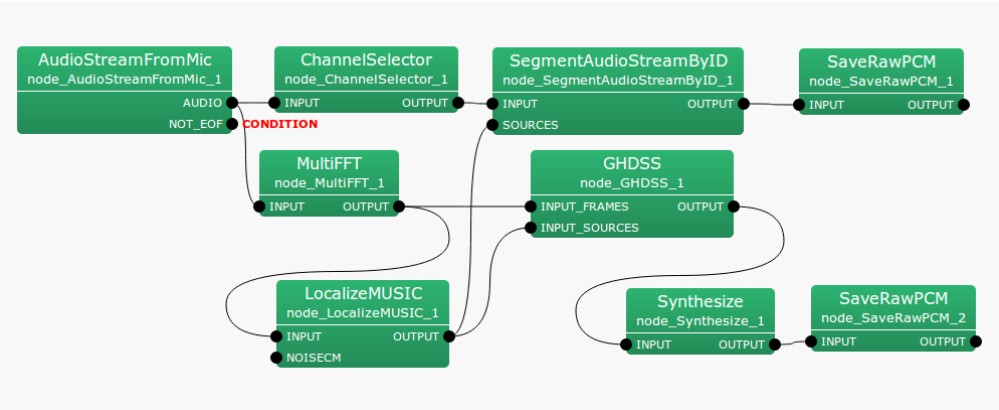


図 6.133: SegmentAudioStreamByID の接続例

ノードの入出力とプロパティ

入力

INPUT : `any` , `Matrix<complex<float> >` や `Matrix<float>` など．

SOURCES : Vector<ObjectRef> , ID 付きの音源方向．各 Vector の中身は，ID 付きの音源情報を示す Source 型になっている．特徴量ベクトルが各音源毎に格納される．このパラメータの指定は必須である．

出力

OUTPUT : Map<int, ObjectRef> , ObjectRef は，Vector<float> , Vector<complex<float> > へのスマートポインタである．

ノードの詳細

ID 情報を利用した音響ストリームを切り出し，ID 情報を付加した出力を行う．このノードは，Matrix<complex<float> > や Matrix<float> を入力で与えられた ID を用いて Map<int, ObjectRef> に変換する現状では入力として 1 ch データしかサポートしていないことに注意．

6.7.31 SourceSelectorByDirection

ノードの概要

入力された音源定位結果のうち，指定した水平・仰角方向の角度の範囲にあるもののみを通過させる，フィルタリングノード.

必要なファイル

無し.

使用方法

どんなときに使うのか

音源の方向に関する事前情報があるとき (前方にしか音源は無いと分かっている場合など) にその方向のみの定位結果を得る場合に使う．あるいは，ノイズ源の方向が分かっているときに，その方向以外を指定すれば，ノイズの定位結果を除去することも可能である.

典型的な接続例

主に，ConstantLocalization，LoadSourceLocation，LocalizeMUSIC などの音源定位結果を接続する.

図 6.134 に示す接続例は，音源定位結果のログファイルのうち，指定した範囲の方向のみを取り出すネットワークである.

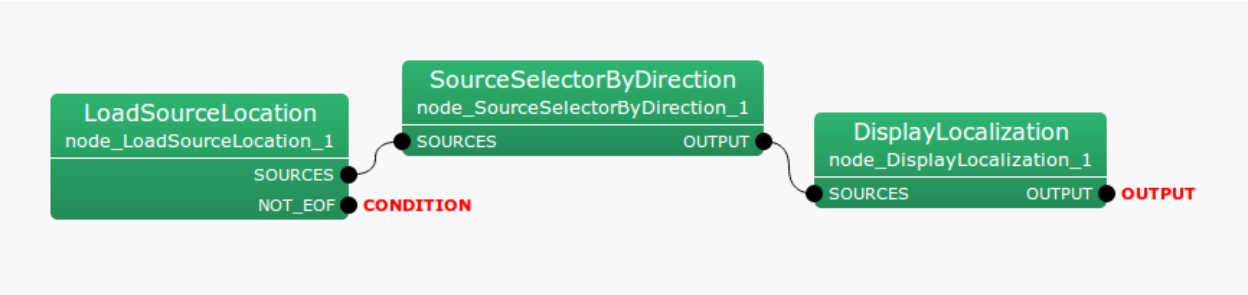


図 6.134: SourceSelectorByDirection の接続例: これは Iterator サブネットワーク

ノードの入出力とプロパティ

入力

SOURCES : `Vector<ObjectRef>` 型．入力となる音源定位結果を接続する．`ObjectRef` が参照するのは，`Source` 型のデータである.

出力

OUTPUT : `Vector<ObjectRef>` 型．フィルタリングされた後の音源定位結果を意味する．`ObjectRef` が参照するのは，`Source` 型のデータである．

パラメータ

表 6.132: SourceSelectorByDirection のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
MIN_AZIMUTH	`float`	-20.0	[deg]	通過させる音源方位角の最小値.
MAX_AZIMUTH	`float`	20.0	[deg]	通過させる音源方位角の最大値.
MIN_ELEVATION	`float`	-90.0	[deg]	通過させる音源仰角の最小値.
MAX_ELEVATION	`float`	90.0	[deg]	通過させる音源仰角の最大値.

MIN_AZIMUTH，MAX_AZIMUTH : `float` 型．角度は通過させる音源の左右方向 (方位角) を表す．角度 θ[deg] が $\theta \in [-180, 180]$ の範囲に無ければ, 範囲に入るように 360 を加算/減算される。

MIN_ELEVATION，MAX_ELEVATION : `float` 型．角度は通過させる音源の上下方向 (仰角) を表す．仰角 ϕ[deg] は $-90 \leq \phi \leq 90$ を満たす必要がある．

ノードの詳細

ビームフォーマなどのマイクロホンアレイ信号処理による空間フィルタリングをするわけではなく，あくまで定位結果の音源方向の情報を元にフィルタリングを行う．

6.7.32 SourceSelectorByID

ノードの概要

複数の音源分離結果のうち，ID が指定した値以上のものだけを出力させたいときに用いる．

必要なファイル

無し．

使用方法

どんなときに使うのか

例えば，ノイズ (ファンの音など) が定常かつ既知，という条件下で音源分離をする場合など，定常ノイズが存在することがわかっている場合に，その ID を ConstantLocalization で指定しておくことで定常ノイズ以外の分離音だけを取り出すことができる．

具体的には, ConstantLocalization ノードでは，音源の ID をそのプロパティである MIN_ID でコントロールすることができるので, 定常ノイズの音源数が 1 の時に, ConstantLocalization の MIN_ID を 0 に設定しておけば，定常ノイズの ID を 0 に固定できる．

この ConstantLocalization を LocalizeMUSIC , SourceTracker , CombineSource の各ノードと 図 6.135 に示すように組み合わせて，GHDSS ノードに接続すれば，常に定常ノイズに対応する分離音の ID を 0 にすることができる．

このとき， GHDSS ノードの後に SourceSelectorByID を接続し， SourceSelectorByID の MIN_ID を 1 に設定しておけば，ID が 0 の音源は無視されるので，定常ノイズ以外の分離音に対応する音源だけを選択することができる．

典型的な接続例

図 6.135 に接続例を示す．図に示すように，このノードは，典型的には GHDSS の後段に接続される．

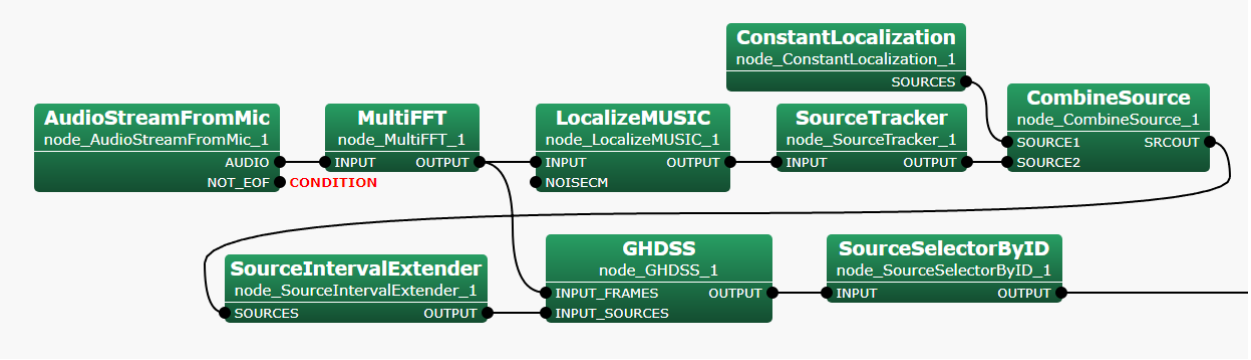


図 6.135: SourceSelectorByID の接続例: これは Iterator サブネットワーク

ノードの入出力とプロパティ

入力

INPUT : Map<int, ObjectRef> 型．通常は音源分離ノードの後段に接続されるので，Map のキーになる int には音源 ID が対応する．ObjectRef は分離を表す Vector<float> 型 (パワースペクトル) か Vector<complex<float>> 型 (複素スペクトル) である．

出力

OUTPUT : Map<int, ObjectRef> 型．音源 ID が MIN_ID より大きいデータだけを抽出したデータが出力される．Map の内容は INPUT と同じになる．

パラメータ

表 6.133: SourceSelectorByID のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
MIN_ID	int	0		この値より大きい ID の音源は通す．

MIN_ID : int 型．このパラメータ値以上の音源 ID を持つ分離音を通過．デフォルト値は 0.

6.7.33 SourceSelectorBySourceInfo

ノードの概要

入力された音源定位結果のうち，その情報（ID，パワー，方位角，仰角）に基づきパラメータで指定した条件を満たすものだけを通過させる，フィルタリングノード．複数音源入力時に音源の ID，パワーに対してその最小，最大，値の範囲を指定して出力することができる．また音源の方向に関しては，複数音源のうち指定した角度方向に最も近い音源を出力したり，方向の範囲を角度で指定し，その範囲内にある音源のみを出力することができる．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源の ID，パワー，方向に関する事前情報があるとき (前方にしか音源は無いと分かっている場合など) にその定位結果を得る場合に使う．あるいは，除去したい音源の事前情報が分かっているときに，その定位結果を除去することも可能である．

典型的な接続例

図 6.136 に接続例を示す．主に，ConstantLocalization，LoadSourceLocation，LocalizeMUSIC などの出力である音源定位結果を入力に接続する．

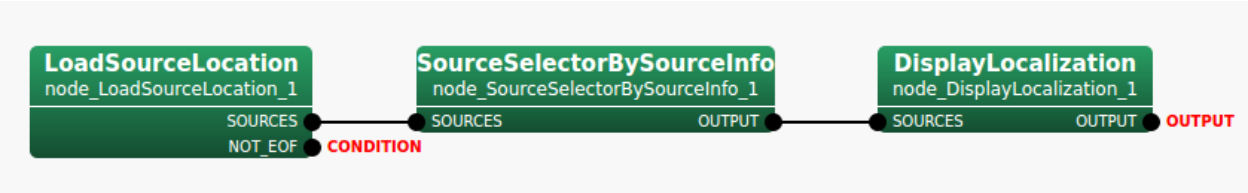


図 6.136: SourceSelectorBySourceInfo の接続例: これは Iterator サブネットワーク

ノードの入出力とプロパティ

入力

SOURCES : `Vector<ObjectRef>` 型．入力となる音源定位結果を接続する．`ObjectRef` が参照するのは，ID 付きの音源情報を示す `Source` 型のデータである．

出力

OUTPUT : Vector<ObjectRef> 型．フィルタリングされた後の音源定位結果を意味する．ObjectRef が参照するのは，ID 付きの音源情報を示す Source 型のデータである．

パラメータ

表 6.134: SourceSelectorBySourceInfo のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
SELECTION_TYPE	string	ID		音源の選択条件の種類．ALL,ID,POWER,DIRECTION_AZIMUTH,DIRECTIO から選択．
ID_SELECTION_TYPE	string	LATEST		音源を ID により選択する場合の選択条件．LATEST，OLDEST，BETWEEN から選択．
ID_RANGE_MIN	int	0		選択する音源の ID の下限値．
ID_RANGE_MAX	int	0		選択する音源の ID の上限値．
POWER_SELECTION_TYPE	string	HIGHEST		音源をパワーにより選択する場合の選択条件．HIGHEST，LOWEST，BETWEEN から選択．
POWER_RANGE_MIN	float	0		選択する音源のパワーの下限値．
POWER_RANGE_MAX	float	40.0		選択する音源のパワーの上限値．
AZIMUTH_SELECTION_TYPE	string	BETWEEN		音源を方位角により選択する場合の選択条件．NEAREST，BETWEEN から選択．
AZIMUTH	float	0	[deg]	選択する音源の近傍の方位角．
AZIMUTH_RANGE_MIN	float	0	[deg]	選択する音源の方位角の下限値．
AZIMUTH_RANGE_MAX	float	360.0	[deg]	選択する音源の方位角の上限値．
ELEVATION_SELECTION_TYPE	string	BETWEEN		音源を仰角により選択する場合の選択条件．NEAREST，BETWEEN から選択．
ELEVATION	float	0	[deg]	選択する音源の近傍の仰角．
ELEVATION_RANGE_MIN	float	0	[deg]	選択する音源の仰角の下限値．
ELEVATION_RANGE_MAX	float	90.0	[deg]	選択する音源の仰角の上限値．

SELECTION_TYPE : string 型．選択条件の種類を ALL，ID，POWER，DIRECTION_AZIMUTH,DIRECTION_ELEVATION から選択する．ALL の場合はすべてのすべての結果を出力する．ID の場合は音源の ID ，POWER の場合は音源のパワー， DIRECTION_AZIMUTH,DIRECTION_ELEVATION の場合は音源の方向，についてそれぞれのパラメータで指定した条件を満たす結果を出力する．

ID_SELECTION_TYPE : string 型．SELECTION_TYPE で ID を選択した場合に，その選択条件を LATEST，OLDEST，BETWEEN から選択する．LATEST では最も新しい音源が， OLDEST では最も古い音源が，BETWEEN ではパラメータ ID_RANGE_MIN と ID_RANGE_MAX で指定された範囲の ID の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

ID_RANGE_MIN : int 型．ID_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の ID の下限値を指定する．

ID_RANGE_MAX : int 型．ID_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の ID の上限値を指定する．

POWER_SELECTION_TYPE : `string` 型．SELECTION_TYPE で POWER を選択した場合に，その選択条件を HIGHEST，LOWEST，BETWEEN から選択する．HIGHEST では 最もパワーの大きな音源が，LOWEST では最もパワーの小さな音源が，BETWEEN ではパラメータ POWER_RANGE_MIN と POWER_RANGE_MAX で指定された範囲のパワーの音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

POWER_RANGE_MIN : `float` 型．POWER_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源のパワーの下限値を指定する．

POWER_RANGE_MAX : `float` 型．POWER_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源のパワーの上限値を指定する．

AZIMUTH_SELECTION_TYPE : `string` 型．SELECTION_TYPE で DIRECTION_AZIMUTH を選択した場合に，その選択条件を NEAREST，BETWEEN から選択する．NEAREST ではパラメータ AZIMUTH に指定された方位角に最も近い音源が，BETWEEN ではパラメータ AZIMUTH_RANGE_MIN と AZIMUTH_RANGE_MAX で指定された方位角の範囲の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

AZIMUTH : `float` 型．AZIMUTH_SELECTION_TYPE で NEAREST を選択した場合に，ここで指定した方位角にもっとも近い音源が出力される．

AZIMUTH_RANGE_MIN : `float` 型．AZIMUTH_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の方位角の下限値を指定する．

AZIMUTH_RANGE_MAX : `float` 型．AZIMUTH_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の方位角の上限値を指定する．

ELEVATION_SELECTION_TYPE : `string` 型．SELECTION_TYPE で DIRECTION_ELEVATION を選択した場合に，その選択条件を NEAREST，BETWEEN から選択する．NEAREST ではパラメータ ELEVATION に指定された仰角に最も近い音源が，BETWEEN ではパラメータ ELEVATION_RANGE_MIN と ELEVATION_RANGE_MAX で指定された仰角の範囲の音源が，出力される．複数の音源がこの条件を満たす場合は複数の音源の結果が出力される．

ELEVATION : `float` 型．ELEVATION_SELECTION_TYPE で NEAREST を選択した場合に，ここで指定した仰角にもっとも近い音源が出力される．

ELEVATION_RANGE_MIN : `float` 型．ELEVATION_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の仰角の下限値を指定する．

ELEVATION_RANGE_MAX : `float` 型．ELEVATION_SELECTION_TYPE で BETWEEN を選択した場合に，出力される音源の仰角の上限値を指定する．

6.7.34 SourceTransformer

ノードの概要

入力された音源定位結果の情報（ID，パワー，方位角，仰角）の値を操作する．`ObjectRef` 型で指定された値で一定値にすることや，加減乗除することが可能．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源定位結果の ID，パワー，方位角，仰角の値を操作したい場合に使う．各値を指定した一定値にする，指定した値で加減乗除する，等を行う．

典型的な接続例

図 6.137 に接続例を示す．主に，ConstantLocalization，LoadSourceLocation，LocalizeMUSIC などの音源定位結果を接続する．

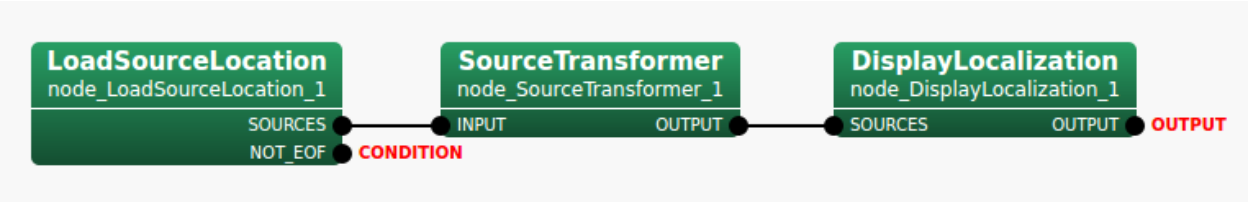


図 6.137: SourceTransformer の接続例: これは Iterator サブネットワーク

ノードの入出力とプロパティ

入力

INPUT ：`Vector<ObjectRef>` 型．入力となる音源定位結果を接続する．`ObjectRef` が参照するのは，ID 付きの音源情報を示す `Source` 型のデータである．

出力

OUTPUT ：`Vector<ObjectRef>` 型．変換された後の音源定位結果．`ObjectRef` が参照するのは，ID 付きの音源情報を示す `Source` 型のデータである．

パラメータ

表 6.135: SourceTransformer のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
ENABLE_ID	bool	false		音源定位結果の ID の編集の可否.
ID_OPERATION_TYPE	string	CONST		音源定位結果の ID の編集の種類（CONST, ADD, SUB, MUL, DIV）.
ID	Object	Vector<int>		音源定位結果の ID の編集に使用する値.
ENABLE_POWER	bool	false		音源定位結果のパワーの編集の可否.
POWER_OPERATION_TYPE	string	CONST		音源定位結果のパワーの編集の種類（CONST, ADD, SUB, MUL, DIV）.
POWER	Object	Vector<float>		音源定位結果のパワーの編集に使用する値.
ENABLE_AZIMUTH	bool	false		音源定位結果の方位角の編集の可否.
AZIMUTH_OPERATION_TYPE	string	CONST		音源定位結果の方位角の編集の種類（CONST, ADD, SUB, MUL, DIV）.
AZIMUTH	Object	Vector<float>		音源定位結果の方位角の編集に使用する値.
ENABLE_ELEVATION	bool	false		音源定位結果の仰角の編集の可否.
ELEVATION_OPERATION_TYPE	string	CONST		音源定位結果の仰角の編集の種類（CONST, ADD, SUB, MUL, DIV）.
ELEVATION	Object	Vector<float>		音源定位結果の仰角の編集に使用する値.

ENABLE_ID : bool 型．true を選択すると，音源定位結果の ID の編集が可能になる．デフォルトは false.

ID_OPERATION_TYPE : string 型．ENABLE_ID に true を指定した場合に，編集の種類を指定する．CONST はパラメータ ID で指定した値で置き換え，ADD,SUB,MUL,DIV は，それぞれ，パラメータ ID で指定した値で加減乗除する．

ID : Object 型．ENABLE_ID に true を指定した場合に，その編集に使用する値を指定する．要素が 1 つしかない場合は、その値を使用して全音源定位結果を操作する. 各音源定位結果ごとに値を指定したい場合は， Vector<int> で指定する. その場合， 要素数が入力の音源定位結果より少なければそれ以降は最後の値が使用される．

ENABLE_POWER : bool 型．true を選択すると，音源定位結果のパワーの編集が可能になる．デフォルトは false.

POWER_OPERATION_TYPE : string 型．ENABLE_POWER に true を指定した場合に，編集の種類を指定する．CONST はパラメータ POWER で指定した値で置き換え，ADD,SUB,MUL,DIV は，それぞれ，パラメータ POWER で指定した値で加減乗除する．

POWER : Object 型．ENABLE_POWER に true を指定した場合に，その編集に使用する値を指定する．要素が 1 つしかない場合は、その値を使用して全音源定位結果を操作する. 各音源定位結果ごとに値を指定したい場合は， Vector<float> で指定する. その場合， 要素数が入力の音源定位結果より少なければそれ以降は最後の値が使用される．

ENABLE_AZIMUTH : bool 型．true を選択すると，音源定位結果の方位角の編集が可能になる．デフォルトは false.

AZIMUTH_OPERATION_TYPE : string 型．ENABLE_AZIMUTH に true を指定した場合に，編集の種類を指定する．CONST はパラメータ AZIMUTH で指定した値で置き換え，ADD,SUB,MUL,DIV は，それぞれ，パラメータ AZIMUTH で指定した値で加減乗除する．

AZIMUTH : Object 型．ENABLE_AZIMUTH に true を指定した場合に，その編集に使用する値を指定する．要素が 1 つしかない場合は、その値を使用して全音源定位結果を操作する. 各音源定位結果ごとに値を指定したい場合は， Vector<float> で指定する. その場合， 要素数が入力の音源定位結果より少なければそれ以降は最後の値が使用される．

ENABLE_ELEVATION : bool 型．true を選択すると，音源定位結果の仰角の編集が可能になる．デフォルトは false.

ELEVATION_OPERATION_TYPE : string 型．ENABLE_ELEVATION true を指定した場合に，その編集に使用する値を指定する．CONST はパラメータ ELEVATION で指定した値で置き換え，ADD,SUB,MUL,DIV は，それぞれ，パラメータ ELEVATION で指定した値で加減乗除する．

ELEVATION : Object 型．ENABLE_ELEVATION に true を指定した場合に，その編集に使用する値を指定する．要素が 1 つしかない場合は、その値を使用して全音源定位結果を操作する. 各音源定位結果ごとに値を指定したい場合は， Vector<float> で指定する. その場合， 要素数が入力の音源定位結果より少なければそれ以降は最後の値が使用される．

6.7.35 Synthesize

ノードの概要

周波数領域の信号を時間領域の波形に変換する.

必要なファイル

無し.

使用方法

周波数領域の信号を時間領域の波形に変換する際に用いる.

典型的な接続例

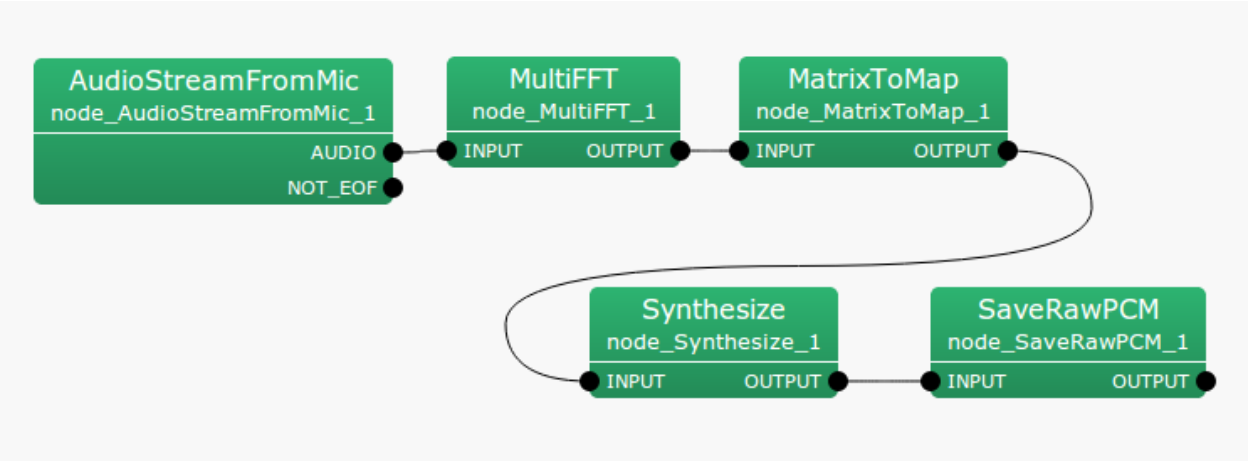


図 6.138: Synthesize の接続例

ノードの入出力とプロパティ

表 6.136: Synthesize のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	FFT 長
ADVANCE	int	160	[pt]	シフト長
SAMPLING_RATE	int	16000	[Hz]	サンプリングレート
MIN_FREQUENCY	int	125	[Hz]	最小周波数
MAX_FREQUENCY	int	7900	[Hz]	最大周波数
WINDOW	string	HAMMING		窓関数
OUTPUT_GAIN	float	1.0		出力ゲイン

入力

INPUT : `Map<int, ObjectRef>` 型． `ObjectRef` は `Vector<complex<float> >` .

出力

OUTPUT : `Map<int, ObjectRef>` 型．`ObjectRef` は `Vector<float>` .

パラメータ

LENGTH FFT 長，他のノード（ MultiFFT ）と値を合わせる必要がある．

ADVANCE シフト長，他のノード（ MultiFFT ）と値を合わせる必要がある．

SAMPLING_RATE サンプリングレート，他のノードと値を合わせる必要がある．

MIN_FREQUENCY 波形生成時に用いる最小周波数値

MAX_FREQUENCY 波形生成時に用いる最大周波数値

WINDOW 窓関数，HAMMING，RECTANGLE, CONJ から選択

OUTPUT_GAIN 出力ゲイン

ノードの詳細

入力された周波数領域の信号に対して，低域 4 バンド分，および，$\omega_s/2 - 100$ [Hz] 以上の周波数ビンについては 0 を代入したのち逆 FFT を適用する．次に，指定された窓をかけ，overlap-add 処理を行う．overlap-add 処理は，フレーム毎に逆変換を行い，時間領域の戻した信号をずらしながら加算することにより，窓の影響を軽減する手法である．詳細は，参考文献で挙げている web ページを参照すること．最後に，得られた時間波形に出力ゲインを乗じて，出力する．

なお，overlap-add 処理を行うために，フレームの先読みをする必要があり，結果として，このノードは処理系全体に遅延をもたらす．遅延の大きさは，下記で計算できる．

$$delay = \begin{cases} |\mathrm{LENGTH}/\mathrm{ADVANCE}| - 1, & \text{if } \mathrm{LENGTH}\ mod\ \mathrm{ADVANCE} \neq 0, \\ \mathrm{LENGTH}/\mathrm{ADVANCE}, & \text{otherwise.} \end{cases} \tag{6.170}$$

HARK のデフォルトの設定では，LENGTH = 512, ADVANCE = 160 であるので，遅延は 3 [frame]，つまり，システム全体に与える遅延は 30 [ms] となる．

参考文献

(1) http://en.wikipedia.org/wiki/Overlap-add

6.7.36 TextConcatenate

ノードの概要

String 型の文字列を連結する．

必要なファイル

無し．

使用方法

どんなときに使うのか

複数の string 型テキストの入力を連結し、1 つの string 型テキストとして出力します。つまり， TextConverter が生成する JSON テキスト，または Constant のパラメータを string 型に設定して生成した文字列などを入力として扱うことができます．出力は HarkDataStreamSender や HARK-Python の PyCodeExecuter などの string 型の入力に対応したノードに接続して用いる．

典型的な接続例

TextConcatenate ノードの接続例を図 6.139 に示す．

図 6.139 は，Constant ノードと TextConverter ノードで生成した文字列を結合して，複数の音源情報を HarkDataStreamSender で送信する例．

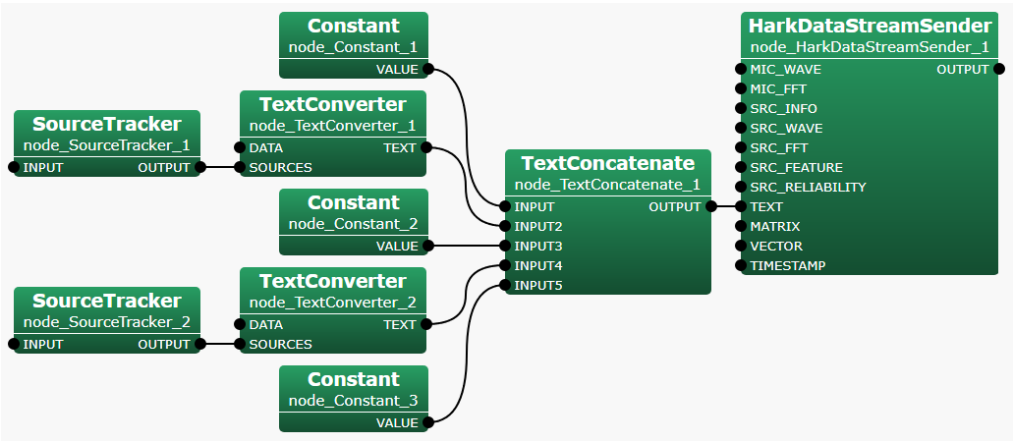


図 6.139: TextConcatenate の接続例 – Constant および TextConverter との接続

ノードの入出力とプロパティ

入力

INPUT ：string 型．入力端子をいくつでも追加することができる．

出力

OUTPUT : string 型.

パラメータ

表 6.137: MatrixToMap パラメータ表

パラメータ名	型	デフォルト値	単位	説明
SEPARATOR	string			テキストの区切り文字列の指定．各 INPUT の間に指定した文字列が挿入される．
ENABLE_DEBUG	bool	false		連結文字列を標準出力に出力するかどうかを選択．

SEPARATOR : string 型．テキストの区切り文字列を指定する．各 INPUT の間に指定した文字列が挿入される．デフォルト値の空欄ではセパレータなしとなり，空白を入力するとセパレータに空白文字が使用される．

ENABLE_DEBUG : bool 型．デフォルトは false．true が与えられると, 連結文字列が標準出力に出力される．

ノードの詳細

＜例＞

```
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}}}
OUTPUT:   , {'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}}}]"
```

注: 実際の文字列に改行とダブルクォートは含まれない.

6.7.37 TextConverter

ノードの概要

HARK がサポートする任意の型から `string` 型の JSON テキストに変換を行う

必要なファイル

無し.

使用方法

どんなときに使うのか

このノードは 型の入力を受け入れる事が出来るノード (例えば HarkDataStreamSender や HARK-Python の PyCodeExecuter) に接続するために用いる．このノードは HARK がサポートする任意の型を JSON テキストとして変換する事が出来るので JSON テキストを理解できる言語 (例えば Python や JavaScript など) で受信する際に扱いやすくする事が出来る．例えば， HARK-Python の PyCodeExecuter では 型で受信したデータを処理するスクリプトで `'import json'` と `'json.load()'` を使用する事で容易に Python オブジェクトへ変換する事が出来る．なお， HARK に存在する多くのノードの出力をこのノードの入力に与える事が出来るが，例外が存在するので注意されたい．これは今後のアップデートで改善する事がある．

典型的な接続例

TextConverter ノードの接続例を図 6.140，6.141 に示す.

図 6.140 は，音源定位結果と分離音を HarkDataStreamSender ノードで送信するネットワークである．node_HarkDataStreamSend は node_HarkDataStreamSender_1 で送信している情報を TextConverter ノードで JSON テキストに変換してから送信する場合の接続例である.

図 6.141 は，音源定位結果と分離音を HARK-Python の PyCodeExecuter ノードに送るための接続である.

ノードの入出力とプロパティ

入力

DATA : `any` . ただし，サポートする型は 表 6.139 を参照．入力端子をいくつでも追加することができる．入力端子の名前は出力する JSON オブジェクトの名前 (マップ、ハッシュ、連想配列などにおけるキー) として使用される.

出力

TEXT : `string` 型 .

パラメータ

ENABLE_DEBUG : `bool` 型．デフォルトは `false`．`true` が与えられると, 変換後の JSON テキストが標準出力に出力される.

ノードの詳細

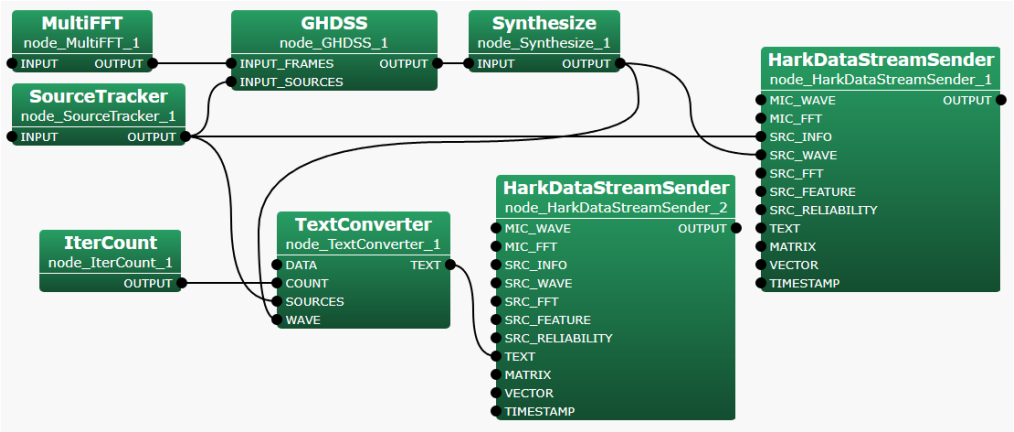


図 6.140: TextConverter の接続例 – HarkDataStreamSender への接続

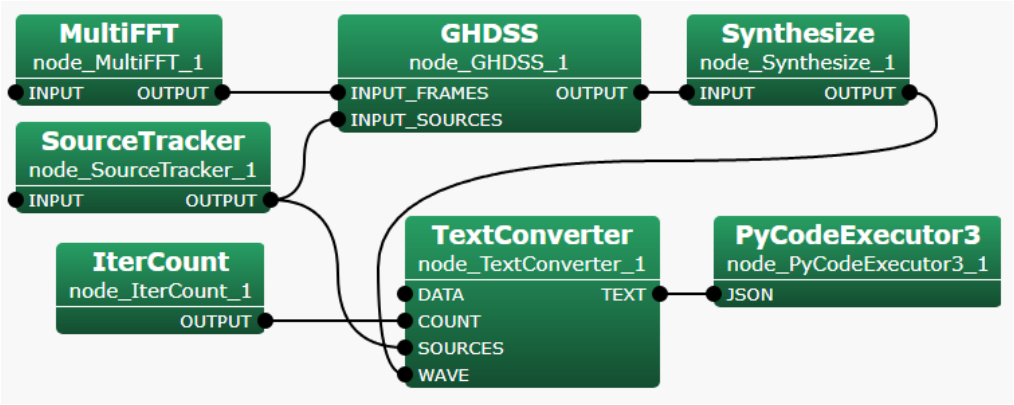


図 6.141: TextConverter の接続例 – HARK-Python の PyCodeExecuter への接続

表 6.138: MatrixToMap パラメータ表

パラメータ名	型	デフォルト値	単位	説明
ENABLE_DEBUG	bool	false		変換された JSON テキストを標準出力に出力するかどうかの選択.

表 6.139: TextConverter 变换表

INPUT	OUTPUT
input object type	output object type
input example	output result
bool	string
true	{'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
$\left\langle \begin{matrix} 1 & -2 & 3 & -4 & 5 & -6 \end{matrix} \right\rangle$	{'DATA': [1, -2, 3, -4, 5, -6]}
Vector<float>	string
$\left\langle \begin{matrix} 1.1 & -2.2 & 3.3 & -4.4 & 5.5 & -6.6 \end{matrix} \right\rangle$	{'DATA': [1.1, -2.2, 3.3, -4.4, 5.5, -6.6]}
Vector<complex<float> >	string
$\left\langle \begin{matrix} 1.1 - 2.2i & 3.3 + 4.4i & -5.5 - 6.6i \end{matrix} \right\rangle$	{'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
$\left\{0, \left\langle \begin{matrix} 1 & -2 & 3 & -4 & 5 & -6 \end{matrix} \right\rangle\right\}$	{'DATA': {0: [1, -2, 3, -4, 5, -6]}}
Map<int , Vector<float> >	string
$\left\{0, \left\langle \begin{matrix} 1.1 & -2.2 & 3.3 & -4.4 & 5.5 & -6.6 \end{matrix} \right\rangle\right\}$	{'DATA': {0: [1.1, -2.2, 3.3, -4.4, 5.5, -6.6]}}
Map<int , Vector<complex<float> > >	string
$\left\{0, \left\langle \begin{matrix} 1.1 - 2.2i & 3.3 + 4.4i & 5.5 - 6.6i \end{matrix} \right\rangle\right\}$	{'DATA': {0: [(1.1, -2.2), (3.3, 4.4), (5.5, -6.6)]}}
Map<int , Matrix<int> >	string
$\left\{0, \begin{bmatrix} 1 & -2 \\ -3 & 4 \\ 5 & 6 \end{bmatrix}\right\}$	{'DATA': {0: [[1, -2], [-3, 4], [5, 6]]}}
Map<int , Matrix<float> >	string
$\left\{0, \begin{bmatrix} 1.1 & -2.2 \\ -3.3 & 4.4 \\ 5.5 & 6.6 \end{bmatrix}\right\}$	{'DATA': {0: [[1.1, -2.2], [-3.3, 4.4], [5.5, 6.6]]}}
Map<int , Matrix<complex<float> > >	string
$\left\{0, \begin{bmatrix} 1.1 - 2.2i & 3.3 + 4.4i \\ -5.5 + 6.6i & -7.7 - 8.8i \end{bmatrix}\right\}$	{'DATA': {0: [[(1.1, -2.2), (3.3, 4.4)], [(-5.5, 6.6), (-7.7, -8.8)]]}}

6.7.38 VectorToMap

ノードの概要

Vector<float> または Vector<complex<float> > 型と Map<int, ObjectRef> 型の変換を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

Vector<float> または Vector<complex<float> > 型 を Map<int, ObjectRef> 型に変換する際に用いる．Vector<float> 型から Map<int , Vector<float> > 型へ, または, Vector<complex<float> > 型から Map<int , Vector<complex<float> > > 型へ変換される．

ノードの入出力とプロパティ

入力

INPUT : any ．ただし，サポートする型は Vector<float> または Vector<complex<float> > 型.

出力

OUTPUT : Map<int, ObjectRef> 型の，Map<int , Vector<float> > または Map<int , Vector<complex<float> > > 型

パラメータ

表 6.140: VectorToMap パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OUTPUT_TYPE	string	map_of_vector		出力される型を指定．キーに 0, 値 ObjectRef に入力の Vector が出力される map_of_vector, キーに入力の Vector のインデクス，値に入力の Vector の各要素が出力される map_of_vectors から 選択.
DEBUG	bool	false		変換状況を出力するかどうかの選択.

OUTPUT_TYPE : string 型．出力される型を指定する．キーに 0, 値 ObjectRef に入力の Vector が出力される「map_of_vector」，キーに入力の Vector のインデクス，値に入力の Vector の各要素が出力される「map_of_vectors」から選択する．デフォルトは map_of_vector

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

表 6.141: VectorToMap 変換表

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	

ノードの詳細

＜例＞

INPUT:

$$< 1 \quad 2 \quad 3 \quad 4 \quad 5 >$$

OUTPUT(1):

$$\{ 0, \quad < 1 \quad 2 \quad 3 \quad 4 \quad 5 > \}$$

OUTPUT(2):

$$\{ 0, \quad < 1 > \}, \quad \{ 1, \quad < 2 > \}, \quad \{ 2, \quad < 3 > \}, \quad \{ 3, \quad < 4 > \}, \quad \{ 4, \quad < 5 > \}$$

6.7.39 VectorToMatrix

ノードの概要

Vector<float> 型から Matrix<float> 型へ，または，Vector<complex<float> > 型から Matrix<complex<float> > 型へ変換を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

Vector<float> 型から Matrix<float> 型へ，または，Vector<complex<float> > 型から Matrix<complex<float> > 型へ変換する際に用いる．

ノードの入出力とプロパティ

入力

INPUT : any ．ただし，サポートする型は Vector<float> または Vector<complex<float> > 型．

出力

OUTPUT : any ．ただし，サポートする型は Matrix<float> または Matrix<complex<float> > 型．

パラメータ

表 6.142: VectorToMatrix パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OUTPUT_TYPE	string	row_vector		行ベクトルを出力するか列ベクトルを出力するかを指定．row_vector, column_vector から選択．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

OUTPUT_TYPE : string 型．行ベクトルを出力するか列ベクトルを出力するかを指定する．row_vector, column_vector から選択．入力 Vector のサイズが N の場合，row_vector が選択されると，1 行 N 列の Matrix に, coulmn_vector が選択されると，N 行 1 列の Matrix になる．デフォルト値は row_vector.

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

表 6.143: VectorToMatrix 変換表

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	

ノードの詳細

＜例＞

INPUT:

$$< 1 \quad 2 \quad 3 \quad 4 \quad 5 >$$

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

ノードの概要

Vector<float> 型と Vector<complex<float> > 型の変換を行う．

必要なファイル

無し．

使用方法

どんなときに使うのか

Vector<float> 型から Vector<complex<float> > 型へ，Vector<complex<float> > 型から Vector<float> 型へ変換する際に用いる．

ノードの入出力とプロパティ

入力

INPUT ：any ．ただし，サポートする型は Vector<float> または Vector<complex<float> > 型．

出力

OUTPUT ：any ．ただし，サポートする型は Vector<float> または Vector<complex<float> > 型．

パラメータ

表 6.144: VectorToVector パラメータ表

パラメータ名	型	デフォルト値	単位	説明
METHOD_COMPLEX_TO_FLOAT	string	magnitude		Vector<complex<float> > 型から Vector<float> 型への変換方法．INPUT の複素数の，絶対値が出力される「magnitude」，実部が出力される「real」，虚部が出力される「imaginary」から選択する．
METHOD_FLOAT_TO_COMPLEX	string	zero		Vector<float> 型から Vector<complex<float> > 型への変換方法．複素数の虚部に，0 が出力される「zero」，実部の絶対値が出力される「helbert」から選択する．
DEBUG	bool	false		変換状況を出力するかどうかの選択．

METHOD_COMPLEX_TO_FLOAT ：string 型．Vector<complex<float> > 型から Vector<float> 型への変換方法を指定する．INPUT の複素数の，絶対値が出力される「magnitude」，実部が出力される「real」，虚部が出力される「imaginary」から選択する．デフォルトは magnitude．

METHOD_FLOAT_TO_COMPLEX : string 型．Vector<float> 型から Vector<complex<float> > 型への変換方法を指定する．複素数の虚部に， 0 が出力される「zero」，実部の絶対値が出力される「helbert」から選択する．デフォルトは zero.

DEBUG : bool 型．true が与えられると, 変換状況が標準出力に出力される．デフォルトは false.

ノードの詳細

表 6.145: VectorToVector 変換表

INPUT	OUTPUT	使用するパラメータ
Vector<float>	Vector<complex<float> >	METHOD_FLOAT_TO_COMPLEX
Vector<complex<float> >	Vector<float>	METHOD_COMPLEX_TO_FLOAT

6.7.41 VectorValueOverwrite

ノードの概要

Vector<ObjectRef> の一部分の要素を指定した値で置き換える．

必要なファイル

無し．

使用方法

どんなときに使うのか

Vector<ObjectRef> 型の Vector<int> または Vector<float> または Vector<complex<float> > の一部分の要素の値を，指定した値で置き換える．指定した値は Vector<ObjectRef> の ObjectRef の型に応じて変換される．

ノードの入出力とプロパティ

入力

INPUT ：any．ただし，サポートする型は Vector<int> または Vector<float> または Vector<complex<float> > 型．

出力

OUTPUT ：any．ただし，サポートする型は Vector<int> または Vector<float> または Vector<complex<float> > 型．

パラメータ

OVERWRITTEN_MIN ：int 型．置き換える Vector<ObjectRef> 要素の開始インデクスを指定する．デフォルトは 0.

OVERWRITTEN_MAX ：int 型．置き換える Vector<ObjectRef> 要素の終了インデクスを指定する．デフォルトは 0.

OVERWRITE_VALUE_REAL ：float 型．置き換える値を指定する．INPUT が， Vector<int> の場合は int に型変換され， Vector<complex<float> > の場合は置き換える複素数の実部となる．デフォルトは 0.

OVERWRITE_VALUE_IMAG ：float 型．置き換える複素数の虚部の値を指定する．INPUT が Vector<complex<float> > の場合のみ有効．デフォルトは 0.

DEBUG ：bool 型．true が与えられると，置換状況が標準出力に出力される．デフォルトは false.

表 6.146: VectorValueOverwrite パラメータ表

パラメータ名	型	デフォルト値	単位	説明
OVERWRITTEN_MIN	int	0		置き換えられる Vector 要素の開始インデクス.
OVERWRITTEN_MAX	int	0		置き換えられる Vector 要素の終了インデクス.
OVERWRITE_VALUE_REAL	float	0		置き換える値. INPUT が, Vector<int> の場合は int に型変換され, Vector<complex<float> > の場合は置き換える複素数の実部となる.
OVERWRITE_VALUE_IMAG	float	0		置き換える複素数の虚部の値. INPUT が Vector<complex<float> > の場合のみ有効で, Vector<int> または Vector<float> の場合は使用されない.
DEBUG	bool	false		置換状況を出力するかどうかの選択.

ノードの詳細

置換の詳細

＜例＞

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

ノードの概要

入力信号に白色ノイズを付加する．

必要なファイル

無し．

使用方法

どんなときに使うのか

分離後の非線形歪みの影響を緩和するために敢えてノイズを付加する場合に用いる．例えば，PostFilter は，非線形処理を行うため，musical ノイズの発生を避けることは難しい．このようなノイズは，音声認識性能に大きく影響する場合がある．適量の既知ノイズを加えることにより，このようなノイズの影響を低減できることが知られている．

典型的な接続例

例を図示．具体的なノード名をあげる．

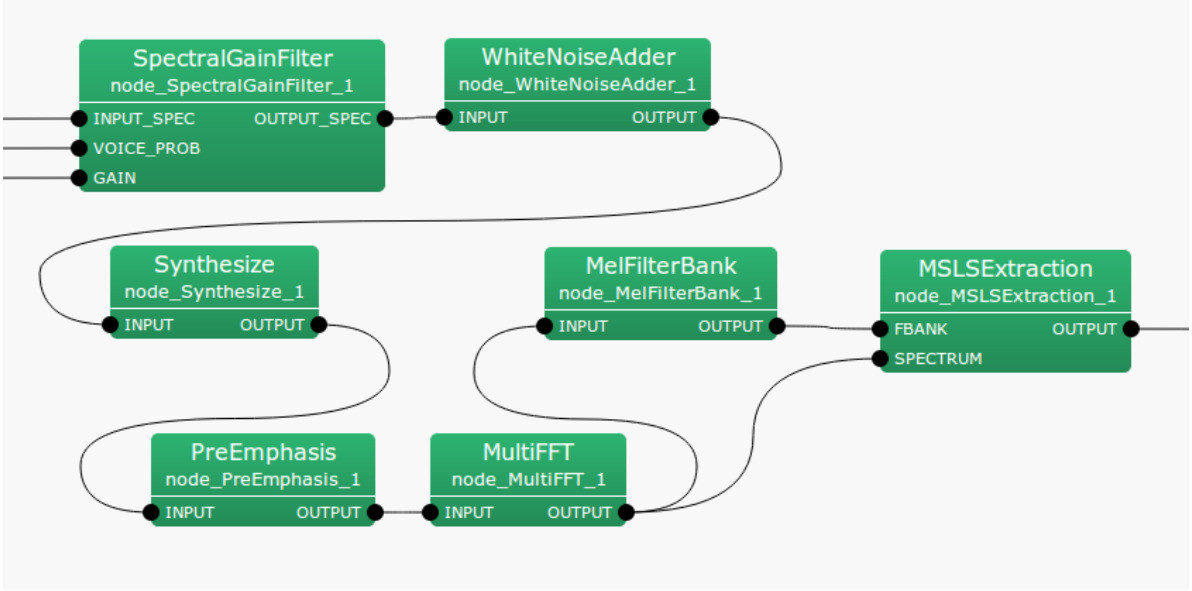


図 6.142: WhiteNoiseAdder の接続例

ノードの入出力とプロパティ

入力

表 6.147: WhiteNoiseAdder のパラメータ表

パラメータ名	型	デフォルト値	単位	説明
LENGTH	int	512	[pt]	FFT 長
WN_LEVEL	float	0		付加ノイズレベル

INPUT : Map<int, ObjectRef> 型. ObjectRef は Vector<complex<float> > であるため，周波数領域の信号を入力することが前提である.

出力

OUTPUT : Map<int, ObjectRef> 型. ObjectRef は Vector<complex<float> > である．白色ノイズが付加された信号が出力される.

パラメータ

LENGTH : FFT 長，他のノードと値を合わせる必要がある.

WN_LEVEL : 付加ノイズレベル，時間領域での最大振幅値を指定.

ノードの詳細

入力信号の各周波数ビンに対して，

$$\frac{\sqrt{\mathrm{LENGTH}}}{2} \cdot \mathrm{WN\_LEVEL} \cdot e^{2\pi jR} \tag{6.171}$$

を加算する．R は，$0 \leq R \leq 1$ となる乱数である (各周波数ビンごとに異なる値となる). $\sqrt{\mathrm{LENGTH}}/2$ は，FFT による周波数解析の際に生じる時間領域と周波数領域のスケーリングのずれを補正するための項である.

6.8 Flow Designer に依存しないモジュール

6.8.1 JuliusMFT

概要

JuliusMFT は，大語彙音声認識システム Julius を HARK 用に改造を行った音声認識モジュールである．HARK 0.1.x 系では，大語彙音声認識システム Julius 3.5 をもとに改良されたマルチバンド版 Julius[4] に対するパッチとして，提供していたが，HARK 1.0 では，Julius 4.1 系をベースに実装および機能を大きく見直した．HARK 1.0 の JuliusMFT では，オリジナルの Julius と比較して，下記の 4 点に対応するための変更を行っている．

- ミッシングフィーチャー理論の導入
- MSLS 特徴量のネットワーク入力 (mfcnet) 対応
- 音源情報 (SrcInfo) の追加に対応
- 同時発話への対応（排他処理）

実装に関しては，Julius 4.0 から導入されたプラグイン機能を用いて極力 Julius 本体に変更を加えない形で，実装を行った，

インストール方法，使用方法を説明するとともに，Julius との違い，FlowDesigner 上の HARK モジュールとの接続について，解説する．

起動と設定

JuliusMFT の実行は，例えば設定ファイル名を julius.conf とすれば，以下のように行う．

```
> julius_mft -C julius.jconf
> julius_mft.exe -C julius.jconf (Windows 版)
```

HARK では，JuliusMFT を起動したのち，IP アドレスやポート番号が正しく設定された SpeechRecognition-Client (または SpeechRecognitionSMNClient) を含んだネットワークを起動することにより，JuliusMFT とのソケット接続が行われ，音声認識が可能な状態となる．

上述の julius.jconf は JuliusMFT の設定を記述したテキストファイルである．設定ファイルの中身は，基本的に “-” で始まる引数オプションからなっており，起動時に直接，julius のオプションとして引数指定することも可能である．また，# 以降はコメントとして扱われる．Julius で用いるオプションは，http://julius.sourceforge.jp/juliusbook/ja/desc_option.html にまとめられているので，そちらを参照していただきたいが，最低限必要な設定は，以下の 7 種類である．

- -notypecheck
- -plugindir /usr/lib/julius_plugin
- -input mfcnet
- -gprune add_mask_to_safe
- -gram grammar
- -h hmmdefs

[4] http://www.furui.cs.titech.ac.jp/mband_julius/

- -hlist allTriphones

- **-notypecheck**
  特徴パラメータの型チェックをスキップする設定．オリジナルの Julius では任意で指定可能なオプションであるが，JuliusMFT では指定が必須のオプションとなっている．このオプションを指定しないとタイプチェックが行われるが，JuliusMFT のプラグインでは，特徴量と共にマスクデータを計算しているため (マスクなしの場合でも 1.0 を出力する)，タイプチェックでサイズ不一致となり，認識が行われない．
- **-plugindir プラグインディレクトリ名**
  プラグイン (*.jpi) が存在するディレクトリを指定する．引数にカレントディレクトリからの相対パス，もしくはプラグインの絶対パスを指定する．このパスは，apt-get でインストールした場合には，/usr/lib/julius_plugin，ソースコードをパス指定せずコンパイルおよびインストールした場合には/usr/local/lib/julius_plugin がデフォルトとなる．なお，このパスは，-input mfcnet や，-gprune add_mask_to_safe などプラグインで実現されている機能の指定よりも前に指定する必要がある．このパス下にある拡張プラグインファイルは，実行時に全て読み込まれるので注意．尚、Windows 版においては、プラグインを使用しないためプラグインディレクトリ名は任意となるが、本オプションは必須となる．
- **-input mfcnet**
  -input 自体はオリジナル Julius で実装されているオプションで，マイクロホン，ファイル，ネットワーク経由の入力などがサポートされている．JuliusMFT では，SpeechRecognitionClient (または，SpeechRecognitionSMNClient) から送信される音響特徴量とマスクをネットワーク経由で受信できるようにこのオプションを拡張し，音声入力ソースとして mfcnet を指定できるようにした．この機能は -input mfcnet と指定することにより，有効にすることができる．また，mfcnet 指定時のポート番号は，オリジナルの Julius で，音声入力ソース adinnet 用ポート番号を指定するために使用される-adport を用いて “-adport ポート番号” のように指定することができる．
- **-gprune**
  既存の出力確率計算にマスクを利用する場合に使用する枝刈りアルゴリズムを指定する．基本的に，HARK 0.1.x で提供していた julius_mft(ver3.5) に搭載された機能を移植したもので，$\{add\_mask\_to\_safe||add\_mask\_to\_heu||add\_m$, $ask\_to\_beam||add\_mask\_to\_none\}$ の 4 種類からアルゴリズムを選択する (指定しない場合はデフォルトの計算方法となる)．それぞれオリジナル Julius の $\{safe||heuristic||beam||none\}$ に対応している．なお，julius_mft(ver3.5) の eachgconst を用いた計算方法は厳密には正確ではないため，オリジナルと比較すると計算結果 (score) に誤差が出てしまっていた．今回，オリジナルと同様の計算方法を取り入れ，この誤差問題を解決している．
- **-gram grammar**
  言語モデルを指定する．オリジナル Julius と同様.
- **-h hmmdefs**
  音響モデル (HMM) を指定する．オリジナル Julius と同様.
- **-hlist allTriphones**
  HMMList ファイルを指定する．オリジナル Julius と同様.

なお，後述のモジュールモードで利用する際には，オリジナル Julius と同様に-module オプションを指定する必要がある．

詳細説明

mfcnet 通信仕様

mfcnet を音声入力ソースとして利用するには，上述のように，JuliusMFT 起動時に “-input mfcnet” を引数として指定する．この際，JuliusMFT は TCP/IP 通信サーバとなり，特徴量を受信する．また，HARK のモジュールである SpeechRecognitionClient や，SpeechRecognitionSMNClient は，音響特徴量とミッシングフィーチャーマスクを JuliusMFT に送出するためのクライアントして動作する．クライアントは，1 発話ごとに JuliusMFT に接続し，送信終了後ただちに接続を切断する．送信されるデータはリトルエンディアンである必要がある（ネットワークバイトオーダーでないことに注意）．具体的には，1 発話に対して以下の流れで通信を行う．

1. ソケット接続

   ソケットを開き，JuliusMFT に接続.

2. **通信初期化（最初に 1 回だけ送信するデータ）**

   クライアントから，ソケット接続直後に 1 回だけ，表 6.148 に示すこれから送信する音源に関する情報を送信する．音源情報は SourceInfo 構造体（表 6.149）で表され，音源 ID，音源方向，送信を開始した時刻を持つ．時刻は，<sys/time.h> で定義されている timeval 構造体で表し，システムのタイムゾーンにおける紀元（1970 年 1 月 1 日 00:00:00）からの経過時間である．以後，時刻は紀元からの経過時間を指すものとする．

表 6.148: 最初に 1 回だけ送信するデータ

サイズ [byte]	型	送信するデータ
4	`int`	28（= sizeof(SourceInfo))
28	SourceInfo	これから送信する特徴量の音源情報

表 6.149: SourceInfo 構造体

メンバ変数名	型	説明
source_id	`int`	音源 ID
azimuth	`float`	水平方向 [deg]
elevation	`float`	垂直方向 [deg]
time	timeval	時刻 (64 bit 処理系に統一，サイズは 16 バイト)

3. **データ送信（毎フレーム）**

   音響特徴量とミッシングフィーチャーマスクを送信する．表 6.150 に示すデータを 1 フレームとし，1 発話の特徴量を音声区間が終了するまで繰り返し送信する．特徴量ベクトルとマスクベクトルの次元数は同じ大きさであることが JuliusMFT 内部で仮定されている．

表 6.150: 毎フレーム送信するデータ

サイズ [byte]	型	送信するデータ
4	`int`	N1=（特徴量ベクトルの次元数）× sizeof(`float`)
N1	`float` [N1]	特徴量ベクトルの配列
4	`int`	N2=(マスクベクトルの次元数) × sizeof(`float`)
N2	`float` [N2]	マスクベクトルの配列

4. **終了処理**

   1 音源分の特徴量を送信し終えたら，終了を示すデータ（表 6.151）を送信してソケットを閉じる．

表 6.151: 終了を示すデータ

サイズ [byte]	型	送信するデータ
4	int	0

モジュールモード通信仕様　-module を指定するとオリジナル Julius と同様にモジュールモードで動作させることができる．モジュールモードでは， JuliusMFT が TCP/IP 通信のサーバとして機能し， JuliusMFT の状態や認識結果を jcontrol などのクライアントに提供する．また，コマンドを送信することにより動作を変更することができる．日本語文字列の文字コードは，通常 EUC-JP を利用しており，引数によって変更可能である．データ表現には XML ライクな形式が用いられており，一つのメッセージごとにデータの終了を表す目印として ”.”（ピリオド）が送信される．JuliusMFT で送信される代表的なタグの意味は以下の通りである．

- INPUT タグ入力に関する情報を表すタグで，属性として STATUS と TIME がある．STATUS の値は LISTEN，STARTREC，ENDREC のいずれかの状態をとる． LISTEN のときは Julius が音声を受信する準備が整ったことを表す．STARTREC は特徴量の受信を開始したことを表す．ENDREC は受信中の音源の最後の特徴量を受信したことを表す．TIME はそのときの時刻を表す．
- SOURCEINFO タグ音源に関する情報を表す，JuliusMFT オリジナルのタグである．属性として ID，AZIMUTH，ELEVATION，SEC，USEC がある．SOURCEINFO タグは第 2 パス開始時に送信される．ID は HARK で付与した音源 ID（話者 ID ではなく，各音源に一意に振られた番号）を，AZIMUTH は音源の最初のフレームのときのマイクロホンアレー座標系からみた水平方向（度）を，ELEVATION は同垂直方向（度）を，SEC と USEC は音源の最初のフレームの時刻を表し SEC が秒の位，USEC がマイクロ秒の位を表す．
- RECOGOUT タグ認識結果を表すタグで，子要素は漸次出力，第 1 パス，第 2 パスのいずれかである．漸次出力の場合は，子要素として PHYPO タグを持つ．第 1 パスと第 2 パス出力の場合は，子要素として文候補の SHYPO タグを持つ．第 1 パスの場合は，最大スコアとなる結果のみが出力され，第 2 パスの場合はパラメータで指定した数だけ候補を出力するので，その候補数だけ SHYPO タグが出力される．
- PHYPO タグ漸次候補を表すタグで，子要素として単語候補 WHYPO タグの列が含まれる．属性として PASS，SCORE，FRAME，TIME がある．PASS は何番目のパスかを表し，必ず 1 である．SCORE はこの候補のこれまでの累積スコアを表す．FRAME はこの候補を出力するのにこれまでに処理したフレーム数を表す．TIME は，そのときの時刻（秒）を表す．
- SHYPO タグ文仮説を表すタグで，子要素として単語仮説 WHYPO タグの列が含まれる．属性として PASS，RANK，SCORE，AMSCORE，LMSCORE がある． PASS は何番目のパスかを表し，属性 PASS があるときは必ず 1 である．RANK は仮説の順位を表し，第 2 パスの場合にのみ存在する．SCORE はこの仮説の対数尤度，AMSCORE は対数音響尤度，LMSCORE は対数言語確率を表す．
- WHYPO タグ単語仮説を表すタグで，属性として WORD，CLASSID，PHONE，CM を含む．WORD は表記を，CLASSID は統計言語モデルのキーとなる単語名を，PHONE は音素列を，CM は単語信頼度を表す．単語信頼度は，第 2 パスの結果にしか含まれない．
- SYSINFO タグシステムの状態を表すタグで，属性として PROCESS がある．PROCESS が EXIT のときは正常終了を，ERREXIT のときは異常終了を，ACTIVE のときは音声認識が動作可能である状態を，SLEEP のときは音声認識が停止中である状態を表す．これらのタグや属性が出力されるかどうかは，Julius MFT の起動時に指定された引数によって変わる．SOURCEINFO タグは必ず出力され，それ以外はオリジナルの Julius と同じなので，オリジナルの Julius の引数ヘルプを参照のこと．

オリジナルの Julius と比較した場合，JuliusMFT における変更点は，以下の 2 点である．

- 上述の音源定位に関する情報用タグである SOURCEINFO タグ関連の追加，および，関連する下記のタグへの音源 ID(SOURCEID) の埋め込み．

  STARTRECOG, ENDRECOG, INPUTPARAM, GMM, RECOGOUT, REJECTED, RECOGFAIL, GRAPHOUT, SOURCEINFO

- 同時発話時の排他制御による処理遅れを改善するために，モジュールモードのフォーマット変更を行った．具体的には，これまで発話単位で排他制御を行っていたが，これをタグ単位で行うよう出力が複数回に分かれており，一度に出力する必要がある下記のタグの出力に改造を施した．

  《開始タグ・終了タグに分かれているもの》

  - <RECOGOUT>...</RECOGOUT>
  - <GRAPHOUT>...</GRAPHOUT>
  - <GRAMINFO>...</GRAMINFO>
  - <RECOGPROCESS>...</RECOGPROCESS>

  《1 行完結のタグであるが，内部では複数回に分けて出力されているもの》

  - <RECOGFAIL ... />
  - <REJECTED ... />
  - <SR ... />

JuliusMFT 出力例

1. 標準出力モードの出力例

```
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)
.....................................................................................
pass1_best: <s> 注文お願いします </s>
pass1_best_wordseq: 0 2 1
pass1_best_phonemeseq: silB | ch u: m o N o n e g a i sh i m a s u | silE
pass1_best_score: 403.611420
### Recognition: 2nd pass (RL heuristic best-first)
STAT: 00 _default: 19 generated, 19 pushed, 4 nodes popped in 202
sentence1: <s> 注文お願いします </s>
wseq1: 0 2 1
phseq1: silB | ch u: m o N o n e g a i sh i m a s u | silE
cmscore1: 1.000 1.000 1.000
score1: 403.611786
```

connection end
ERROR: an error occured while recognition, terminate stream ←このエラーログが出るのは仕様

2. モジュールモードの出力サンプル

   下記の XML ライクな形式でクライアント (jcontrol など) に出力される．行頭の “>” は，jcontrol を用いた場合に jcontrol によって出力される（出力情報には含まれない）．

```
> <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="注文お願いします" CLASSID="2" PHONE="ch u: m o N o n e g a i sh i m a s u" CM="1.
>     <WHYPO WORD="</s>" CLASSID="1" PHONE="silE" CM="1.000"/>
>   </SHYPO>
> </RECOGOUT>
```

注意事項

- -outcode オプションの制約

  タグ出力をプラグイン機能を用いて実装したため，出力情報タイプを指定できる -outcode オプションもプラグイン機能を用いて実現するように変更した．このため，プラグインが読み込まれていない状態で -outcode オプションを指定すると，エラーとなってしまう．

- 標準出力モードの発話終了時のエラーメッセージ

  標準出力モードで出力されるエラー “ERROR: an error occured while recognition, terminate stream” （出力例を参照）は，作成した特徴量入力プラグイン (mfcnet) で生成した子プロセスを終了する際に，強制的にエラーコードを julius 本体側に返しているため出力される．Julius 本体に極力修正を加えないようこのエラーに対する対処を行わず，仕様としている．なお，モジュールモードでは，このエラーは出力されない．

インストール方法

- apt-get を用いる方法

  apt-get の設定ができていれば，下記でインストールが完了する．なお，Ubuntu では オリジナルの Julius もパッケージ化されているため，オリジナルの Julius がインストールされている場合には，これを削除してから，以下を実行すること．

```
> apt-get install julius-4.1.4-hark julius-4.1.3-hark-plugin
```

- ソースからインストールする方法

  1. julius-4.1.4-hark と julius_4.1.3_plugin をダウンロードし，適当なディレクトリに展開する．
  2. julius-4.1.4-hark ディレクトリに移動して以下のコマンドを実効する．デフォルトでは，/usr/local/bin にインストールされてしまうため，パッケージと同様に /usr/bin にインストールするためには，以下のように -prefix を指定する．

     ```
     ./configure --prefix=/usr --enable-mfcnet; make; sudo make install
     ```

  3. 実行して以下の表示が出力されれば Julius のインストールは正常に終了している．

     ```
     > /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. 次にプラグインをインストールする． julius_4.1.3_plugin ディレクトリに移動して以下のコマンドを実行する．

     ```
     > export JULIUS_SOURCE_DIR=../julius_4.1.4-hark; make; sudo make install
     ```

     JULIUS_SOURCE_DIR には julius_4.1.4-hark のソースのパスを指定する．今回は同じディレクトリに Julius と plugin のソースを展開した場合を想定した．

     以上でインストール完了である．

  5. /usr/lib/julius_plugin 下にプラグインファイルがあるかどうかを確認する．

     ```
     > ls /usr/lib/julius_plugin
     calcmix_beam.jpi calcmix_none.jpi mfcnet.jpi calcmix_heu.jpi calcmix_safe.jpi
     >
     ```

     以上のように 5 つのプラグインファイルが表示されれば，正常にインストールできている．

- Windows 版のインストール方法については、3.2 章のソフトウェアのインストール方法を参照.

6.8.2 KaldiDecoder

概要

KaldiDecoder は，深層学習を利用した音声認識ツールキット Kaldi [5] のライブラリを用いて HARK 用に作られたデコーダである．HARK 2.2.0 までは，大語彙音声認識システム Julius をもとに追加機能を実装した JuliusMFT を提供していたが，近年の動向を反映し HARK 2.3.0 からは，Kaldi に対応する KaldiDecoder の提供も開始する．

Kaldi の標準デコーダと比較して，KaldiDecoder には下記に挙げる特徴がある．

1. HARK モジュールとの接続性 (JuliusMFT と同じように扱える)
   - MSLS，MFCC 特徴量のネットワーク経由の入力 (mfcnet) に対応
   - 音源情報 (SrcInfo) の追加に対応
   - 同時発話への対応（排他処理）

   いずれも，従来の JuliusMFT と同様に SpeechRecognitionClient (または SpeechRecognitionSMNClient) を使って HARK と接続出来る．

2. JuliusMFT との互換性
   - JuliusMFT 出力エミュレーション (モジュールモード及び標準出力の書式) に対応

   KaldiDecoder は JuliusMFT の出力を可能な範囲で再現しているため，JuliusMFT を用いて構築された既存システム (デモやスコアリング評価のシステム) に対する変更が最小限で済むように作られている．

3. Kaldi への追加機能
   - nnet1 モデルのオンラインデコードに対応

   Kaldi の標準デコーダでは nnet1 モデルについてオフラインデコードのみ対応となっている．

4. 未実装機能
   - ミッシングフィーチャー (ただし，mfcnet のマスク付きデータ構造には対応している)

上記 1〜3 の実装に関しては，Kaldi 本体に変更を加えない形で実装を行った．

以下のセクションでは KaldiDecoder のインストール方法，使用方法を説明するとともに，FlowDesigner 上の HARK モジュールとの接続について，解説する．

起動と設定

KaldiDecoder の実行は，例えば設定ファイル名を kaldi.conf とすれば，以下のように行う．

```
> kaldidecoder --config=kaldi.conf (Ubuntu 版)
> kaldidecoder.exe --config=kaldi.conf (Windows 版)
```

[5] http://kaldi-asr.org/

HARK では，KaldiDecoder をオンラインモードで起動したのち，IP アドレスやポート番号が正しく設定された SpeechRecognitionClient (または SpeechRecognitionSMNClient) を含んだネットワークを起動することにより，KaldiDecoder とのソケット接続が行われ，音声認識が可能な状態となる．

上述の kaldi.conf は KaldiDecoder の設定を記述したテキストファイルである．設定ファイルの中身は，基本的に “--” で始まる引数オプションからなっており，起動時に直接，KaldiDecoder のオプションとして引数指定することも可能である．また，# 以降はコメントとして扱われる．KaldiDecoder で用いるオプションは，

```
> kaldidecoder --help (Ubuntu 版)
> kaldidecoder.exe --help (Windows 版)
```

を実行することで確認する事が出来るので，そちらを参照していただきたいが，nnet1 のモデルを使用する場合に最低限必要な設定は，以下の 7 種類である．

- `--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`

nnet3 のモデルを使用する場合に最低限必要な設定は，以下の 4 種類である．

- `--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`

chain モデルの場合は，nnet3 の設定に加えて次の設定が必要である．

- `--frame-subsampling-factor=3`

nnet3/chain のモデル学習の時に iVector を使用している場合は，次の設定が追加で必要である．

- `--ivector-extraction-config=<YOUR_PATH>/ivector_extractor.conf`

オフラインデコードの場合には，下記のように評価対象の特徴量ファイルリストを指定する．指定しない場合は，自動的にオンラインデコード (mfcnet 入力) で起動したと判断される．HARK 2.5.0 よりオフラインデコードの場合でも並列デコードが可能となったので，同時に起動するデコーダインスタンスを `--max-tasks=<Core 数>` オプションで制限することを推奨する．HARK 2.4.0 以前と同様に特徴量ファイルリストの記述順での出力を保証したい場合は下記のように 1 を指定する．

- `--filename-features-list=<YOUR_PATH>/features_list.txt`
- `--max-tasks=1`

オンラインデコードで mfcnet 入力ポート，または結果出力ポートを変更したい場合には下記のように指定出来る．下記はデフォルトポート (変更なし) である．

- `--port-mfcnet=5530`

- `--port-result=10500`

1. 基本設定 (入力ファイルと動作モードの設定)

   - **--nnet-type=モデル形式**
     Kaldi における nnet モデル形式を指定する．初期値は 1 である．所有しているモデルの種類に応じて，次のように設定を変更することが出来る．Karel Vesely の手法で作成された nnet1 モデルを指定する場合は 1，Daniel Povey の手法で作成された nnet3/chain モデルを指定する場合は 3 を指定する．
   - **--filename-words=単語リストファイル名**
     単語リストファイルを指定する．引数にカレントディレクトリからの相対パス，もしくは絶対パスを指定する．単語リストファイルの書式は Kaldi の学習や検証で使用する Lexicon ディレクトリまたは，評価で使用する言語モデルに含まれる `words.txt` と同じ形式である．なお，認識結果の出力に必要であるため本オプションは必須である．
   - **--filename-phones=音素リストファイル名**
     音素リストファイルを指定する．引数にカレントディレクトリからの相対パス，もしくは絶対パスを指定する．音素リストファイルの書式は Kaldi の学習や検証で使用する Lexicon ディレクトリまたは，評価で使用する言語モデルに含まれる `phones.txt` と同じ形式である．なお，音素アラインメントを行う場合にのみ必要であり，本オプションは任意である．
   - **--filename-align-lexicon=語彙ファイル名**
     語彙ファイルを指定する．引数にカレントディレクトリからの相対パス，もしくは絶対パスを指定する．語彙ファイルの書式は Kaldi の学習や検証で使用する Lexicon ディレクトリまたは，評価で使用する言語モデルに含まれる `align_lexicon.int` と同じ形式である．ここで指定する語彙ファイルは，`prepare_lang.pl` を用いて lang ディレクトリを作成しているなら `lang/phones/aligned_lexicon.int` に出力される．なお，認識結果の出力に必要であるため本オプションは必須である．
   - **--filename-feature-transform=FeatureTransform ファイル名**
     FeatureTransform ファイルを指定する．ここで指定する FeatureTransform ファイルは，DNN 学習を行った出力先の `exp/<model_dir>/final.feature_transform` に出力される．なお，nnet1 形式では音響モデルと別ファイルとなっているため nnet1 形式の場合，本オプションは必須である．
   - **--filename-nnet=nnet ファイル名**
     nnet ファイルを指定する．ここで指定する nnet ファイルは，DNN 学習を行った出力先の `exp/<model_dir>/final.nnet` (注:これは SymbolicLink である) に出力される．なお，nnet1 形式では音響モデルと別ファイルとなっているため nnet1 形式の場合，本オプションは必須である．
   - **--filename-mdl=mdl ファイル名**
     mdl ファイルを指定する．ここで指定する mdl ファイルは，DNN 学習を行った出力先の `exp/<model_dir>/final.mdl` (注:これは SymbolicLink である) に出力される．なお，音響モデルは認識結果の出力に必要であるため本オプションは必須である．
   - **--filename-class-frame-counts=クラスフレーム数ファイル名**
     クラスフレーム数ファイルを指定する．ここで指定する class-frame-counts ファイルは，DNN 学習を行った出力先の `exp/<model_dir>/ali_train_pdf.counts` に出力される．なお，nnet1 形式では音響モデルと別ファイルとなっているため nnet1 形式の場合，本オプションは必須である．
   - **--filename-fst=FST ファイル名**
     FST ファイルを指定する．ここで指定する FST ファイルは，`mkgraph.sh` を用いて graph ディレクトリを作成しているなら `graph*/HCLG.fst` に出力される．なお，FST ファイルは認識結果の出力に必要であるため本オプションは必須である．
   - **--filename-features-list=特徴量リストファイル名**

KaldiDecoder がオフラインデコードを行う場合に，評価したい特徴量のファイル名 (Path 付き) がリストアップされたテキストファイルを指定する．(特徴量のファイル名ではないので注意する事) 本オプションが指定されない場合は，オンラインデコードを行う．

- **--ivector-extraction-config=iVector 抽出設定ファイル名**
  iVector 抽出設定ファイルを指定する．ここで指定する iVector 抽出設定ファイルは，DNN 学習を行った出力先の `exp/<model_dir>/ivector*/conf/ivector_extractor.conf` に出力される．なお，nnet3/chain 形式のモデルを学習する際に iVector を使用している場合，本オプションは必須である．
- **--frame-subsampling-factor=フレームサブサンプリング係数**
  フレームサブサンプリング係数を指定する．ここで指定するフレームサブサンプリング係数が書かれたファイルは，DNN 学習を行った出力先の `exp/<model_dir>/frame-subsampling-factor` (ファイルに書かれた整数値のみが必要である) に出力される．なお，chain 形式のモデルを使用している場合，本オプションは必須である．以下のように設定する必要がある．
  `--frame-subsampling-factor=3`
- **--frames-per-chunk=チャンクあたりのフレーム数**
  ニューラルネットによって別々に評価される各チャンク内のフレーム数を指定する．
  `--frame-subsampling-factor` オプションを使用する場合，任意のサブサンプリングの前に測定される．(すなわち入力フレームをカウントする) このオプションは， Kaldi のデコードレシピで使用されるシェルスクリプト `step/nnet3/decode.sh` や `steps/nnet3/decode_looped.sh` を参照されたい．初期値は 20 であるが，上記のデコードレシピでは 50 に設定されている．
- **--port-mfcnet=ポート番号**
  SpeechRecognitionClient (または，SpeechRecognitionSMNClient) から送信される音響特徴量とマスクをネットワーク経由で受信するポート番号を指定する．これは JuliusMFT の mfcnet 入力ポート指定 “-adport” と同様である．指定がない場合は JuliusMFT のデフォルトと同じ 5530 が設定される．オンラインデコード時のみ有効なオプション．
- **--port-result=ポート番号**
  認識結果の出力をネットワーク経由で送信するポート番号を指定する．これは JuliusMFT のモジュールモード出力ポート指定 “-module” と同様である．指定がない場合は JuliusMFT のデフォルトと同じ 10500 が設定される．
- **--host-mfcnet=ホスト名**
  --port-mfcnet で指定したポートでリッスンするサーバの IP またはホスト名を指定する．初期値は “localhost” である．
- **--host-result=ホスト名**
  --port-result で指定したポートでリッスンするサーバの IP またはホスト名を指定する．初期値は “localhost” である．
- **--lm-name**
  言語モデル名を指定する．指定するとモジュールモード出力時に LMNAME 属性を付与する．初期値は指定なし．

2. デコーダのチューニング関連設定 (重み付けと枝刈りの設定) 本項目は Kaldi のデコーダ（クラス）で実装されている機能から継承されたオプションである．

   - **--acoustic-scale=音響スケール**
     音響対数尤度のスケーリング係数を指定する．初期値は 0.1 である．一般的にスコアリングで得られた最適な LM ウェイトの逆数となる．

参考: https://sourceforge.net/p/kaldi/discussion/1355348/thread/924c555b/
しかしながら， chain モデルにおける最適設定は 1.0 である．
詳細については http://kaldi-asr.org/doc/chain.html#chain_decoding を参照されたい．

- **--max-active**
  デコーダの最大活性状態数を指定する．大きくするとより正確になるが遅くなる．初期値は 2147483647(int32 の最大値) である．経験上，2000 から 5000 の間ぐらいで指定する事を推奨.
- **--min-active**
  デコーダの最小活性状態数を指定する．初期値は 200 である．
- **--beam=ビーム幅**
  デコード処理のビーム幅を指定する．大きくするとより正確になるが遅くなる．初期値は 16.0 である． (> 0.0) 詳細は引用 6.152 を参照.
- **--beam-delta=ビーム差分**
  デコード処理のビーム差分を指定する．デコード処理に使われるこのパラメータは漠然と max-active の制約が適用された方法での高速化に関係する．大きくするとより正確になる．初期値は 0.5 である． (> 0.0) 詳細は引用 6.152 を参照.
- **--delta=デルタ**
  決定化に使用される許容範囲．初期値は 0.000976562 である．詳細は引用 6.152 を参照.
- **--hash-ratio**
  デコーダで使用されるハッシュ動作を制御するための指定．初期値は 2.0 である． (>= 1.0)
- **--prune-interval=フレーム数**
  指定したフレーム間隔でトークンを枝刈りする．初期値は 25 である． (> 0)
- **--splice=スプライス数**
  DNN 入力のスプライス数を指定する．現在フレーム周辺のコンテキストのフレーム数．初期値は 3 である．初期値 3 の場合，前後に 3 フレームある事を意味する．

これは Kaldi のサイト (http://www.danielpovey.com/kaldi-docs/decoders.html) からの引用である．

表 6.152: FasterDecoder: a more optimized decoder からの引用

```
  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:
引用元: [http://www.danielpovey.com/kaldi-docs/decoders.html#decoders_faster]: 3 項: [2016 年 12 月 16 日]

3. Lattice 関連設定本項目は Kaldi のデコーダ（クラス）で実装されている機能から継承されたオプションである．

   - **--determinize-lattice**
     Lattice を決定化する．Lattice 決定化は各単語順序のために最良の PDF（確率分布関数）順序のみを保持する．
   - **--lattice-beam=ビーム幅**
     Lattice 生成ビーム幅．大きいほど Lattice が深くなり遅い．初期値は 10 である．
   - **--max-mem=最大メモリ確保サイズ**
     Lattice の決定化における，おおよその最大メモリ確保サイズを指定する．但し，実際のメモリ使用量はここで指定された確保サイズで，複数回行われる可能性がある．
   - **--minimize**
     このオプションを指定すると，決定化の後で Lattice の最小化を行う．
   - **--phone-determinize**
     このオプションを指定すると，音素と単語両方に関する決定化の第一パスを行う．“--word-determinize” を参照.
   - **--word-determinize**
     このオプションを指定すると，単語のみに関する決定化の第二パスを行う．“--phone-determinize” を参照.

4. その他

   - **--config=コンフィグファイル**
     コンフィグファイルの指定．繰り返し指定可能.
   - **--enable-debug**
     このオプションを指定すると，デバッグ出力を有効にする．初期値は無効.
   - **--help**
     ヘルプの表示．このオプションが指定された場合，他のすべてのオプションは無視されます.
   - **--print-args**
     このオプションを指定すると，コマンドライン引数の内容を標準出力へ出力する．初期値は有効．無効にする場合，“--print-args=false” と指定する．
   - **--verbose=ログレベル**
     ログの詳細レベルを指定する．大きいほどより詳細にログを取る．初期値は 0 である．

なお，Julius や，JuliusMFT でモジュールモードと呼ばれていた機能はオンラインデコードを選択すると自動的に有効になる．その場合，Julius や，JuliusMFT のように標準出力が停止することはなく標準出力とソケット出力の両方が利用できる．

詳細説明

mfcnet 通信仕様

mfcnet を入力として利用するには，上述のように，KaldiDecoder 起動時に “--filename-features-list” を引数として指定しなければ良い．この際，KaldiDecoder は TCP/IP 通信サーバとなり，接続待ちとなる．また，HARK のモジュールである SpeechRecognitionClient や，SpeechRecognitionSMNClient は，音響特徴量とミッシングフィーチャーマスクを KaldiDecoder に送出するためのクライアントとして動作する．クライアントは，1 発話ごとに KaldiDecoder に接続し，送信終了後ただちに接続を切断する．送信されるデータはリトルエンディアンである必要がある（ネットワークバイトオーダーでないことに注意）．具体的には，1 発話に対して以下の流れで通信を行う．

1. ソケット接続

   ソケットを開き，KaldiDecoder の mfcnet 指定ポート番号に接続.

2. **通信初期化（最初に 1 回だけ送信するデータ）**

   クライアントから，ソケット接続直後に 1 回だけ，表 6.153 に示すこれから送信する音源に関する情報を送信する．音源情報は SourceInfo 構造体（表 6.154）で表され，音源 ID，音源方向，送信を開始した時刻を持つ．時刻は，`<sys/time.h>`で定義されている timeval 構造体で表し，システムのタイムゾーンにおける紀元（1970 年 1 月 1 日 00:00:00）からの経過時間である．以後，時刻は紀元からの経過時間を指すものとする．

3. **データ送信（毎フレーム送信するデータ）**

   音響特徴量とミッシングフィーチャーマスクを送信する．表 6.155 に示すデータを 1 フレームとし，1 発話の特徴量を音声区間が終了するまで繰り返し送信する．特徴量ベクトルとマスクベクトルの次元数は同じ大きさであることが KaldiDecoder 内部で仮定されている．

4. **終端処理（最後に 1 回だけ送信するデータ）**

   1 音源分の特徴量を送信し終えたら，終端を示すデータ（表 6.156）を送信する．KaldiDecoder は終端を示すデータが送信されるかソケットが切断されない限り，データが継続するものとして次フレームの受信を待機し続ける．よってある程度不安定な通信環境であってもデータ受信を再開する事が出来る．

5. ソケット切断

   終端処理を行った後，ソケットを閉じる．終端処理を行わずにソケットを閉じた場合，異常系の処理が走るため認識結果の出力が遅れる場合がある．また，終端処理後にデータを送出した場合，ソケットを閉じていなくても該当のデータは無視される．

表 6.153: 最初に 1 回だけ送信するデータ（音源情報を示すデータ）

サイズ [byte]	型	送信するデータ
4	int	28 (= sizeof(SourceInfo))
28	SourceInfo	これから送信する特徴量の音源情報

表 6.154: SourceInfo 構造体

メンバ変数名	型	説明
source_id	int	音源 ID
azimuth	float	水平方向 [deg]
elevation	float	垂直方向 [deg]
time	timeval	時刻 (64 bit 処理系に統一，サイズは 16 バイト)

表 6.155: 毎フレーム送信するデータ（特徴量とマスクデータ，および次元数情報）

サイズ [byte]	型	送信するデータ
4	int	N1=（特徴量ベクトルの次元数）× sizeof(float)
N1	float [N1]	特徴量ベクトル (float 配列)
4	int	N2=(マスクベクトルの次元数) × sizeof(float)
N2	float [N2]	マスクベクトル (float 配列)

表 6.156: 最後に 1 回だけ送信するデータ（終端を示すデータ）

サイズ [byte]	型	送信するデータ
4	int	0

モジュールモード通信仕様

オンラインデコードを指定すると Julius と同様にモジュールモードで動作させることができる．モジュールモードでは，KaldiDecoder が TCP/IP 通信のサーバとして機能し，認識結果を jcontrol などのクライアントに提供する．日本語文字列の文字コードは，言語モデルの文字コードに依存する．データ表現には Julius と同様に XML ライクな形式が用いられており，一つのメッセージごとにデータの終了を表す目印として “.”（ピリオド）が送信される．また，KaldiDecoder の追加機能で “.”（ピリオド）を使用せず XML 準拠で出力することも出来る．KaldiDecoder で送信される代表的なタグの意味は以下の通りである．

- INPUT タグ

  入力に関する情報を表すタグで，属性として STATUS と TIME がある．STATUS の値は LISTEN，STARTREC，ENDREC のいずれかの状態をとる． LISTEN のときは KaldiDecoder が音声を受信する準備が整ったことを表す．STARTREC は特徴量の受信を開始したことを表す．ENDREC は受信中の音源の最後の特徴量を受信したことを表す．TIME はそのときの時刻を表す．

- SOURCEINFO タグ

  音源に関する情報を表す，KaldiDecoder オリジナルのタグである．属性として ID，AZIMUTH，ELEVATION，SEC，USEC がある．SOURCEINFO タグは認識開始時に送信される．ID は HARK で付与した音源 ID（話者 ID ではなく，各音源に一意に振られた番号）を，AZIMUTH は音源の最初のフレームのときのマイクロホンアレー座標系からみた水平方向（度）を，ELEVATION は同垂直方向（度）を，SEC と USEC は音源の最初のフレームの時刻を表し SEC が秒の位，USEC がマイクロ秒の位を表す．

- RECOGOUT タグ

  認識結果を表すタグで，子要素は漸次出力，最終出力のいずれかである．漸次出力の場合は，子要素として PHYPO タグを持つ．最終出力の場合は，子要素として文候補の SHYPO タグを持つ．最終出力の場合はパラメータで指定した数だけ候補を出力するので，その候補数だけ SHYPO タグが出力される．

- PHYPO タグ

  漸次候補を表すタグで，子要素として単語候補 WHYPO タグの列が含まれる．属性として PASS，SCORE，FRAME，TIME がある．PASS は何番目のパスかを表し，必ず 1 である．SCORE はこの候補のこれまでの累積スコアを表す．FRAME はこの候補を出力するのにこれまでに処理したフレーム数を表す．TIME は，そのときの時刻（秒）を表す．

- SHYPO タグ

  文仮説を表すタグで，子要素として単語仮説 WHYPO タグの列が含まれる．属性として PASS，RANK，SCORE，AMSCORE，LMSCORE がある． PASS は何番目のパスかを表し，属性 PASS があるときは必ず 1 である．RANK は仮説の順位を表す．SCORE はこの仮説の対数尤度，AMSCORE は対数音響尤度，LMSCORE は対数言語確率を表す．

- WHYPO タグ

  単語仮説を表すタグで，属性として WORD，CLASSID，PHONE，CM を含む．WORD は表記を，CLASSID は統計言語モデルのキーとなる単語名を，PHONE は音素列を，CM は単語信頼度を表す．単語信頼度は，Julius の出力形式との互換性のために存在し，値に意味はなく常に 1.0 である．

- SYSINFO タグ

システムの状態を表すタグで，属性として PROCESS がある．PROCESS が EXIT のときは正常終了を，ERREXIT のときは異常終了を，ACTIVE のときは音声認識が動作可能である状態を，SLEEP のときは音声認識が停止中である状態を表す．

これらのタグや属性が出力されるかどうかは，KaldiDecoder の起動時に指定された引数によって変わる．SOURCEINFO タグは必ず出力され，それ以外はオリジナルの Julius と同じなので，オリジナルの Julius の引数ヘルプを参照のこと．

オリジナルの Julius と比較した場合，KaldiDecoder における変更点は，以下の 2 点である．

- 上述の音源定位に関する情報用タグである

  SOURCEINFO タグ関連の追加，および，関連する下記のタグへの音源 ID(SOURCEID) の埋め込み．

  STARTRECOG, ENDRECOG, INPUTPARAM, GMM, RECOGOUT, REJECTED, RECOGFAIL, GRAPHOUT, SOURCEINFO

- 同時発話時の排他制御による処理遅れを改善するために，

  モジュールモードのフォーマット変更を行った．具体的には，これまで発話単位で排他制御を行っていたが，これをタグ単位で行うよう出力が複数回に分かれており，一度に出力する必要がある下記のタグの出力に改造を施した．

  《開始タグ・終了タグに分かれているもの》

  - `<RECOGOUT> ... </RECOGOUT>`
  - `<GRAPHOUT> ... </GRAPHOUT>`
  - `<GRAMINFO> ... </GRAMINFO>`
  - `<RECOGPROCESS> ... </RECOGPROCESS>`

  《1 行完結のタグであるが，内部では複数回に分けて出力されているもの》

  - `<RECOGFAIL ... />`
  - `<REJECTED ... />`
  - `<SR ... />`

KaldiDecoder 出力例

1. 標準出力の出力例

```
source_id = 0, azimuth = 5.000000, elevation = 16.700001, sec = 1268718777, usec =
 474575
### Recognition: 2nd pass (RL heuristic best-first)
STAT: 00
sentence1: 注文 お 願い し ます
wseq1: 注文+名詞 お+接頭辞 願い+名詞 し+動詞 ます+助動詞
phseq1: ch ch ch u: u: u: u: u: m m m o o N N N o o n n n e e e g g a a i i sh sh
i i i m m a a a a a s s s u u u
cmscore1: 1.000 1.000 1.000 1.000 1.000
score1: 403.611786 ( AM: 409.323194, LM: -5.711408 )
```

2. ソケット出力（モジュールモード）の出力例

```
<SOURCEINFO SOURCEID="0" AZIMUTH="5.000000" ELEVATION="16.700001" SEC="1268718638"
USEC="10929"/>
.
<STARTRECOG SOURCEID="0"/>
.
<ENDRECOG SOURCEID="0"/>
.
<RECOGOUT SOURCEID="0">
  <SHYPO RANK="1" SCORE="403.611786" AMSCORE="409.323194" LMSCORE="-5.711408">
    <WHYPO WORD="注文" CLASSID="注文+名詞" PHONE="" CM="1.000"/>
    <WHYPO WORD="お" CLASSID="お+接頭辞" PHONE="" CM="1.000"/>
    <WHYPO WORD="願い" CLASSID="願い+名詞" PHONE="" CM="1.000"/>
    <WHYPO WORD="し" CLASSID="し+動詞" PHONE="" CM="1.000"/>
    <WHYPO WORD="ます" CLASSID="ます+助動詞" PHONE="" CM="1.000"/>
  </SHYPO>
</RECOGOUT>
.
```

注意事項

- PHONE タグの制約

  JuliusMFT では WORD 毎の PHONE 出力をサポートしているが，Kaldi の仕組み上の理由でこれを実装すると遅くなってしまうためサポートしていない．このため，ソケット出力での WHYPO タグ出力毎に音素情報を出力する機能はサポートしていない．標準出力では単語間のパイプ"|"が入っていない状態の出力のみサポートされる．

- Windows 版の制約

KaldiDecoder を Windows のプロンプトで実行すると標準出力の認識結果が文字化けするケースを確認している．これは Ubuntu ターミナルの標準出力が UTF-8 となっているため，同じ言語モデルを Windows 上で流用すると発生する問題である．つまり言語モデルに使用されている文字コードに起因する．この場合，KaldiDecoder の起動時に出力リダイレクション"> `filename`"を付与し書き出したテキストファイルを適切なエディタで開くことで文字化けしていない認識結果を得ることが出来るので確認頂きたい．この制約がある理由は，Julius では文字コード変換機能の iconv を内部で持っているため出力時の文字コード変換機能が存在していたが，Kaldi は文字コード変換機能を持たず KaldiDecoder でも追加実装はされていないからである．

インストール方法

- apt を用いる方法

  HARK のリポジトリが登録されていれば，下記でインストールが完了する．

  ```
  > sudo apt install kaldidecoder-hark
  ```

- ソースからインストールする方法

  KaldiDecoder は Kaldi のライブラリを使用しているため事前にビルドが必要である．しかし Ubuntu では Kaldi のパッケージが提供されていないので，以下のコマンドを実行して Kaldi 本体もソースからビルドを行う必要があり，ビルドには時間がかかる．

```
> 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
```

デフォルトでは，/usr/local/bin にインストールされてしまうため，パッケージと同様に/usr/bin にインストールするためには，以下のように -DCMAKE_INSTALL_PREFIX を指定する．

```
> 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
```

実行して以下の表示が出力されれば KaldiDecoder のインストールは正常に終了している.

```
> kaldidecoder --help
usage: If you requests need use ONLINE decoding with nnet1 model.
       (ONLINE mode is default)
...
```

以上でインストール完了である.

- Windows 版のインストール方法については、3.2 章のソフトウェアのインストール方法を参照.

第7章　サポートツール

7.1　HARKTOOL

7.1.1　概要

HARKTOOL は，GHDSS で用いる分離用伝達関数ファイルと LocalizeMUSIC で用いる定位用伝達関数ファイルを生成・可視化するツールである．なお，ファイル作成方法には，下記の GUI 画面から作成する方法と，コマンドのみで作成する方法の 2 種類がある．
前者については,HARKTOOL5-GUI ドキュメント, 後者については HARKTOOL5 ドキュメントを参照のこと.

7.2 wios

7.2.1 概要

wios とは, HARK がサポートしている 2 種類のデバイス (1) ALSA がサポートするデバイス, (2) RASP シリーズ の録音, 再生, 同時録音再生を行うツールである. デバイスに関しては**??** を参照．特に同時録音再生が可能なので, 音源定位、音源分離に必要なインパルス応答の測定に使える.

7.2.2 インストール方法

HARK がサポートしているディストリビューション/バージョンであれば, `sudo apt-get install wios` でインストールが可能. リポジトリの登録は HARK のウェブページを参照.

7.2.3 使用方法

重要なオプションは 3 種類で, 動作モード (録音, 再生, 同時録音再生) と, 使用デバイス (ALSA, RASP)，そして動作指定オプションである. それぞれ自由に組み合わせた指定が可能である．wios の詳細なオプションは、引数なしで実行すると見ることができる.

たとえば ALSA デバイス `plughw:1,0` を使用して `44.1kHz` で 2 秒 録音し，`voice.wav` に保存したい時は次のようにすればよい.

```
wios -r -x 0 -a plughw:1,0 -f 44100 -t 2 -o voice.wav
```

RASP デバイス `192.168.33.24` を使用して `tsp.wav` を再生する場合は次のようになる.

```
wios -p -x 1 -a 192.168.33.24 -i tsp.wav
```

RASP デバイス `192.168.33.24` を使用して `tsp.wav` を再生しながら `rec.wav` に保存する (同時録音再生する) 場合は次のようになる． `-a` は同一デバイスであるため省略できる.

```
wios -s -x 1 -d 192.168.33.24 -i tsp.wav -o rec.wav
```

ALSA デバイス `plughw:1,0` を使用して `tsp.wav` を再生しながら ALSA デバイス `plughw:2,0` を使用して `rec.wav` に保存する (同時録音再生する) 場合は次のようになる． `-a` は別のデバイスであるため省略できない.

```
wios -s -x 0 -d plughw:1,0 -a plughw:2,0 -i tsp.wav -o rec.wav
```

RASP デバイス `192.168.33.24` を使用して `tsp.wav` を再生しながら ALSA デバイス `plughw:2,0` を使用して `rec.wav` に保存する (同時録音再生する) 場合は次のようになる． 再生デバイスと録音デバイスの種別が異なるため`-x` の代わりに`-y` と`-z` を使用する.

```
wios -s -y 1 -d 192.168.33.24 -z 0 -a plughw:2,0 -i tsp.wav -o rec.wav
```

他のコマンド例は, HARK cookbook の「多チャネル録音したい」を参照.

次に，オプションの種類ごとに説明する.

動作モード

録音：オプションに -r or --rec を与える．-o or --ad-file オプションで wave ファイル名を与えると，指定されたデバイスを通してマルチチャネルの音を録音できる．ファイル名を与えない場合のデフォルト名は ad.wav.

再生：オプションに-p or --play を与える．-i or --da-file オプションで wave ファイル名を与えると，指定されたデバイスを通して音声を再生できる．ファイル名を与えない場合のデフォルト名は da.wav.

同時録音再生：オプションに -s or --sync を与える．-i or --da-file オプションに再生するファイル名を，-o or --ad-file オプションに録音するファイル名を与える．すると wios はファイルの再生と録音を同時に開始する．

デバイスの選択

デバイスは, デバイスの種別とデバイス名を表す 2 つのオプションを使って指定する．デバイスの種別を指定するには-x or --dev-type を使用する．--sync モードにおいて D/A デバイスと A/D デバイスを個別に指定する場合は-y or --da-dev-type (D/A デバイス) と-z or --ad-dev-type (A/D デバイス) を使用する．デバイス名を指定するには-d (D/A デバイス) と-a (A/D デバイス) を使用する．

ALSA：ALSA のデバイス種別は 0 である．つまり-x 0 や-y 0，-z 0 と指定する．使用するデバイスは-d or --da-dev-name (D/A：再生時) または-a or --ad-dev-name (A/D：録音時) で指定する. デフォルトは plughw:0,0

RASP：RASP のデバイス種別は 1 である．つまり-x 1 や-y 1，-z 1 と指定する．使用するデバイスの IP アドレスを-d or --da-dev-name (D/A：再生時) または-a or --ad-dev-name (A/D：録音時) で指定する．デフォルトは 192.168.33.24

上記のオプションに加えて, -x の値ごとに，デバイス専用の追加オプションを指定することも可能．詳しくは wios を引数なしで実行した際のオプションを参照.

動作指定

- -t: 録音/再生時間.
- -e: 量子化ビット数．デフォルトは 16bit.
- -f: サンプリング周波数．デフォルトは 16000Hz.
- -c: チャネル数．デフォルトは 1ch.