HARK version 1.0.0 Document
(document version 0.1.0)

HARK Documentation Team

平成 23 年 2 月 2 日

[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 が現在ミドルウエアとして利用している FlowDesigner 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，
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．

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

このような設計思想の下に，2008 年 4 月に HARK 0.1.7 をオープンソースとして公開し，開発者自身での改良，ユーザからのフィードバックの反映，バグフィックス，ドキュメントの充実などを通じて HARK 1.0.0 プレリリースを 2009 年 11 月に公開し，確定版 HARK 1.0.0 を 2010 年 10 月に公開予定である．以下では,「HARK 1.0.0」は「確定版 HARK 1.0.0」を指す．HARK 1.0.0 の主たる新機能は以下の通りである:

1. 音源分離の新規実装による音源分離の高性能化・ロバスト化,
2. 比較的複雑なロボット形状への信号処理レベルでの対応,
3. ロボットの定常雑音を抑制する単純な自己生成音抑制機能,
4. 移動音源を見据えた音源定位・音源分離法の開発,
5. 音源定位・音源分離に関するパラメータ数の削減とそれらの詳細設定機能,
6. 新音声特徴量の開発と音声認識向上のための機能,
7. 設定データ可視化，及び，作成ツールの提供,
8. ミドルウエア FlowDesigner の操作性の向上.

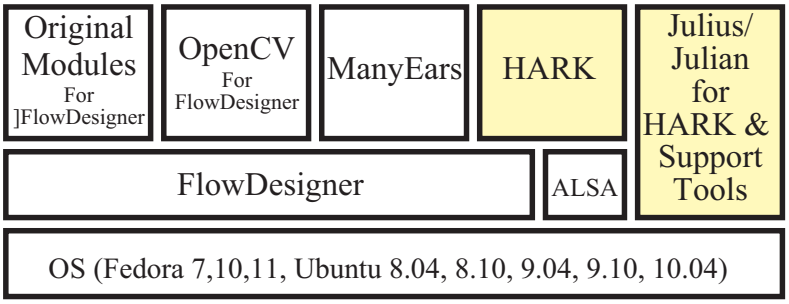

図 1.1: ロボット聴覚ソフトウエア HARK とミドルウエア FlowDesigner，OS との関係

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

図 1.1 から分かるように，Linux 系の OS しかサポートされていない．この 1 つの理由は，複数のマルチチャネル A/D 装置をサポートするために ALSA (Advanced Linux Sound Architecture) という API を使用しているためである．最近 PortAudio が Windows 系で利用されるようになっているので，PortAudio を使用した HARK も開発中である．

ミドルウエア FlowDesigner

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．

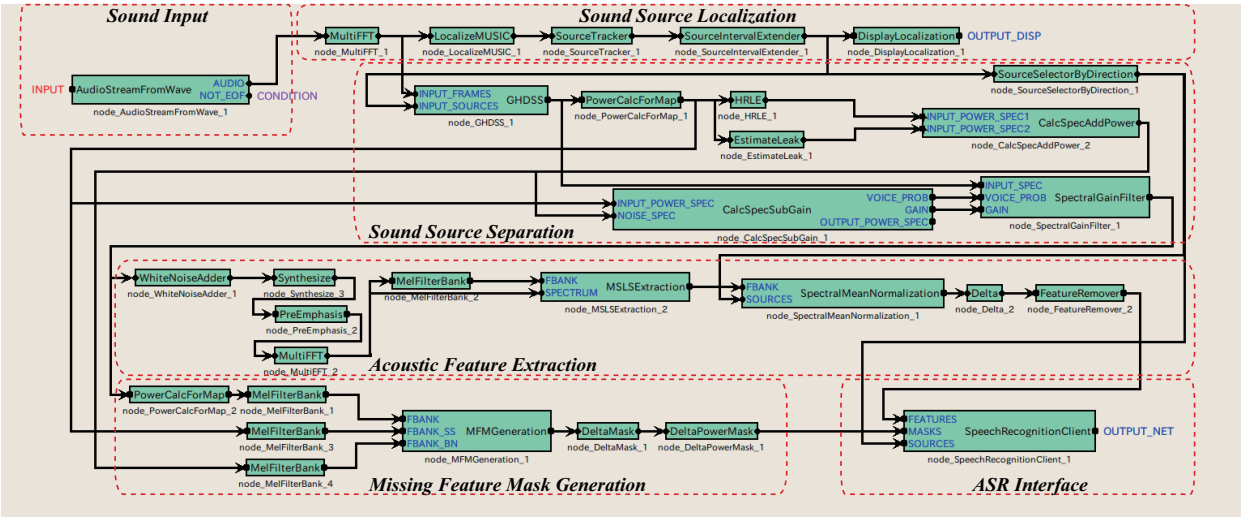

図 1.2: HARK を用いた典型的なロボット聴覚の FlowDesigner 上でのモジュール構成

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 GUI 開発環境である FlowDesigner [2] を採用した．FlowDesigner は，CORBA 等汎用的にモジュール統合に用いることが可能な枠組みと比較して，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[2]，高速・軽量なモジュール統合を実現したデータフロー指向の GUI 開発環境を備えたフリー（LGPL/GPL）のミドルウエアである．FlowDesigner では，各モジュールは C++ のクラスとして実現される．これらのクラスは，共通のスーパークラスを継承するため，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，FlowDesigner 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．

我々は，これまでの使用経験に基づき，メモリリーク等のバグに対処するとともに，操作性の向上（主に属性設定部）を図った FlowDesigner も同時に公開している[3]．

HARK を用いた典型的なロボット聴覚に対する FlowDesigner 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 3 種類である．

[2]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．

[3]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，http://winnie.kuis.kyoto-u.ac.jp/HARK/ からそれぞれダウンロードできる．

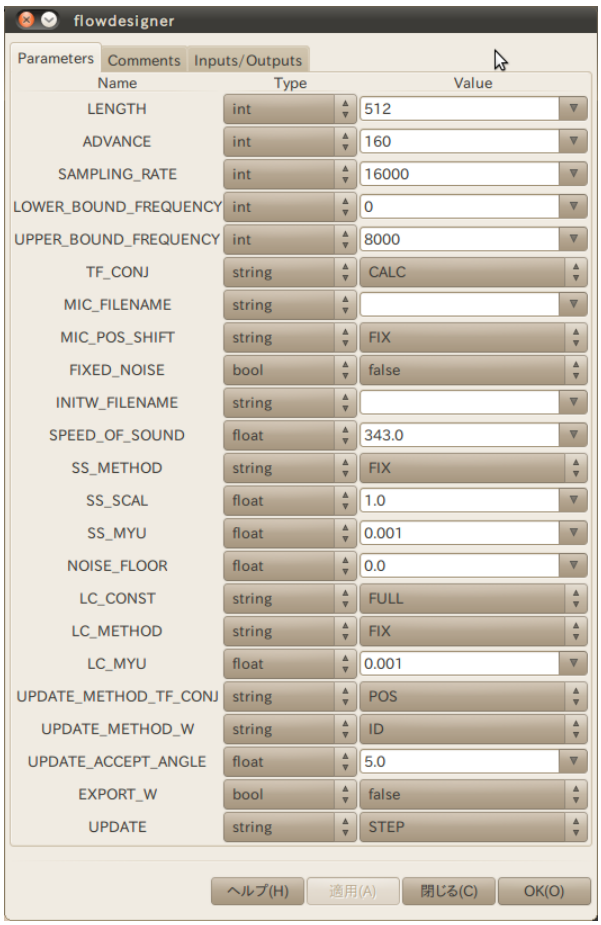

図 1.3: GHDSS の属性設定画面の例


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）

- 日本電子システムテクノロジー社製，RASP シリーズ，
- 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），
- 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.

これらの A/D 装置はいずれも 16 チャネル入力であるので，HARK での内部パラメータを変更すれば，16 チャネル入力にも対応できる．ただし，処理速度は低下する．また，信号の表現法を 24 ビットでサンプリングしても，そのまま対応ができる．なお，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．

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．

表 1.1: Modules and Tools provided by HARK 1.0.0

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
		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
		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
音源分離	Separation	BGNEstimator	背景雑音推定
		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
		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
		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
		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	hark-tool	データ可視化・各種設定ファイル作成

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．

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) [7]，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, 7].

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

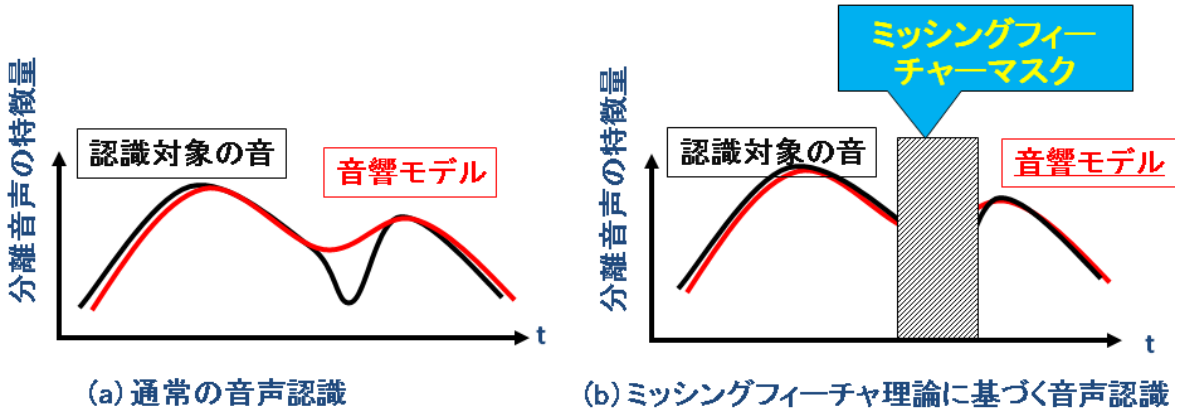

図 1.5: ミッシングフィーチャ理論による音声認識の概念図

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))．MTF-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．

MTF-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 [12]．

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 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 [15]. 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おう得なかった．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 [16].

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 [17, 18].

1.4.1 3 話者同時発話認識

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，

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

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[4]．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 [16].

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 [16] 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．

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

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 [19] 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．

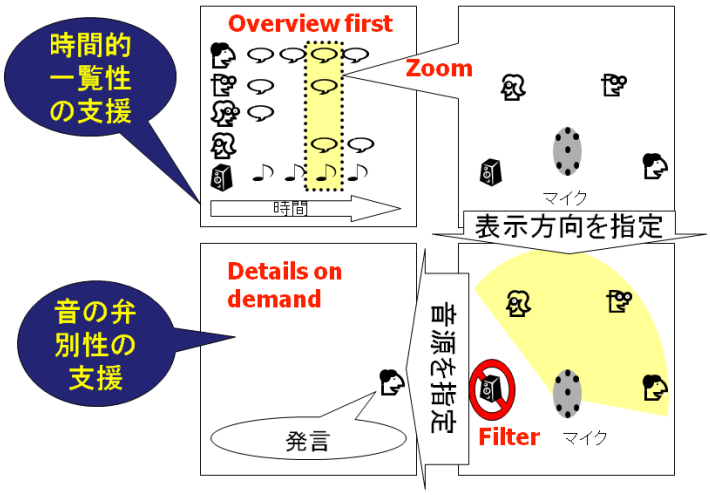

図 1.7: CASA 3D Visualizer: Visual Information-Seeking Matra “Overview first，zoom and filter, then details on demand” に従った HARK 出力の可視化

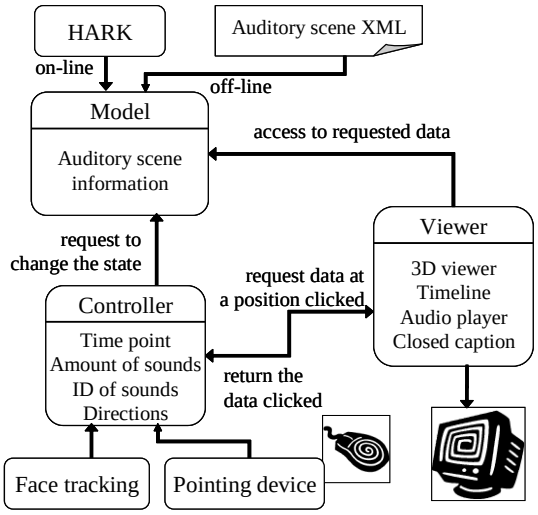

図 1.8: CASA 3D Visualizer の MVC (Model-View-Control) モデルを使用した実装法

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 [17, 18]． 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．

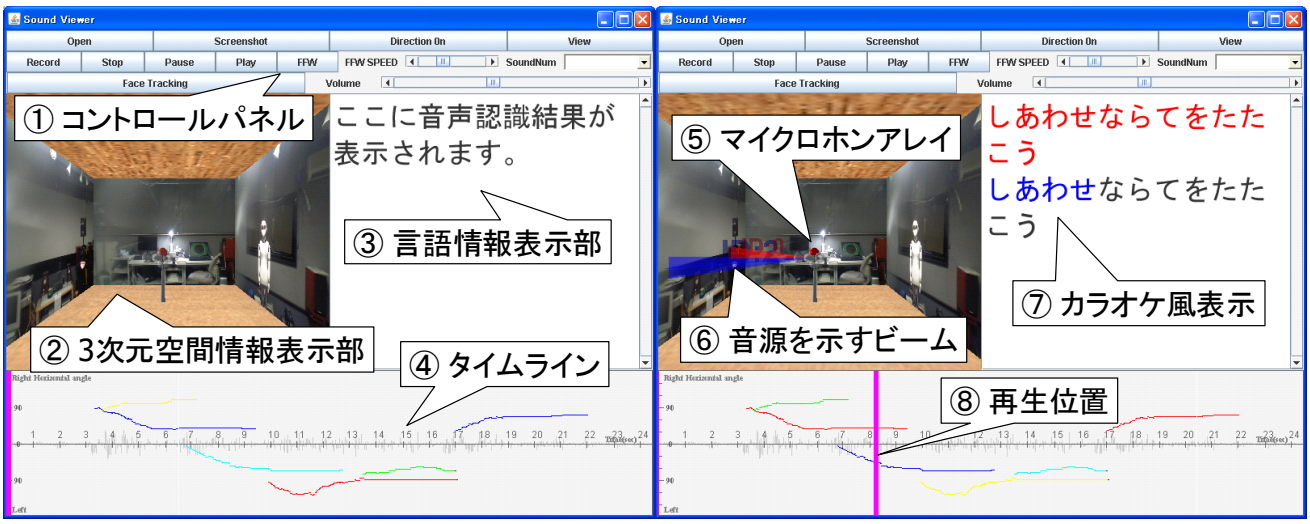

図 1.9: CASA 3D Visualizer の GUI

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 [17],
2. Visualizer の結果をヘッドマウントディスプレィ (HMD) に表示 [20].

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[5]．テ

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

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 fig:newTexai). 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．

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．

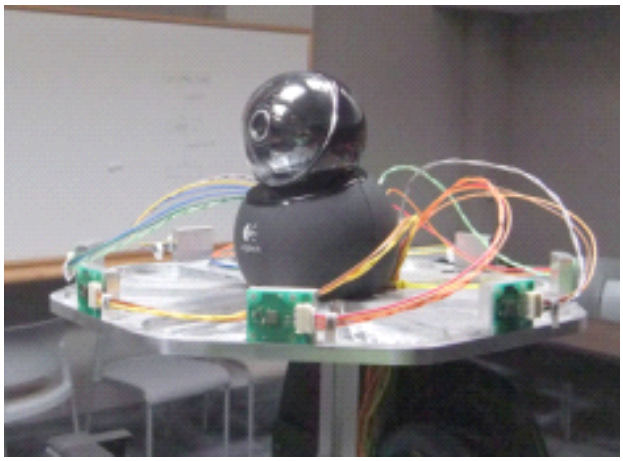

図 1.11: Texai の最初の頭部の拡大: 8 個の MEMS マイクロフォンを円盤上に設置

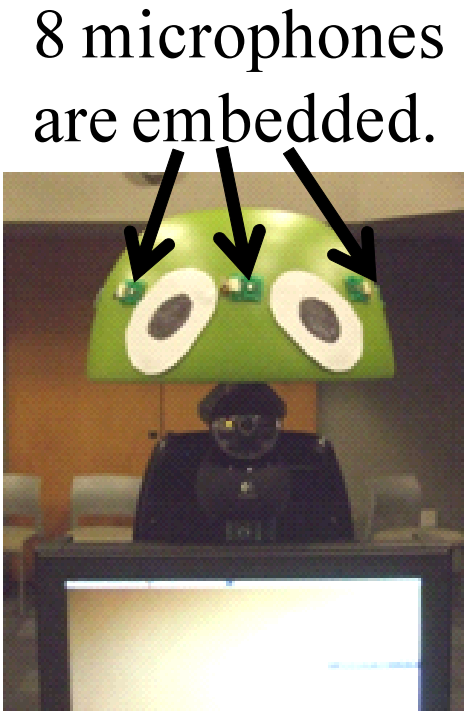

図 1.12: Texai の頭部の拡大: 8 個の MEMS マイクロフォンを円周状に設置

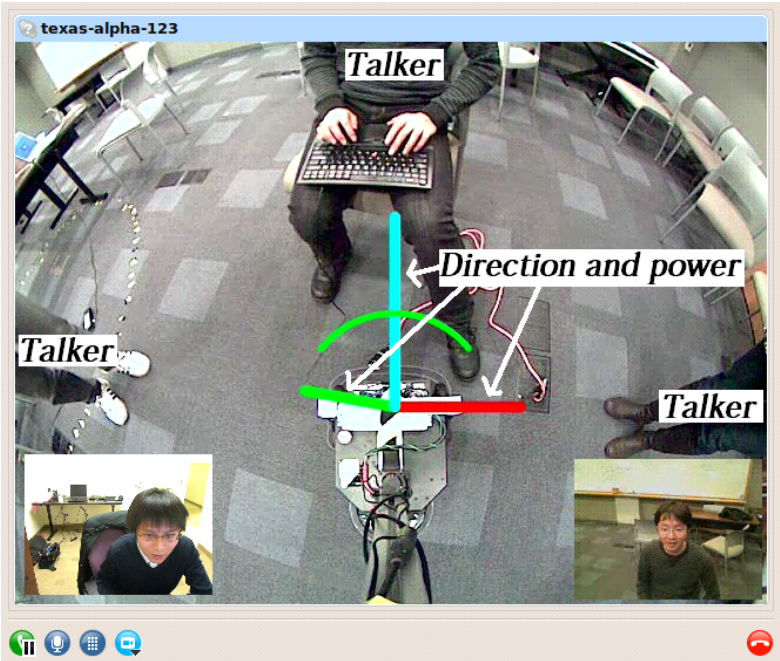

図 1.13: Texai を通じて，remote operator に見える画面

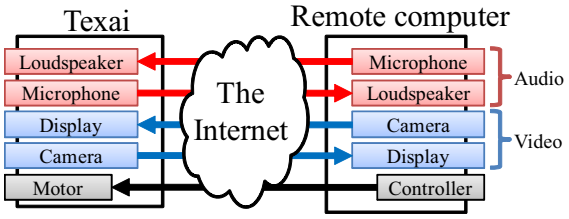

図 1.14: Texai の Teleoperation の方法

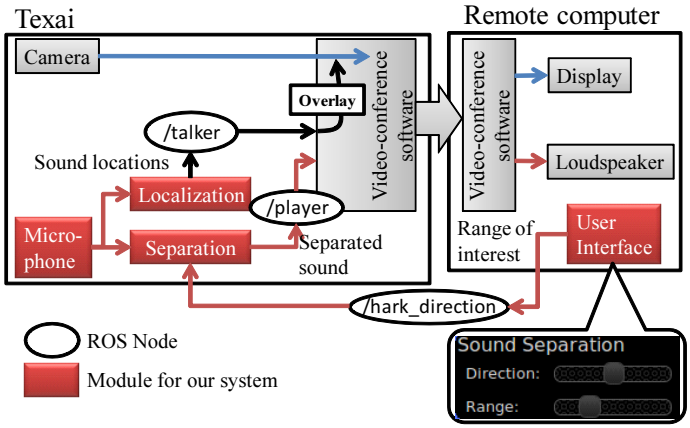

図 1.15: Texai への HARK の組込方法

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) [8] 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の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，

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

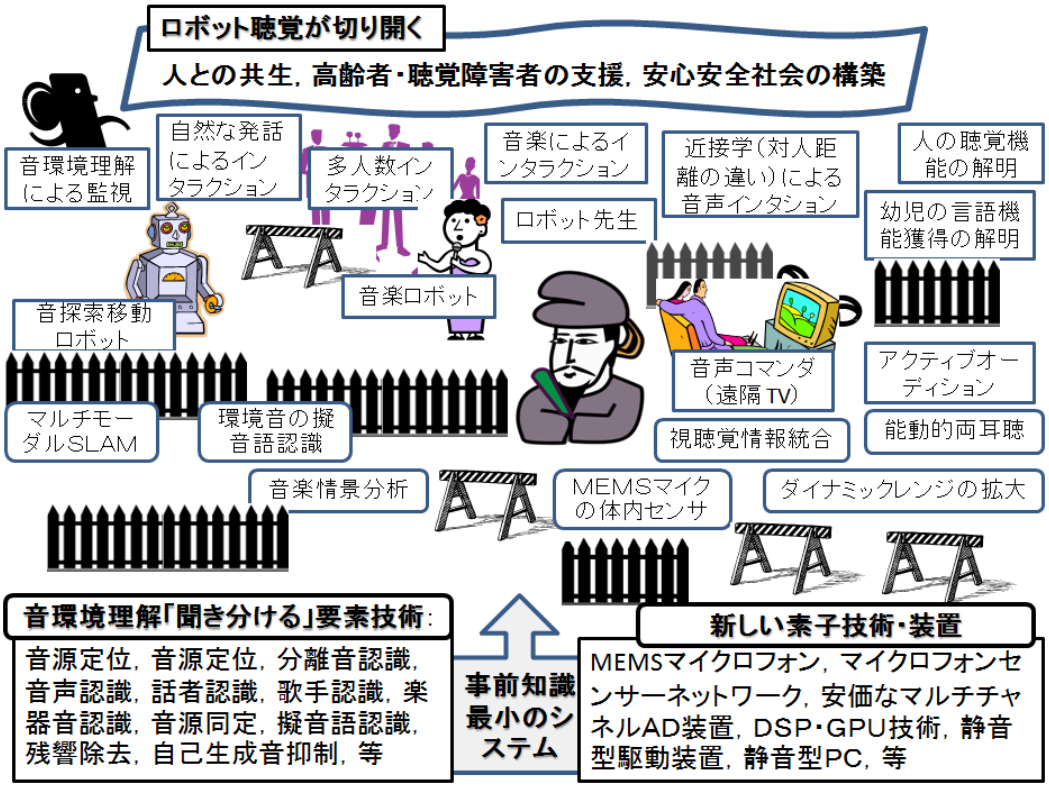

図 2.1: 音環境理解をベースとしたロボット聴覚の展開

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，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 [10, 11]．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より安定した

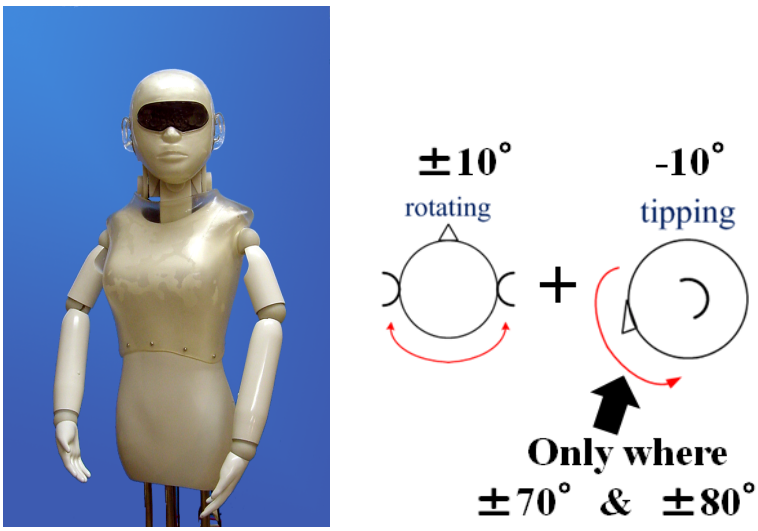

図 2.2: SIG2 のアクティブオーディション：周辺部の音に対しては首を左右と下に動かして前後問題の曖昧性を解消する．

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 [13, 14, 26]．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 [11]．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ができないから

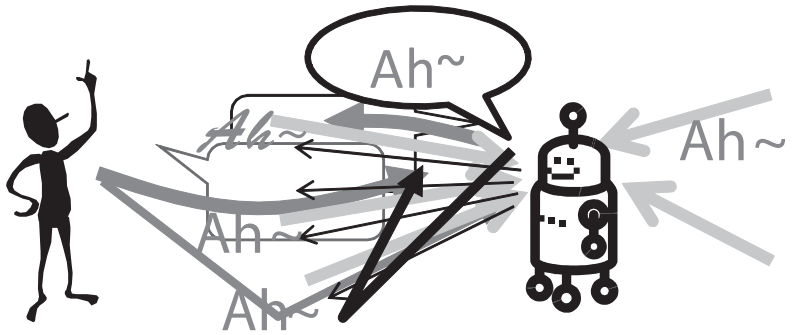

図 2.3: 自分の話声が残響を伴って自分の耳に入り，さらに，相手の割り込み発話 (バージイン) も聞こえる

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 [11]．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 [11]．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 [11].

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 [11]．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 [11]．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] 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.

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

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

[10] 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.

[11] 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/

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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

[17] 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

[18] 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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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 サイトにアクセスしダウンロード・インストールする．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と，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の 2 つの方法がある．

ソースコードの公開サイトは，http://winnie.kuis.kyoto-u.ac.jp/HARK/ である．こちらのサイトから，ダウンロードする．

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

本節の説明で，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．

3.2.1　パッケージからのインストール方法

Ubuntu 8.04, 8.10, 9.04, 9.10, 10.04 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．

配布サーバの設定は，root 権限を使い，/etc/apt/sources.list.d 以下に harklist ファイル名で以下の内容のテキストファイルを用意する．

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

具体的な作業例を以下に示す.

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

次にパッケージのインストールを行う．apt-get を使用する場合の 具体的な作業例を以下に示す.

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

Synaptic パケージ・マネージャでもインストール可能である．Synaptic パッケージマネージャの実行には root 権限が必要である．検索窓に flowdesignerhri を指定し，flowdesignerhri パッケージを選択し，適用を右クリックすることでインストールできる．harkfd, harktool, libharkio, julius-4.1.5-hark も同様にインストールする.

3.3 FlowDesigner

FlowDesigner は，オープンソースのミドルウエアの一つである．現在 HARK がベースにしているミドルウエアは，FlowDesigner のみである．今後，他のミドルウエア上への実装も視野に入っている.

FlowDesigner の特徴は， GUI を通じてシステムを構築できる点である．simulink や LabView に見られるグラフィカルプログラミングスタイルが採用されており，プログラム作成経験の少いユーザも容易にプログラム作成可能である.

FlowDesigner でのプログラム作成は，モジュールの配置と結線，モジュールのプロパティ値の設定によって行われる．図 3.1 に FlowDesigner の概観を示す．以下の作業で FlowDesigner を起動できる.

```
> flowdesigner
```

3.3.1 FlowDesigner の基本操作

モジュールは，データを処理する処理単位である．最初にモジュールの例を図 3.2 に示す．長方形で囲まれた緑色の部分が，モジュールである．このモジュールの中央には，モジュール名が表示される．この例では，PostFilter である．モジュール下に表示される名前 node_Postfilter_1 は，モジュールのインスタンス名である．同種のモジュールが複数ある場合には，モジュールのインスタンスを生成した順番に node_Postfilter_2, node_Postfilter_3 と名付けられる．個々のモジュールを区別するのに役立つ.

モジュールは，処理するデータの入力と出力の口になるターミナルをそれぞれ複数もつことが可能である．モジュールの左右にある黒い点がターミナルである．左側のターミナルが入力ターミナルで，右側のターミナルが出力ターミナルである．ターミナルには名前が付けられている場合と名前が付けられていない場合がある．この例では，全てのターミナルに名前が付けられている．モジュールは，少なくとも１つの出力ターミナルがある．入力ターミナルは１つも無いこともある．例えば FlowDesigner に標準で含まれる **constant** モジュールは，入力がなく出力ターミナルを 1 つ持つ.

複数のモジュールを配置し，モジュールのターミナル同士を接続することによって，データの処理の流れを定義する．接続されてた一連のモジュールをネットワーク呼ぶ．ネットワークを作成すること自体が，プログラム作成になっている．ネットワークは，ファイルの保存メニューから保存できる．一度保存すれば，いつでも FlowDesigner

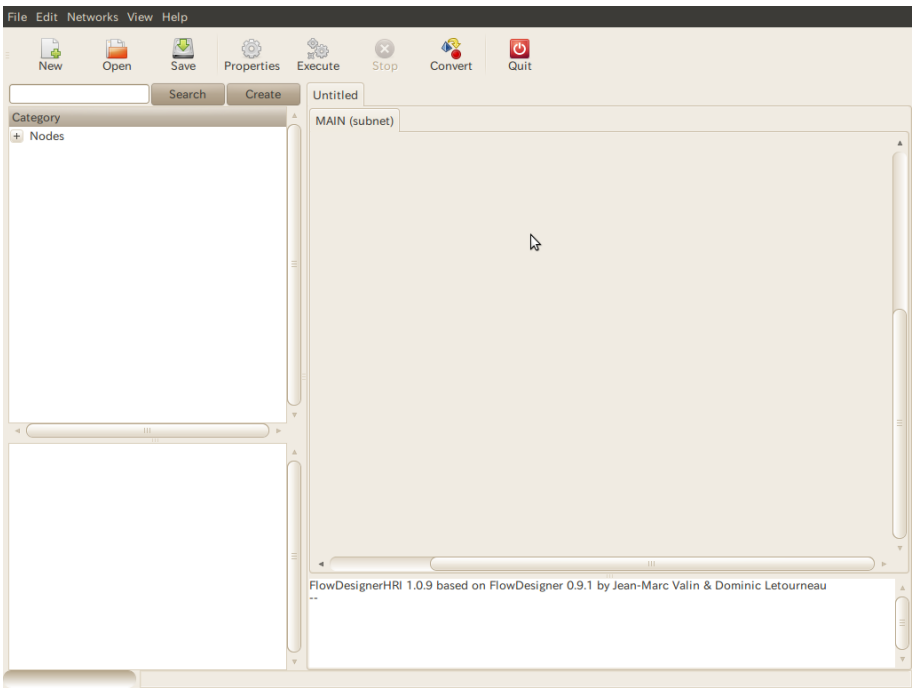

図 3.1: FlowDesigner の概観

図 3.2: モジュールの概観

でロードできる．保存したファイルをテキストエディタで開くと XML ライクな記述を確認できる．慣れればテキストエディタで直接ネットワークを編集することもできるようになる．

モジュールには，プロパティをもつものがあり，プロパティ値を設定することでモジュールの処理を詳細に指定可能である．プロパティ値の設定には，モジュール上にマウスカーソルを移動させ，左ダブルクリックする．設定ダイアログウィンドウが開き，各種プロパティ値をキーボードで入力あるいは選択できる．

プロパティ値には値のデータ型あり，データ型と値をペアで設定する必要がある．データ型の詳細は，データ型の章を参考のこと．データ型は，プルダウンメニュー形式の中から選択する．値は，キーボードから値を入力する場合と，プルダウンメニュー形式で値を選択する場合がある．

3.3.2 モジュールの操作・ターミナルの接続・プロパティの設定

以下では，具体的なモジュールの操作方法，ターミナルの接続方法，プロパティの設定方法を述べる．

- **モジュールの作成**
  ネットワーク構築タブでマウスの右ボタンをクリックすると，モジュールのカテゴリ一覧が表示される．目的のカテゴリ名の上にマウスカーソルを移動させると，モジュールが表示される．そのまま，目的のモジュール名の上にマウスカーソルを移動させ，左ボタンをクリックすると，モジュールが作成される．カテゴリ一覧

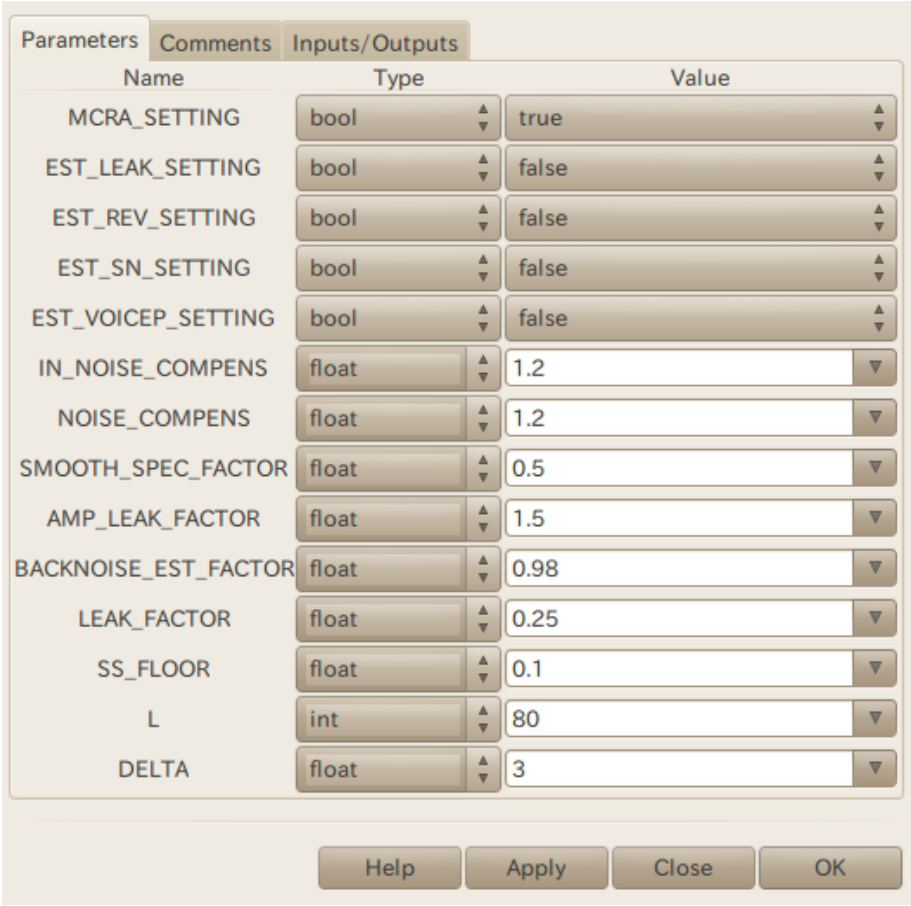

図 3.3: プロパティの概観

や，モジュール一覧が表示された状態から，モジュールの配置を中止する場合には，ネットワーク構築タブ内の地の部分を（左右どちらのボタンでも可）クリックする．

- **モジュールの再配置**
  一度配置したモジュールを見やすい位置に再配置するには，モジュールの上にマウスカーソルを移動し，左ボタンでドラッグする．

- **モジュールの削除**
  モジュールの上にマウスカーソルを移動し，右ボタンをクリックすると，モジュールに対する操作メニューが表示される．Delete と表示された部分にマウスカーソルを移動し，左クリックするとモジュールを削除できる．

- **モジュールの複製**
  Shift キーを押下したまま，モジュール上で左クリックすると，そのモジュールを複製できる．属性値を含め，複製モジュールが生成できるため，同様の処理を並列・並行して処理するネトワークを構築する場合に便利な機能である．

- **モジュールの接続**
  モジュール同士は，ターミナルを矢線で結線し，接続する．接続元のターミナル上で左ボタンを押下し，ドラッグして接続先のターミナルでボタンを放すとターミナル間が矢線で結線される．これで，モジュールの接続が完了である．

入力と出力ターミナルの接続が可能で，入力ターミナル同士や出力ターミナル同士を接続することはできない．１つの出力ターミナルから複数の接続を引出すことが可能である．モジュールの処理結果が，複数のモジュールに送られる．一方，１つの入力ターミナルで複数の接続を受け入れることはできない．入力と出力ターミナルの接続でも，接続できないことがある．ターミナルが処理できるデータ型に対応していない場合は接続できず，矢線が赤色に表示される．ターミナルが処理できるデータ型の自動判定に失敗し，矢線が黒色であるにもかかわらず，プログラム実行時にエラーが起る場合もある．データ型の自動判定は，補助的なものと考え，正確な接続には，モジュールリファレンスで，対応するデータ型を確認する方がよい．

- **モジュールの切断**
  切断したい接続の矢線の始点または終点で，Shift キーを押下しながら，左クリックすることで接続を切断できる．

- **モジュール接続線の再配置**
  モジュールを移動させると，接続を維持したままモジュールが移動する．多数のモジュールを配置すると，モジュールと接続線が重なり，見ずらいネットワークになる．そこで，線を曲，重なりの少い見やすいネットワークにするとよい．接続線の途中の部分の上にマウスカーソルを移動させ，そこでマウス左ボタンでドラッグすると，接続線がその点で折れ曲る．曲げた点上でマウス左ボタンでドラッグすると，点を移動できる．

- **プロパティ値の設定**モジュールの上にマウスカーソルを移動し，右ボタンをクリックすると，モジュールに対する操作メニューが表示される．Properties と表示された部分にマウスカーソルを移動し，左クリックするとプロパティ値の設定ダイアログが開く．モジュールリファレンスを参考にして値とそのデータ型を入力する．Apply ボタンをクリックすれば設定が反映される． OK ボタンをクリックすれば設定が反映され，かつダイアログが閉じる．

以上で，モジュールの基本操作の説明を終え，実際に音源定位ネットッワークを構築しながら，ネットワーク構築に必要な基本操作を述べる．

3.3.3 はじめてのネットワークの作成

はじめてネットワークを作成する人を対象に，音源定位ネットワークを構築する例を示す．メインネットワークタブの他に Iteration 用のサブネットワークタブが必要になる．まず，メインタブで音源波形読込部分を構築し，続いて，サブネットワークを作成し，サブネットワーク内に，音源定位部，結果表示部を構築する．

音源波形読込部分のモジュール配置，接続，プロパティの設定について解説する．FlowDesigner を起動すると，ウィンドウ内右側に MAIN (subnet) というタブが現れる．始めに，このタブの中で作業を進める．Constant モジュールと， InputStream モジュールを図 3.4 のように配置する．タブ内の地の部分を右クリックすると，プルダウンメニューが現れる．その中の New Node 上にマウスカーソルを移動すると， FlowDesigner に登録されているモジュール一覧が表示される．ここでは， General カテゴリを選ぶ．マウスカーソルをそのまま General という表示の上まで移動さる．ここで，プルダウンメニューが 1 段展開され，General カテゴリに登録されているモジュールの一覧が表示される．この中に Constant モジュールがある．マウスカーソルをそのまま Constant という表示の上まで移動させると，MAIN (subnet) タブに Constant モジュールを配置できる．以後，この様にプルダウンメニューからモジュールを選択する操作を，簡略化して,「New Node → General → Constant」と表記する．同様に，InputStream モジュールを New Node → IO → InputStream と辿り，モジュールを配置する．

次に Constant モジュールと， InputStream モジュールを図 3.5 のように接続する．

次に モジュールのプロパティの設定を行う．ここでは， Constant モジュールのみ設定する．InputStream モジュールには設定可能なプロパティ値がない．

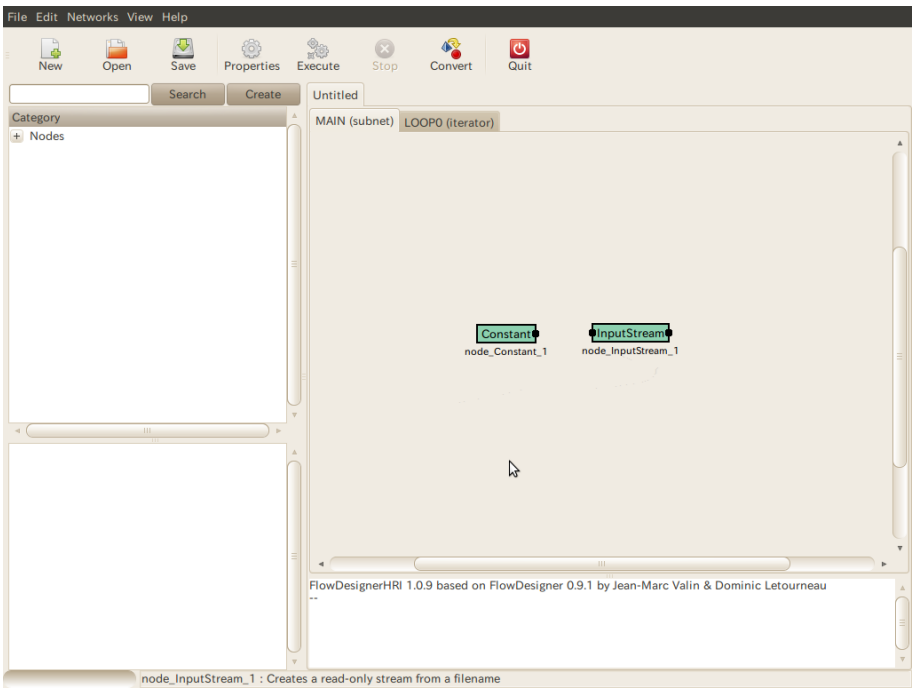

図 3.4: 2 つのモジュールを配置した様子

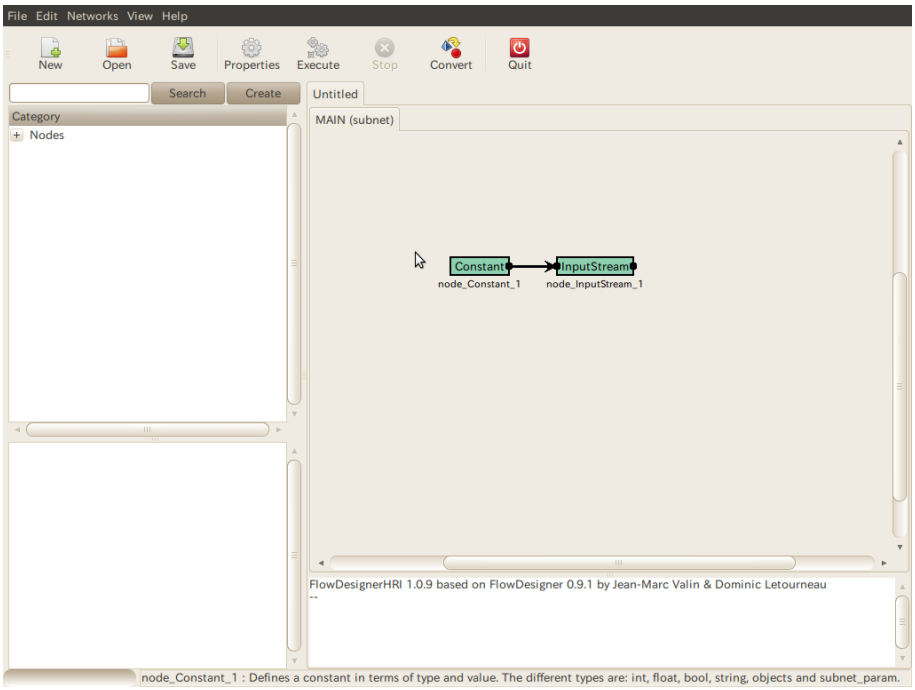

図 3.5: 2 つのモジュールを接続した様子

Constant モジュールを右クリックすると，プルダウンメニューが現れる．Properties という表示の上にマウスカーソルを移動させ Properties を左クリックすると，プロパティ設定ダイアログが開く．ダイアログには， Parameters タブ，Comments タブ， Inputs/Ouptuts タブがある．デフォルトでは Parameters タブが開かれている．Comments タブ，Inputs/Ouptuts タブを使用するこはない．

Parameters タブ内には，NAME，Type，Value という３つの項目がある．NAME は，プロパティ名であり， Type は，そこに設定する値のデータ型を表す，Value は，属性に指定する値である．Constant モジュールには，設定できるプロパティは１つしかない．VALUE という属性名である．このモジュールで，ファイル名を表したいので，プロパティの値にファイル名を tutorial1.wav とする．データ型は `string` である．最後にプロパティウンドウの適用を押すことで設定が反映される．ここで閉じるを押すと設定内容が破棄される．OK を押すと適用を押した後に，閉じると押した動作と等価な処理が行われる．プロパティウィンドウ表示中は，FlowDesigner の別のウィンドウの操作が無効状態になっているので，操作へ復帰するには，プロパティ設定を終える必要がある．以上でプロパティの設定は完了である．これで音源波形読込部分のネットワークが完成した．

続いて，サブネットワーク作成について述べる．サブネットワークは， FlowDesigner のウィンドウメニュー，Networks から add Iterator を選択して作成する．add Iterator を選択すると，ダイアログウィンドウが起動し，追加する Iterator タブの名前の入力が促される．デフォルトの LOOP0 でよければ OK ボタンを押す．キャンセルボタンを押すとサブネットワーク作成を中止する．

LOOP0 で OK すると，MAIN (subnet) タブの隣に，LOOP0 というタブが現れ， LOOP0 タブがアクティブになる．このタブの中に音源定位部と結果表示部を構築する．

音源定位部を，ファイルの読込，FFT，MUSIC 法による音源定位，音源追従，定位結果の表示のためのモジュールで構築する．以下の手順でモジュールを配置すると，図 3.6 のようになる．

New Node → HARK → AudioIO → AudioStreamFromMic
New Node → HARK → MISC → MultiFFT
New Node → HARK → Localization → LocalizeMUSIC
New Node → HARK → Localization → SourceTracker
New Node → HARK → Localization → DisplayLocalization

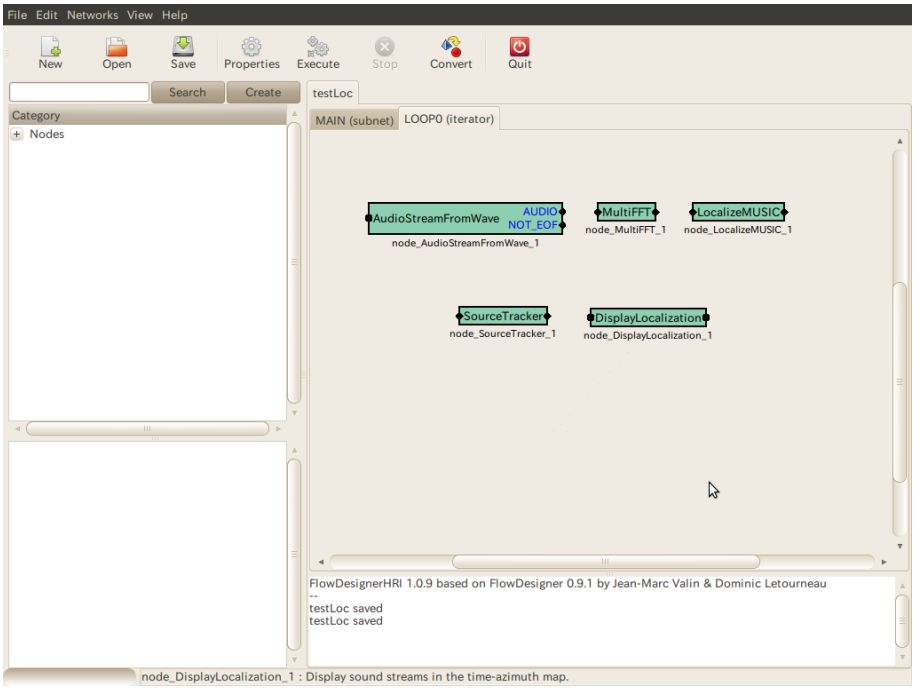

図 3.6: 音源定位部分のモジュールを配置した様子

これらのモジュールを図 3.7 の様に接続する．

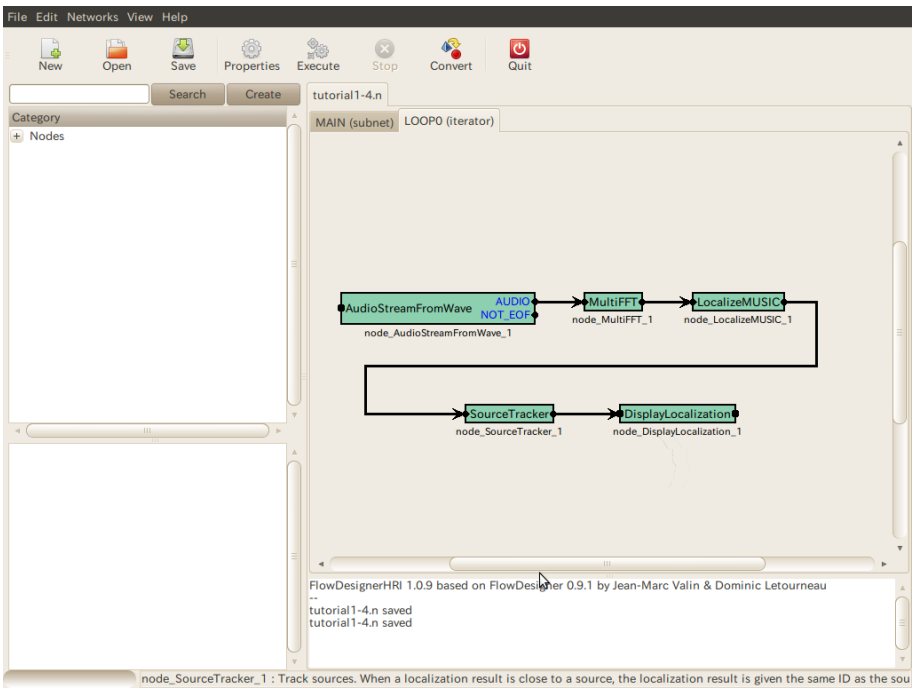

図 3.7: 音源定位部分のモジュールを接続した様子

接続が終ったら，モジュールのプロパティを設定する．処理する音声ファイルが 16000 [Hz] であることを前提に設定する．AudioStreamFromWave モジュールには，LENGTH，ADVANCE，USE_WAIT の 3 個所のプロパティ値の設定場所がある．LENGTH と ADVANCE は，音声の分析フレーム長とフレームシフト長を単位サンプルで表している．データ型は `int` である．デフォルト値 512，160 が設定されており，それぞれ 32 msec， 10 msec に対応している．変更する必要はない．データ型 `bool` の USE_WAIT にもデフォルト値が設定されており，`false` が設定されている．変更する必要はない．

MultiFFT モジュールには，LENGTH，WINDOW，WINDOW_LENGTH の 3 個所のプロパティ値の設定場所がある．音声の分析フレーム長と窓関数の種類，窓の長さを単位サンプルで表している．デフォルト値 512，CONJ，512 が設定されており，LENGTH と WINDOW_LENGTH は， 32 msec に対応し，データ型は， `int` である．CONJ は，分析窓に CONJ 窓を使うことを指定している．データ型は，`string` である．これらの値を変更する必要はない．

LocalizeMUSIC モジュールは，NB_CHANNELS，LENGTH，SAMPLING_RATE，A_MATRIX，ELEVATION，PERIOD，NUM_SOURCE，MIN_DEG，MAX_DEG，LOWER_BOUND_FREQUENCY，DEBUG UPPER_BOUND_FREQUENCY の１２個所のプロパティ値の設定場所がある．

- **MB_CHANNELS**
  NB_CHANNELS は，扱うマイクロホン数あるいは音声ファイルのチャネル数を設定する．データ型は `int` である．今回は， 8 を設定する．音声の分析フレーム長は，デフォルト値の 512 サンプルとする．MultiFFT の LENGTH と揃える必要がある．

- **SAMPLING_RATE**
  SAMPLING_RATE は，最初に仮定したとおり 16000 [Hz] を指定する．データ型は `int` である．

- **A_MATRIX**
  A_MATRIX は，伝達関数のファイル名を設定する．データ型は `string` である．伝達関数ファイルの作成方法は harktool の解説を参照のこと．ここでは，sample_tf.dat を設定する．

- **ELEVATION**
  ELEVATION は，音源の仰角を表す．データ型は，`float` である．デフォルトで，16.7 [deg] が設定されている．16.7[deg] は，音源がマイクロホンから 1 [m] 離れており，マイクロホンと音源の高さ方向の相対位置が 0.3 [m] の仰角方向にあると仮定された値である．

- **PERIOD**
  PERIOD は，相関関数の計算に使う分析フレーム数である．データ型は `int` で，デフォルト値は， 50 であり変更する必要はない．

- **NUM_SOURCE** NUM_SOURCE には，仮定する音源数を設定する．データ型は，`int` で，デフォルト値は， 2 であり変更する必要はない．

- **MIN_DEG，MAX_DEG**
  MIN_DEG と MAX_DEG で音源定位処理の対処となる範囲を指定する．データ型は，`int` である．デフォルトでは，-180 と 180 が設定されており，全方位を音源定位処理対象としている．これらを変更する必要はない．

- **LOWER_BOUND_FREQUENCY，UPPER_BOUND_FREQUENCY**
  LOWER_BOUND_FREQUENCY と UPPER_BOUND_FREQUENCY で，定位処理で使用する周波帯域を制限する．`int` である．デフォルトでは，500 と 2800 が設定されており，音声のエネルギーが大きい周波数帯域を指定している．これらを変更する必要はない．

- **DEBUG** DEBUG で，デバッグ様の情報の表示・非表示を切り換える．`bool` 型の値で，デフォルトで非表示を意味する `false` が入っている．これを変更する必要はない．

SourceTracker モジュールは，THRESH，PAUSE_LENGTH，MIN_SOURCE_INTERVAL，DEBUG の 4 個所のプロパティ値の設定場所がある．

- **THRESH**
  音源のパワーが THRESH より大きい間，音源を検出する．データ型は， `float` であり，デフォルト値はない．ここでは，とりあえず 25 を設定する．値のチューニング方法は，モジュールリファレンスを参照のこと．

- **PAUSE_LENGTH**
  発話中のポーズ区間で，音源のパワーが一時的に THRESH を下回る場合がある．その都度，音源検出が途絶えては，検出後の処理が困難である．ポーズと見做される時間内であれば一時的に THRESH を下回っても音源追従を継続し，音源があるとみなす方が都合がよい．データ型は `float` であり，デフォルト値は， 800 である．これを変更する必要はない．

  PAUSE_LENGTH を長くとると，音源検出が継続しやすい傾向になるが，発話終了時刻が PAUSE_LENGTH 分延長される．次の発話開始とオーバーラップし，２つの発話が１つの発話に見えるという問題が発生する．短過ぎると，１つの発話と看做してほしい発話が，時間的に細切れに検出される傾向が強くなる．

- **MIN_SOURCE_INTERVAL**
  音源を追従する上で，同一音源が常に同一方向に検出され続けることは稀である．検出方向がゆらぐため，揺らぎ幅内であれば同一音源みなさなす必要がある．許容する揺らぎ幅を角度で指定する．データ型は，`float` で，デフォルト値が 20 である． これを変更する必要はない．

DisplayLocalization モジュールは，LOG_IS_PROVIDED， １個所のプロパティ値設定場所がある．
音源位置のログが欲しいときに， `true` にする．データ型は， `bool` でデフォルト値は `false` である．これを変更する必要はない．以上で，音源定位部のプロパティ設定が完了である．

続いて，Iterator の設定を行う．Iterator は，MAIN (subnet) からサブルーチンのように使用される．以下の設定を行い MAIN (subnet) との関係を設定する必要がある．

Iterator に必須な項目は，ネットワークに INPUT と OUTPUT と CONDITION があることである．INPUT はデータの入力部分で，OUTPUT は，データの出力部分である．サブルーチンの入力と出力と対応付けて考えると理解しやすい．Iterator は，サブルーチンでありながら繰り返し処理を行う構造になっている．そのため，CONDITION つまり繰り返し処理の停止条件を記述する必要がある．

図 3.8 のように設定する．INPUT の設定は，入力ターミナル部分で Shift を押下しながら，左クリックすると，ダイアログが現れ，名前を変更することができる．OK を押すと，設定され，入力部分に赤字で INPUT と表示される．OUTPUT の設定は，出力ターミナル部分で Shift を押下しながら，左クリックすると，ダイアログが現れ，名前を変更することができる．OK を押すと，設定され，入力部分に青字で OUTPUT と表示される．CONDITION の設定は，出力ターミナル部分で Control を押下しながら，左クリックすると，設定される．紫色で CONDITION と表示される．

意図しない場所に INPUT，OUTPUT，CONDITION を設定した場合は，これらの文字の上で Shift を押下しながら，左クリックすると，解除できる．

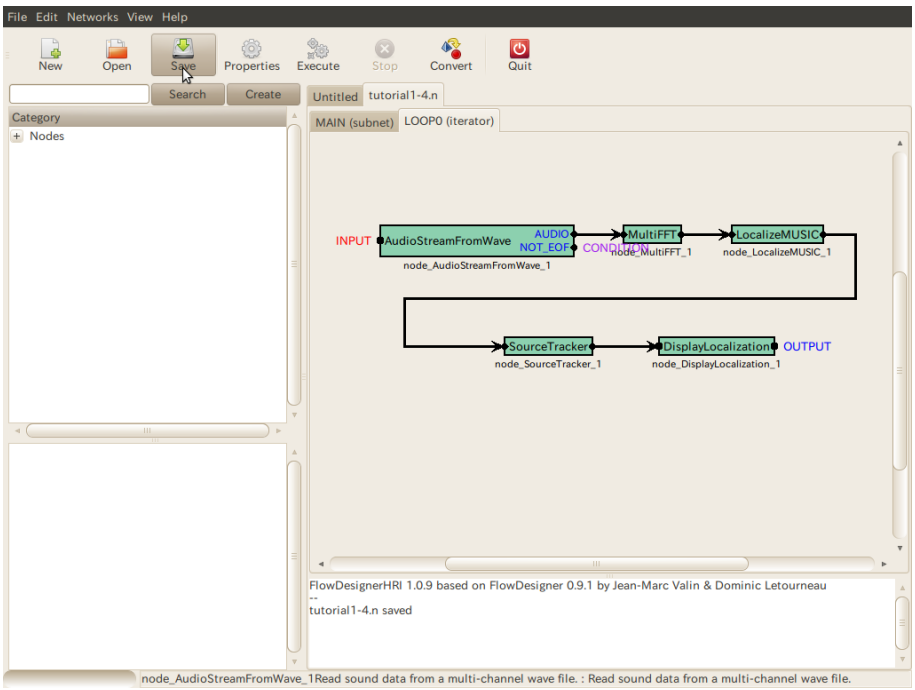

図 3.8: 音源定位部分のモジュールに INPUT，OUTPUT，CONDITION を設定した様子

最後に，subnet と iterator を統合する．MAIN (subnet) タブを押し，MAIN (subnet) のネットワーク表示に戻る．地を左クリックし， New Node 上にマウスカーソル移動させると，モジュールカテゴリ名に新しく subnet という項目ができている．subnet カテゴリ上にマウスカーソル移動させると， LOOP0 という項目が現れる．モジュールを配置するのと同様に LOOP0 上で左クリックすると， LOOP0 という 1 入力，1 出力のモジュールが配置される．つまり，これまでの作業で，LOOP0 (iterator) 内に記述したサブネットワークを仮想的なモジュールにしたことになる．

図 3.9 の様に，InputStream と LOOP0 を接続し， LOOP0 に OUTPUT を設定し，音源定位ネットワークの完成である．

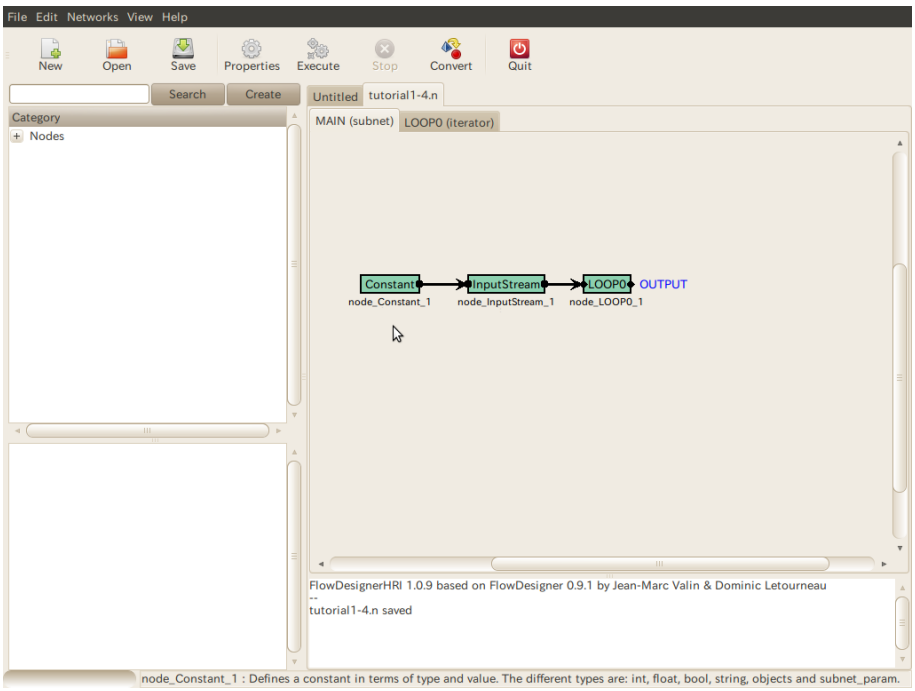

図 3.9: 完成した音源定位ネットワーク

第4章　データ型

本章では，FlowDesigner と HARK のモジュール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．

- モジュールのプロパティ設定
- モジュール同士の接続（モジュール間通信）

モジュールのプロパティ設定で用いるデータ型

モジュール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	FlowDesigner 固有型
subnet_param	サブネットパラメータ型	FlowDesigner 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，FlowDesigner 固有のデータ型である．Object は, FlowDesigner 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，テキスト形式での記述が実装されているクラスは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，複数のモジュール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．

モジュール同士の接続の際に用いるデータ型

モジュールの接続（モジュール間通信）は，異なるモジュールのターミナル（モジュールの左右に黒点として表示される）を FlowDesigner 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	FlowDesigner 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	FlowDesigner オブジェクト型
Matrix	行列型	FlowDesigner オブジェクト型
Int	整数型	FlowDesigner オブジェクト型
Float	単精度浮動小数点実数型	FlowDesigner オブジェクト型
String	文字列型	FlowDesigner オブジェクト型
Complex	複素数型	FlowDesigner オブジェクト型
TrueObject	論理型（真）	FlowDesigner オブジェクト型
FalseObject	論理型（偽）	FlowDesigner オブジェクト型
nilObject	オブジェクト型 (nil)	FlowDesigner オブジェクト型
ObjectRef	オブジェクト参照型	FlowDesigner 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，FlowDesigner 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，FlowDesigner の Object を継承する形で定義されているデータ型を FlowDesigner 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．FlowDesigner オブジェクト型で最もよく使われるものは，Vector, Matrix であろう．これらは，C++ の STL を継承した FlowDesigner 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 FlowDesigner 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．

Map も，C++ の STL を継承しているが FlowDesigner ではなく，HARK 固有のデータ型である．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．

モジュールのターミナルの型 は，FlowDesigner 上で型を知りたいモジュールのターミナルにカーソルをフォーカスすると， FlowDesigner の最下部に表示される．図 4.1 に，AudioStreamFromMic モジュールの AUDIO ターミナルにマウスをフォーカスした例を示す．FlowDesigner の最下部に AUDIO (Matrix<float>Windowed wave form. A row index is a channel, and a column index is time. が表示され，AUDIO ポートが Matrix<float>をサポートしていること，窓掛けされた音声波形を出力すること，行列の行がチャネルを表し，列が時刻を表していることがわかる．

一般的に，モジュール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

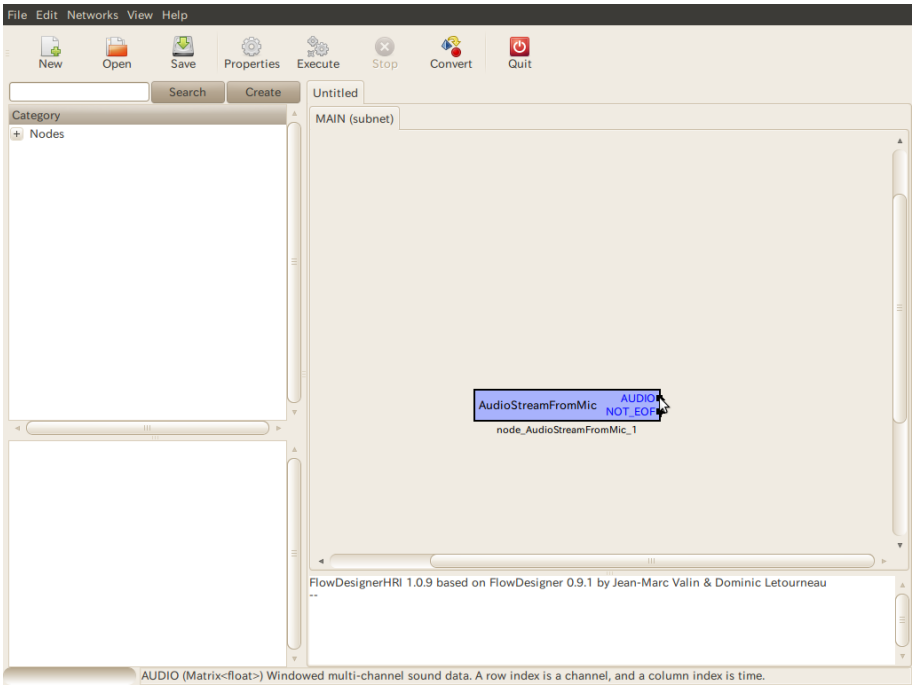

図 4.1: メッセージバーの表示例

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，黒矢印で表示される．この条件を満たさないターミナル同士を接続した場合は，警告の意味で赤矢印で表示される．

以降の節では，上述について基本型，FlowDesigner オブジェクト型，FlowDesigner 固有型 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，モジュール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，FlowDesigner としてはサポートしているが，HARK では，Source を除いて利用していない．

この型に変換するモジュール: Conversion カテゴリの To* モジュールが各型に変換する．int は ToInt，float は ToFloat，bool は ToBool，string は ToString を用いる．

4.2 FlowDesigner オブジェクト型

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 FlowDesigner 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，FlowDesigner 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 モジュールは，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，モジュール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 1.5，20.4 を持つ float 型の Vector をパラメータに入力したい時は，図 4.2 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．

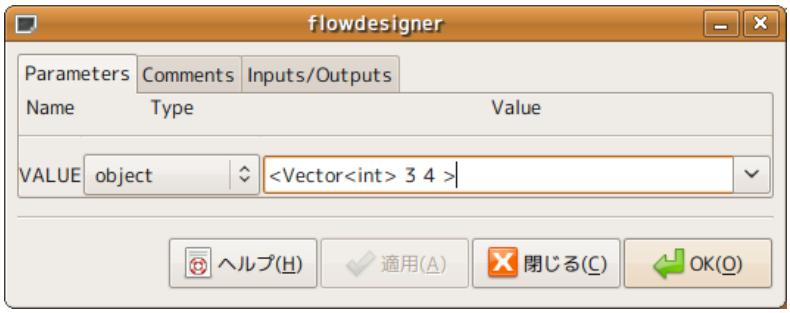

図 4.2: Vector の入力例

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 をモジュール間通信に用いるモジュールとしては，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，モジュール間通信に 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 FlowDesigner 固有型

FlowDesigner 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，モジュールのターミナルが any 型である場合は，他のモジュール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モジュール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モジュールに使用を限定している．

4.3.2 ObjectRef

FlowDesigner 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．FlowDesigner オブジェクト型，FlowDesigner 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

主にモジュール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．

参考までに，以下に，入力できる例，できない例を挙げる．

○	Vector< float > 0.0 1.0>	一般的な Vector の入力法
○	Complex<float(0,0)>	complex も Complex とすれば入力できる
×	Vector<complex<float> >(0.0, 1.0) >	complex の入力はサポートされていないためＮＧ
○	Vector< ObjectRef > Complex<float> (0.0, 1.0)>>	Complex として入力すれば問題ない
○	<Int 1>	int も Int として入力できる

4.3.4 subnet_param

モジュールのプロパティで用いられるデータ型である．subnet の複数のモジュール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 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モジュールを配置した際に，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) 上のモジュールのプロパティを subnet_param 型にすると，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．引数として指定できるかどうかは，FlowDesigner のプロパティをクリックすることによって確認できる．このプロパティのダイアログに値を記述しておくと，バッチ実行の際にデフォルト値として指定した値が用いられる．

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．

こうしたモジュールと，通常，音源追従処理より前に使う Matrix ベースで通信を行うモジュールを接続するために，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．定位された音源がそれ以降どれだけ続くと仮定するかの指標．

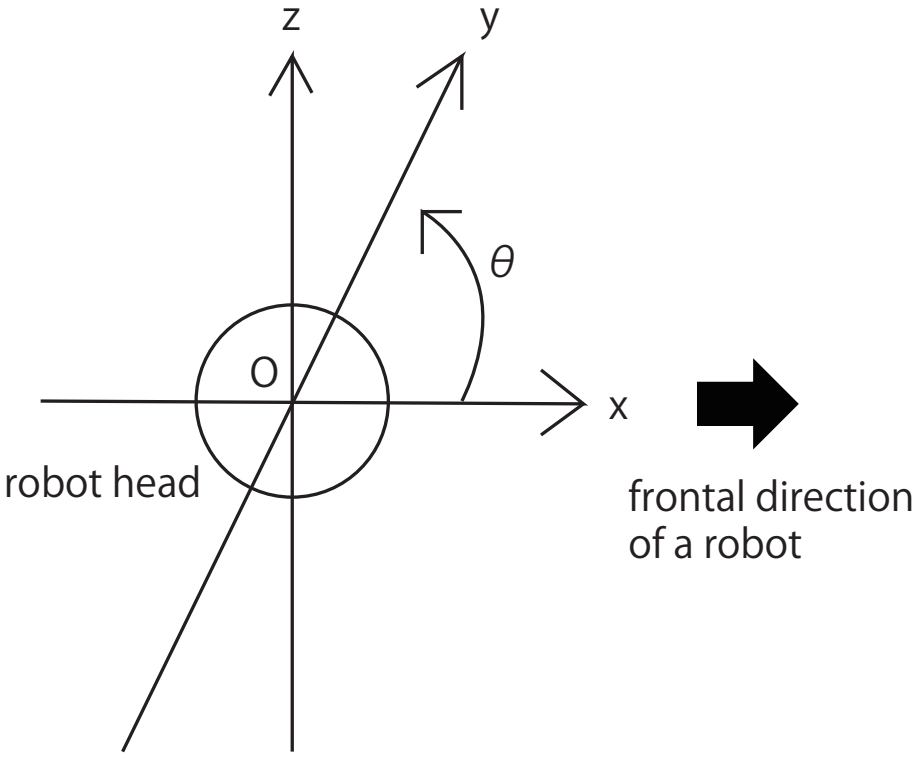

図 4.3: HARK 標準座標系

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]．)

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．HARK では，モジュール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

表 5.1: ファイル入出力を指定する HARK モジュール一覧

モジュール名	使用箇所	ファイル種類	ファイル形式
SaveRawPCM	出力	PCM raw ファイル	PCM バイナリ
LocalizeMUSIC	プロパティ設定	音源定位伝達関数ファイル	HGTF バイナリ
SaveSourceLocation	出力	音源定位結果ファイル	定位結果テキスト
LoadSourceLocation	プロパティ	音源定位結果ファイル	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	HGTF バイナリ
	プロパティ設定	マイクロホン位置ファイル	HARK テキスト
	プロパティ設定	定常ノイズ位置ファイル	HARK テキスト
	プロパティ設定	初期分離行列ファイル	HGTF バイナリ
	出力	分離行列ファイル	HGTF バイナリ
SaveFeatures	出力	特徴量ファイル	`float` バイナリ
DataLogger	出力	`Map` データファイル	`Map` テキスト
JuliusMFT	起動時引数	設定ファイル	jconf テキスト
	設定ファイル中	音響モデル・音素リスト	julius 形式
	設定ファイル中	言語モデル・辞書	julius 形式
harktool	harktool	音源位置リストファイル	srcinf テキスト
	harktool	インパルス応答ファイル	`float` バイナリ

これらのファイルのうち，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 の説明を参照してほしい．

以降では，Julius 形式以外のファイル形式について説明する．なお，実装面では，HGTF (HARK General Transfer Function) バイナリ形式，HARK テキスト形式は，HARK の入出力ライブラリ harkio で統合的に入出力をサポートしており，総称して，HARK ファイル形式と呼ぶ．それ以外の srcinf テキスト形式については，PCM バイナリ，`float` バイナリ，定位結果テキスト，`Map` テキスト形式についても，順次 harkio でサポートを行うよう変更していく予定である．

5.1　HGTF ファイル形式

HGTF(HARK General Transfer Function) ファイル形式は RIFF (Resource Interchange File Format) ライクなバイナリファイル形式であり，LocalizeMUSIC の音源定位伝達関数ファイル，GHDSS の音源分離伝達関数ファイル，および，分離行列ファイルを表すために用いる表現である．データ構造を表 5.2 に示す．

フォームタイプ “HGTF” のバージョンは，以下の 3 種類である．

表 5.2: HGTF(汎用伝達関数) のデータ構造

<table>
<tr><td colspan="2">識別子 “HARK”</td><td>4 bytes</td></tr>
<tr><td colspan="2">サイズフォームタイプ以降のサイズ</td><td>4 bytes</td></tr>
<tr><td rowspan="4">データチャンク</td><td>フォームタイプ “HGTF”</td><td>4bytes</td></tr>
<tr><td colspan="2">fmt サブチャンク (必須)
(フォーマット ID, マイクロホン数
サンプリング周波数などの基本データ)</td></tr>
<tr><td colspan="2">M2PG/GTK/SM サブチャンク（排他的)</td></tr>
<tr><td colspan="2">SPEC サブチャンク（M2PG があるときのみ存在)
（付加情報）</td></tr>
</table>

- M2PG 1.0
- GTF 1.0
- SM 1.0

M2PG (MUSIC 2D Polar Grid) は，LocalizeMUSIC の音源定位伝達関数ファイルのために用いられる．GTF (GHDSSTransfer Function) は，GHDSS の音源分離伝達関数ファイルにまた，SM (Separation `Matrix`) は，GHDSS の分離行列ファイルを表すために用いる．

fmt は，基本的なデータを入力するために使われる必須のサブチャンクであり，詳細を．表 5.3 に示す．なお，チャンクのサイズは，チャンク名の 4 バイトとチャンクサイズの 4 バイトを除いた部分のサイズの合計である．

表 5.3: fmt サブチャンクの定義

型	値
char[4]	“ fmt “
`int`	チャンクのサイズ，24+12*Nmic [bytes]
short[2]	fmt バージョン（メジャー，マイナー）
`int`	フォーマット ID (0: 未定義，1:M2PG, 2:GTF, 3:SM)
`int`	マイクロホン数 (Nmic)
`int`	FFT 長 [サンプル] (Nfft)
`int`	インパルスレスポンス長 [サンプル]
`int`	サンプリング周波数 [Hz]
Vector3[nmic]	マイクロホン位置の座標 [m]

M2PG/GTF/SM サブチャンクは，個々のファイル（音源定位伝達関数ファイル，音源分離伝達関数ファイル，分離行列ファイル）のデータが格納されるチャンクである．次節以降で，別途説明を行う．

SPEC は，ファイル作成条件など付加的な情報を入力するために用いられる．表 5.4 に SPEC のフォーマットを示す．各項目は，0 終端文字列とし，文字コードは ASCII （基本的に英数字など 1 バイト文字しか利用しない），改行コードは LF とする．

表 5.4: SPEC サブチャンクの定義

型	値
char[4]	“ SPEC ”
int	チャンクのサイズ，1551 [bytes]
short[2]	SPEC バージョン（メジャー，マイナー）
char[11]	作成日 (例：2009/05/19)
char[128]	場所（例：Smart room）
char[128]	ロボット（例：HRP-2, SIG2, ···）
char[128]	マイクロホンアレイのタイプ（例：8ch circular，8ch cubic, ···）
char[128]	作成者（例：Kazuhiro Nakadai）
char[1024]	備忘録（MUSIC の TF 作成のパラメータなど）

5.1.1 LocalizeMUSIC 用音源定位伝達関数

音源定位伝達関数は，音源からマイクロホンアレイまでの伝達関数を表し，M2PG チャンクを用った HGTF 形式のファイルとして表現する，音源定位伝達関数ファイルは，LocalizeMUSIC のプロパティ A_MATRIX に指定する．具体的にどのように使われるかは，LocalizeMUSIC を参照されたい．

M2PG の定義を表 5.5 に示す．なお，多次元配列のメモリの並びは row major (C 言語の場合と同じ) である．

表 5.5: M2PG サブチャンクの定義

型	値
char[4]	”M2PG”
int	チャンクのサイズ， $12 + 12 * Nd * Nr + 8 * Nd * Nr * nmic * (Nfft/2 + 1)$ [bytes]
short[2]	M2PG バージョン（メジャー，マイナー）
int	水平方向の要素数 (Nd)
int	距離の要素数 (Nr)
Vector3[Nd][Nr]	音源位置の座標
complex<float> [Nd][Nr][nmic][Nfft/2+1]	伝達関数

本ファイルは，インパルス応答形式のファイルから，サポートツールである harktool を使用して作成する．具体的な作成方法は 7.1 節で述べる．

5.1.2 GHDSS 用音源分離伝達関数

音源分離用伝達関数も，音源定位用伝達関数と同様に音源からマイクロホンアレイまでの伝達関数を表し，GTF サブチャンクを用った HGTF 形式のファイルとして表現する，音源分離伝達関数ファイルは，GHDSS のプロパティ TF_CONJ が true の時に，TF_CONJ_FILENAME に指定する．具体的にどのように使われるかは，GHDSS を参照されたい．

GTF サブチャンクの定義を表 5.6 に示す．多次元配列のメモリの並びは row major (C 言語の場合と同じ) である．

本ファイルは，インパルス応答形式のファイル，もしくは，マイクロホン位置と音源位置リストファイルからサポートツールである harktool を使用して作成する．具体的な作成方法は，7.1 節で述べる．

表 5.6: GTF サブチャンクの定義

型	値
char[4]	”GTF ”
int	チャンクのサイズ， 8+12*Ns+4+4*Nf+8*Ns*nmic*Nf [bytes]
short[2]	GTF バージョン（メジャー，マイナー）
int	音源位置数 (Ns)
Vector3[Ns]	音源位置の座標 [m]
int	周波数ライン数 (Nf)
int[Nf]	インデックスから周波数ビンへのマップ
complex<float> [Ns][nmic][Nf]	伝達関数

5.1.3 GHDSS 用 分離行列

分離行列は，SM サブチャンクを用った HGTF 形式のファイルとして表現する，音源分離伝達関数ファイルは，GHDSS のプロパティ INITW_FILENAME に指定することができる．また，EXPORT_W を true にした際に現れる EXPORT_W_FILENAME を指定することで分離行列ファイルを出力することができる．具体的にどのように使われるかは，GHDSS を参照されたい．

SM サブチャンクの定義を表 5.7 に示す．多次元配列のメモリの並びは row major (C 言語の場合と同じ) である．

表 5.7: SM サブチャンクの定義

型	値
char[4]	”SM ”
int	チャンクのサイズ， 8+12*Ns+4+4*Nf+8*Ns*nmic*Nf [bytes]
short[2]	SM バージョン（メジャー，マイナー）
int	音源位置数 (Ns)
Vector3[Ns]	音源位置の座標 [m]
int	周波数ライン数 (Nf)
int[Nf]	インデックスから周波数ビンへのマップ
complex<float> [Ns][nmic][Nf]	分離行列

5.2 HARK テキスト形式

HARK テキスト形式は，マイクロホン位置情報，ノイズ位置情報をサポートしているテキスト形式ファイルである．それぞれ，ファイル識別子とバージョン番号を持つヘッダーをファイルの先頭に記述する．

```
MICARY-LocationFile
Version=バージョン番号

Noise-LocationFile
Version=バージョン番号
```

行頭がシャープの行は，コメント行であるが，ファイルの作成条件等を記述しておく．

5.2.1 マイクロホン位置テキスト形式

マイクロホン位置テキスト形式は，GHDSS や harktool で使用するマイクロホンアレイの各マイクロホン位置を記述したファイルである．GHDSS モジュールの TF_CONJ プロパティで CALC を選択すると，MIC_FILENMAE プロパティが現れる．この欄にマイクロホン位置形式で保存されたマイクロホンアレイの各マイクロホン座標を与えると，インパルス応答をシミュレーションで生成，音源分離を行う．

以下にフォーマットを記す．

```
MICARY-LocationFile
Version=1.0
# date 2010/07/13
# place office
# robot HRP-2
# array_type   8ch circular
# author HARK team
# memo nothing special
# MUSIC_DIR=32
NumOfPosition=N
Position 0,x,y,z
Position 1,x,y,z
    ...
Position N-1,x,y,z
```

1 行目はヘッダ，2 行目はファイルバージョンである．3 行目から 9 行目はコメントで，ファイルの作成条件等を記述しておく．10 行目はマイクロホン数である．11 行目以降は各マイクロホンの位置を指定する．Position の次にマイクロホンの通し番号 (0 から) を書き，その後にコンマで区切って HARK 標準座標系で座標を指定する (単位はメートル [m])．

実際に 8 チャネルのマイクロホンアレイのマイクロホン位置ファイルの例を以下に示す．

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

```
Position 5,-0.0557,0.07815,0.0699
Position 6,0.0,0.0842,0.0808
Position 7,0.0479,0.0571,0.0958
```

5.2.2 ノイズ位置テキスト形式

ノイズ源ファイルは，GHDSS で固定方向からの定常ノイズ方向を指定するためのファイルである．GHDSS モジュールの FIXED_NOISE プロパティで true を選択すると， FIXED_NOISE_FILENAME プロパティが現れる．この欄にノイズ源形式の音源座標を与えると，その方向のノイズを既知ノイズ源として処理する．

以下にフォーマットを示す．

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

1 行目はヘッダ，2 行目はファイルバージョンである．3 行目から 9 行目はコメントであり，ファイルの作成条件等を記述しておく．10 行目はノイズ源数であり，1 を指定する．11 行目はノイズ源方向である．Positions の次に音源の通し番号を書き，原点からのノイズ源方向を，3 次元のベクトル（大きさ 1）で示す．音源 ID は，音源分離処理内で，-1 に割り振られるため，Position にも -1 を指定することが望ましい．なお，指定できるノイズ源の数は，1 つである．

以下のノイズ源ファイルは，ロボットの後方に定常ノイズを仮定する場合の記述例である．

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

5.3 その他のファイル形式

その他のファイル形式には， srcinf テキスト形式，PCM バイナリ, float バイナリ，定位結果テキスト，Map テキスト形式が挙げられる．これらについて説明する．

5.3.1 音源位置リスト情報 (srcinf) 形式

音源定位伝達関数ファイル（LocalizeMUSIC の A_MATRIX に設定）や音源分離伝達関数ファイル（GHDSS の TF_CONJ_FILE に設定）を作成するため harktool で用いるファイルである．測定方向とそれに対応するインパルス応答ファイルを「**測定座標**，%%**ファイル名**，**オフセット**」の形で 1 行 1 レコードとして (「%%」の前に半角スペースが必要)，インパルス応答の測定点数分，羅列したテキストファイルである．

測定座標 マイクロホンアレイの中心（原点）から測定場所への方向ベクトル (x，y，z) を表す．ベクトルの大きさ $\sqrt{x^2+y^2+z^2}$ が 1.0 になるように記述する．また，各値はそれぞれコンマで区切る．

ファイル名 測定したインパルス応答のファイル名を記述する．ファイル名は example-1.flt，example-2.flt，...，example-N.flt (”exampl“ 部分は任意，N はチャネル番号) の形式で命名する必要がある．Srcinfo.txt ではこれらをまとめて， example-*.flt と指定する．チャネル番号は 1 から始まる点に注意．

オフセット インパルス応答ファイルの先頭から，何番目のデータ（サンプル）から読込みを開始するかを指定する．例えば 1 を指定すれば，1 サンプル目から読み込みを開始する．通常は 1 で問題はない．オフセットは正の整数とする必要がある．

以下に具体例を示す．ここでは，4 チャネルマイクロホンアレーを用い，HARK 標準座標系を使うと仮定している．

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

ただし，本設定ファイル例はカレントディレクトリに以下のインパルス応答ファイル群があることを仮定している．

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

5.3.2 PCM バイナリ形式

音声波形を PCM 形式で表したファイル形式．分離音の時間波形をファイル保存する際などに用いる．各サンプル値は 16 ビットあるいは 24 ビット符号付整数をリトルエンディアンで表し，ヘッダなど付加的情報は用いない．

HARK では，SaveRawPCM が出力し，標準の拡張子は .sw である．複数チャネル存在する音声波形の場合には，チャネルでマルチプレクスして表現する．従って，本形式のファイルを別プログラムで読み書きする際には，sox などを用いて wav ファイル等に変換して使用するか，適切なサンプリング周波数とトラック数，量子化ビット数を指定する必要がある．

5.3.3 float バイナリ

実数あるいは複素数を IEEE 758 の 32 ビット浮動小数点数，リトルエンディアンで格納したファイルであり，HARK では，主に，特徴ベクトルやインパルス応答データに用いられるファイル形式である．

特徴ベクトル

特徴ベクトルは，実数あるいは複素数を要素にもつベクトル値である．SaveFeatures で出力され，拡張子は .spec が用いられる．ベクトルの次元要素は低次元から高次元方向に書き出される．ベクトル値がもつ意味は，ユーザの使用に依存する．HARK で通常用いる特徴は，パワースペクトル，複素スペクトル，MFCC，MSLS, ミシングフィーチャマスクである．複素スペクトルは，複素数を要素にもつベクトル値で，その他は実数を要素にもつベクトル値である．ヘッダ情報はないため， 特徴ベクトルの次元数や，ベクトルの数は，記録されないことに注意されたい．

インパルス応答形式

音源位置とマイクロホン間のインパルス応答を表す．標準の拡張子は .flt であり，harktool で伝達関数の行列を作成するために用いられる．

本形式のファイルをテキストに変換するには，例えば次のようなプログラムを作成する．

```
#include <stdio.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
  int fd=0;
  float data=0;

  fd = open(argv[1], O_RDONLY);
  while(1){
    int ret = read(fd, &data, 4); /* 4 [byte] = 32 [bit] */
    if( ret != 4){
      break;
    }
    printf("%f\n", data);
  }
  close(fd);
  return (0);
}
```

5.3.4 定位結果テキスト

音源定位結果を記録するファイル形式である．LoadSourceLocation, SaveSourceLocation モジュールで用いられる．プロパティの FILENAME に本形式で保存されたファイルを指定することで読み書きできる．

音源定位結果フォーマットは，次の情報で構成されている．フレーム番号を先頭に，N 個分の音源情報を 1 行に表示する構成となっている．

例えば，N 個の音源が M フレーム分定位された場合，次の形式のテキストファイルが保存される．

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

1 列目は入力されたフレームの通し番号で，それ以降は，4 列を 1 セット（1 つ分の音源情報）として，同時に検出された音源数に応じて列が増える．音源情報 1 セット中の 1 列目は音源の ID，2-4 列目は，音源の方向を単位球上に直交座標系である．

5.3.5 Map テキスト

DataLogger で出力されるファイル形式である．`Map<int, float>`, `Map<int, int>`または `Map<int, ObjectRef>`をサポートしている（`Map<int, ObjectRef>`の `ObjectRef` は，`Vector< float >` または `Vector< complex >` のみ）．

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 . . .

1 フレーム 1 行形式で，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．

第6章　モジュールリファレンス

本章では，各モジュールの詳細な情報を示す．はじめに，モジュールリファレンスの読み方について述べる．

モジュールリファレンスの読み方

1. **モジュールの概要:** そのモジュール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:** そのモジュール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:** どんなときにそのモジュールを使えばよいのかと，具体的な接続例について述べる．とにかく当該モジュール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. **モジュールの入出力とプロパティ:** モジュール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. **モジュールの詳細:** そのモジュールの理論的背景や実装の方法を含む詳細な解説を述べる．詳しく当該モジュール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，JEOL 日本電子システムテクノロジー（株），RASP シリーズ，東京エレクトロンデバイス TD-BD-16ADUSB，ALSA ベースのデバイス (例，RME Hammerfall DSP Multiface シリーズ) である．各種デバイスの導入は，第 8 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 に AudioStreamFromMic モジュールの使用例を示す．

デバイスの写真

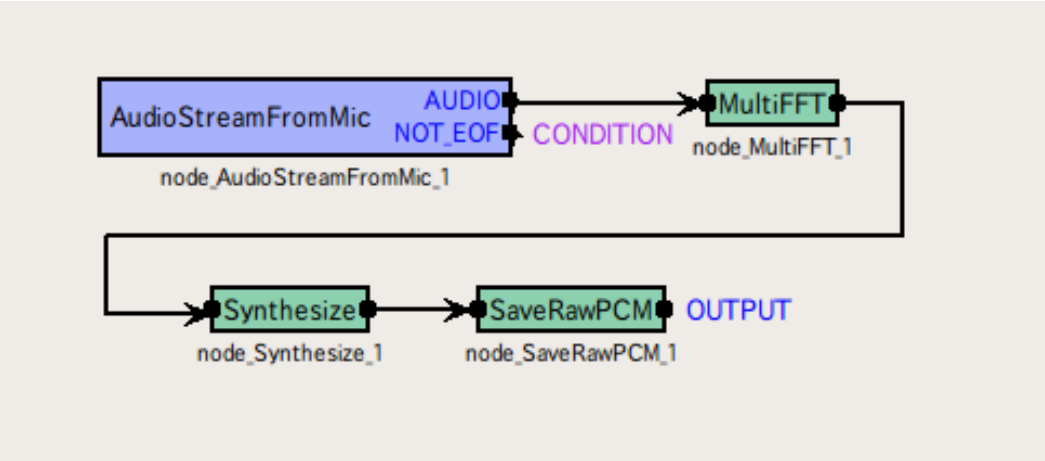

図 6.1: AudioStreamFromMic の接続例

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 6.2 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．

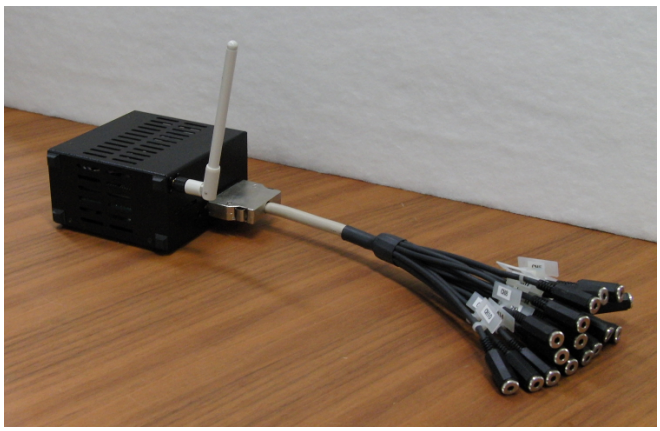

図 6.2: 無線 RASP

2．RME Hammerfall DSP Multiface シリーズ　図 6.3，6.4 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 6.4)．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．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 6.3: RME Hammerfall DSP Multiface の正面

図 6.4: RME Hammerfall DSP Multiface の背面

モジュールの入出力とプロパティ

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		[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．
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．

入力

無し．

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．

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 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，SINICH，RASP，WS 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．TD-BD-16ADUSB を使用する場合には SINICH を選択する．RASP2 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．

DEVICE : `string` 型．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，

1. JEOL 日本電子システムテクノロジー（株）．RASP シリーズ，
2. 東京エレクトロンデバイス TD-BD-16ADUSB，
3. ALSA ベースのデバイス (例，RME Hammerfall DSP シリーズ Multiface) である．

の 3 種である．以下ではそれぞれのデバイスを用いる際の注意点を記す．

RASP シリーズ: ここでは，**RASP-2** と，**無線 RASP** 利用する際のパラメータ設定を記す．

RASP-2

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

無線 RASP

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，AudioStream-FromMic 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．

TD-BD-16ADUSB:

CHANNEL_COUNT	16
DEVICETYPE	SINICH
DEVICE	SINICH

ALSA 対応デバイス:

CHANNEL_COUNT	8
DEVICETYPE	ALSA
DEVICE	plughw:0,1
備考	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．

6.1.2 AudioStreamFromWave

モジュールの概要

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

このモジュールは，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 6.5，6.6 に AudioStreamFromWave モジュールの使用例を示す．

図 6.5 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 モジュール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 6.6 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 モジュールでファイルディスクリプタを生成する．そして，InputStream モジュールの出力を，AudioStreamFromWave など HARK の各種モジュールのネットワークがある iterator サブネットワーク（図 6.6 中の LOAD_WAVE）に接続する．

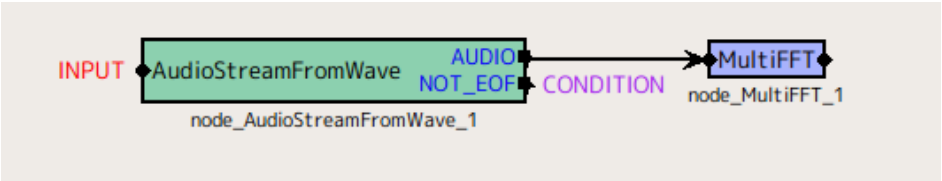

図 6.5: AudioStreamFromWave の接続例 1

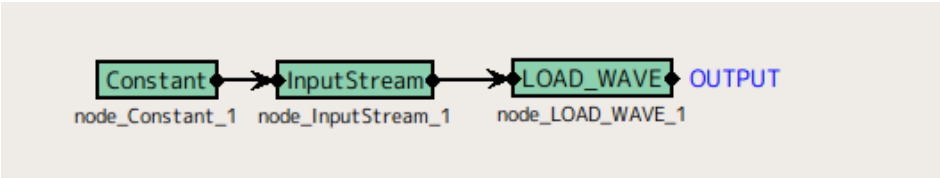

図 6.6: AudioStreamFromWave の接続例 2

モジュールの入出力とプロパティ

表 6.3: 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 標準モジュールの, IO カテゴリにある InputStream モジュール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.

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.

モジュールの詳細

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モジュールがある (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.

6.1.3 SaveRawPCM

モジュールの概要

時間領域の音声波形データをファイルに保存する．出力されるバイナリファイルは，サンプル点が 16 [bit] または 24 [bit] 整数で記録された Raw PCM サウンドデータである．入力データの型によって，マルチチャネル音声データ，または，分離音ごとのモノラル音声データが出力される．

必要なファイル

無し．

使用方法

どんなときに使うのか

分離音を Synthesize モジュールで波形にして，音を実際に確認したい場合や，AudioStreamFromMic モジュールと組みわせて，マイクロホンアレイの録音を行う場合に用いる．

典型的な接続例

図 6.7，6.8 に SaveRawPCM の使用例を示す．

図 6.7 は，AudioStreamFromMic からの多チャネル音響信号を SaveRawPCM モジュールでファイルに保存する例である．例のように，ChannelSelector モジュールで保存するチャネルを選択することが出来る．また，SaveRawPCM は `Map<int, ObjectRef>`型入力を受け付けるので，MatrixToMap モジュールによって `Matrix<float>`型から `Map<int, ObjectRef>`型へ変換している．

図 6.8 は，分離音を SaveRawPCM モジュールによって保存する例である．GHDSS モジュールや，分離後のノイズ抑圧を行う PostFilter モジュールから出力される分離音は周波数領域にあるので，Synthesize モジュールによって時間領域の波形に変換されたのち，SaveRawPCM モジュールに入力される．WhiteNoiseAdder モジュールは，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．

モジュールの入出力とプロパティ

表 6.4: 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プレフィックス．
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．

入力

INPUT : `Map<int, ObjectRef>`または `Matrix<float>`型．前者は分離音など，音源 ID と波形データの構造体，後者はマルチチャネルの波形データ行列．

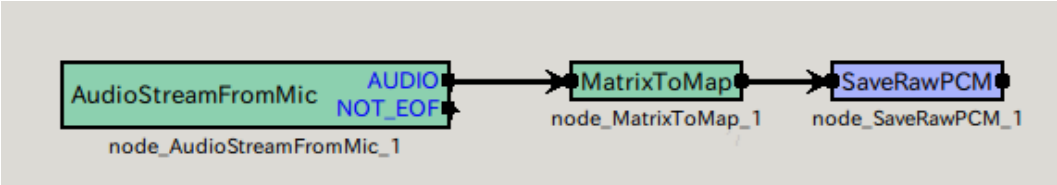

図 6.7: SaveRawPCM の接続例 1

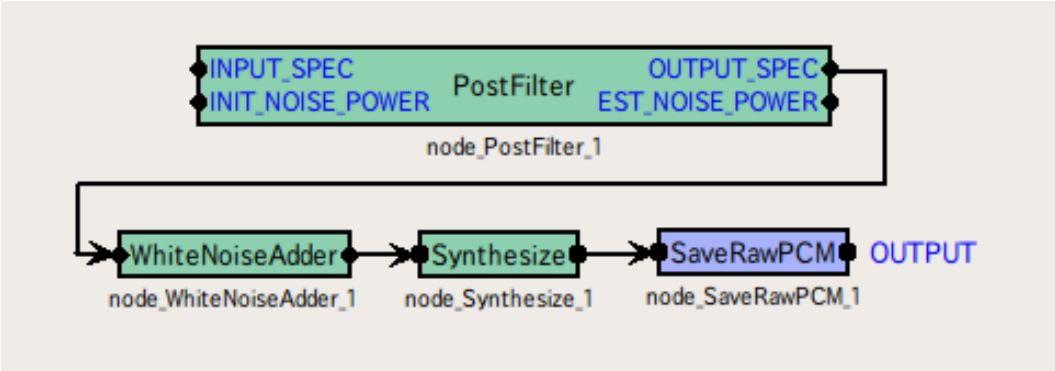

図 6.8: SaveRawPCM の接続例 2

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_．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 “BASENAMEID．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．他のモジュールの 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波形の量子化ビット数．16 または 24 を指定可．

モジュールの詳細

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 サウンド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マルチチャネル音声ファイルとなる．

`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ファイルが書き出される．

6.1.4 HarkDataStreamSender

モジュールの概要

以下の音響信号処理結果をソケット通信で送信するモジュールである．

- 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

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 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 6.9 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「モジュールの詳細」を参照．

モジュールの入出力とプロパティ

表 6.5: 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	[bytes]	ソケット通信のために確保するバッファサイズ
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

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）

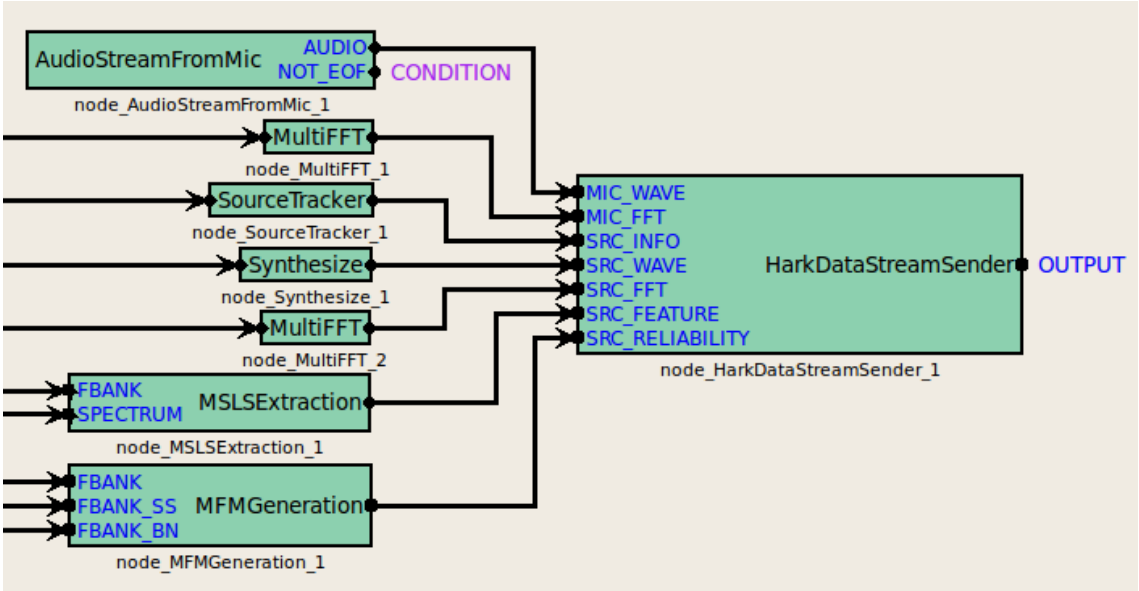

図 6.9: HarkDataStreamSender の接続例

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．

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．

出力

OUTPUT : `ObjectRef` 型．ダミー出力．

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．

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．

モジュールの詳細

(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．
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 6.16 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)`

図 6.10: 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 6.11 参照.
advance	int	フレームのシフト長
count	int	HARK のフレーム番号

図 6.11: 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

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，送信されたデータを解釈することができる．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 0000010 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 0100101 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 0000000 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)`

図 6.12: 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)`

図 6.13: 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)`

図 6.14: 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

図 6.15: 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
(a)	HD_Header	`sizeof(HD_Header)`	○	○	○	○	○	○	○
(b)	HDH_MicData	`sizeof(HDH_MicData)`	○						
(c)	float[]	`HDH_MicData.data_bytes`	○						
(d)	HDH_MicData	`sizeof(HDH_MicData)`		○					
(e)	float[]	`HDH_MicData.data_bytes`		○					
(f)	float[]	`HDH_MicData.data_bytes`		○					
(g)	int	`1 * sizeof(int)`			○	○*	○*	○*	○*
(h)	HDH_SrcInfo	`sizeof(HDH_SrcInfo)`			○	○*	○*	○*	○*
(i)	HDH_SrcData	`sizeof(HDH_SrcData)`				○*			
(j)	short int[]	`HDH_SrcData.data_bytes`				○*			
(k)	HDH_SrcData	`sizeof(HD_SrcData)`					○*		
(l)	float[]	`HDH_SrcData.data_bytes`					○*		
(m)	float[]	`HDH_SrcData.data_bytes`					○*		
(n)	HDH_SrcData	`sizeof(HD_SrcData)`						○*	
(o)	float[]	`HDH_SrcData.data_bytes`						○*	
(p)	HDH_SrcData	`sizeof(HD_SrcData)`							○*
(q)	float[]	`HDH_SrcData.data_bytes`							○*

図 6.16: 送信データ詳細

	説明	Debug
(a)	送信データヘッダ．表 6.10 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 6.12 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 6.12 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 6.13 参照．	○
(i)	音源 ID 毎の音響信号の構造（フレーム長，送信バイト数）を表す構造体．表 6.14 参照．	○
(j)	音源 ID 毎の音響信号（フレーム長 の float 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 6.14 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 6.14 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 6.14 参照．	○
(q)	音源 ID 毎の MFM （特徴量次元数 の float 型の一次元配列）	

送信データは各フレーム毎に，表 6.15，表 6.16 の (a)-(q) のように，分割されて行われる．表 6.15 に，送信データ (a)-(q) と，接続された入力端子の関係を，表 6.16 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．

```
calculate{

   Send (a)

   IF MIC_WAVE is connected
       Send (b)
       Send (c)
   ENDIF

   IF MIC_FFT is connected
       Send (d)
       Send (e)
       Send (f)
   ENDIF

   IF SRC_INFO is connected

       Send (g) (Let the number of sounds 'src_num'.)

       FOR i = 1 to src_num (This is a sound ID based routine.)

           Send (h)

           IF SRC_WAVE is connected
               Send (i)
               Send (j)
           ENDIF

           IF SRC_FFT is connected
               Send (k)
               Send (l)
               Send (m)
           ENDIF

           IF SRC_FEATURE is connected
               Send (n)
               Send (o)
           ENDIF

           IF SRC_RELIABILITY is connected
               Send (p)
               Send (q)
           ENDIF

       ENDFOR

   ENDIF

}
```

ここで，コード内の (a)-(q) が，表 6.15 と表 6.16 の (a)-(q) に対応している．

6.2 Localization カテゴリ

6.2.1 ConstantLocalization

モジュールの概要

一定の音源定位結果を出力し続けるモジュール．パラメータは，ANGLES と ELEVATIONS の二つであり，それぞれに音源が到来する方位角 (ANGLES) と仰角 (ELEVATIONS) を設定する．各パラメータは `Vector` になっているので，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 6.17 に接続例を示す．このネットワークでは，一定の定位結果を表示し続ける．

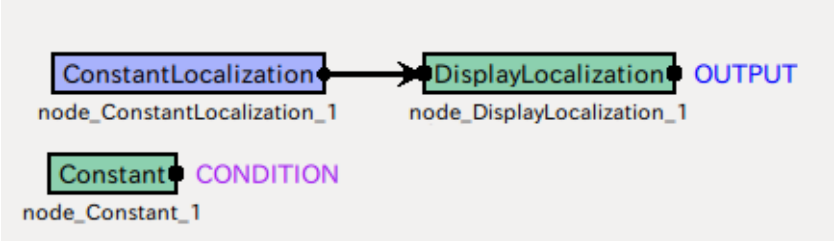

図 6.17: ConstantLocalization の接続例

モジュール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 6.6: 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	`Vector< float >`		[deg]	音源の方位角 (左右の向き)
ELEVATIONS	`Vector< float >`		[deg]	音源の仰角 (上下の向き)

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.

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.

モジュールの詳細

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 >` $a_1 \ldots a_N$>

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

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．

$$x_i = \cos(a_i\pi/180)\cos(e_i\pi/180) \tag{6.1}$$

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

$$z_i = \sin(e_i\pi/180) \tag{6.3}$$

音源の座標の他に， ConstantLocalization は音源のパワー (1.0 で固定) と音源の ID (i に対応) も出力する．

6.2.2 DisplayLocalization

モジュールの概要

音源定位結果を GTK を使って表示するモジュールである．

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，定位モジュールの後に接続する．図 6.18 では，ConstantLocalization からの固定の定位結果を表示し続ける．

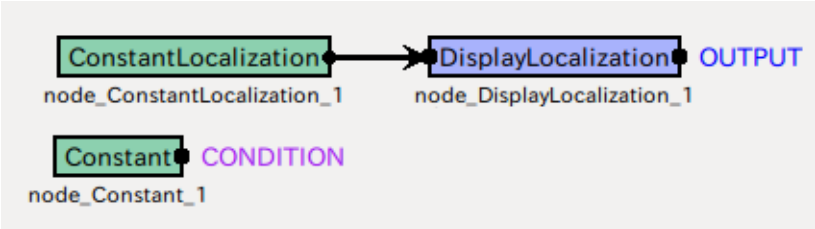

図 6.18: DisplayLocalization の接続例

モジュール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 6.7: 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_LENGTH	`int`	1000	Frame	音源定位結果を表示するディスプレイの長さ

WINDOW_LENGTH]: `int` 型．デフォルトは 1000．見たい音源定位結果の長さに応じてこのパラメータを調整することができる．

モジュールの詳細

異なる ID は異なる色で表示されるが，色自体に意味は無い.

6.2.3 LocalizeMUSIC

モジュールの概要

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．

必要なファイル

ステアリングベクトルからなる行列 (A_MATRIX) 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

ロボットに音源の方向を向かせるなどの用途で，音源の方向を推定したい場合，および GHDSS などの音源分離などの入力として必要な音源方向を推定する場合に用いる．

典型的な接続例

典型的な接続例を図 6.19 に示す．

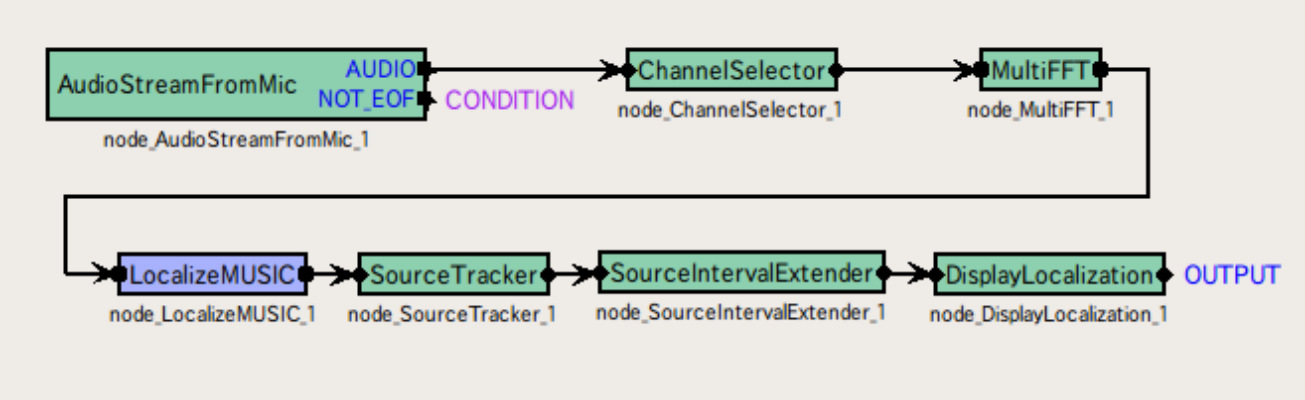

図 6.19: LocalizeMUSIC の接続例

モジュール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 $M \times (NFFT/2+1)$.

出力

OUTPUT : `Vector<ObjectRef>`，音源位置（方向），`ObjectRef` は，音源方向と MUSIC スペクトルのパワーからなる構造体，`Vector` の要素数は音源数 (N) .

パラメータ

表 6.8: 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
NB_CHANNELS	int	8	[ch]	入力信号のチャネル数
LENGTH	int	512	[pt]	FFT の窓長 (=FFT のポイント数)
SAMPLING_RATE	int	16000	[Hz]	サンプリングレート
A_MATRIX	string			ステアリングベクトルのファイル名
ELEVATION	float	16.7	[deg]	音源の仰角（固定）
PERIOD	int	50	[frame]	定位の平滑化フレーム数
NUM_SOURCE	int	2		MUSIC で仮定する音源数
MIN_DEG	int	-180	[deg]	ステアする方向（方位角）の最小値
MAX_DEG	int	180	[deg]	ステアする方向（方位角）の最大値
LOWER_BOUND_FREQUENCY	int	500	[Hz]	定位処理で用いる周波数の最小値
UPPER_BOUND_FREQUENCY	int	2800	[Hz]	定位処理で用いる周波数の最大値
DEBUG	bool	false		デバッグ情報出力の ON/OFF

NB_CHANNELS 入力の信号のチャネル数．入力信号とそろえる必要がある．

LENGTH 前段の MultiFFT，AudioStreamFromWave，AudioStreamFromMic といったモジュールと合わせる必要がある．

SAMPLING_RATE LENGTH と同様，他のモジュールとそろえる必要がある．

A_MATRIX ファイルの作成については，tftool を参照．

ELEVATION 仰角は，ロボットと人との位置関係から想定される値を設定すること．たとえば，ロボットのマイクロホンアレイの高さが 1.2 [m]，人の口の位置が 1.5 [m]，人とロボットの距離を約 1.0 [m] とした場合，$\arctan \frac{1.5-1.2}{1.0}$[rad]= 16.7 [deg] となる．

PERIOD 定位の平滑化フレーム数を表す．モジュール詳細の式 6.7 中に *PERIOD* で表わされる値．この値を大きくすると，相関行列は安定するが，定位結果を得るための時間間隔が大きくなり，また音源が移動する場合には，音源の追従性が悪くなる．

NUM_SOURCE MUSIC で仮定する音源数であり，モジュール詳細で N_s で表わされる値である．事前に決めておく必要がある．一般的には 1〜2 の値を選択すれば十分である．

MIN_DEG 音源探索する際の最小角度であり，モジュール詳細で 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 音源探索する際の最大角度であり，モジュール詳細で 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 周波数方向に統合を行う際に利用する最小周波数値であり，下記では，ω_{min} で表わされている．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 周波数方向に統合を行う際に利用する最大周波数値であり，下記では，ω_{max} で表わされている．LOWER_BOUND_FREQUENCY < UPPER_BOUND_FREQUENCY である必要がある．

DEBUG 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，音源数分だけ，ID，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．方向は，degree で方位角を小数の位を丸めた整数値が表示される．パワーは MUSIC スペクトルのパワー値（下記の $P_a(\theta)$ の値）がそのまま出力される．次に，改行後，“MUSIC spectrum: ” と出力され，$P_a(\theta)$ の値が，すべての θ について表示される．

モジュールの詳細

MUSIC 法は，入力信号の相関の固有値を利用して，音源方向の推定を行う手法である．

伝達関数の生成:

まず，事前に音源から各マイクロホンまでの伝達関数を生成する．マイクロホンマイクロホンアレイからみて，θ 方向にある音源 S_θ から i 番目のマイク M_i までの伝達関数を $h_i(\theta, \omega)$ と周波数領域で定義を行い，これらをまとめて，音源 S_θ からマイクロホンアレイへの伝達関数を下記のように定義する．

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

適当な間隔 $\Delta\theta$ 毎に，事前に $\boldsymbol{H}(\theta, \omega)$ を計算もしくは計測し，用意しておく．HARK のサイトから，$\Delta\theta = 5[deg]$ として，水平方向に一周計測した伝達関数をダウンロードできるようになっている．また，tftool を用いれば，マイクロホン配置情報を入力することによって伝達関数を自動的に計算することもできる．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．

相関行列の推定:

次に，時間領域での N チャネルの入力信号を $\mathrm{x}(t) = [x_1(t), x_2(t), x_3(t), \cdots, x_N(t)]^T$ とする．周波数領域では以下のように表現できる．

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

ここで，ω は周波数， f はフレームを表す．時間 t とフレーム f の関係は MultiFFT を参照されたい．

入力信号 $\boldsymbol{X}(\omega, f)$ 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．

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

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．

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

この平滑化を行うため，LocalizeMUSIC の出力は，各フレームごとではなく，PERIOD フレームごとに一回となる．

MUSIC スペクトルの算出:

以降では，表記の簡略化のために，f を省略する．MUSIC 法では，この入力の相関行列 $\boldsymbol{R}'$ の性質を積極的に利用する．まず，$\boldsymbol{R}'$ に対して固有値分解を行う．

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

ここで，$\boldsymbol{P}$ は互いに直交する固有ベクトルからなる行列 $\boldsymbol{P}(\omega) = [e_1(\omega), e_2(\omega), \cdots, e_M(\omega)]$ を，$\boldsymbol{D}(\omega)$ 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，$\boldsymbol{D}(\omega)$ 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．

$\boldsymbol{P}(\omega)$ の固有ベクトルに対応する固有値のうち，値が大きいものを音源，小さいものをノイズに由来するという仮定を置く．音源数を N_s とすれば，れば，$[e_1(\omega), \cdots, e_{N_s}(\omega)]$ が音源に対応する固有ベクトル，$[e_{N_s+1}(\omega), \cdots, e_M(\omega)]$ がノイズに対応する固有ベクトルとなる．

MUSIC 法では入力信号から得られる固有ベクトルのうち，ノイズに由来すると仮定する固有ベクトルと事前に用意した伝達関数を用いて，下記のようにして MUSIC スペクトルを求める．

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

右辺の分母は，入力のうちノイズ源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作られる空間上では，小さい固有値に対応するノイズと大きい固有値に対応する目的信号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，$P(\theta, \omega)$ 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．

$P(\theta, \omega)$ 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．HARK では，周波数方向統合の重みとして，便宜上，最大固有値の平方根を利用している．

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

音源の探索:

次に $P_a(\theta)$ について $\theta_{min} \sim \theta_{max}$ の範囲からローカルピークを検出し，値の大きい順に，上位 N_s 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 N_s 個に満たない場合は，出力が N_s 以下になることもある．

参考文献

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

(2) 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.

6.2.4 LoadSourceLocation

モジュールの概要

SaveSourceLocation モジュールで保存された音源定位結果を読み込むモジュール．

必要なファイル

SaveSourceLocation 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

音源定位した結果を再度使いたいとき．完全に同じ音源定位結果を元に，異なる音源分離手法を評価するときなど．

典型的な接続例

図 6.20 は，保存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．このように，定位した結果を後で使う場合に用いる．

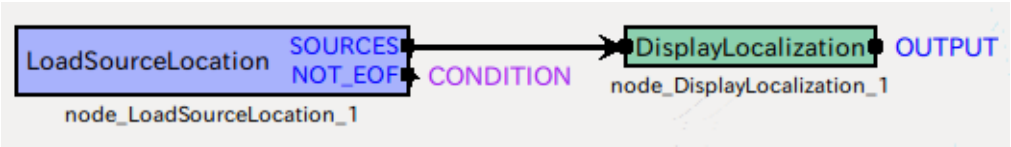

図 6.20: LoadSourceLocation の接続例

モジュール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，音源定位モジュール (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，Iteration のサブネットワークの終了条件をこの端子にするとファイルを最後まで読ませることができる．

パラメータ

表 6.9: 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`	無し		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．

モジュールの詳細

モジュール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.

6.2.5 SaveSourceLocation

モジュールの概要

音源定位結果をファイルに保存するモジュール．

必要なファイル

無し．

使用方法

どんなときに使うのか

定位結果を後で解析するために，定位結果をファイルに保存するときに用いる．保存したファイルの読み込みには， LoadSourceLocation モジュールを用いる．

典型的な接続例

図 6.21 に典型的な接続例を示す．例で示すネットワークは，固定の定位結果をファイルに保存する機能をもつ．保存形式は，5.3.4 を参照のこと．その他にも，定位結果を出力するモジュール LocalizeMUSIC，ConstantLocalization，LoadSourceLocation など) に接続される．

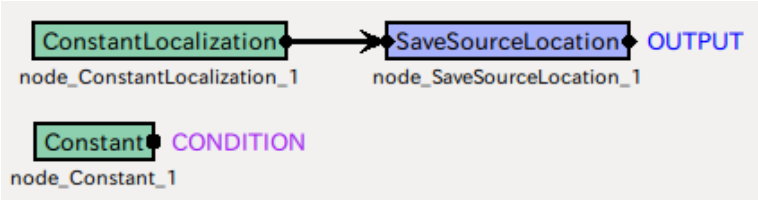

図 6.21: SaveSourceLocation の接続例

モジュール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` 型のデータである．

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 6.10: 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`			セーブしたいファイルの名前

6.2.6 SourceIntervalExtender

モジュールの概要

音源定位結果を，実際よりも早く出力させたいときに使うモジュール．パラメータ PREROLL_LENGTH に与えた分だけ，実際の定位結果が出力されるよりも定位結果が早く出力される．

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，分離の前処理として用いる．音源定位は音が入力されてから定位するので，実際の音が発生した時間よりもやや遅れてしまう．そのため，この遅延時間分だけ分離音の先頭が切れてしまう．この問題を防ぐために用いる．

典型的な接続例

図 6.22 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，その間に SourceIntervalExtender モジュールを挿入することで，音源分離の開始が遅れる問題を回避できる．

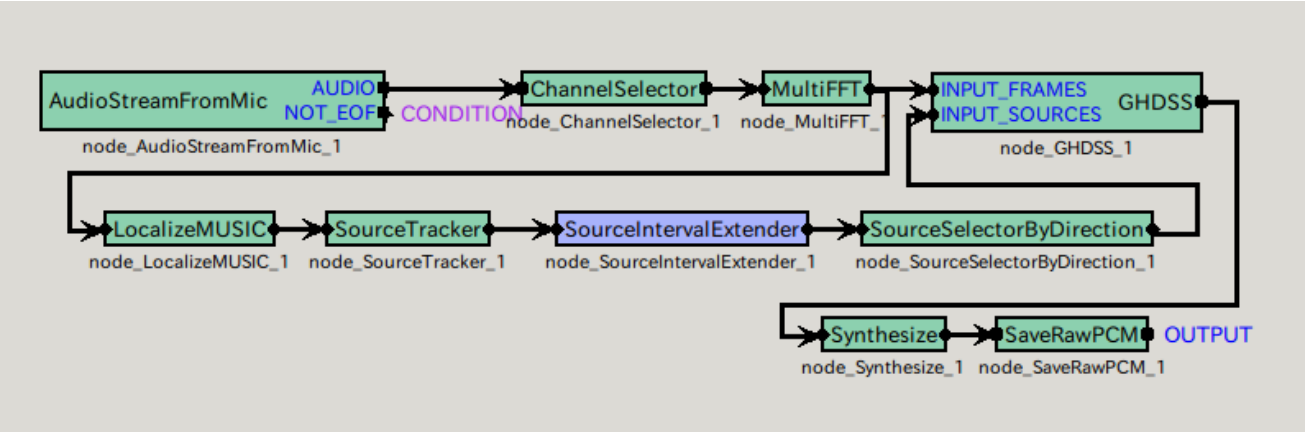

図 6.22: SourceIntervalExtender の接続例

モジュール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．音源定位結果が早く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 6.11: 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．どれだけ早く定位結果を出力するかを決定する．値が小さすぎると音源分離の開始が遅れるので，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．

モジュールの詳細

SourceIntervalExtender 無しで定位結果を元に音源分離を行ったとき，図 6.23 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，本モジュールを使って定位結果を事前に出力させる必要がある．

本モジュールは，各繰り返しで PREROLL_LENGTH 分だけ入力を先読みし，先読みした先に定位結果があれば，定位結果を発見した時点から定位結果の出力を開始する (図 6.24 参照．)

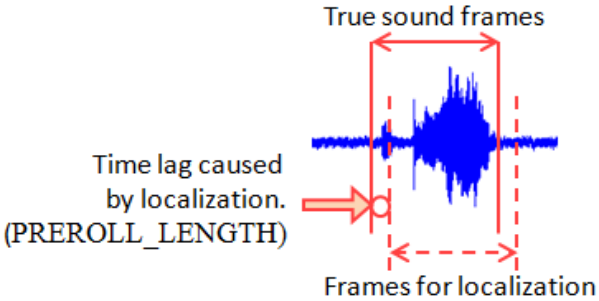

図 6.23: 音源定位結果を実際より早く出力する必要性

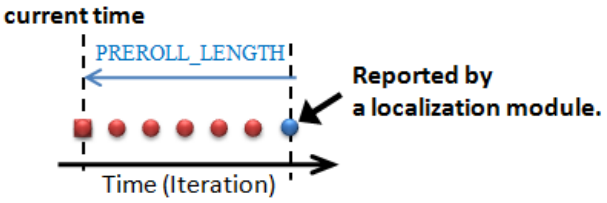

図 6.24: SourceIntervalExtender の動作

6.2.7 SourceTracker

モジュールの概要

時系列で入力された，ID 無しの音源定位結果に対して，近い方向から到来した音源定位結果には同じ ID を，異なる方向から到来した音源定位結果には異なる ID を与えるモジュール．本モジュール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

音源定位結果は，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），変動する．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，音源定位結果が，同一音源からの定位結果であるように音源 ID を統一するためには，音源定位結果を追従する必要がある．SourceTracker では，音源定位結果を十分近い音源定位結果に対して同じ音源の ID を与えるというアルゴリズムを用いている．十分近い音源かどうかを判断する基準として，閾値を角度で設定できる．本モジュールを用いることで，音源に ID が付与され， ID 毎の処理を行える.

典型的な接続例

通常は，ConstantLocalization，LocalizeMUSIC などの音源定位モジュールの出力を本モジュール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モジュール GHDSS などや音源定位結果の表示モジュール (DisplayLocalization) に接続する.

図 6.25 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 Constant-Localization 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.

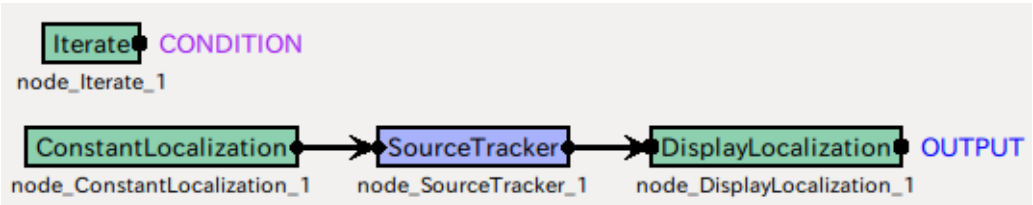

図 6.25: SourceTracker の接続例

モジュール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．

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 6.12: 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`			音源のパワーが THRESH より小さければ無視．
PAUSE_LENGTH	`float`	800	[frame*10]	音響ストリーム終了判定に必要な時間長．この間定位が観測されないストリームは終了する．
MIN_SRC_INTERVAL	`float`	20	[deg]	同一の音源とみなす角度差の閾値

THRESH : `float` 型．音源定位結果を無視すべきノイズか否かを，そのパワーで判定する．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．

MIN_SRC_INTERVAL : `float` 型．音源定位結果が 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とによって，音源定位のゆらぎの影響を軽減する．

モジュールの詳細

記号の定義:

まず，本節で用いる記号を定義する．

1. ID : 音源の ID
2. パワー p: 定位された方向のパワー．
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．

現在モジュール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 とし，それぞれに ^{last}，^{cur} の添字をつけて区別する．例えば，i 番号めの新たに入力された音源のパワーは 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 θ 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，単位円上の座標 $\boldsymbol{q} = (x, y, z)$ と $\boldsymbol{q}' = (x', y', z')$ で表現すると次のように求まる．

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

ここで，$\boldsymbol{q}$，$\boldsymbol{q}'$ は単位円上の点なので，$|\boldsymbol{q}| = |\boldsymbol{q}'| = 1$ である．従って絶対値の項は消去できて，

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

ここで逆三角関数を用いると，θ が求まる．

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

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．

音源追従方法:

SourceTracker が音源追従の際に行う処理を図 6.26 にまとめる．図は，横軸が時間 (=繰り返し回数) で，縦軸が音源方向を表す．また，青い丸が既にモジュール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，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，既にモジュール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．モジュールが保持しているが，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，既にモジュール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．

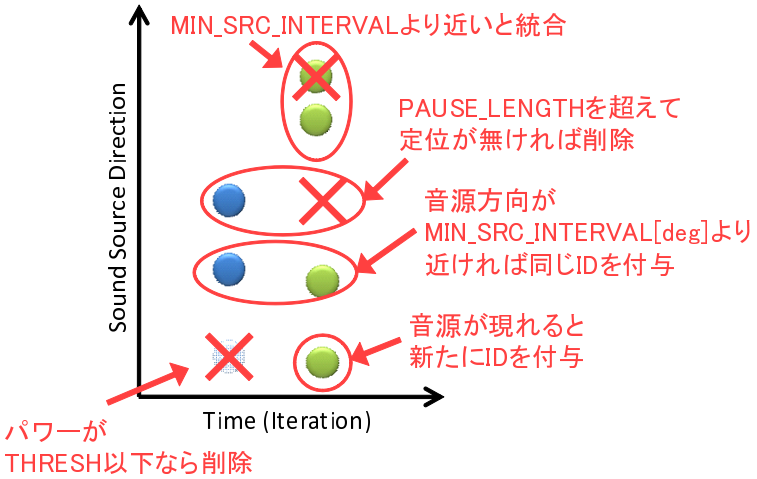

図 6.26: SourceTracker の音源追従方法

6.3 Separation カテゴリ

6.3.1 BGNEstimator

モジュールの概要

マルチチャネル信号のパワースペクトルから，信号に含まれるファンノイズなどの定常ノイズ (あるいは，背景ノイズ，BackGround Noise) を推定する．推定した定常ノイズは，PostFilter モジュールで使用される．

必要なファイル

無し．

使用方法

どんなときに使うのか

信号に含まれるファンノイズなどの定常ノイズ (あるいは，背景ノイズ，BackGround Noise) を推定する．この推定値が必要になるモジュールは，PostFilter である．PostFilter モジュールでは，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 モジュールの接続例を図 6.27 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 モジュールで利用される．

モジュールの入出力とプロパティ

表 6.13: 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

入力

INPUT : `Matrix<float>`型． マルチチャネルパワースペクトル

出力

OUTPUT : `Matrix<float>`型． 推定された定常ノイズのパワースペクトル．

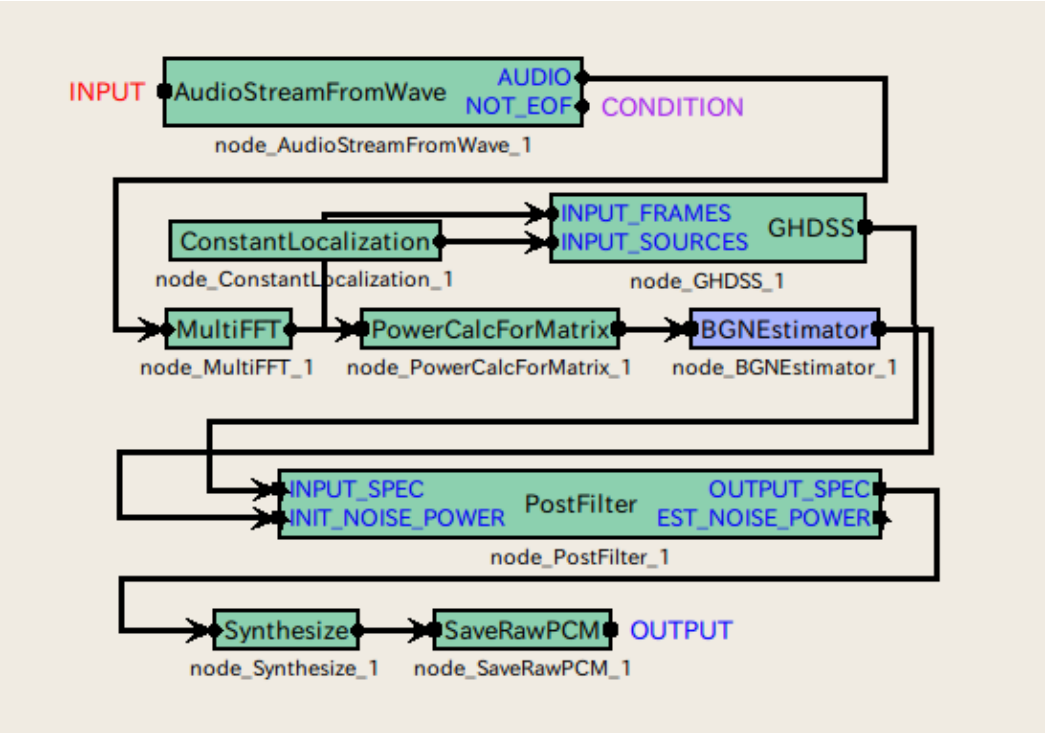

図 6.27: BGNEstimator の接続例

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 モジュールなどで指定される 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．

モジュールの詳細

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 6.28 の通り．

表 6.14: 変数表

変数名	対応パラメータ，または，説明
$\boldsymbol{S}(f,k_i) = [S_1(f,k_i),...,S_M(f,k_i)]^T$	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
$\boldsymbol{\lambda}(f,k_i) = [\lambda_1(f,k_i),...,\lambda_M(f,k_i)]^T$	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

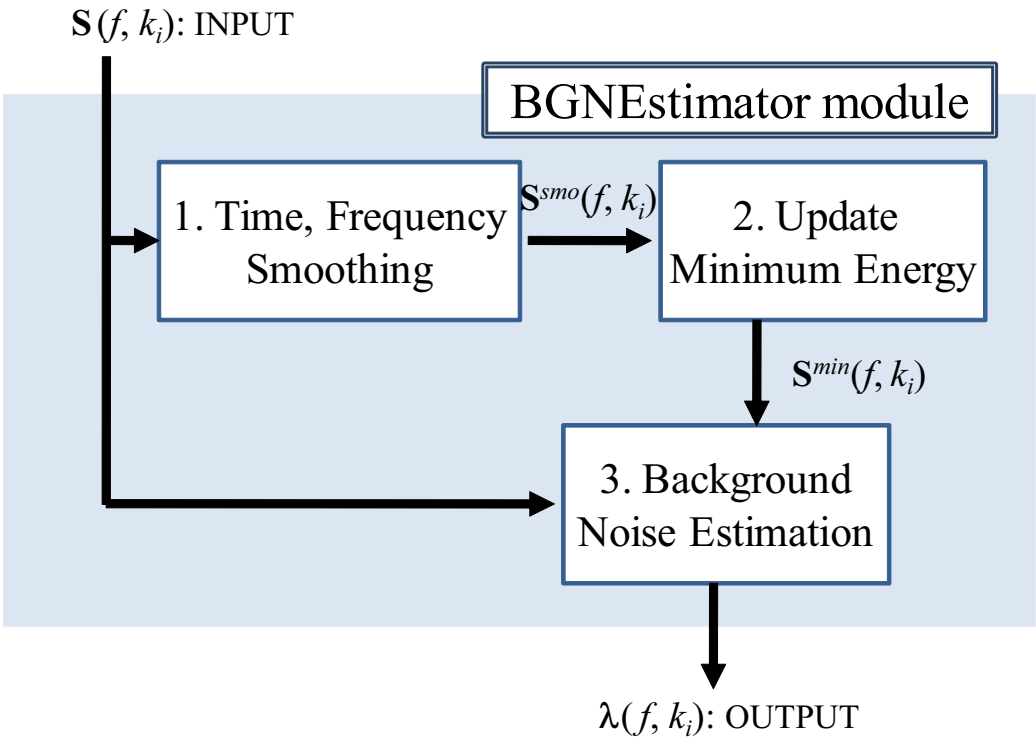

図 6.28: 定常ノイズ推定の流れ

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．

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

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．

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

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．

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

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

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．

$$S_m^{smo}(f,k_i) < \delta S_m^{min}(f,k_i) \text{ または} \tag{6.18}$$

$$f < N \text{ または} \tag{6.19}$$

$$S_m^{smo}(f,k_i) < \lambda_m(f-1,k_i) \tag{6.20}$$

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,

$$\alpha_d = \begin{cases} \frac{1}{f+1}, & \text{if } (\frac{1}{f+1} \geq \alpha_d^{min}) \\ \alpha_d^{min}, & \text{if } (\frac{1}{t+1} < \alpha_d^{min}) \\ 0 & \text{(目的音が含まれるとき)} \end{cases} \tag{6.21}$$

   として計算する．定常ノイズは以下の式によって求める．

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

6.3.2 CalcSpecSubGain

モジュールの概要

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モジュールである．その他，音声存在確率（6.3.7 節参照）を出力する．ただし，このモジュール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 モジュール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 6.29 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．

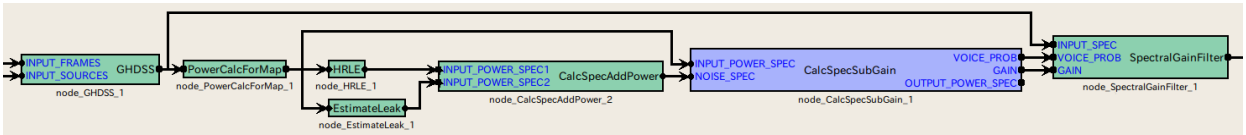

図 6.29: CalcSpecSubGain の接続例

モジュールの入出力とプロパティ

入力

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 無し．

モジュールの詳細

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モジュールである．音声存在確率（6.3.7 節参照）も出力する．ただし，このモジュール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．

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

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．

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

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モジュールの狙いである．

6.3.3 CalcSpecAddPower

モジュールの概要

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 モジュールを用いたノイズ推定時に用いる．HRLE モジュール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 6.30 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．

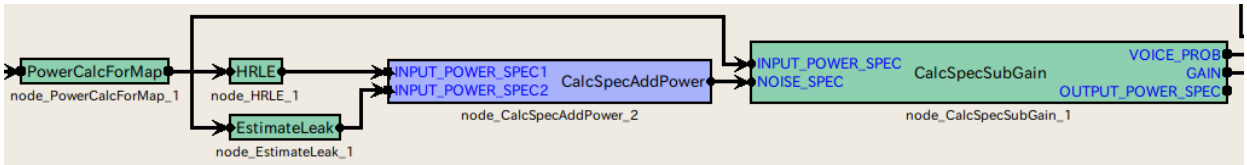

図 6.30: CalcSpecAddPower の接続例

モジュール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．

モジュールの詳細

本モジュールは，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．

6.3.4 EstimateLeak

モジュールの概要

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 6.31 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．

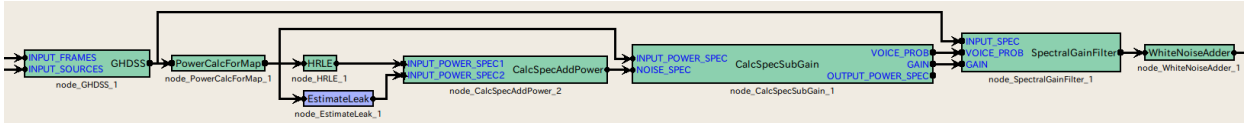

図 6.31: EstimateLeak の接続例

モジュール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 6.15: 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．

モジュールの詳細

本モジュールは，他チャネルからの漏れ成分の推定を行う．詳細は 6.3.7 節の PostFilter モジュール 1-b) 漏れノイズ推定を参照のこと．

6.3.5 GHDSS

モジュールの概要

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，音源方向のデータである．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 6.16: 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
MIC_FILENAME	マイクロホン位置の座標
FIXED_NOISE_FILENAME	ノイズ音源位置の座標
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 モジュールの接続例を図 6.32 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 モジュールを利用して，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 6.32 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 6.32 右下参照）．

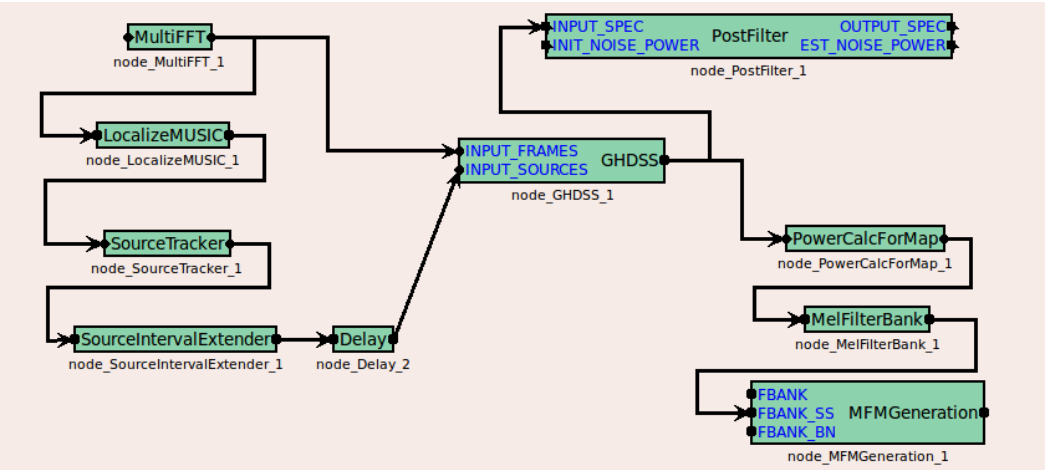

図 6.32: GHDSS の接続例

モジュール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 モジュール，SourceIntervalExtender モジュールと繋げ，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．

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．

表 6.17: 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	string	CALC		CALC，DATABASE から選択
TF_CONJ==DATABASE				以下 TF_CONJ が DATABASE の時に有効.
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.
TF_CONJ==CALC				以下 TF_CONJ が CALC の時に有効.
MIC_FILENAME	string			マイクロホン位置の座標を記したファイル名.
MIC_POS_SHIFT	string	FIX		マイクロホン座標の原点をシフトするか決める. FIX，SHIFT から選択．FIX ならマイクロホンの原点は変化しない．SHIFT なら，マイクロホン座標の重心を原点とする.
SPEED_SOUND	float	343	[m/s]	音の速さ.
FIXED_NOISE	bool	false		固定ノイズ音源を指定するかどうか.
FIXED_NOISE==true				以下 FIXED_NOISE が true の時に有効
FIXED_NOISE_FILENAME	string			固定ノイズ音源位置の座標を記述したファイル名.
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	DIAG		幾何制約の手法を決める．DIAG, FULL から選択．DIAG は目的音源方向に焦点を合わせるのみ．FULL では目的方向の焦点に加えて，非目的音源方向に死角を形成する.
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_ANGLE	float	5	[deg]	分離処理において，同一音源とみなす角度差の閾値.
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.

LOWER_BOUND_FREQUENCY GHDSS 処理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 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 ：`string` 型．伝達関数をシミュレーションで求めるか，測定値を用いるかを選ぶ．シミュレーションを用いるときは CALC，測定値を用いるときは DATABASE を指定する．シミュレーションで計算する場合は，通常の伝達関数を計算するが，測定値は，伝達関数の複素共役である必要がある．hark-tools によって伝達関数データを生成すると，自動的に複素共役の値に変換される．

1. DATABASE のとき: TF_CONJ_FILENAME を設定する．
   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 5.1.2 節を参照．
2. CALC のとき: MIC_FILENAME，MIC_POS_SHIFT，SPEED_SOUND を設定する．
   MIC_FILENAME: `string` 型．マイクロホン位置の座標を記したテキストファイルの名前を指定する．ファイルフォーマットは 5.2 節を参照．
   MIC_POS_SHIFT: `string` 型．FIX または SHIFT を指定．FIX がデフォルト．SHIFT を指定すると，マイクロホン座標系の原点を，上記ファイルの重心に移動する．FIX を指定した場合，何もしない．
   SPEED_SOUND: `float` 型．343 がデフォルト．音速 [m/s] を指定する．

FIXED_NOISE ：`bool` 型．固定ノイズ，例えばロボットの背中のファンノイズなどを GHDSS 処理の対象に含むかどうかを設定する．`true` の時，FIXED_NOISE_FILENAME を指定する．

FIXED_NOISE_FILENAME ：`string` 型．FIXED_NOISE が `true` のとき有効．固定ノイズの位置座標を記述したテキストファイルの名前を指定する．ファイルフォーマットは 5.2 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 5.1.3 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．目的音源方向に焦点を合わせる幾何制約のみの場合は DIAG 目的方向の焦点に加えて，非目的音源方向に死角を形成する場合は 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 DIAG.

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 や音源方向の角度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，さらに音源方向の角度を比較して判断を行う．

UPDATE_ACCEPT_ANGLE : `float` 型．5 がデフォルト．単位は [deg]．UPDATE_METHOD_TF_CONJ や，UPDATE_METHOD_W で POS，ID_POS を設定した場合の，同一音源と判断する角度の許容誤差を設定する．

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 5.1.3 節を参照.

モジュールの詳細

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 6.18 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 6.18: 変数の定義

変数	説明
$\boldsymbol{S}(k_i) = [S_1(k_i), \ldots, S_N(k_i)]^T$	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．
$\boldsymbol{X}(k_i) = [X_1(k_i), \ldots, X_M(k_i)]^T$	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．
$\boldsymbol{N}(k_i) = [N_1(k_i), \ldots, N_M(k_i)]^T$	各マイクロホンに作用する加法性ノイズ．
$\boldsymbol{H}(k_i) = [H_{m,n}(k_i)]$	反射，回折などを含む伝達関数行列 $(M \times N)$．
$\boldsymbol{H}_D(k_i) = [H_{Dm,n}(k_i)]$	直接音の伝達関数行列 $(M \times N)$．
$\boldsymbol{W}(k_i) = [W_{n,m}(k_i)]$	分離行列 $(N \times M)$．
$\boldsymbol{Y}(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	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 $\boldsymbol{H}(k_i)$ の影響を受け，M 個のマイクロホンを通じて式 (6.25) のように観測される．

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

一般に，伝達関数 $\boldsymbol{H}(k_i)$ 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 $\boldsymbol{H}_D(k_i)$ 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 (6.26) のように計算可能である．

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

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

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，$\boldsymbol{H}_D(k_i)$ 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 $\boldsymbol{Y}(k_i)$ は，

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

として求める．GHDSS アルゴリズムは，$\boldsymbol{Y}(k_i)$ が $\boldsymbol{S}(k_i)$ に近づくように，分離行列 $\boldsymbol{W}(k_i)$ 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 モジュール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 $\boldsymbol{H}_D(k_i)$ (測定する or 式 (6.26) 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 $\boldsymbol{H}(k_i)$
2. 観測ノイズ $\boldsymbol{N}(k_i)$

分離行列の更新式　GHDSS は，下記を満たすように分離行列 $\boldsymbol{W}(k_i)$ 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 $\boldsymbol{Y}(k_i)$ の高次相関行列 $\boldsymbol{R}^{\phi(y)y}(k_i) = E[\phi(\boldsymbol{Y}(k_i))\boldsymbol{Y}^H(k_i)]$ 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，本モジュールでは下記で定義される双曲線正接関数を用いている．

$$\phi(\boldsymbol{Y}) = [\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_N)]^T \tag{6.29}$$

$$\phi(Y_k) = \tanh(\sigma|Y_k|)\exp(j\angle(Y_k)) \tag{6.30}$$

   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 $\boldsymbol{W}(k_i)$ と 直接音の伝達関数 $\boldsymbol{H}_D(k_i)$ の積が単位行列になるようにする ($\boldsymbol{W}(k_i)\boldsymbol{H}_D(k_i) = \boldsymbol{I}$).

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．

$$J(\boldsymbol{W}) = \alpha J_1(\boldsymbol{W}) + \beta J_2(\boldsymbol{W}), \tag{6.31}$$

$$J_1(\boldsymbol{W}) = \sum_{i \neq j} |R^{\phi(y)y}_{i,j}|^2, \tag{6.32}$$

$$J_2(\boldsymbol{W}) = \|\boldsymbol{W}\boldsymbol{H}_D - \boldsymbol{I}\|^2, \tag{6.33}$$

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 $\|\boldsymbol{M}\|^2 = tr(\boldsymbol{M}\boldsymbol{M}^H) = \sum_{i,j}|m_{i,j}|^2$ として定義される．

式 (6.31) を最小化するための分離行列の更新式は，複素勾配演算 $\frac{\partial}{\partial \boldsymbol{W}^*}$ を利用した勾配法から，

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

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 (6.34) の右辺にある複素勾配を求めると，$\boldsymbol{R}^{xx} = E[\boldsymbol{X}\boldsymbol{X}^H]$ や $\boldsymbol{R}^{yy} = E[\boldsymbol{Y}\boldsymbol{Y}^H]$ などの期待値計算に複数のフレームの値を要する．GHDSS モジュールでの計算は，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 (6.35) を用いる．

$$\boldsymbol{W}(k_i, f+1) = \boldsymbol{W}(k_i, f) - \left[\mu_{SS}\frac{\partial J_1}{\partial \boldsymbol{W}^*}(\boldsymbol{W}(k_i, f)) + \mu_{LC}\frac{\partial J_2}{\partial \boldsymbol{W}^*}(\boldsymbol{W}(k_i, f))\right], \tag{6.35}$$

$$\frac{\partial J_1}{\partial \boldsymbol{W}^*}(\boldsymbol{W}) = \left(\phi(\boldsymbol{Y})\boldsymbol{Y}^H - \mathrm{diag}[\phi(\boldsymbol{Y})\boldsymbol{Y}^H]\right)\tilde{\phi}(\mathrm{boldmath}WX)\boldsymbol{X}^H, \tag{6.36}$$

$$\frac{\partial J_2}{\partial \boldsymbol{W}^*}(\boldsymbol{W}) = 2\,(\boldsymbol{W}\boldsymbol{H}_D - \boldsymbol{I})\,\boldsymbol{H}_D^H, \tag{6.37}$$

ここで，$\tilde{\phi}$ は ϕ の偏微分であり，下記で定義される．

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

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

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．

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

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

式 (6.36，6.37) 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．

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

ただし，M はマイクロホン数である．

処理の流れ:

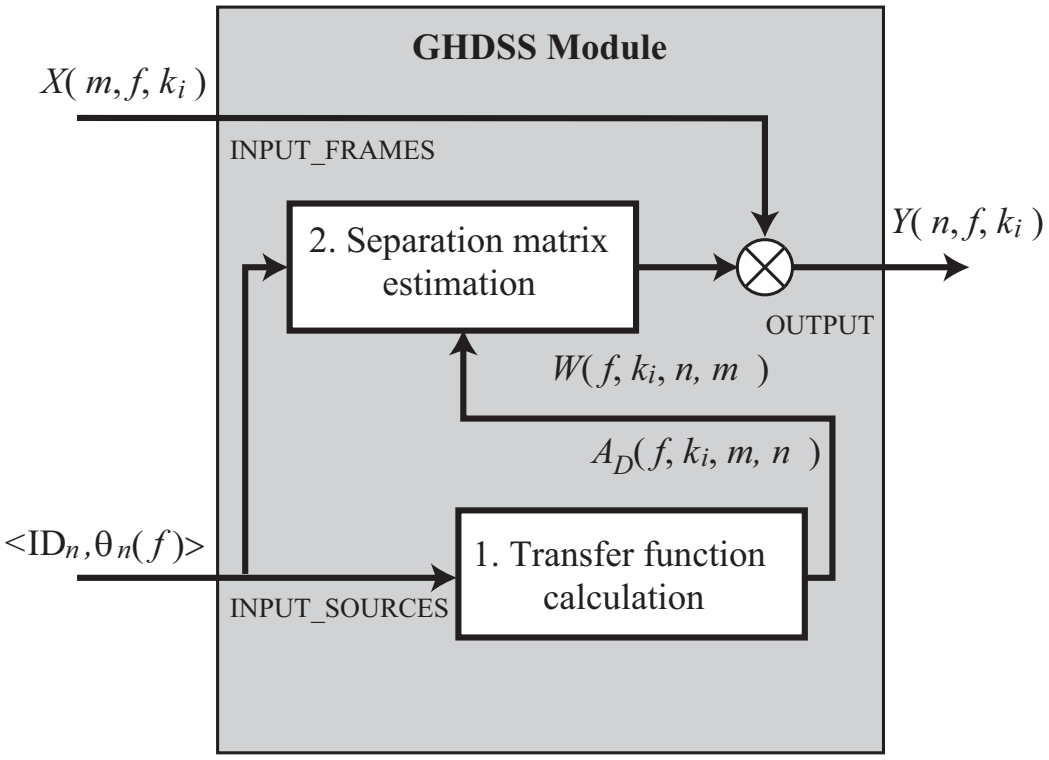

図 6.33: GHDSS の流れ図

GHDSS モジュールにおける，時間フレーム f における主な処理を図 6.33 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. 固定ノイズの有無をチェック
2. (直接音) 伝達関数の取得
3. 分離行列 $\boldsymbol{W}$ 推定
4. 式 (6.28) に従って音源分離処理を実行
5. 分離行列の書き出し (EXPORT_W が `true` のとき)

固定ノイズの有無をチェック: FIXED_NOISE が `true` のとき，音源定位結果の中で，固定ノイズ方向からの音源があれば，その音源に対しては ID として -1 が付与された状態で音源分離が行われる．

伝達関数の取得: 伝達関数の初期値は，パラメータ TF_CONJ の値によって異なる．
パラメータ TF_CONJ が CALC のときは，入力された音源定位結果とパラメータ MIC_FILENAME で指定されたマイクロホン位置座標を用い，式 (6.26) に従って伝達関数 $\boldsymbol{H}_D$ を計算する．
パラメータ TF_CONJ が DATABASE のときは，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．

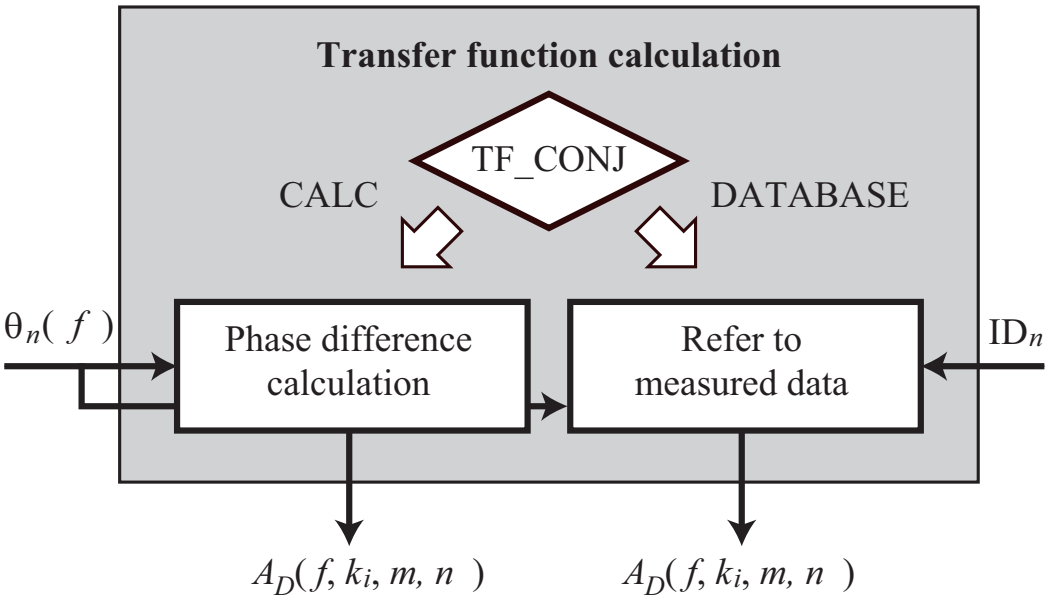

図 6.34: 伝達関数計算の流れ図

伝達関数を得るまでの流れを図 6.34 に示す．

- パラメータ TF_CONJ が CALC のときは，入力された音源定位結果を用い，式 (6.26) に従って伝達関数 $\boldsymbol{H}_D$ を計算する．
- パラメータ TF_CONJ が DATABASE のときは，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．

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_ANGLE 未満: 引き継ぐ
   - 誤差が UPDATE_ACCEPT_ANGLE 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 $\boldsymbol{H}_D$ から分離行列 $\boldsymbol{W}$ 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．

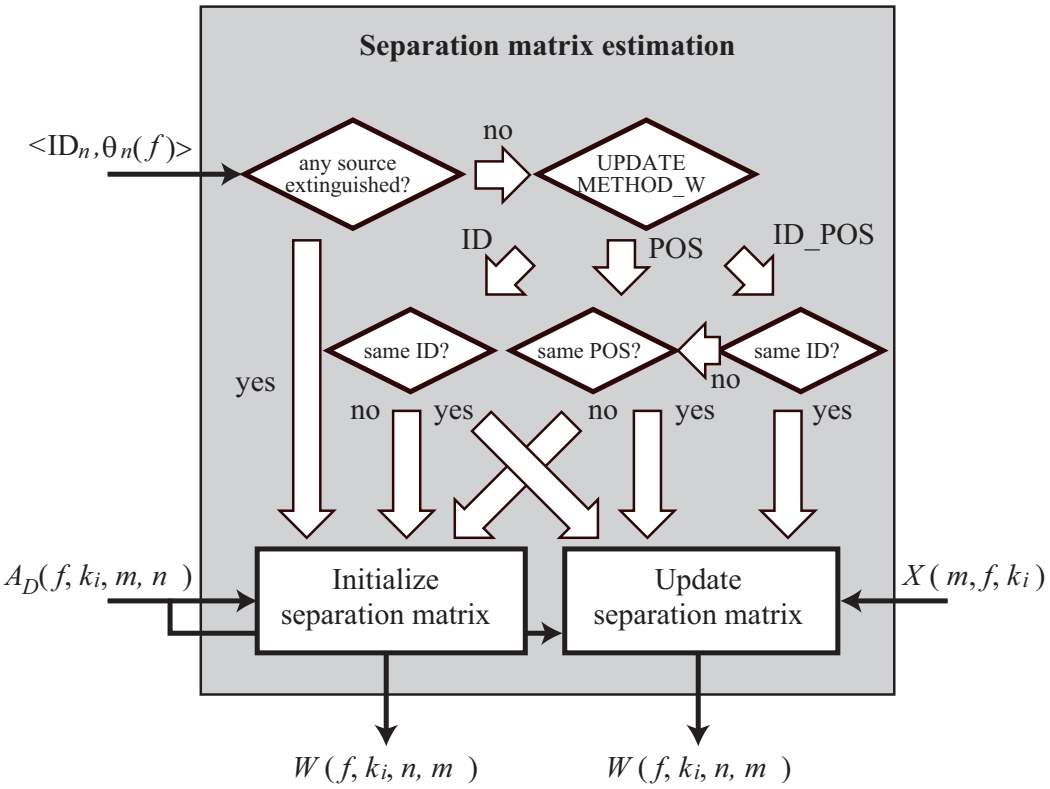

図 6.35: 分離行列推定の流れ図

分離行列を推定するまでの流れを図 6.35 に示す．ここでは，式 (6.35) に従って，前フレームの分離行列を更新するか，式 (6.42) 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，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: $\boldsymbol{W}$ を更新
   - 異なる: $\boldsymbol{W}$ 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_ANGLE 未満: $\boldsymbol{W}$ を更新
   - 誤差が UPDATE_ACCEPT_ANGLE 以上: $\boldsymbol{W}$ を初期化

UPDATE_METHOD_W が ID_POS

1. 前フレーム ID と比較
   - 同じ: $\boldsymbol{W}$ を更新
2. ID が異なった場合，定位方向を比較
   - 誤差が UPDATE_ACCEPT_ANGLE 未満: $\boldsymbol{W}$ を更新
   - 誤差が UPDATE_ACCEPT_ANGLE 以上: $\boldsymbol{W}$ 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_ANGLE 未満: $\boldsymbol{W}$ を上書き保存
   - 誤差が UPDATE_ACCEPT_ANGLE 以上: $\boldsymbol{W}$ を追加保存

6.3.6 HRLE

モジュールの概要

本モジュールは，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 6.36 に示すように，入力は GHDSS などの分離モジュールの後に接続し，出力は CalcSpecSubGain などの最適ゲインを求めるモジュールに接続する．図 6.37 は，EstimateLeak を併用した場合の接続例である．

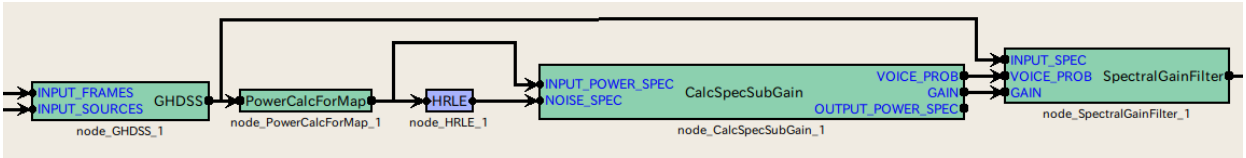

図 6.36: HRLE の接続例 1

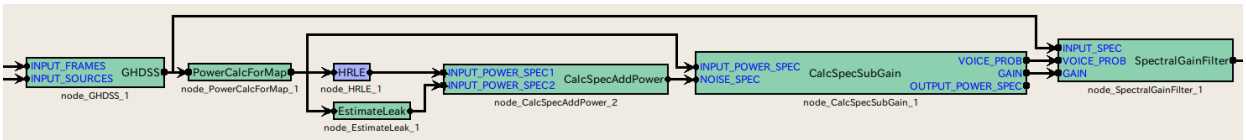

図 6.37: HRLE の接続例 2

モジュールの入出力とプロパティ

入力

INPUT_SPEC ：`Map<int, float>`型． 入力信号のパワースペクトル

出力

NOISE_SPEC ：`Map<int, float>`型． 推定ノイズのパワースペクトル

表 6.19: 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	float	16000	[pt]	時定数.
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.

パラメータ

LX : float 型．デフォルトは 0.85 ．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．

TIME_CONSTANT : float 型．デフォルトは 16000 ．時定数（0 以上）を時間サンプル単位で指定する．

NUM_BIN : float 型．デフォルトは 1000 ．ヒストグラムのビン数を指定する．

MIN_LEVEL : float 型．デフォルトは -100 ．ヒストグラムの最小レベルを dB 単位で指定する．

STEP_LEVEL : float 型．デフォルトは 0.2 ．ヒストグラムのビンの幅を dB 単位で指定する．

DEBUG : bool デフォルトは false．デバッグモードを指定する． デバッグモード (true) の場合，累積ヒストグラムの値が標準出力にコンマ区切りテキストファイル形式で 100 フレーム毎に 1 回出力される． 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：

```
---------- 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
```

モジュールの詳細

図 6.38 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 6.39 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．

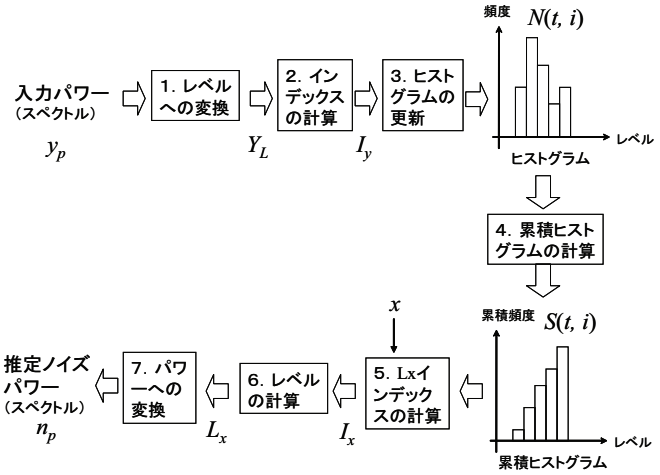

図 6.38: HRLE の処理フロー

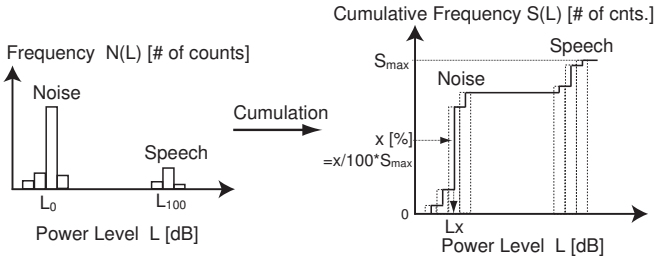

図 6.39: Lx 値の推定

HRLE の具体的な処理手順は，下記の 7 つの数式（図 6.38 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．

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

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

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

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

$$
\begin{aligned}
I_x(t) &= \underset{I}{\operatorname{argmin}} \left[S(t, I_{max}) \frac{x}{100} - S(t, I) \right], && (6.47) \\
L_x(t) &= L_{min} + L_{step} \cdot I_x(t), && (6.48) \\
n_p(t) &= 10^{L_x(t)/10} && (6.49)
\end{aligned}
$$

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).

6.3.7 PostFilter

モジュールの概要

このモジュールは，音源分離モジュール 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

このモジュールは，GHDSS モジュール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 モジュールの接続例は図 6.40 の通り．入力の接続として，INPUT_SPEC は GHDSS モジュールの出力，INIT_NOISE_POWER は BGNEstimator モジュールの出力と接続する．

出力について，図 6.40 では

1. 分離音 (OUTPUT_SPEC) の音声特徴抽出 (MSLSExtraction モジュール),
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 モジュール)

の接続例を示している．

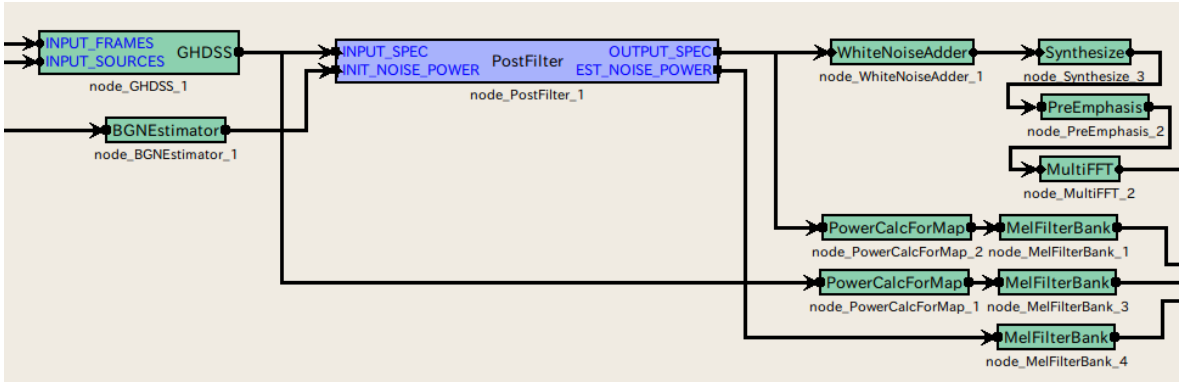

図 6.40: PostFilter の接続例

モジュールの入出力とプロパティ

入力

INPUT_SPEC : `Map<int, ObjectRef>`型．GHDSS モジュール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 モジュール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．

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

モジュールの詳細

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 6.41 は，PostFilter モジュールの流れ図である．入力としては，GHDSS モジュールからの分離音スペクトルと，BGNEstimator モジュール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 6.42 に示す．PostFilter モジュール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，

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

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 6.20: 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	float	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を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を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を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．

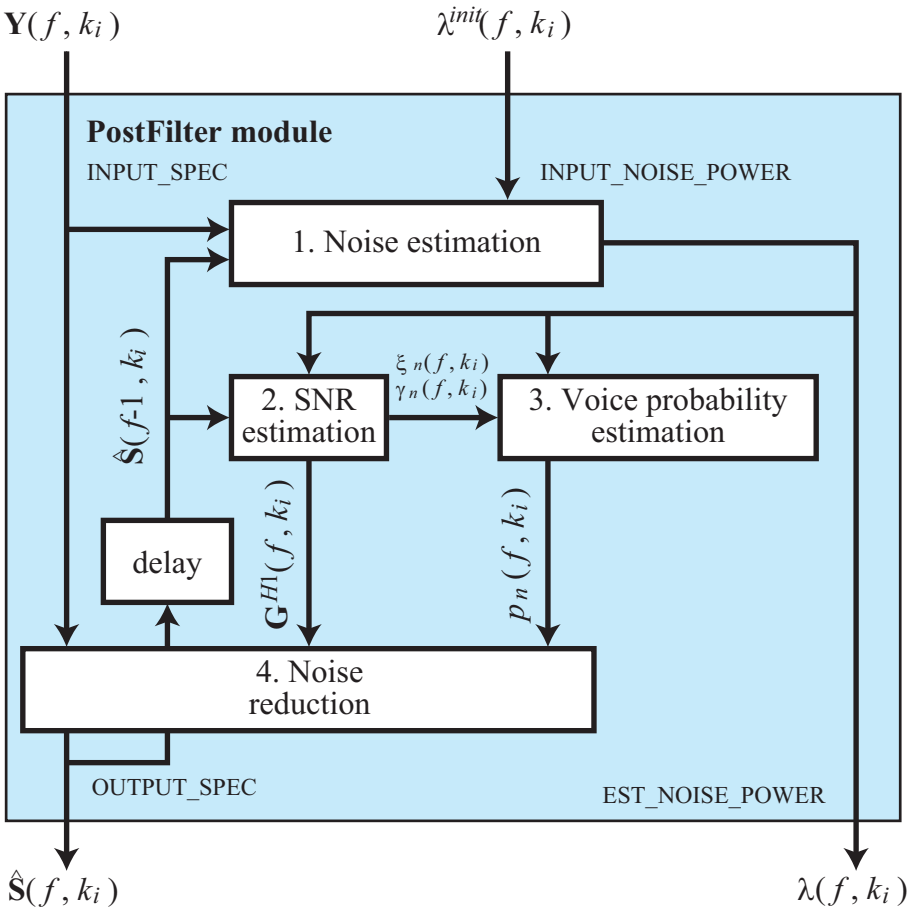

図 6.41: PostFilter の流れ図

1-a) MCRA 法による定常ノイズ推定　1-a) で用いる変数は 表 6.22 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 $\boldsymbol{S}(f,k_i) = [S_1(f,k_i),\ldots,S_N(f,k_i)]$ を求める．

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

次に，$\boldsymbol{S}^{tmp}$，$\boldsymbol{S}^{min}$ を更新する．

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

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

ただし，n は任意の整数である．$\boldsymbol{S}^{min}$ はノイズ推定を始めてからの最小パワーを保持し，$\boldsymbol{S}^{tmp}$ は最近の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 $\boldsymbol{S}^{tmp}$ 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．

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

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

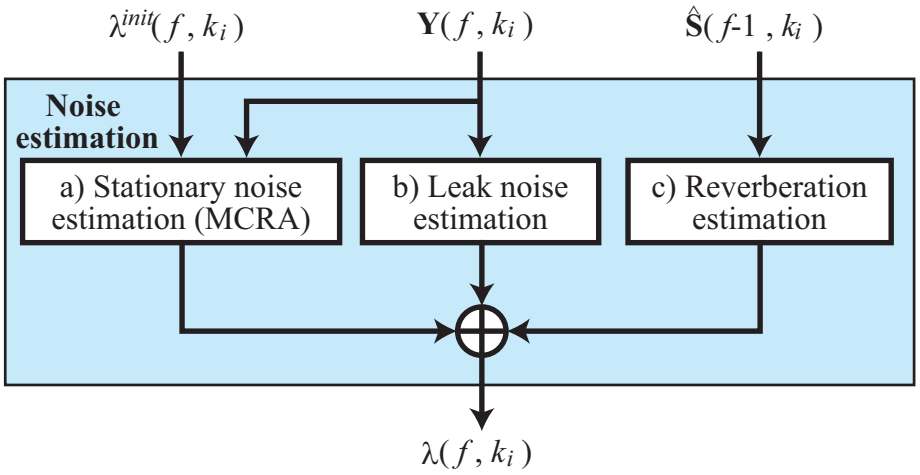

図 6.42: ノイズ推定の手順

$I_n(k_i)$ が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 $\alpha^C_{d,n}(k_i)$ を決める．

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

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．

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

$$S^0_n(k_i) = |Y_n(k_i)|^2 - qS^{leak}_n(k_i), \tag{6.58}$$

ただし，$S^0_n(k_i) < S_{floor}$ のとき，

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

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 $S^0_n(f, k_i)$ と，前フレームの推定定常ノイズ $\boldsymbol{\lambda}^{sta}(f-1, k_i)$ 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．

$$\lambda^{sta}_n(f, k_i) = \begin{cases} \alpha^C_{d,n}(k_i)\lambda^{sta}_n(f-1, k_i) + (1 - \alpha^C_{d,n}(k_i)rS^0_n(f, k_i) & \text{if音源位置に変更なし} \\ \alpha^C_{d,n}(k_i)\lambda^{init}_n(f, k_i) + (1 - \alpha^C_{d,n}(k_i)rS^0_n(f, k_i) & \text{if音源位置に変更あり} \end{cases} \tag{6.60}$$

1-b) 漏れノイズ推定　1-b) で用いる変数は 表 6.23 に基づく．

いくつかのパラメータを次のように計算する．

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

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

このパラメータを用いて，平滑化されたスペクトル $\boldsymbol{S}(k_i)$ と，式 (6.57) 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．

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

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 $\boldsymbol{\lambda}^{leak}(k_i)$ は，

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

として求める．

1-c) 残響推定　1-c) で用いる変数は 表 6.24 に基づく．

残響のパワーは，前フレームの推定残響パワー $\boldsymbol{\lambda}^{rev}(f-1,k_i) = [\lambda_1^{rev}(f-1,k_i),\ldots,\lambda_N^{rev}(f-1,k_i)]^T$ と，前フレームの分離スペクトル $\hat{\boldsymbol{S}}(f-1,k_i) = [\hat{S}_1(f-1,k_i),\ldots,\hat{S}_N(f-1,k_i)]^T$ 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．

$$\lambda_n^{rev}(f,k_i) \quad = \quad \gamma\left(\lambda_n^{rev}(f-1,k_i) + \Delta|\hat{S}_n(f-1,k_i)|^2\right) \tag{6.65}$$

2) SNR 推定:

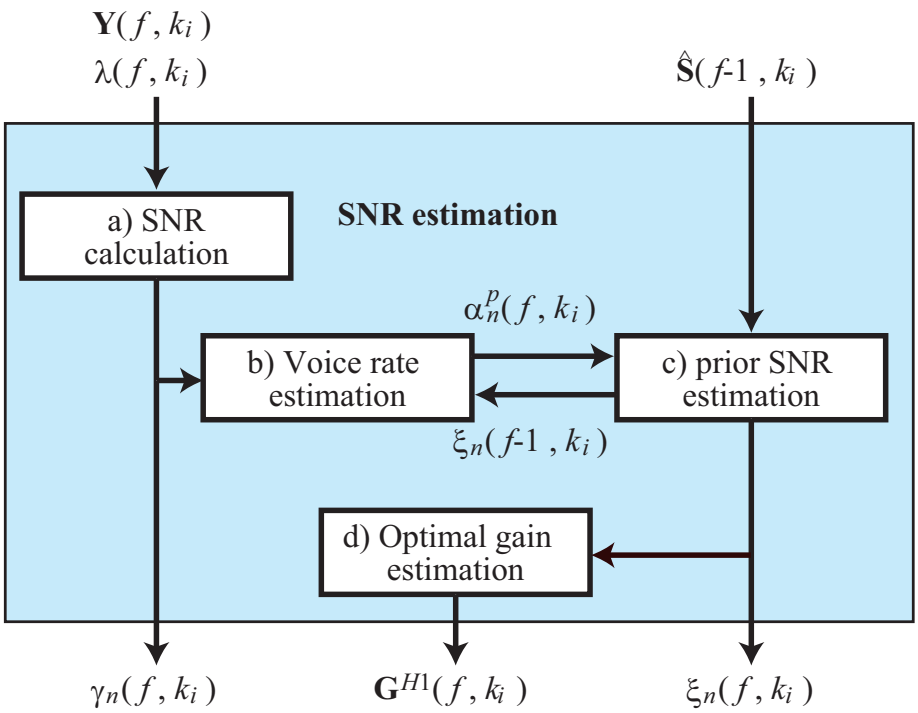

図 6.43: SNR 推定の手順

SNR 推定の流れを図 6.43 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 6.25 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 6.25 に従う．ここでは，入力の複素スペクトル $\boldsymbol{Y}(k_i)$ と，前段で推定されたノイズのパワースペクトル $\boldsymbol{\lambda}(k_i)$ を元に，SNR $\gamma_n(k_i)$ が計算される．

$$\gamma_n(k_i) = \frac{|Y_n(k_i)|^2}{\lambda_n(k_i)} \tag{6.66}$$

$$\gamma_n^C(k_i) = \begin{cases} \gamma_n(k_i) & \text{if } \gamma_n(k_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.67}$$

2-b) 音声含有率の推定　2-b) で用いる変数は，表 6.26 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．

$$\alpha_n^p(f, k_i) = \alpha_{mag}^p \left(\frac{\xi_n(f-1, k_i)}{\xi_n(f-1, k_i) + 1} \right)^2 + \alpha_{min}^p \tag{6.68}$$

2-c) ノイズ混入前の事前 SNR 推定　2-c) で用いる変数は，表 6.27 に従う．

事前 SNR $\xi_n(k_i)$ は，次のようにして計算する．

$$\xi_n(k_i) = \left(1 - \alpha_n^p(k_i)\right)\xi_{tmp} + \alpha_n^p(k_i)\gamma_n^C(k_i) \tag{6.69}$$

$$\xi_{tmp} = a\frac{|\hat{S}_n(f-1, k_i)|^2}{\lambda_n(f-1, k_i)} + (1-a)\xi_n(f-1, k_i) \tag{6.70}$$

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 6.28 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．

$$v_n(k_i) = \frac{\xi_n(k_i)}{1 + \xi_n(k_i)}\gamma_n(k_i) \tag{6.71}$$

$v_n(k_i) > \theta^{max}$ 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 $\boldsymbol{G}^{H1}(k_i) = [G_1^{H1}(k_i), \ldots, G_N^{H1}(k_i)]$ は，

$$G_n^{H1}(k_i) = \frac{\xi_n(k_i)}{1 + \xi_n(k_i)} \exp\left\{ \frac{1}{2} \mathtt{int}_{v_n(k_i)}^{\inf} \frac{e^{-t}}{t} \mathrm{d}t \right\} \tag{6.72}$$

として求める．ただし，

$$\begin{aligned} G_n^{H1}(k_i) &= 1 \quad \text{if} v_n(k_i) < \theta^{min} \\ G_n^{H1}(k_i) &= 1 \quad \text{if} G_n^{H1}(k_i) > 1. \end{aligned} \tag{6.73}$$

3) 音声存在確率推定:

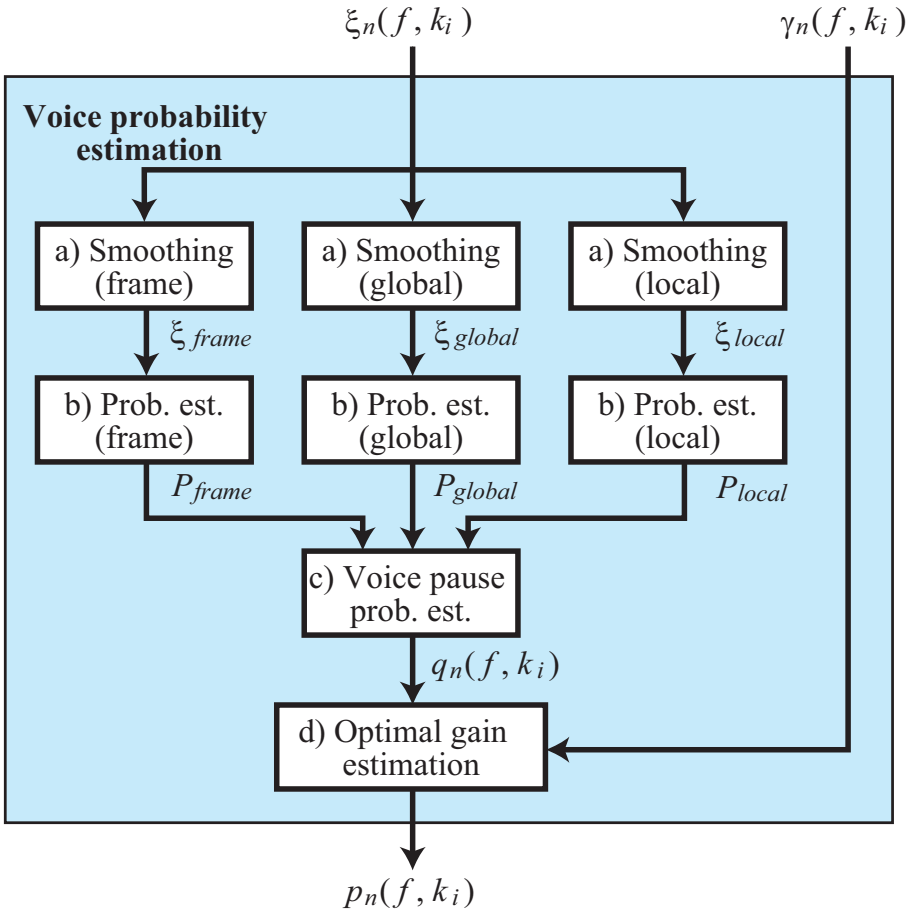

図 6.44: 音声存在確率推定の手順

音声存在確率推定の流れを図 6.44 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
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 6.29 にまとめる．

まず，式 (6.69) 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．

$$\zeta_n(f, k_i) = b\zeta_n(f-1, k_i) + (1-b)\xi_n(f, k_i) \tag{6.74}$$

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．

$$\zeta_n^f(k_i) = \frac{1}{F_{en} - F_{st} + 1} \sum_{k_j=F_{st}}^{F_{en}} \zeta_n(k_j) \tag{6.75}$$

- 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．

$$\zeta_n^g(k_i) = \sum_{j=-(G-1)/2}^{(G-1)/2} w_{han}(j+(G-1)/2)\zeta_n(k_{i+j}), \tag{6.76}$$

$$w_{han}(j) = \frac{1}{C}\left(0.5 - 0.5\cos\left(\frac{2\pi j}{G}\right)\right), \tag{6.77}$$

  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．

$$\zeta_n^l(k_i) = 0.25\zeta_n(k_i-1) + 0.5\zeta_n(k_i) + 0.25(k_i+1) \tag{6.78}$$

3-b) 暫定音声確率を推定　3-b) で用いる変数を表 6.30 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．

$$\zeta_n^{peak}(f,k_i) = \begin{cases} \zeta_n^f(f,k_i), & \text{if } \zeta_n^f(f,k_i) > Z_{thres}\zeta_n^f(f-1,k_i) \\ \zeta_n^{peak}(f-1,k_i), & \text{if otherwise.} \end{cases} \tag{6.79}$$

  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，

$$\zeta_n^{peak}(k_i) = \begin{cases} Z_{min}^{peak}, & \text{if } \zeta_n^{peak}(k_i) < Z_{min}^{peak} \\ Z_{max}^{peak}, & \text{if } \zeta_n^{peak}(k_i) > Z_{max}^{peak} \end{cases} \tag{6.80}$$

  次に，$P_n^f(k_i)$ を次のように求める．

$$P_n^f(k_i) = \begin{cases} 0, & \text{if } \zeta_n^f(k_i) < \zeta_n^{peak}(k_i)Z_{min}^f \\ 1, & \text{if } \zeta_n^f(k_i) > \zeta_n^{peak}(k_i)Z_{max}^f \\ \frac{\log\left(\zeta_n^f(k_i)/\zeta_n^{peak}(k_i)Z_{min}^f\right)}{\log\left(Z_{max}^f/Z_{min}^f\right)}, & \text{otherwise} \end{cases} \tag{6.81}$$

- $P_n^g(k_i)$ の計算
  次の通りに計算する．

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

- $P_n^l(k_i)$ の計算
  次の通りに計算する．

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

3-c) 音声休止確率推定　3-c) で用いる変数を表 6.31 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．

$$q_n(k_i) \quad = \quad 1 - \left(1 - a^l + a^l P_n^l(k_i)\right)\left(1 - a^g + a^g P_n^g(k_i)\right)\left(1 - a^f + a^f P_n^f(k_i)\right), \tag{6.84}$$

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 (6.71) 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．

$$p_n(k_i) \quad = \quad \left\{1 + \frac{q_n(k_i)}{1 - q_n(k_i)}\,(1 + \zeta_n(k_i))\exp\left(-v_n(k_i)\right)\right\}^{-1} \tag{6.85}$$

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．

$$\hat{S}_n(k_i) \quad = \quad Y_n(k_i) G_n^{H1}(k_i) p_n(k_i) \tag{6.86}$$

表 6.21: 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 6.22: 変数の定義

変数	説明，対応するパラメータ
$\boldsymbol{Y}(k_i) = [Y_1(k_i), \ldots, Y_N(k_i)]^T$	周波数ビン k_i に対応する分離音複素スペクトル
$\boldsymbol{\lambda}^{init}(k_i) = \left[\lambda_1^{init}(k_i), \ldots, \lambda_N^{init}(k_i)\right]^T$	定常ノイズ推定に用いる初期値パワースペクトル
$\boldsymbol{\lambda}^{sta}(k_i) = \left[\lambda_1^{sta}(k_i), \ldots, \lambda_N^{sta}(k_i)\right]^T$	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
$\boldsymbol{S}^{tmp}(k_i) = \left[S_1^{tmp}(k_i), \ldots, S_N^{tmp}(k_i)\right]$	最小パワー計算用のテンポラリ変数．
$\boldsymbol{S}^{min}(k_i) = \left[S_1^{min}(k_i), \ldots, S_N^{min}(k_i)\right]$	最小パワーを保持する変数．
L	$\boldsymbol{S}_{tmp}$ 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
$\boldsymbol{Y}^{leak}(k_i)$	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 6.23: 変数の定義

変数	説明，対応するパラメータ
$\boldsymbol{\lambda}^{leak}(k_i)$	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 (6.51) で求める平滑化パワースペクトル

表 6.24: 変数の定義

変数	説明，対応するパラメータ
$\boldsymbol{\lambda}^{rev}(f, k_i)$	時間フレーム f での残響のパワースペクトル
$\hat{\boldsymbol{S}}(f-1, k_i)$	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 6.25: 主な変数の定義

変数	説明，対応するパラメータ
$\boldsymbol{Y}(k_i)$	PostFilter モジュールの入力である分離音の複素スペクトル
$\hat{\boldsymbol{S}}(k_i)$	PostFilter モジュールの出力となる，整形された分離音複素スペクトル
$\boldsymbol{\lambda}(k_i)$	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
$\boldsymbol{G}^{H1}(k_i)$	分離音の SNR を向上させるための最適ゲイン

表 6.26: 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 6.27: 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 6.28: 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 6.29: 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 6.30: 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 6.31: 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

6.3.8 SpectralGainFilter

モジュールの概要

本モジュールは，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 6.45 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．

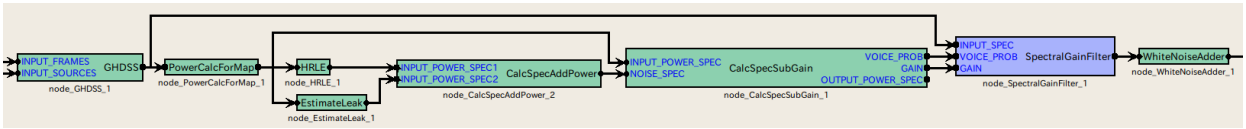

図 6.45: SpectralGainFilter の接続例

モジュール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

モジュールの詳細

本モジュールは，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．

$$Y_n(k_i) = X_n(k_i)G_n(k_i)p_n(k_i) \tag{6.87}$$

6.4 FeatureExtraction カテゴリ

6.4.1 Delta

モジュールの概要

本モジュールは，静的特徴ベクトルから動的特徴量ベクトルを求める．特徴抽出モジュールである 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モジュールは，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 モジュールでは，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モジュールを使う．通常，MFCCExtraction や MSLSExtraction の後に用いる．

典型的な接続例

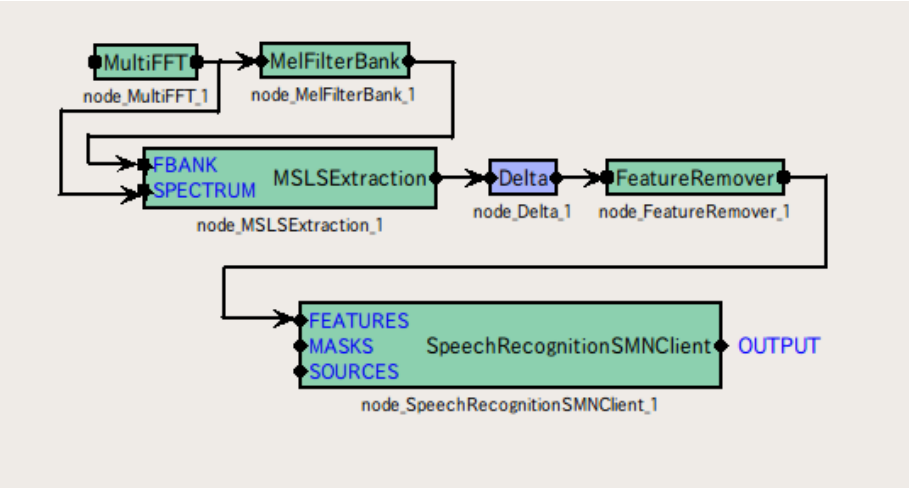

図 6.46: Delta の典型的な接続例

表 6.32: 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

モジュール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．

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．特徴量抽出モジュール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．

モジュールの詳細

本モジュールは，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，

$$\boldsymbol{x}(f) = [x(f,0), x(f,1), \ldots, x(f,P-1)]^T \tag{6.88}$$

と表す．ただし，P は，FBANK_COUNT である．

$$\boldsymbol{y}(f) = [x(f,0), x(f,1), \ldots, x(f,2P-1)]^T \tag{6.89}$$

出力ベクトルの各要素は，

$$y(f,p) = \begin{cases} x(f,p), & \text{if } p = 0, \ldots, P-1, \\ w \sum_{\tau=-2}^{2} \tau \cdot x(f+\tau, p), & \text{if } p = P, \ldots, 2P-1, \end{cases} \tag{6.90}$$

である．ただし，$w = \frac{1}{\sum_{\tau=-2}^{\tau=2} \tau^2}$ である．図 6.47 に Delta の入出力フローを示す．

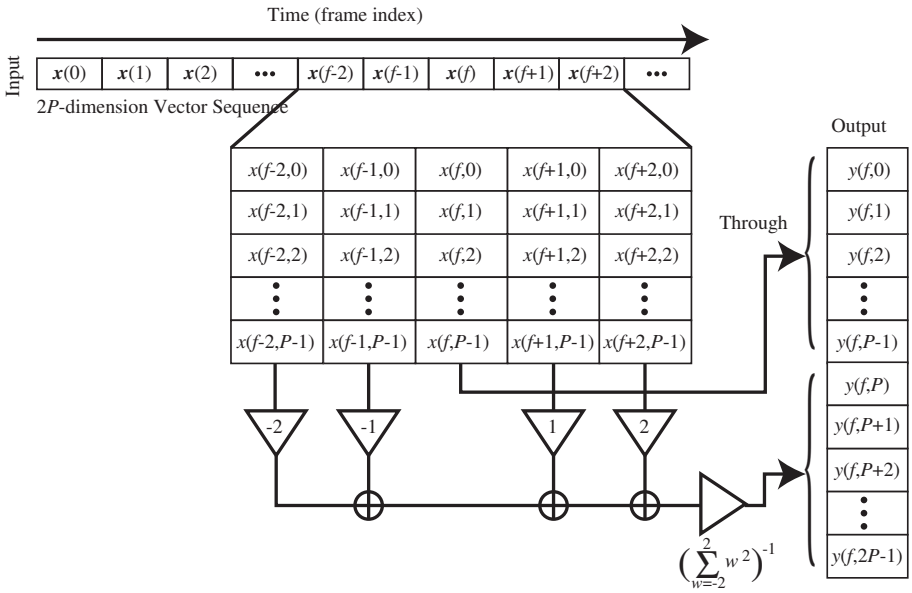

図 6.47: Delta の入出力フロー.

6.4.2 FeatureRemover

モジュールの概要

本モジュールは，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，本モジュールを使用する．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．本モジュールは，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．

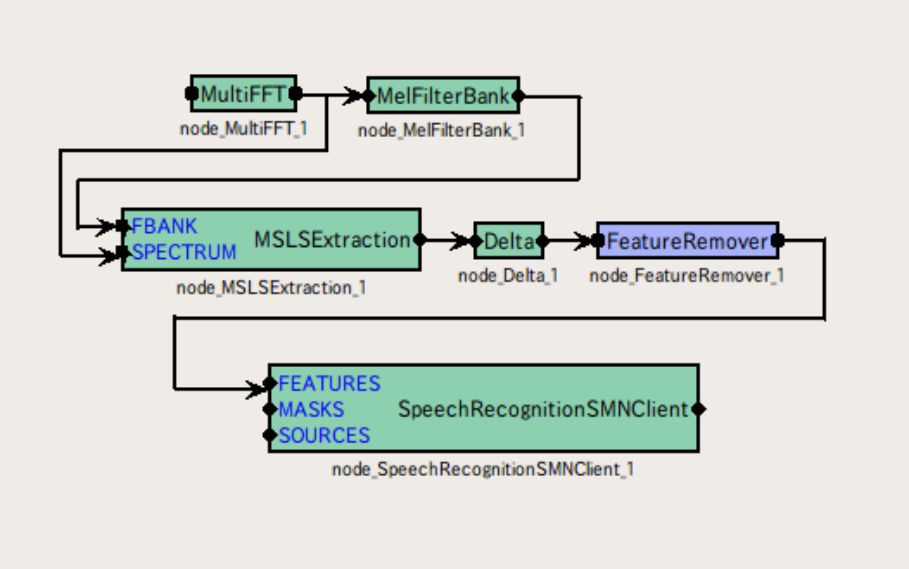

図 6.48: FeatureRemover の典型的な接続例

表 6.33: 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	Vector<int>			次元インデックスからなるベクトル（複数指定可）

モジュール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．

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．

モジュールの詳細

本モジュールは，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

モジュールの概要

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

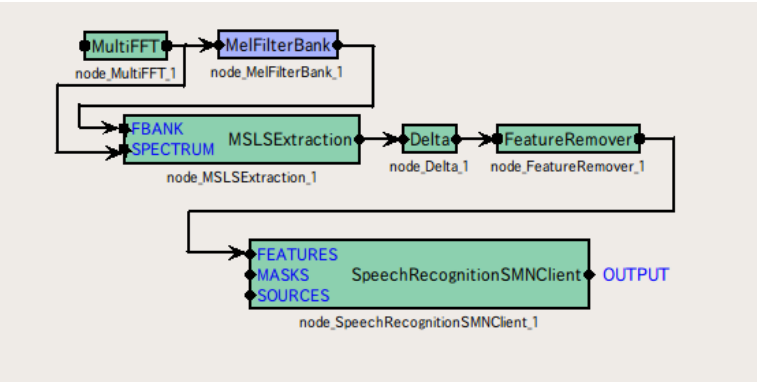

図 6.49: MelFilterBank の接続例

モジュールの入出力とプロパティ

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

表 6.34: 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

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．

CUT_OFF : 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 CUT_OFF 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 CUT_OFF 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．

モジュールの詳細

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_RATECUT_OFF/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，

$$m = 1127.01048 \log(1.0 + \frac{\lambda}{700.0}) \tag{6.91}$$

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 6.50 に 8000 Hz までの変換例を示す．赤点は，SAMPLING_RATE が 16000 Hz，CUT_OFF 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 6.51 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 6.52 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 6.52 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．

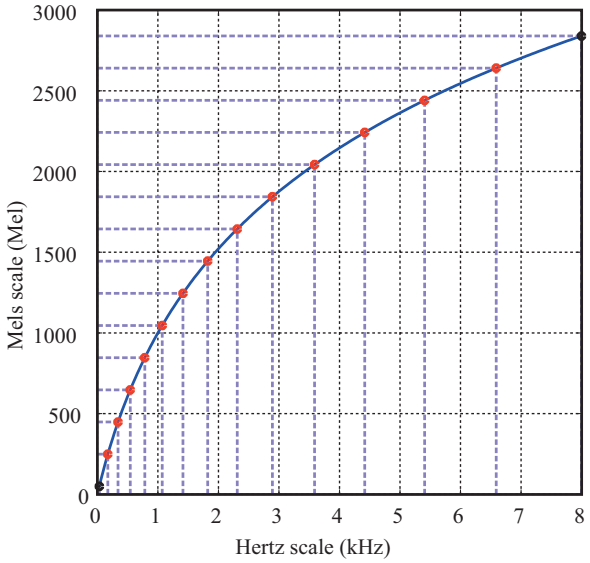

図 6.50: リニアスケールとメルスケールの対応

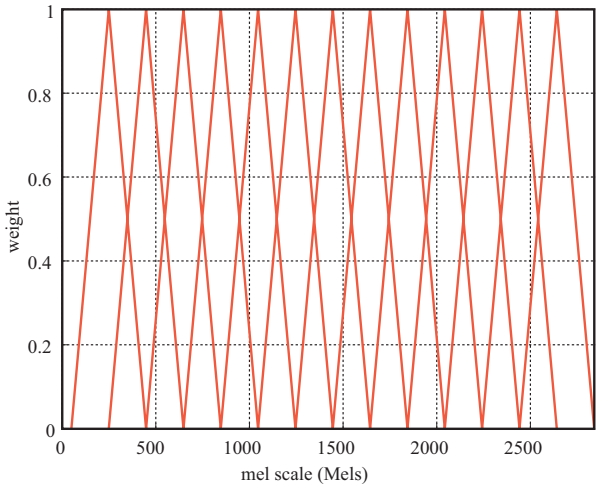

図 6.51: メルスケール上での窓関数

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.

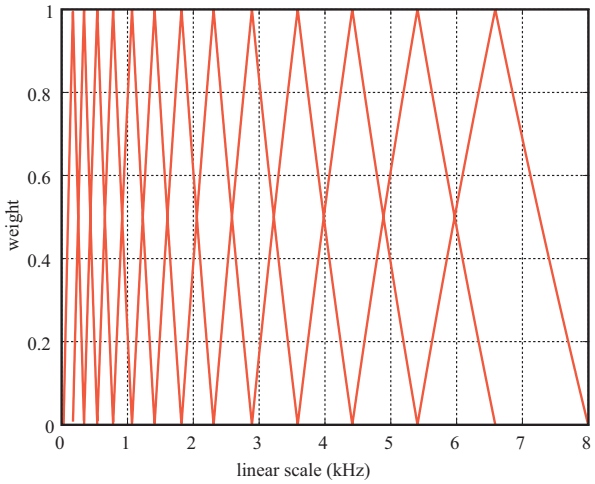

図 6.52: リニアスケール上での窓関数

6.4.4 MFCCExtraction

モジュールの概要

本モジュールは，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モジュールに入力し，音韻や話者を識別する．

典型的な接続例

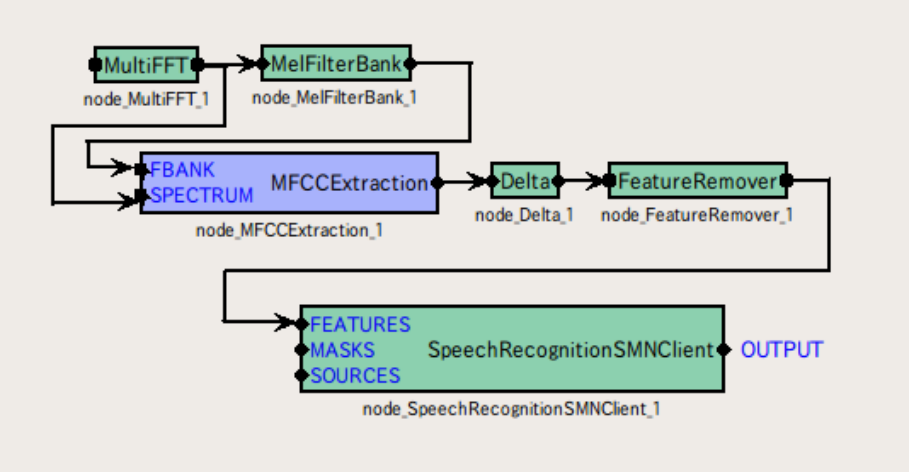

図 6.53: MFCCExtraction の典型的な接続例

モジュールの入出力とプロパティ

表 6.35: 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<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．

モジュールの詳細

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．

本モジュールの入力部の 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，

$$\boldsymbol{x}(f) = [x(f,0), x(f,1), \cdots, x(f,P-1)]^T \tag{6.92}$$

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．本モジュールの出力ベクトルは，

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

$$y(f,p) = L(p) \cdot \sqrt{\frac{2}{P}} \cdot \sum_{q=0}^{P-1} \left\{ \log(x(q)) \cos\left(\frac{\pi(p+1)(q+0.5)}{P} \right) \right\} \tag{6.94}$$

ただし，E は，パワー項（後述）で，リフタリング係数は，

$$L(p) = 1.0 + \frac{Q}{2} \sin\left(\frac{\pi(p+1)}{Q} \right), \tag{6.95}$$

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

$$\boldsymbol{s} = [s(0), \ldots, s(K-1)]^T, \tag{6.96}$$

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，

$$E = \log\left(\frac{1}{K}\sum_{k=0}^{K-1} s(k)\right) \tag{6.97}$$

6.4.5 MSLSExtraction

モジュールの概要

本モジュールは，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モジュールに入力し，音韻や話者を識別する．

典型的な接続例

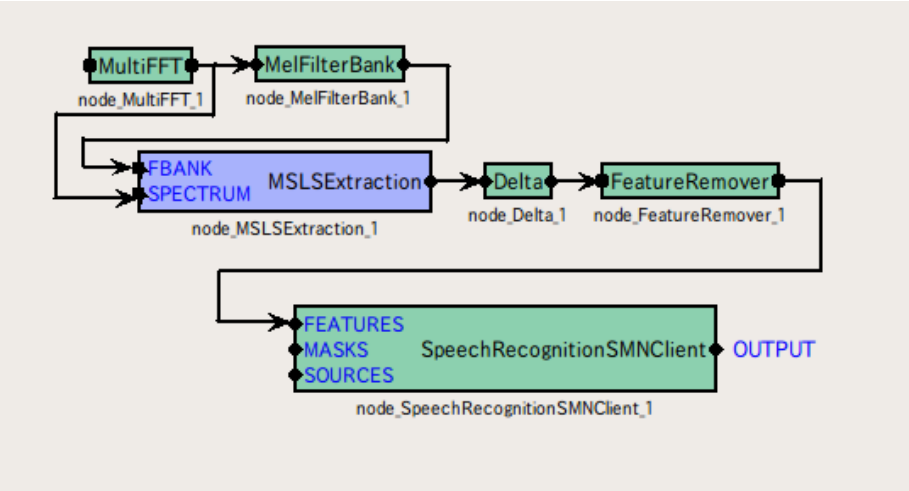

図 6.54: MSLSExtraction の典型的な接続例

モジュール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．

表 6.36: 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
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

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<float>型のデータのペア．

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．本モジュールは， MSLS の静的特徴を求めるモジュールであるが，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 6.55 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．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．

モジュールの詳細

本モジュールは，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．

本モジュールの 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モジュールの出力ベクトルの計算方法を示す．

CEPSTRAL : FBANK 端子への入力を，

$$\boldsymbol{x} = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.98}$$

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．本モジュールの出力ベクトルは，

$$\boldsymbol{y} = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.99}$$

$$y(p) = \frac{1}{P}\sum_{q=0}^{P-1}\left\{L(q)\cdot\sum_{r=0}^{P-1}\left\{\log(x(r))\cos\left(\frac{\pi q(r+0.5)}{P}\right)\right\}\cos\left(\frac{\pi q(p+0.5)}{P}\right)\right\} \tag{6.100}$$

である．ただし，リフタリング係数は，

$$L(p) = \begin{cases} 1.0, & (p = 0, \ldots, P-1), \\ 0.0, & (p = P, \ldots, 2P-1), \end{cases} \tag{6.101}$$

とする．ただし，$Q = 22$ である．

SPECTRAL：FBANK 部の入力を

$$\boldsymbol{x} = [x(0), x(1), \cdots, x(P-1)]^T \tag{6.102}$$

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．本モジュールの出力ベクトルは，

$$\boldsymbol{y} = [y(0), y(1), \ldots, y(P-1), E]^T \tag{6.103}$$

$$y(p) = \begin{cases} (\log(x(p)) - \mu) - 0.9(\log(x(p-1)) - \mu), & if \;\; p = 1, \ldots, P-1 \\ \log(x(p), & if \;\; p = 0, \end{cases} \tag{6.104}$$

$$\mu = \frac{1}{P}\sum_{q=0}^{P-1}\log(x(q)), \tag{6.105}$$

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

$$\boldsymbol{s} = [s(0), s(1), \ldots, s(N-1)]^T \tag{6.106}$$

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，

$$p = \log\left(\frac{1}{N}\sum_{n=0}^{N-1}s(n)\right) \tag{6.107}$$

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 6.55: MSLSExtraction の出力パラメータ

6.4.6 PreEmphasis

モジュールの概要

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

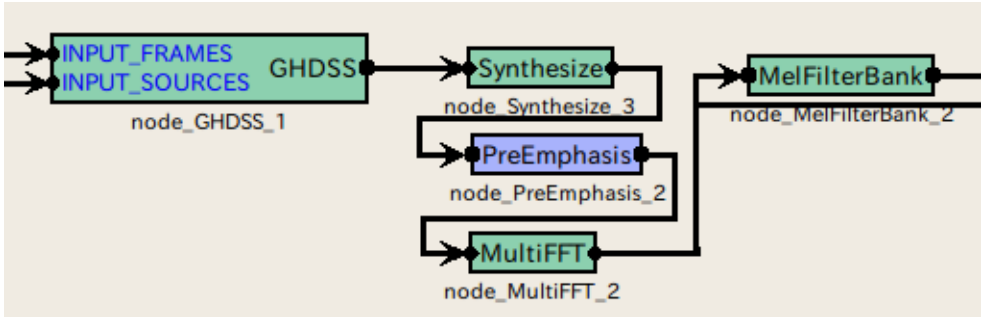

図 6.56: PreEmphasis の接続例

モジュールの入出力とプロパティ

表 6.37: 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

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

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，同様に前段のモジュール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，他のモジュール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．

モジュールの詳細

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．

$$p[t] = \begin{cases} s[t] - c_p \cdot s[t-1] & t > 0 \\ (1 - c_p) \cdot s[0] & t = 0 \end{cases} \tag{6.108}$$

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

モジュールの概要

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

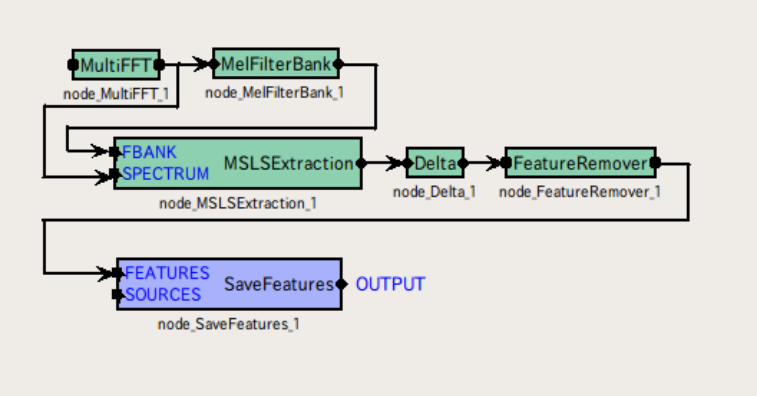

図 6.57: SaveFeatures の接続例

モジュールの入出力とプロパティ

表 6.38: 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>型または Matrix<float>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．

モジュールの詳細

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 758 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

モジュールの概要

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

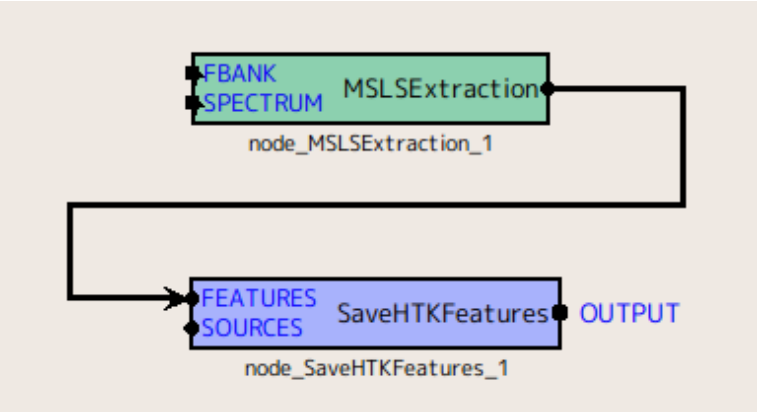

図 6.58: SaveHTKFeatures の接続例

モジュールの入出力とプロパティ

表 6.39: 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>型または Matrix<float>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．

モジュールの詳細

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 758 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

モジュールの概要

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モジュールである．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

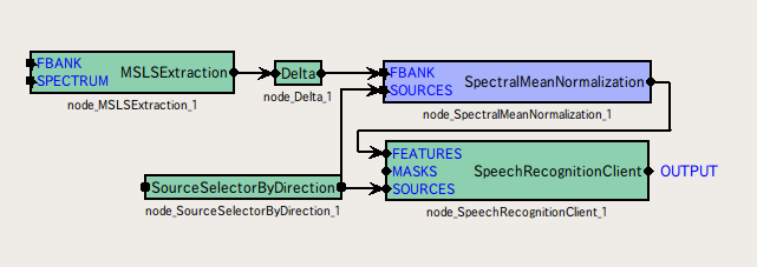

図 6.59: SpectralMeanNormalization の接続例

モジュール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．

SOURCES ：`Vector<ObjectRef>`型である．音源位置．

表 6.40: 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

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．

モジュールの詳細

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モジュールである．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

モジュールの概要

本モジュールは，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

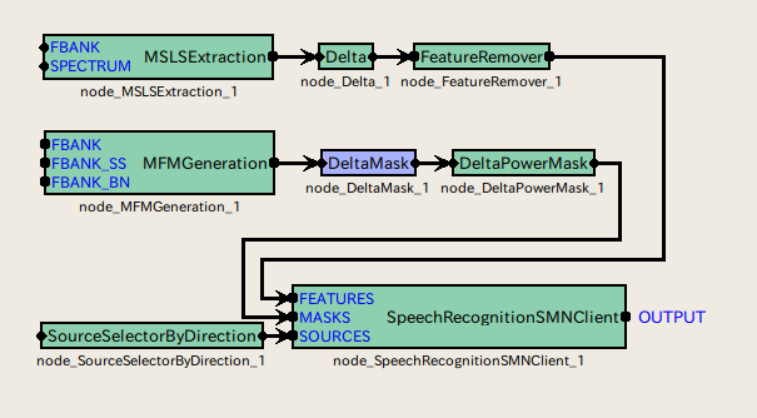

図 6.60: DeltaMask の典型的な接続例

モジュールの入出力とプロパティ

表 6.41: 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．

モジュールの詳細

本モジュールは，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，

$$\boldsymbol{m}(f) = [m(f,0), m(f,1), \ldots, m(f,2P-1)]^T \tag{6.109}$$

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 $\boldsymbol{m}'(f)$ は，

$$\boldsymbol{y}'(f) = [m'(f,0), m'(f,1), \ldots, m'(f,2P-1)]^T \tag{6.110}$$

$$m'(f,p) = \begin{cases} m(f,p), & \text{if } p = 0, \ldots, P-1, \\ \prod_{\tau=-2}^{2} m(f+\tau, p), & \text{if } p = P, \ldots, 2P-1, \end{cases} \tag{6.111}$$

である．図 6.61 に DeltaMask の入出力フローを示す．

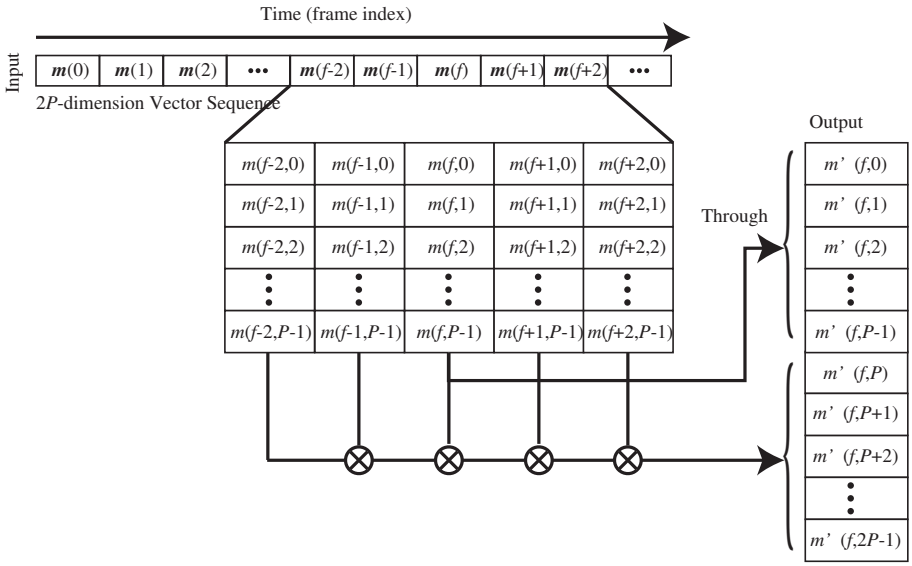

図 6.61: DeltaMask の入出力フロー.

6.5.2 DeltaPowerMask

モジュールの概要

本モジュールは，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

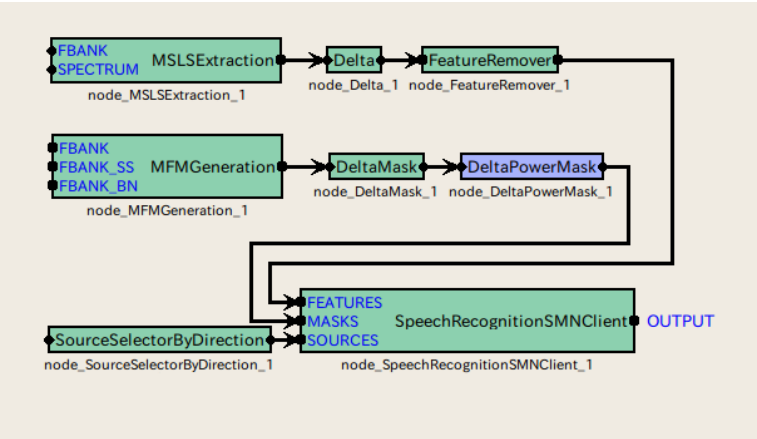

図 6.62: DeltaPowerMask の典型的な接続例

モジュール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

モジュールの詳細

本モジュールは，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

モジュールの概要

本モジュールは，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

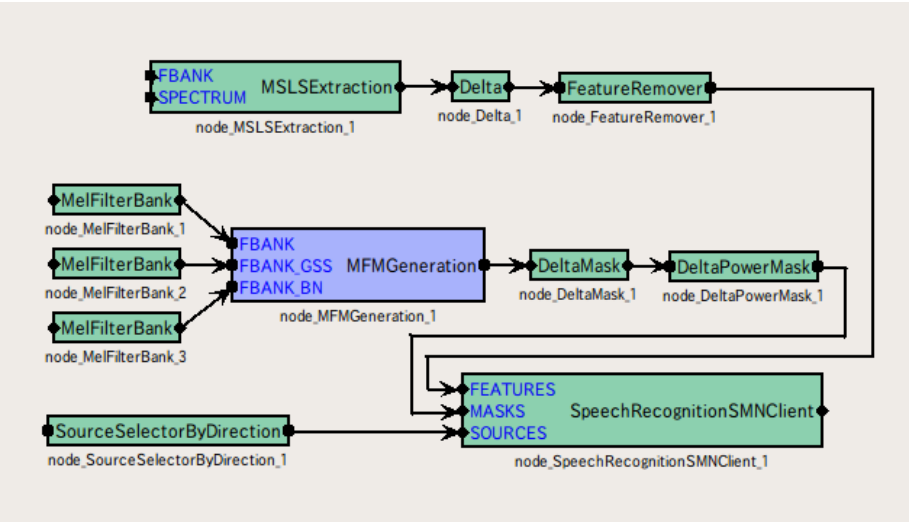

図 6.63: MFMGeneration の接続例

モジュールの入出力とプロパティ

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．

表 6.42: 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

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．モジュール内部で計算する 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．

モジュールの詳細

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，

$$\boldsymbol{m}(f) = [m(f,0), m(f,1), \ldots, m(f,P-1)]^T \tag{6.112}$$

$$m(f,p) = \begin{cases} 1.0, & r(p) > \mathrm{THRESHOLD} \\ 0.0, & r(p) \leq \mathrm{THRESHOLD} \end{cases}, \tag{6.113}$$

$$r(p) = \min(1.0, (f(p) + 1.4 * b(p))/(fg(p) + 1.0)), \tag{6.114}$$

$$\tag{6.115}$$

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 6.64 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 6.64: MFMGeneration の出力ベクトル列

6.6 ASRIF カテゴリ

6.6.1 SpeechRecognitionClient

モジュールの概要

音響特徴量をネットワーク経由で音声認識モジュールに送信するモジュールである．

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

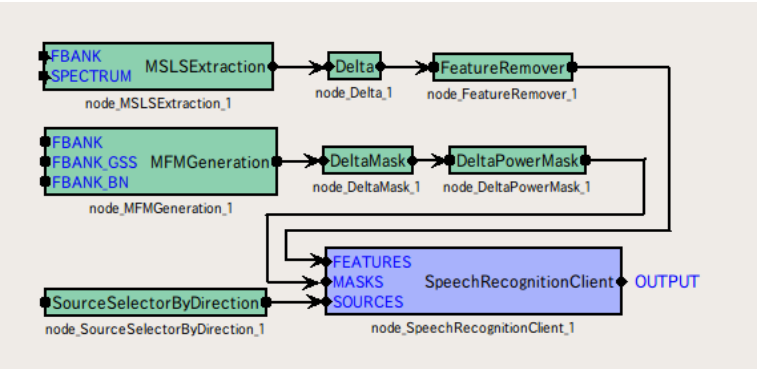

図 6.65: SpeechRecognitionClient の接続例

モジュールの入出力とプロパティ

表 6.43: 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
MFT_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

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 : Source 型．

出力

OUTPUT : Vector<ObjectRef>型．

パラメータ

MFT_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．

モジュールの詳細

MFT_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モジュールに送信するモジュールである．MFT_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，特徴量を音声認識モジュールに送信しない．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

モジュールの概要

音響特徴量をネットワーク経由で音声認識モジュールに送信するモジュールである．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，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．近似処理の詳細は，モジュール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

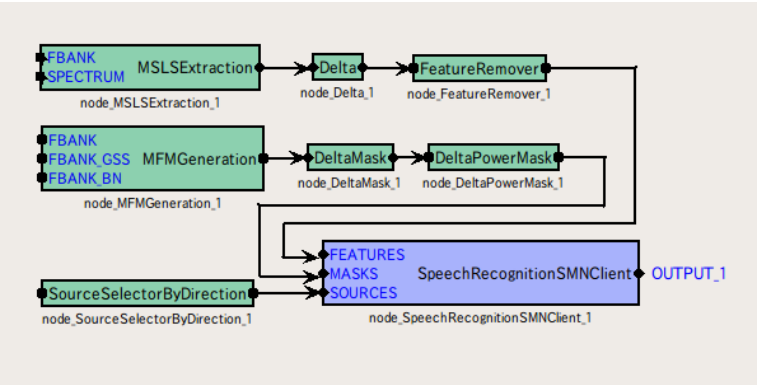

図 6.66: SpeechRecognitionSMNClient の接続例

モジュール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 ：`Source` 型．

出力

表 6.44: 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
MFT_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

OUTPUT : Vector<ObjectRef>型.

パラメータ

MFT_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.

モジュールの詳細

MFT_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モジュールに送信するモジュールである. MFT_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, 特徴量を音声認識モジュールに送信しない. 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

モジュールの概要

マルチチャネルの音声波形データから，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

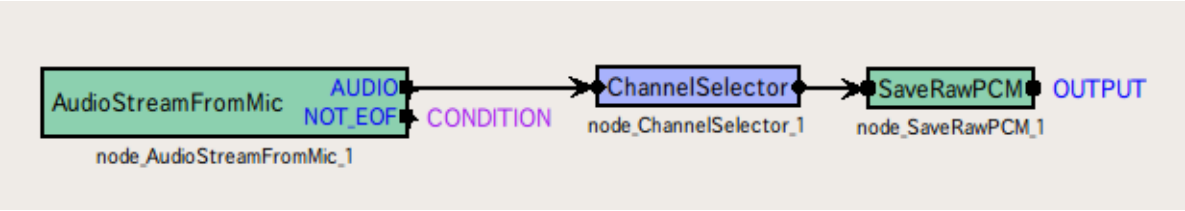

図 6.67: ChannelSelector の典型的な接続例

図 6.67 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，主な出力先は SaveRawPCM，MultiFFT などである．

モジュールの入出力とパラメータ

入力

INPUT ：`Matrix<float>`型．マルチチャネルの音声波形データ．

出力

OUTPUT ：`Matrix<float>`型．マルチチャネルの音声波形データ．

パラメータ

表 6.45: 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	Vector< int >	Vector< int >		出力するチャネルの番号を指定

SELECTOR : Vector<int>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>

モジュールの詳細

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 DataLogger

モジュールの概要

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

モジュールのデバッグの際や，モジュール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．

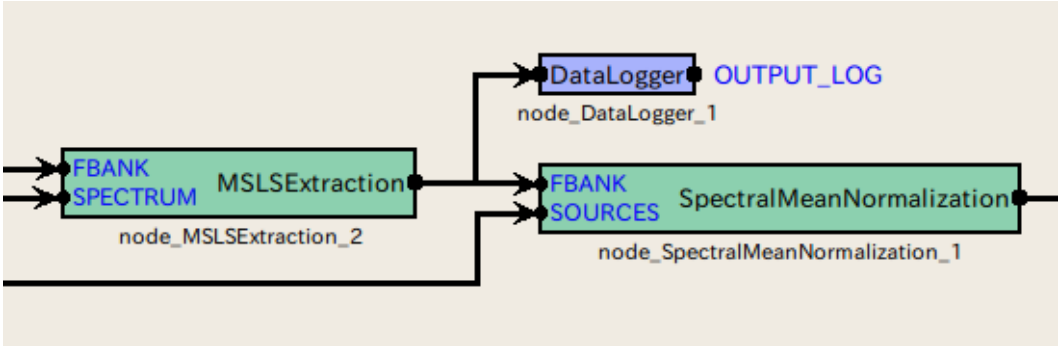

図 6.68: DataLogger の接続例

モジュールの入出力とプロパティ

表 6.46: 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．

モジュールの詳細

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 . . .

本モジュールの 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．

6.7.3 MatrixToMap

モジュールの概要

Matrix<float>型や，Matrix<complex<float> >型のデータを Map<int, ObjectRef>型に変換する．

必要なファイル

無し．

使用方法

どんなときに使うのか

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 モジュールの接続例を図 6.69，6.70 に示す．

図 6.69 は，AudioStreamFromMic モジュールでマイクロホンから音声波形データを取り込み，ChannelSelector モジュールにて必要なチャネルを選別し，MatrixToMap モジュールによって Matrix<float>型データを Map<int, ObjectRef>型に変換する．その出力を SaveRawPCM モジュールに接続し，波形をファイルとして保存する．

図 6.70 は，波形のスペクトルを Map<int, ObjectRef>型で得たいときの MatrixToMap モジュールの使い方である．図のように，MultiFFT モジュールを通すのは，MatrixToMap の前でも後でも良い．

モジュールの入出力とプロパティ

入力

INPUT : Matrix<float>または Matrix<complex<float> >型．

出力

OUTPUT : Map<int, ObjectRef>型．入力データに ID が付与された構造体．

パラメータ

無し．

モジュールの詳細

ID の付き方: ID の値は常に 0 となる．

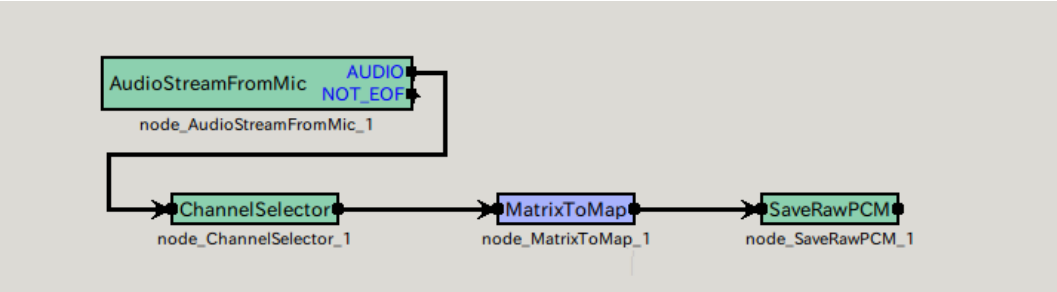

図 6.69: MatrixToMap の接続例 – SaveRawPCM への接続

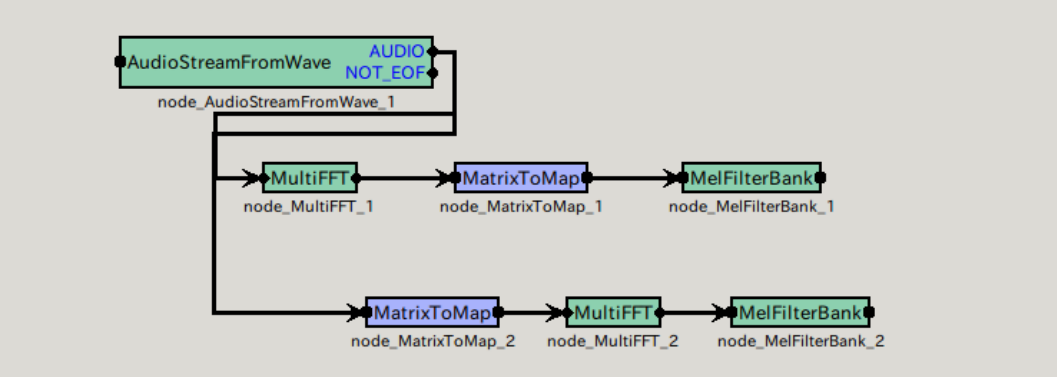

図 6.70: MatrixToMap の接続例 – MultiFFT との接続

6.7.4 MultiDownSampler

モジュールの概要

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 モジュール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，前段の入力モジュール，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，このモジュール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．このモジュール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）：このモジュールの ADVANCE は，後段モジュール（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，後段モジュールで 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，このモジュールや前段モジュールの ADVANCE は 480 に設定する必要がある．

注意 3（前段モジュールでの LENGTH 値の要求）：このモジュール以前（AudioStreamFromMic など）における LENGTH 値も，後段モジュール（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．

モジュール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）.

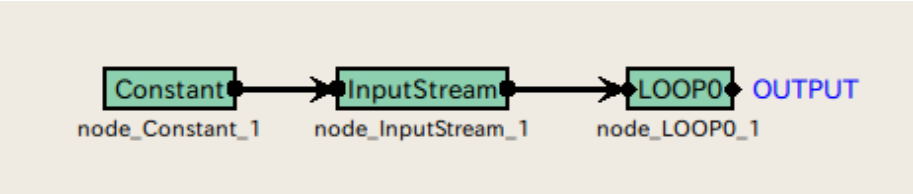

図 6.71: MultiDownSampler の接続例: Main ネットワーク

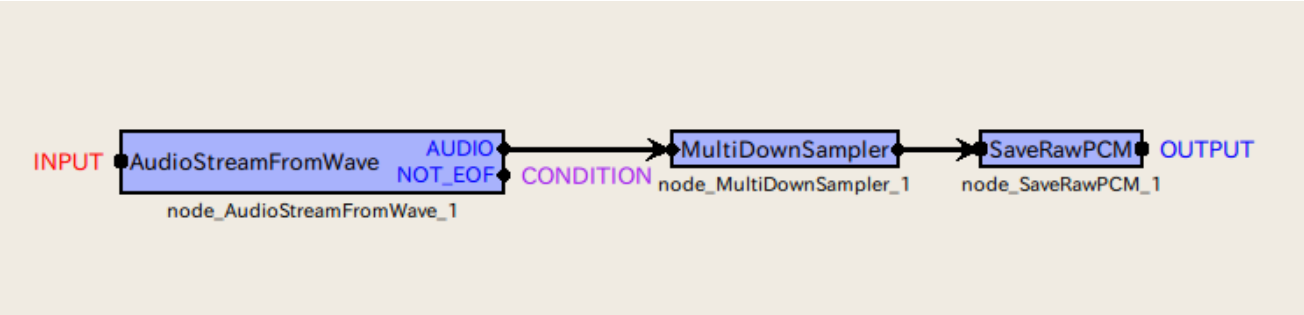

図 6.72: MultiDownSampler の接続例: Iteration(LOOP0) ネットワーク

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 6.47: 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 6.73 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 以前のモジュール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．

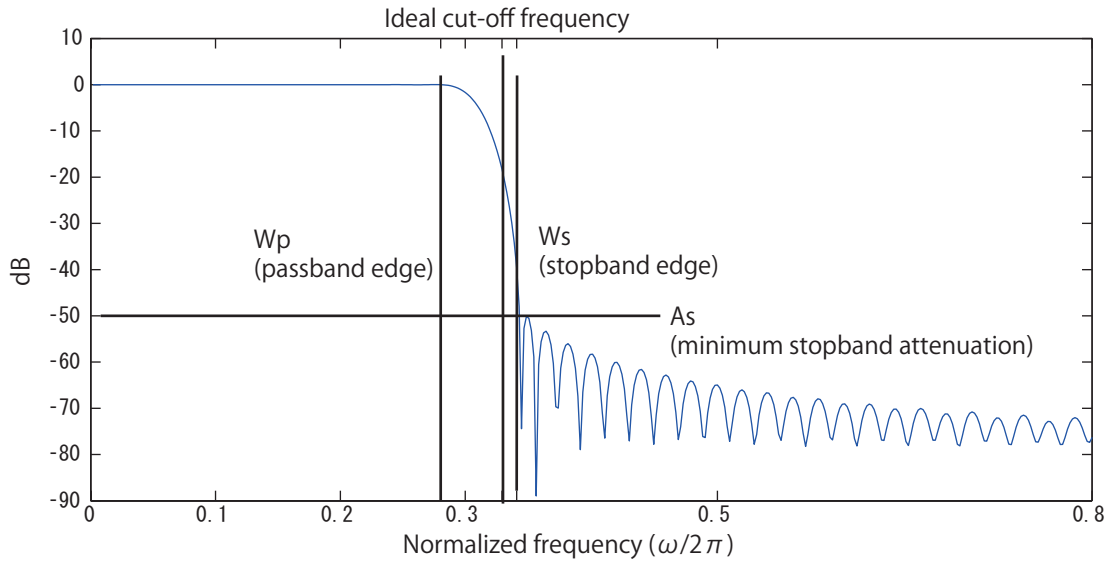

図 6.73: デフォルト設定のフィルタ特性: 横軸 正規化周波数 [$\omega/2\pi$]，縦軸 ゲイン [dB]

モジュールの詳細

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モジュールである．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．

$$s_{\mathrm{out}}(t) = \sum_{i=0}^{N} h(n) s_{\mathrm{in}}(t-n) \tag{6.116}$$

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．

$$H_i(e^{j\omega}) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \text{otherwise} \end{cases} \tag{6.117}$$

このインパルス応答は,

$$h_i(n) = \frac{\omega_c}{\pi}\left(\frac{sin(\omega n)}{\omega n}\right), \quad -\infty \leq n \leq \infty \tag{6.118}$$

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．

$$h(n) = \begin{cases} h_i(n), & |n| \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.119}$$

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．

$$h(n) = h_i(n)v(n) \tag{6.120}$$

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．

$$v(n) = \begin{cases} \frac{I_0\left(\beta\sqrt{1-(nN/2)^2}\right)}{I_0(\beta)}, & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6.121}$$

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，

$$I_0(x) = 1 + \sum_{k=1}^{\infty}\left(\frac{(0.5x)^k}{k!}\right) \tag{6.122}$$

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．

$$\beta = \begin{cases} 0.1102(As - 8.7) & As > 50, \\ 0,5842(As - 21)^{0.4} + 0.07886(As - 21) & 21 < As < 50, \\ 0 & As < 21 \end{cases} \tag{6.123}$$

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，

$$N \approx \frac{As - 7.95}{14.36\Delta f} \tag{6.124}$$

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.

6.7.5 MultiFFT

モジュールの概要

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

このモジュールは，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 6.74 で，MultiFFT モジュールに `Matrix<float>`，`Map<int, ObjectRef>`型の入力を与える例を示す．

図 6.74 の上のパスは AudioStreamFromWave モジュールから `Matrix<float>`型の多チャネル音響信号を受け取り，MultiFFT モジュールで `Matrix<complex<float> >`型の複素スペクトルに変換したのち，LocalizeMUSIC モジュールに入力される．

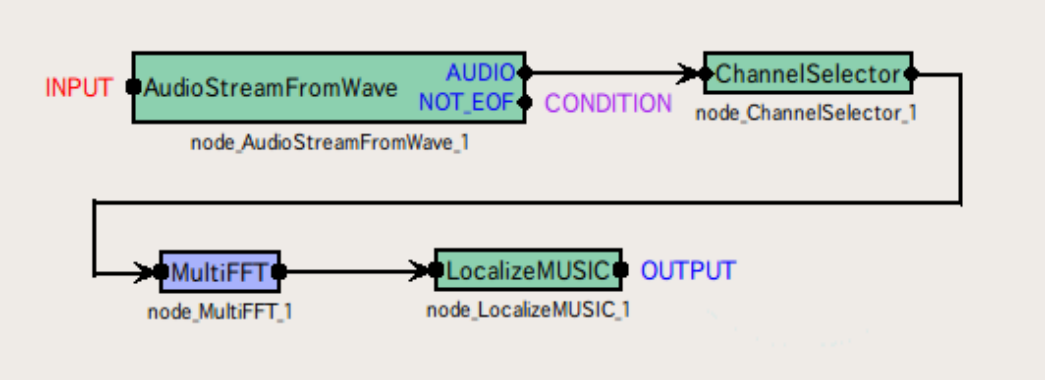

図 6.74: MultiFFT の接続例

モジュール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．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．

出力

表 6.48: 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 から選択する．それぞれ，複素窓，ハミング窓，矩形窓を示す．
WINDOW_LENGTH	int	512	[pt]	フーリエ変換を行う際の窓関数の長さ．

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．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 から選択する．それぞれ，複素窓，ハミング窓，矩形窓を意味する．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．

モジュールの詳細

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 $0 \le k < L$ の範囲を動く．FFT 長が窓の長さよりも大きいとき，$NFFT \le k < L$ 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．

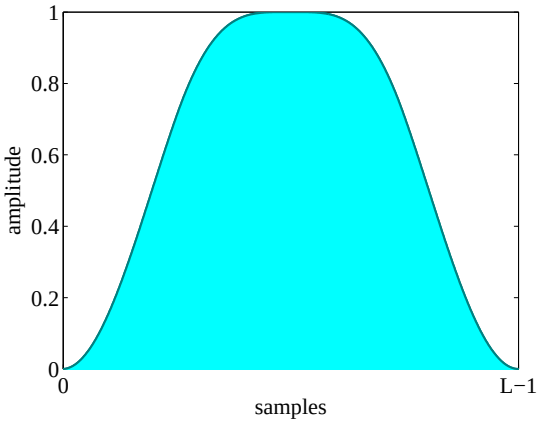

図 6.75: 複素窓関数の形状

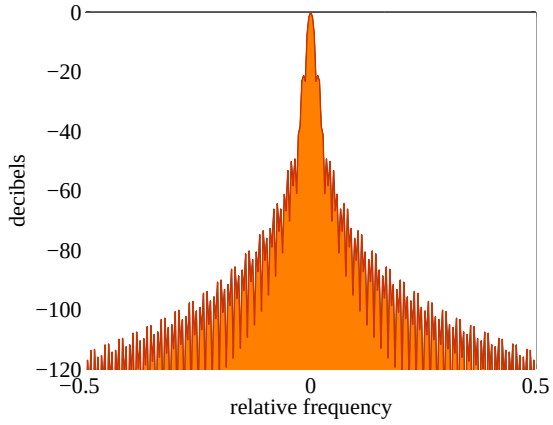

図 6.76: 複素窓関数の周波数応答

図 6.75 と図 6.76 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 6.76 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 6.77，6.77 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 6.79，6.79 はそれぞれ，矩形窓関数の形状と周波数応答である．

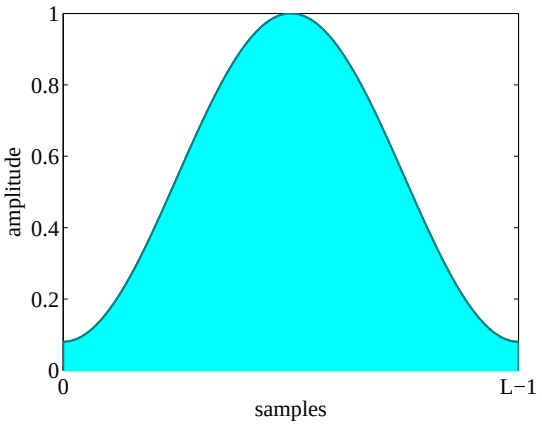

図 6.77: ハミング窓関数の形状

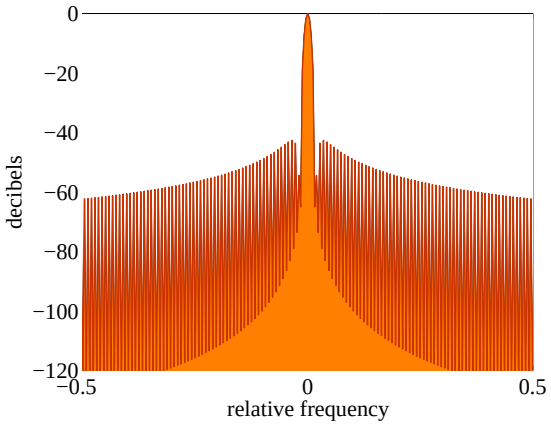

図 6.78: ハミング窓関数の周波数応答

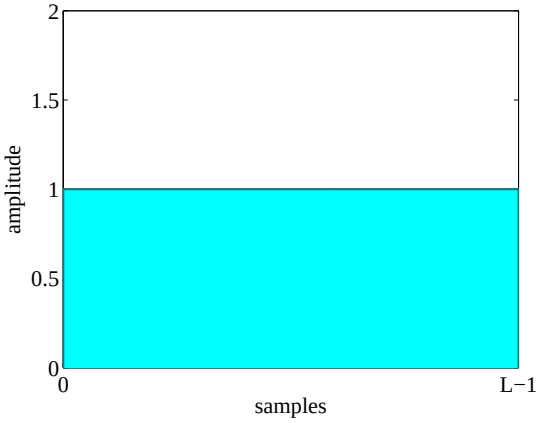

図 6.79: 矩形窓関数の形状

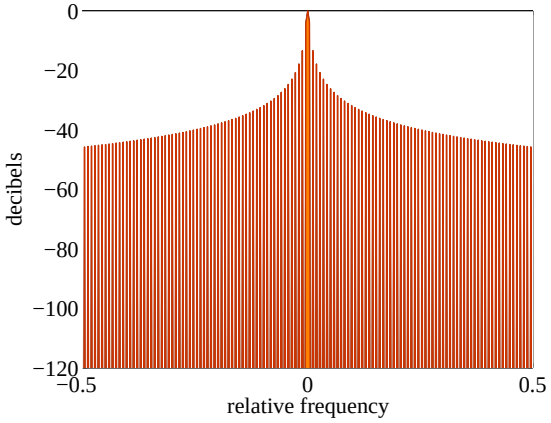

図 6.80: 矩形窓関数の周波数応答

6.7.6 MultiGain

モジュールの概要

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，このモジュールを利用して，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．

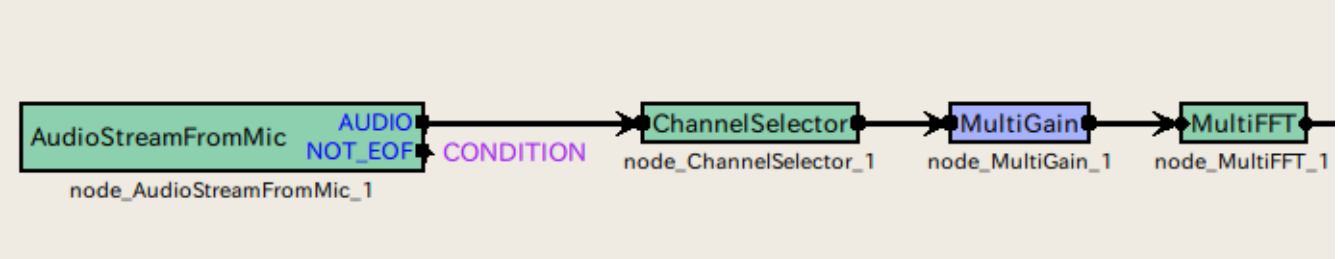

図 6.81: MultiGain の接続例

モジュールの入出力とプロパティ

表 6.49: 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

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）．

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．

モジュールの詳細

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．

$$20 \log x = -40 \tag{6.125}$$

$$x = 0.01 \tag{6.126}$$

6.7.7 PowerCalcForMap

モジュールの概要

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>型のときはこのモジュールを用いる．入力が Matrix<complex<float> >型の時は，PowerCalcForMatrix モジュールを用いる．

典型的な接続例

図 6.82 に PowerCalcForMap モジュールの使用例を示す．MultiFFT モジュールから得られた 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 モジュールに入力している．

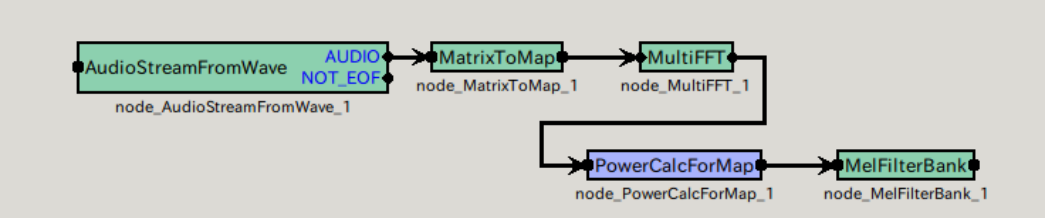

図 6.82: PowerCalcForMap の接続例

モジュールの入出力とプロパティ

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，絶対値を取った実行列が格納されている．

パラメータ

無し．

モジュールの詳細

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．

$$N_{i,j} \quad = \quad M_{i,j} M^*_{i,j},$$

ただし，$M^*_{i,j}$ は，$M_{i,j}$ の複素共役を表す．

6.7.8 PowerCalcForMatrix

モジュールの概要

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> >型のときはこのモジュールを用いる．入力が Map<int, ObjectRef>型のときは PowerCalcForMap モジュールを用いる．

典型的な接続例

図 6.83 に PowerCalcForMatrix モジュールの使用例を示す．MultiFFT モジュールから得られた 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 モジュールに入力している．

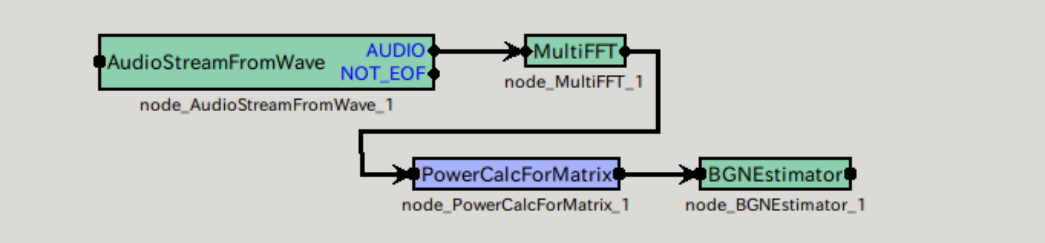

図 6.83: PowerCalcForMatrix の接続例

モジュールの入出力とプロパティ

入力

INPUT ：Matrix<complex<float> >型．各要素が複素数の行列．

出力

OUTPUT ：Matrix<float>型．入力の各要素の絶対値を取った実行列．

パラメータ

無し．

モジュールの詳細

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．

$$N_{i,j} \quad = \quad M_{i,j} M^*_{i,j},$$

ただし，$M^*_{i,j}$ は，$M_{i,j}$ の複素共役を表す．

6.7.9 SegmentAudioStreamByID

モジュールの概要

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モジュールである．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モジュールに入力する．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．

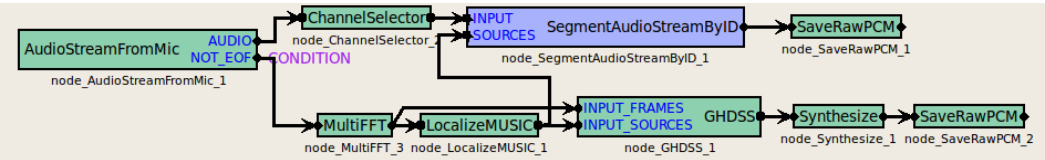

図 6.84: SegmentAudioStreamByID の接続例

モジュール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．

モジュールの詳細

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．このモジュールは，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.

6.7.10 SourceSelectorByDirection

モジュールの概要

入力された音源定位結果のうち，指定した水平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

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 6.85 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．

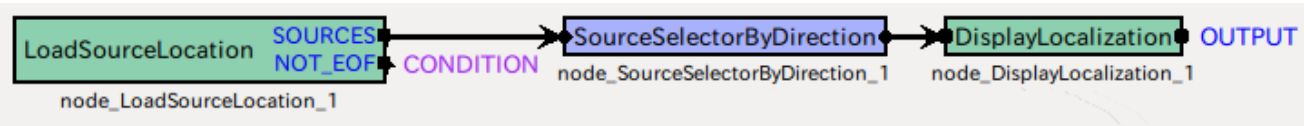

図 6.85: SourceSelectorByDirection の接続例

モジュール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

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．

表 6.50: 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]	通過させる音源方向の最小値
MAX_AZIMUTH	`float`	20.0	[deg]	通過させる音源方向の最大値

モジュールの詳細

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．

6.7.11 SourceSelectorByID

モジュールの概要

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．特に， GHDSS モジュールの FIXED_NOISE プロパティを true にした場合は，定常ノイズ分離結果に負の ID が振られるので，それ以外の音を処理するためのフィルタとして使用する．

必要なファイル

無し．

使用方法

どんなときに使うのか

音源分離モジュール GHDSS は，ロボットの電源を入れるが移動はさせない．すなわちノイズ (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，そのノイズの分離結果の ID を-1 として出力する．このとき，GHDSS モジュールの後に SourceSelectorByID を接続し，閾値を 0 に設定すると，定常ノイズの分離音を以後の処理で無視することができる．

典型的な接続例

図 6.86 に接続例を示す．図に示すように，このモジュールは，典型的には GHDSS の後段に接続される．

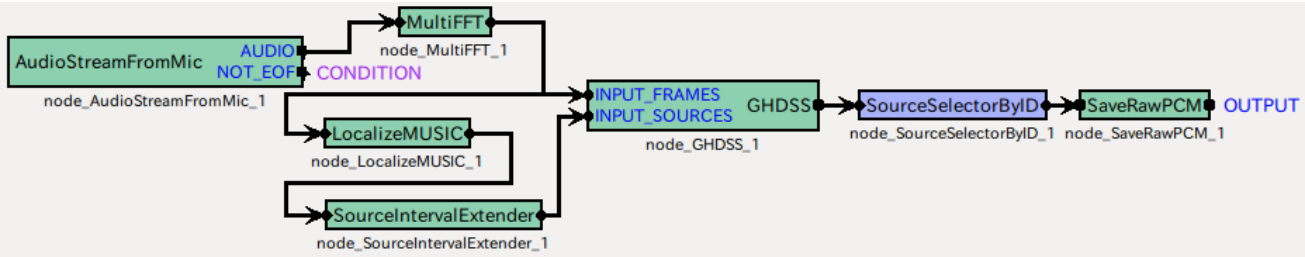

図 6.86: SourceSelectorByID の接続例

モジュールの入出力とプロパティ

入力

INPUT ：`Map<int, ObjectRef>`型．通常は音源分離モジュール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 と同様．

パラメータ

表 6.51: 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 . GHDSS の後段に接続するのであれば変更不要.

6.7.12 Synthesize

モジュールの概要

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

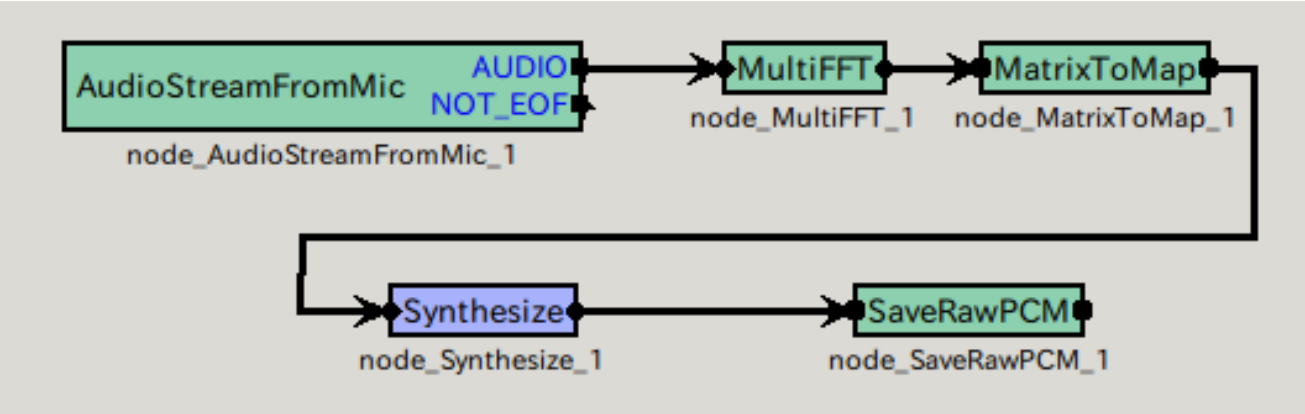

図 6.87: Synthesize の接続例

モジュールの入出力とプロパティ

表 6.52: 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，他のモジュール（ MultiFFT）と値を合わせる必要がある．

ADVANCE シフト長，他のモジュール（ MultiFFT）と値を合わせる必要がある．

SAMPLING_RATE サンプリングレート，他のモジュール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

モジュールの詳細

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，このモジュール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．

$$delay = \begin{cases} |\mathrm{LENGTH}/\mathrm{ADVANCE}| - 1, & \text{if } \mathrm{LENGTH}\ mod\ \mathrm{ADVANCE} \neq 0, \\ \mathrm{LENGTH}/\mathrm{ADVANCE}, & \text{otherwise.} \end{cases} \tag{6.127}$$

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

6.7.13 WhiteNoiseAdder

モジュールの概要

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

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．具体的なモジュール名をあげる．

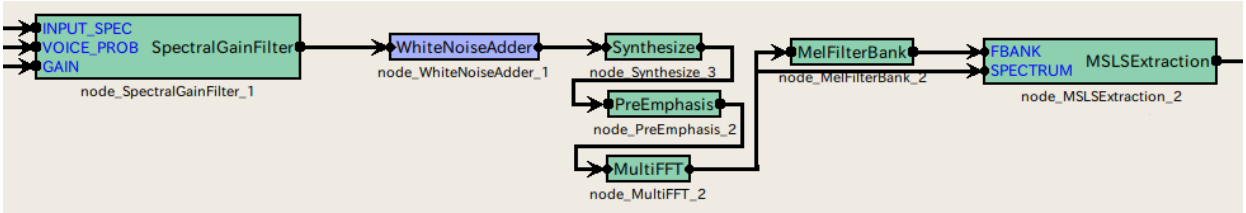

図 6.88: WhiteNoiseAdder の接続例

モジュールの入出力とプロパティ

表 6.53: 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

入力

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，他のモジュール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．

モジュールの詳細

入力信号の各周波数ビンに対して，

$$\frac{\sqrt{\mathrm{LENGTH}}}{2} \cdot \mathrm{WN\_LEVEL} \cdot e^{2\pi jR} \tag{6.128}$$

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[1] 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，Julius4.0 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 -C julius.jconf
```

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 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，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
- -hlist allTriphones

[1] http://www.furui.cs.titech.ac.jp/mband_julius/

- **-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.
- **-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 6.54 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 6.55）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 6.54: 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 6.55: 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 6.56 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 6.56: 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 6.57）を送信してソケットを閉じる．

表 6.57: 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．

第7章　サポートツール

7.1　harktool

7.1.1　概要

harktool は，GHDSS と LocalizeMUSIC の，伝達関数を生成，可視化するツールである．
伝達関数の生成に必要なファイルは，以下の 2 つである．

1. マイクロホン配置 (5.2 参照)
2. インパルス応答ファイル (5.3.1)

これら 2 種類のファイルをウィンドウから指定することで，伝達関数を生成する．LocalizeMUSIC に関しては，同様に提供しているツール mkmusictf を用い，シミュレーションで生成可能である．この場合，上記 1 のマイクロホン配置を記述したファイルのみで伝達関数を生成する．ただし，マイクロホンは自由空間に置かれたと仮定して生成するので，マイクロホンが実際に設置されている物体の反響など（e.g. ロボットの頭の反射など）は無視される．
伝達関数を生成した後は，gnuplot を用いて形状を確認可能である．

7.1.2　インストール方法

HARK と同様に，ウェブページ http://winnie.kuis.kyoto-u.ac.jp/HARK で公開している．
コンパイルは，アーカイブを解凍したディレクトリで make をする．

7.1.3　使用方法

現在， GHDSS と LocalizeMUSIC 用の伝達関数の生成，視覚化に対応している．ウィンドウ最上段のタブでノードの名前を選択する．
詳細は付属マニュアルを参照.

第8章　HARK 対応マルチチャネル A/D 装置の紹介と設定

HARK にマルチチャネル A/D 装置（以後，単に A/D 装置と呼ぶ）を接続することで，マイクロホンアレイ処理が可能になる．マイクロホンアレイ処理を必要とするユーザは，本章を参考にし，必要な A/D 装置の設定を行う．

HARK でサポートする A/D 装置は，以下の 3 機種である．

1. System In Fronteir， Inc． RASP シリーズ，
2. ALSA ベースのデバイス (例，RME Hammerfall DSP Multiface シリーズ)，
3. 東京エレクトロンデバイス TD-BD-16ADUSB．

各装置についてインストール方法と設定方法を説明する．ただし，TD-BD-16ADUSB はサードパーティ版 HARK のみが対応している．

8.1　System In Fronteir， Inc． RASP シリーズ，無線 RASP

本節では，HARK 開発チームが動作確認を行っているマルチチャネル A/D 装置の一つである System In Fronteir, Inc． の RASP シリーズの中の１つである無線 RASP（以下，単に無線 RASP と呼ぶ）を HARK で使用するための設定手順について説明する．

図 8.1 に無線 RASP の概観を示す．無線 RASP は，単体で 16ch A/D 変換と 2 ch D/A 変換が可能な A/D，D/A 変換装置である．TCP/IP 経由でコマンドを送受信し，ホスト PC から A/D，D/A 変換処理の設定とデータの送受信を行う．サンプリング周波数や，アナログ入力部分のフィルタをソフトウェアから容易に変更でき使い勝手が良い．

無線 RASP のインストール方法，および，HARK 上での無線 RASP を接続したマイクロホンからの音声録音方法を取り上げる．説明と動作確認は，Ubuntu 9．10 で行っている．

8.1.1　無線 RASP の PC への接続

無線 RASP は，単体のボックスで構成されている．ホスト PC との接続は，無線 LAN で行うため，別途無線 LAN に対応したルーターが必要である．無線 RASP には，事前に IP アドレスを設定する他，無線ルーターの SSID を登録しておく必要がある．IP アドレスは出荷時設定をそのまま使用してもよいが，SSID の設定は必須である．添付マニュアルを参照して設定を完了させておく．

8.1.2　無線 RASP 用ソフトウェアのインストールと設定

無線 RASP を操作するためには，操作するホスト PC に，ライブラリのインストールとサンプルプログラムをインストールする必要がある．無線 RASP に同梱の CD からファイルをコピーし，インストール作業を行う．詳細は，添付資料を参考に行う．

図 8.1: System In Fronteir, Inc. Wireless RASP

HARK の使用には，少なくとも無線 RASP のアナログフィルタの設定を初期化できる環境を用意する必要がある．具体的には， ws_fpaa_config というコマンドのインストールが必須である．ライブラリをインストールし，サンプルプログラムをコンパイルすると ws_fpaa_config を生成できる．ws_fpaa_config は，無線 RASP の電源を切ると設定値が消えるため，電源を入れた直後に必ず実行する必要がある．

8.1.3 無線 RASP を用いた HARK での録音テスト

録音に先立ち機材を確認する．使用機材を表 8.1 に示す．

表 8.1: 使用機材一覧

機材名	説明
無線 RASP	A/D，D/A 変換装置
無線 LAN 対応ルーター	RASP と ホスト PC 接続用
PC	無線 RASP 操作用 PC

HARK を用いた録音テストには，図 8.2 に示すネットワークファイル test．n を用いる．AudioStreamFromMic から ChannelSelector を経由し，MatrixToMap から SaveRawPCM への接続となっている．各プロパティの設定を表 8.2 に示す．

```
> ./test.n
```

とすることで，無線 RASP のアナログ入力 1 の音声が 3 秒間録音される．PCM 16 ビット量子化，16kHz サンプリングの音声がリトルエンディアン RAW 形式で保存される．

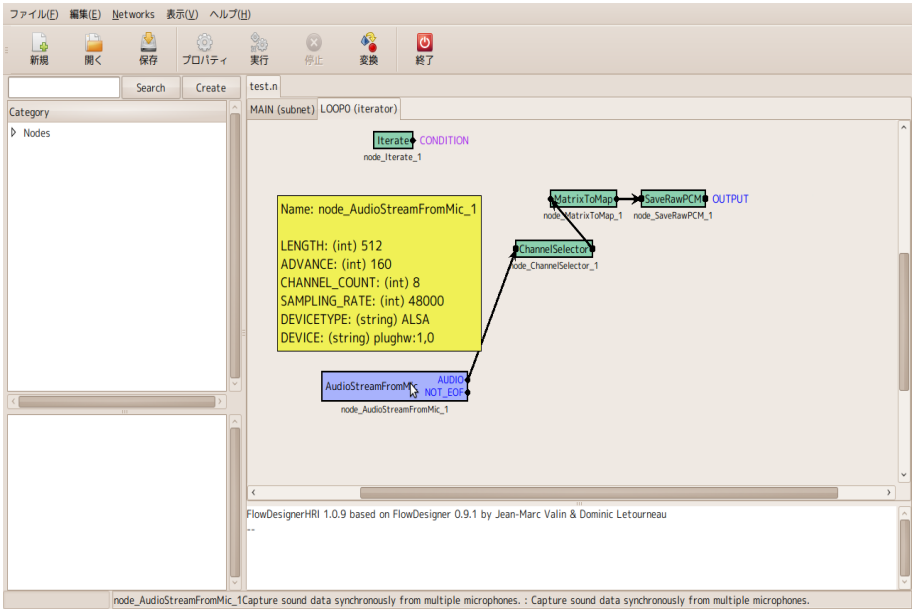

図 8.2: ネットワーク (test.n)

表 8.2: 録音ネットワークのプロパティ

モジュール名	プロパティ名	値の型	設定値
AudioStreamFromMic	LENGTH	int	512
	ADVANCE	int	160
	CHANNEL_COUNT	int	16
	SAMPLING_RATE	int	16000
	DEVICETYPE	string	WS
	DEVICE	string	192．68．0．1
ChannelSelector	SELECTOR	Object	<Vector<int> 0 >
SaveRawPCM	BASENAME	string	sep_
	ADVANCE	int	160
	BITS	int	16
Iterate	MAX_ITER	int	300

8.2 RME Hammerfall DSP シリーズ Multiface AE

本節では，HARK 開発チームが動作確認を行っているマルチチャネル A/D 装置の一つである RME 社の Hammerfall DSP Multiface シリーズ（以下，Hammerfall DSP Multiface シリーズ を単に Multiface と呼ぶ）を HARK で使用するための設定手順について説明する．

Multiface は，単体で 10 ch A/D 変換と 10 ch D/A 変換可能な A/D，D/A 装置である．10 ch のうちアナログ部が 8 ch で，SPDIF 部が 2 ch である．Linux で標準的なインタフェースである ALSA 準拠デバイスである．ALSA 準拠デバイスの代表として設定方法を紹介する．他の，ALSA 準拠デバイスを ALSA ドライバを通じて OS に認識させることができれば，Multiface と同様に， HARK でマルチチャネル録音が可能である．Multiface 以外の ALSA 準拠デバイスはサポート外であるが，本節は，サポート外の ALSA 対応の A/D，D/A 装置を使用する際に参考になる．

Multiface のインストール方法，および，HARK 上での Multiface を接続したマイクロホンからの音声録音方法を取り上げる．説明と動作確認は，Ubuntu 9.10 で行っている．

8.2.1 Multiface の PC への接続

RME 社製の 24bit/96kHz マルチチャネル A/D 装置は，下記の 3 つの装置から構成される．

- **RME HDSP I/O ボックス**: RME HDSP Multiface AE,
- **RME HDSP インタフェース**: RME HDSP PCI カード，RME HDSP CardBus カードのいずれか
  RME HDSP PCI-Express カード，および RME HDSP Express カードは，動作未確認であり，現在サポート外である．
- **マイクロホン用プリアンプ**: RME OctaMic-II．RME OctaMic や YAMAHA HA-8 も使用実績があるが，これらのモデルは製造中止になっている．

ただし，マイクロホン用プリアンプは，必ずしも必要ではなく，録音レベルを確保できるならば，接続する必要はない．これらのハードウエアは，添付マニュアルを参照して，PC に接続する．

Multiface 用ソフトウェアのインストールと設定

Multiface を Linux にマルチチャネル A/D 機器として認識させるためには，ALSA デバイス用汎用ドライバと，Multiface 用のファームウエア及び，ソフトウェアをインストールし，パラメータを設定する必要がある．

以下のインストールは，パッケージからインストールする方法を強く推奨する．ソースからコンパイル，インストールという方法もあるが，他のソフトウェアとの競合が起り，コンパイルとインストールが見かけ上，成功しても，実際には音が再生・録音できないという現象に陥りやすい．特に pulseAudio との競合が起るので注意が必要である．各自のシステム環境に強く依存するため，ソースからのインストールの具体的な解説は割愛する．

以下パッケージから，ALSA デバイス用汎用ドライバのインストール方法を説明した後に，Multiface の初期化に必要な hdsp コマンドを使用可能にするためのインストール作業について述べ，hdsp コマンドによる設定について述べる．最後に，HARK を使った録音ネットワークの例題について解説する．

ALSA デバイス用汎用ドライバのインストール

ALSA デバイス用汎用ドライバ (alsa-lib， alsa-driver) は，インストールされていることが多い．インストールされているか確認するには，以下の太字の部分を入力する．> はコマンドプロンプトを表す．イタリック体部分のメッセージが表示されれば，alsa-lib， alsa-drive がインストールあされているので，これらのインストールは必要ない．Version 番号は，使用環境によって異なるので作業例中の Version 1.0.23 は，各自の使用環境で異なるバージョン番号を示す場合がある．適宜読み換えて作業を進める．今回のテスト環境では， Version 1.0.23 を使用した．

> **cat /proc/asound/version**
Advanced Linux Sound Architecture Drive Version 1.0.23.

インストールされていなければ，次の節の作業で自動的に自動インストールされるので，このまま次節の作業に入ればよい．

Multiface の初期化に必要なコマンドを使用可能にするためのインストール作業

必要なパッケージは， 以下の 2 つである．

- alsa-firmware-loaders
- alsa-tools-gui

これらのパッケージのインストールには,

- Synaptic パッケージマネージャ
- apt-get

のいずれかを使用する．以上のパッケージをインストールすることで，Multiface の設定に必要な以下の 3 つのコマンドが使用可能になる．

- hdsploader (Package : alsa-firmware-loaders)
- hdspconf (Package : alsa-tools-gui)
- hdspmixer (Package : alsa-tools-gui)

apt-get を使用したインストール例を示す.

```
> sudo apt-get upadte
> sudo apt-get install alsa-firmware-loaders
> sudo apt-get install alsa-tools-gui
```

最後に，multiface_firmware_rev11.bin を入手するために，alsa-firmware を alsa の Web サイトからソースをダウンロードする．(http://www．alsa-project．org/main/index．php/Main_Page)．2010/8/9 現在，利用可能な最新バージョンは， 1.0.23 である．alsa-firmware-1.0.23.tar.bz2 をダウンロードする．ダウンロード後，ファイルは，システムメニューの「場所」「ダウンロード」に保存される．これを,「場所」「ホーム」にコピーすると作業しやすい．

適当なディレクトリにダウンロードしたファイル alsa-firmware-1.0.23.tar.bz2 をコピーする．最初に bunzip2 と tar でファイルを展開する.「アプリケーション」「アクセサリ」「端末」を選ぶ．ダウンロードしたファイルを「場所」「ホーム」にコピーしてあるならば，開いたウィンドウで以下のコマンドを実行する．異なるディレクトリにコピーした場合には，cd コマンドを用いてそのディレクトリに 移動する．その後，コンパイル作業に入る．具体的な作業手順を以下に示す.

```
> bunzip2 -d alsa-firmware-1.0.23.tar.bz2
> tar vfx alsa-firmware-1.0.23.tar
alsa-firmware-1.0.23/
alsa-firmware-1.0.23/multisound/
  中略
alsa-firmware-1.0.23/hdsploader/
alsa-firmware-1.0.23/hdsploader/digiface_firmware_rev11.dat
alsa-firmware-1.0.23/hdsploader/Makefile.in
alsa-firmware-1.0.23/hdsploader/tobin.c
alsa-firmware-1.0.23/hdsploader/digiface_firmware.dat
alsa-firmware-1.0.23/hdsploader/README
alsa-firmware-1.0.23/hdsploader/Makefile.am
alsa-firmware-1.0.23/hdsploader/multiface_firmware_rev11.dat
alsa-firmware-1.0.23/hdsploader/multiface_firmware.dat
  中略
alsa-firmware-1.0.23/echoaudio/Makefile.am
alsa-firmware-1.0.23/echoaudio/fw_writer.c
> cd alsa-firmware-1.0.23
> sh configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
  中略
config. status: creating aica/Makefile
config. status: executing depfiles commands
> make
> sudo mkdir -p /lib/firmware/hdsploader
> sudo cp hdsploader/multiface_firmware_rev11.dat /lib/firmware/hdsploader/
```

以上で， hdsp コマンドがインストールされて Multiface の操作と設定が可能になった．

hdsp コマンドによる設定

必要な hdsp コマンドは hdsploader， hdspconf， hdspmixer の３つである．これらのコマンドを順に説明する．

- **hdsploader**
  hdsploader は，multiface の FPGA を初期化する firmware プログラム（multiface_firmware_rev11.dat）をアップロードするコマンドである．以下に実行例を示す．エラーメッセージ (Hwdep ioctl error on card hw:0 : Device or resource busy.) が表示されるが，ここでは問題ないので無視してよい．同一システム上で 2 度以上実行すると，エラーメッセージが表示される．OS 起動中に実行されるシステムの場合には，この作業で必ずこのエラーメッセージが表示される．

```
# hdsploader
hdsploader - firmware loader for RME Hammerfall DSP cards
Looking for HDSP + Multiface or Digiface cards :
Card 0 : RME Hammerfall DSP + Digiface at 0xf9df0000, irq 16
Upload firmware for card hw:0
Hwdep ioctl error on card hw:0 : Device or resource busy.
Card 1 : HDA Intel at 0xfdffc000 irq 30
```

- **hdspconf**
  hdspconf を実行すると，Multiface の設定用のウィンドウが開く．サンプリング周波数を設定できる．他の項目は Multiface のドキュメントを参考にする．図 8.3 に設定用のウィンドウ例を示す．設定可能なサンプリングレートは，32kHz， 44.1kHz， 48kH， 64kHz， 88.2KkHz， 96kHz である．ここでは，32kHz が選択されている．

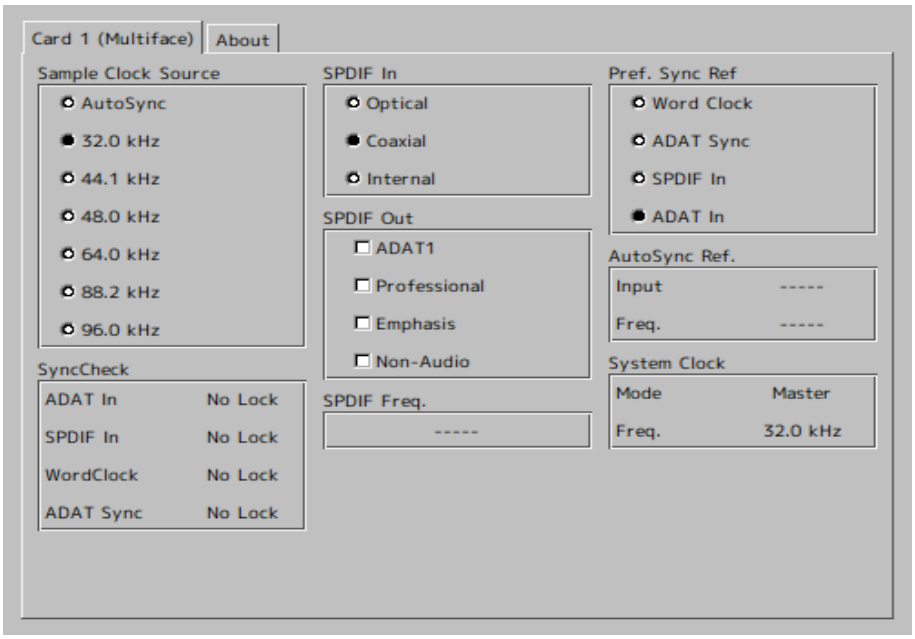

図 8.3: パラメータ設定ウィンドウ

- **hdspmixer**
  hdspmixer を実行すると，ミキサーの GUI が表示される．入力レベル，出力レベルの調節と確認が可能である．図 8.4 にミキサーの表示例を示す．

8.2.2 Multiface を用いた HARK での録音テスト

録音に先立ち機材を確認する．使用機材を表 8.3 に示す．

HARK を用いた録音テストには，図 8.2 と同じネットワークファイル test.n を用い，各モジュールのプロパティのみを変更する．各プロパティの設定を表 8.4 に示す．

```
> ./test.n
```

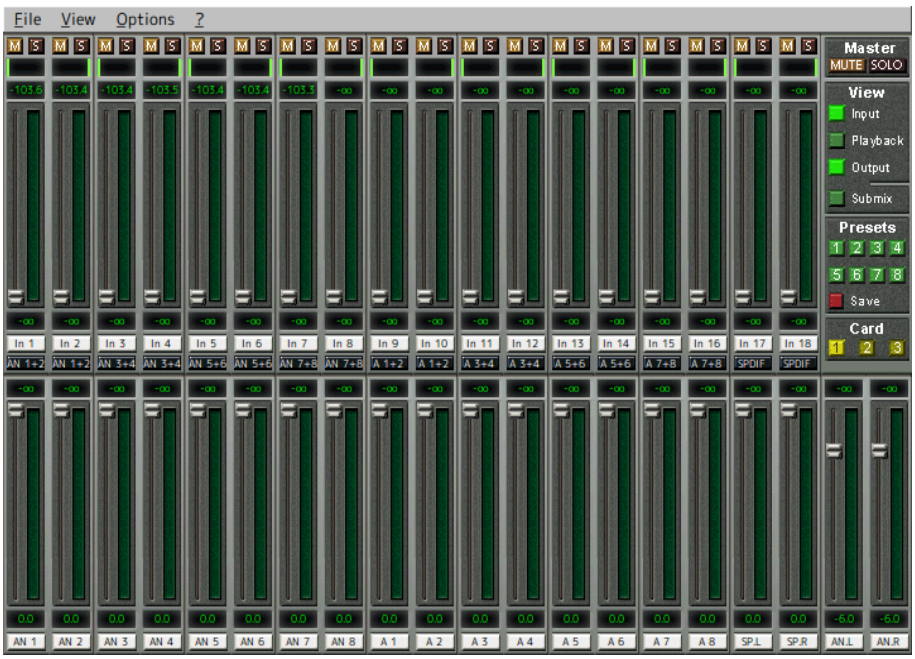

図 8.4: パラメータ設定ウィンドウ

表 8.3: 使用機材一覧

機材名	説明
Hammerfall DSP Multiface	A/D，D/A 変換装置
Digital Audio CardBus Interface	Multiface 接続専用ポートを PCMCIA スロットに増設するカード
Octa Mic	マイクロホンプリアンプ

図 8.5: RME Hammerfall DSP Multiface (front)

とすることで，Multiface のアナログ入力 1 の音声が 3 秒間録音される．PCM 16 ビット量子化，32kHz サンプリングの音声がリトルエンディアン RAW 形式で保存される．

8.3　東京エレクトロンデバイス TD-BD-16ADUSB

本節では，HARK 開発チームが動作確認を行っているマルチチャネル A/D 装置の一つである東京エレクトロンデバイス の Inrevium シリーズの中の 1 つである TD-BD-16ADUSB （以下，東京エレクトロンデバイス TD-BD-16ADUSB

図 8.6: RME Hammerfall DSP Multiface (Rear)

図 8.7: PCMCIA CardBus Interface for Hammerfall DSP System

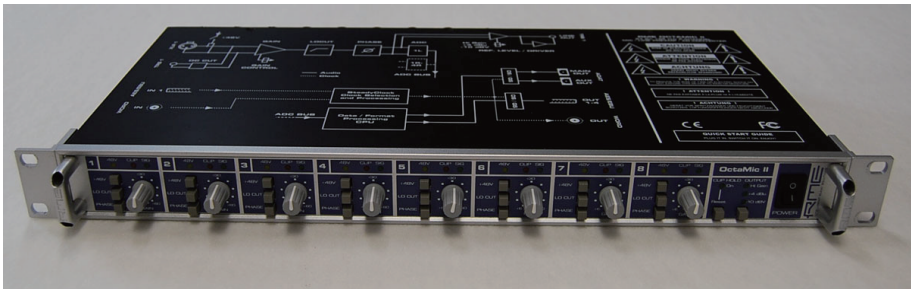

図 8.8: RME OctaMic (Front)

を単に 16ADUSB と呼ぶ）を HARK で使用するための設定手順について説明する．なお，TD-BD-16ADUSB を HARK で使うためには，サードパーティ版のみに含まれる AudioStreamFromMic2 を用いる必要がある．

16ADUSB は，単体で 16ch A/D 変換と 4 ch D/A 変換が可能な A/D，D/A 変換装置である．16ADUSB のドライ

図 8.9: RME OctaMic (Rear)

表 8.4: 録音ネットワークのプロパティ

モジュール名	プロパティ名	値の型	設定値
AudioStreamFromMic	LENGTH	int	1024
	ADVANCE	int	320
	CHANNEL_COUNT	int	8
	SAMPLING_RATE	int	32000
	DEVICETYPE	string	ALSA
	DEVICE	string	plughw:1，0
ChannelSelector	SELECTOR	Object	<Vector<int> 0 >
SaveRawPCM	BASENAME	string	sep_
	ADVANCE	int	320
	BITS	int	16
Iterate	MAX_ITER	int	300

バソフトウェアは，製品に附属している．Linux 用のカーネルモードドライバが附属しており使い勝手が良い．マイクロホンの接続では，プラグインパワー供給に対応しているため，プラグインパワー供給可能なコンデンサマイクロホンをそのまま接続できる．

16ADUSB のインストール方法，および，HARK 上での 16ADUSB を接続したマイクロホンからの音声録音方法を取り上げる．説明と動作確認は，Ubuntu 9.10 で行っている．

8.3.1 16ADUSB の PC への接続

16ADUSB は，名刺サイズの基盤に実装されている．ホスト PC との接続は，USB で行うため，接続が容易である．ハードウエアは，添付マニュアルを参照して，PC に接続する．

8.3.2 16ADUSB 用ソフトウェアのインストールと設定

16ADUSB を操作するためには，操作するホスト PC に，カーネルモードドライバをインストールする必要がある．

8.3.3 TD-BD-16ADUSB を用いた HARK での録音テスト

録音に先立ち機材を確認する．使用機材を表 8.5 に示す．

表 8.5: 使用機材一覧

機材名	説明
TD-BD-16ADUSB	A/D，D/A 変換装置
PC	TD-BD-16ADUSB 操作用 PC

HARK を用いた録音テストには，図 8.2 と同じネットワークファイル test.n を用い，各プロパティのみを変更する．ただし，ここで AudioStreamFromMic ではなくサードパーティ版のみに含まれる AudioStreamFromMic2 を用いる点に注意が必要である．各プロパティの設定を表 8.6 に示す．

```
> ./test.n
```

とすることで，16ADUSB のアナログ入力 1 の音声が 3 秒間録音される．PCM 16 ビット量子化，16kHz サンプリングの音声がリトルエンディアン RAW 形式で保存される．

表 8.6: 録音ネットワークのプロパティ

モジュール名	プロパティ名	値の型	設定値
AudioStreamFromMic2	LENGTH	int	512
	ADVANCE	int	160
	CHANNEL_COUNT	int	16
	SAMPLING_RATE	int	16000
	DEVICETYPE	string	SINICH
	DEVICE	string	SINICH
ChannelSelector	SELECTOR	Object	<Vector<int> 0 >
SaveRawPCM	BASENAME	string	sep_
	ADVANCE	int	160
	BITS	int	16
Iterate	MAX_ITER	int	300